Diseño e Implementación de Herramientas para autómatas programables desarrolladas en sistemas Android

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Electrónica Industrial y Automática

Alumno: Brian Vargas Perera

Tutor: Roberto Luis Marichal Plasencia

ÍNDICE DE FIGURAS

1. RESUMEN

El objetivo de este proyecto ha sido desarrollar un editor/simulador de diagramas *GRAFCET* para dispositivos *Android*. Para ello, hemos partido de una aplicación ya existente, programada para funcionar en computadores, llamada *ULLSIMGRAF* y presentada anteriormente como Proyecto de Fin de Carrera por *Adrián David Estévez Corona* y *José Manuel Royo Hardisson*. Por ello, el problema del que se parte en el presente trabajo reside en la dificultad de adaptar un programa preparado para un ordenador a un sistema móvil. A lo largo de esta memoria, comentaremos algunas de las características que provocan que *Android* destaque sobre el resto de sistemas operativos en el mercado de dispositivos móviles, y que, por ello, se convertirán en una de nuestras referencias a la hora de preparar nuestra aplicación para futuros usuarios.

2. ABSTRACT

The objective of this Project has been to develop a *GRAFCET* diagram editor/simulator for *Android* devices. For this, we have started from an old application developed by former students like a Final Career Project. In the current memory we will do a short review about the history of the industrial automation, the importance of *GRAFCET in* today's industry and we will also comment on the use of *XML* in this kind of projects.

In addition, Android will be one of the main topics in the project. For this reason, it is important to clarify and define all the elements which form the big platform that we know as *Android*.

The main problem of this Project will be to adapt a computer-based software to a mobile device.

3. INTRODUCCIÓN

3.1. LA AUTOMATIZACIÓN EN EL ENTORNO INDUSTRIAL

"La automatización industrial se basa en el uso de sistemas de control, como ordenadores y robots, y tecnologías de información para manejar distintos procesos y maquinarias en una industria con el objetivo de reemplazar al ser humano" **[1]**.

Partiendo de esta definición de la automatización industrial, podemos localizar su origen en plena revolución industrial, creándose en 1771 el primer molino de hilado que funcionó de forma completamente automatizada **[2]**. A pesar de contar con un nacimiento tan temprano, la automatización industrial no cuenta con un gran desarrollo hasta los inicios del siglo XX, cuando, en 1913, *Ford Motor Company* introdujo por primera vez una línea de montaje de producción de automóviles, incrementando de forma exponencial la producción de sus fábricas.

Desde la creación de esa primera línea de montaje, los avances en este terreno han sido bastante notorios, siendo actualmente la automatización de procesos en el entorno industrial una de las bases de la industria contemporánea.

Con el avance del siglo pasado, se han ido sumando numerosos avances. Durante la década de 1930, comienza la introducción de controladores "con capacidad de realizar cambios calculados como respuesta a las desviaciones de una cifra de control" **[2]**. A partir de este momento, Japón pasa a liderar la carrera de la automatización industrial, destacando en el diseño de algunos componentes como microinterruptores, relés de protección o temporizadores eléctricos.

3.1.1. ELEMENTOS DE UN SISTEMA AUTOMATIZADO

3.1.1.1. PARTE DE MANDO

Normalmente, se refiere como parte de mando a la unidad que permite la comunicación entre todos los elementos que constituyen un sistema automatizado, por lo que se le considera el centro de dicho sistema. Suele ser un *PLC*, aunque antes se utilizaban otro tipo de dispositivos, como relés electromagnéticos o tarjetas electrónicas.

- **VENTAJAS DEL USO DE PLC**

"Un Controlador Lógico Programable (*PLC*), es un equipo electrónico, programable en lenguaje no informático, diseñado para controlar en tiempo real y en ambiente de tipo industrial, procesos secuenciales. Un PLC trabaja en base a la información recibida por los captadores y el programa lógico interno, actuando sobre los accionadores de la instalación". **[3]**

Estos *PLC* son bastante sencillos de instalar, gracias a sus pequeñas dimensiones. El uso de *PLC* como parte de mando implica numerosas ventajas para el sistema. Podríamos destacar el aumento de calidad en la monitorización de los procesos automatizados, lo que implica una detección más rápida de los errores durante el procedimiento. También merece la pena nombrar que, a largo plazo, el uso de *PLC* permite el ahorro de costes adicionales, como suponen algunos costos de operación y mantenimiento.

- **APLICACIONES DEL PLC**

Gracias a sus características y a los constantes avances que existen en este ámbito, su campo de aplicación es muy extenso. "Su utilización se da fundamentalmente en aquellas instalaciones en donde es necesario un proceso de maniobra, control, señalización, etc., por tanto, su aplicación abarca desde procesos de fabricación industriales de cualquier tipo a transformaciones industriales, control de instalaciones, etc." **[3]**.

Figura 1. Modelo de PLC

- **MODELOS DE AUTÓMATAS PROGRAMABLES**

Actualmente numerosas empresas desarrollan sus propios PLC, teniendo cada uno una programación específica para sus autómatas. Entre los más usados destacan los desarrollados por empresas como *Siemens*, *Toshiba*, *Koyo* o *General Electric*.

El concepto de programación usado en todos estos autómatas es similar. La diferencia entre ellos radica en la forma de direccionar los dispositivos usados en la programación **[4].**

Como ejemplo, podríamos tomar la familia de *PLCs Simatic S7* de *Siemens*, siendo estos los más utilizados gracias a su simplicidad y a poseer una estructura modular fácilmente ampliable.

3.1.1.2. PARTE OPERATIVA

"La parte operativa es la parte que actúa directamente sobre la máquina. Son los elementos que hacen que la máquina se mueva y realice la operación deseada. Los elementos que forman la parte operativa son los accionadores de las máquinas como motores, cilindros, compresores y los captadores como fotodiodos, finales de carrera, etc." **[3]**.

3.1.2. LA AUTOMATIZACIÓN EN LA INDUSTRIA ACTUAL

Actualmente, a pesar de la gran cantidad de avances con los que cuenta nuestra industria, sigue siendo necesaria la intervención humana en algunas de las etapas de los procesos de manufactura. Dentro de la evolución contemporánea de la automatización cabe destacar la incorporación de sistemas de control por visión y el gran desarrollo de computadores de alta capacidad.

3.1.2.1. INDUSTRIA 4.0

La cuarta revolución industrial, también conocida como *Industria 4.0*, "está cambiando la forma en que los negocios operan y, por lo tanto, los entornos en los que se ven obligados a competir" **[5]**. El objetivo de este nuevo concepto de industria es combinar

versiones más avanzadas de las técnicas de producción con nuevas tecnologías inteligentes que se integran a la producción.

Una de las grandes diferencias con la industria pasada reside en la incorporación de nuevos elementos como la robótica, la inteligencia artificial o la nanotecnología en distintas fases del proceso.

En este proyecto creemos que es de suma importancia resaltar el valor que tiene este nuevo concepto de industria, pues la aplicación que desarrollamos supone además una inclusión completa dentro de la *Industria 4.0*, combinando todas las ventajas previamente comentadas con las altas capacidades y ayudas que aportan los dispositivos móviles, principalmente en fases del proceso que requieren la comunicación o el análisis de los datos obtenidos.

3.1.3. CARACTERÍSTICAS DE LA AUTOMATIZACIÓN INDUSTRIAL

3.1.3.1. ALTA PRODUCTIVIDAD

Gracias a la automatización de las plantas industriales y una menor necesidad de personal para trabajar en ellas, las fábricas multiplican sus horas de producción, incrementando de forma bastante notable su oferta.

3.1.3.2. ALTA SEGURIDAD

La seguridad de los trabajadores y trabajadoras se ve incrementada, gracias a que se automatizan algunas de las etapas más peligrosas del proceso de producción, como el transporte de grandes cargas o fases de trabajo con materiales a altas temperaturas.

3.1.3.3. ALTOS COSTES INICIALES

Aunque a largo plazo la automatización es claramente rentable, la inversión inicial en maquinaria, infraestructuras y personal cualificado es bastante superior a las inversiones requeridas en la industria tradicional.

4. GRAFCET

4.1. ORIGEN DEL MODELADO GRAFCET

La norma *IEC 60848:2013* define *GRAFCET* como "un lenguaje que permite modelar el comportamiento de la parte secuencial de un sistema automatizado" **[7]**. La concepción de *GRAFCET* surge de un modelado gráfico más general, como podrían ser las redes de Petri. Gracias a su sencillez, ha terminado por convertirse en una de las mejores herramientas para representar sistemas automatizados.

GRAFCET es el acrónimo de *Graph Fonctionnel de Commande Etape-Transition* (en español, grafo funcional de control etapa-transición). Tiene su origen en Francia, durante el año 1977, surgiendo como iniciativa de algunos fabricantes de autómatas en acuerdo con organismos oficiales como *AFCET* (*Asociación Francesa para la Cibernética, Economía y Técnica*) y *ADEPA* (*Agencia Nacional para el Desarrollo de la Producción Automatizada*) **[7]**.

"La construcción de un sistema automático requiere, entre otras cosas, establecer relaciones causa/efecto entre los eventos de entrada y las acciones deseadas (salidas). En este contexto, se denomina parte secuencial del sistema la que se circunscribe a las relaciones entre variables entrada y salida de tipo booleano" **[7]**.

4.2. NIVELES DE DIAGRAMAS GRAFCET

Durante el desarrollo de automatismos, se pasa por numerosas fases, partiendo desde un análisis de la viabilidad económica del proyecto hasta un estudio de su diseño e implementación. La transición entre estas fases requiere la adaptación a las necesidades de cada una de ellas. Esto implica que en cada etapa necesitaremos conocer detalles más o menos concretos de nuestro proyecto.

GRAFCET consigue adaptarse a las necesidades previamente nombradas pudiendo organizarse en niveles, de mayor a menor grado de abstracción, en función de la cantidad de detalle que se requiera en la fase actual del desarrollo.

Cabe destacar que, a pesar de existir una definición clara de las características que debe tener el etiquetado del diagrama *GRAFCET* en función del nivel, no existen reglas que

dicten cuál es la representación que se debe escoger en cada etapa del proyecto. De hecho, en la práctica, lo más habitual es combinar los tres tipos de sintaxis para ayudar a la legibilidad de los esquemas en su representación global.

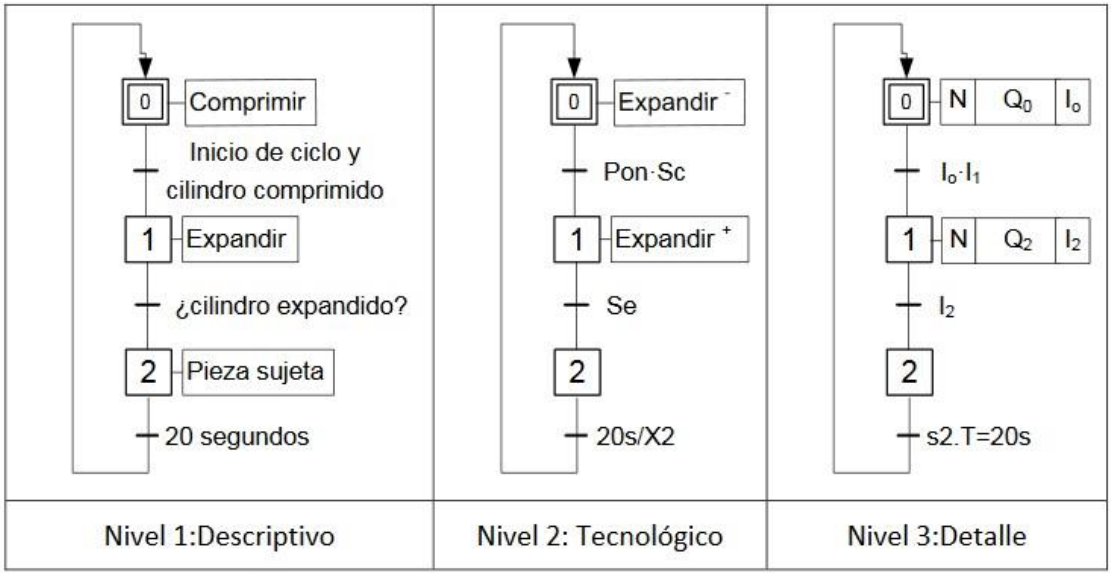

Figura 3-1. Ejemplo de diagrama con distintos Niveles de GRAFCET [7].

4.2.1. GRAFCET DE NIVEL 1

Este primer nivel implica descripciones poco detalladas del automatismo. La ventaja de este nivel reside en que permite entender el funcionamiento general del proceso a automatizar de forma muy rápida. Si recapitulamos, este nivel sería el apropiado para mostrar en las primeras fases del proyecto al tratar con personas ajenas al entorno de la programación, pero que son responsables de la parte económica del proyecto. Por ello, como se puede observar en la ***Figura 2-1***, el lenguaje que se utiliza en las etiquetas asociadas a las etapas y transiciones del diagrama es un lenguaje natural cuyo objetivo es únicamente describir las acciones que van a tener lugar en esas etapas del proceso, sin hacer referencias a los conceptos más técnicos de serán necesarios en fases posteriores.

4.2.2. GRAFCET DE NIVEL 2

Este nivel implica una descripción más técnica de las etapas y transiciones. Lo fundamental para este nivel es que en las etiquetas asociadas queden representados todos los elementos mecánicos y tecnológicos utilizados en la automatización asociada a

nuestro diagrama. La especificación utilizada, tanto en este nivel como en el anterior, es la norma *IEC 60848.*

4.2.3. GRAFCET DE NIVEL 3

Este nivel describe la realización del automatismo esquematizado. Por ello, se utiliza un lenguaje de programación gráfico para este nivel. Es importante incluir junto al diagrama una tabla que relacione los símbolos utilizados en el esquema con los elementos a los que corresponden.

Generalmente, lo más adecuado es seguir la norma *IEC 61131-3.*

4.3. ELEMENTOS BÁSICOS DE UN DIAGRAMA GRAFCET

4.3.1. ETAPAS

Podemos definir un diagrama *GRAFCET* como "una sucesión de etapas y transiciones conectadas entre sí por arcos orientados" **[7]**. Estas etapas están asociadas a una o varias acciones, y se representan con un cuadrado y un número que indica su posición y además la identifica dentro del diagrama.

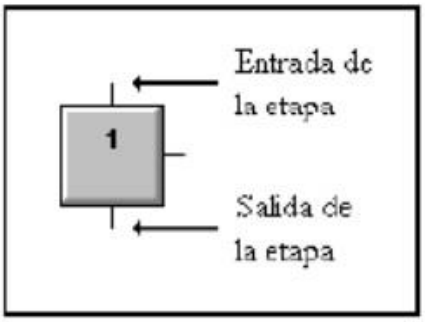

Figura 3-2. Representación gráfica de una etapa [4].

En cualquier diagrama *GRAFCET*, independientemente de su complejidad o extensión, siempre debe definirse una ***etapa inicial***. Entendemos por etapa inicial "al estado de control correspondiente al arranque, el estado inicial del diagrama *GRAFCET*" **[7]**. Cabe destacar que un diagrama puede contar con más de un estado inicial, lo que implica la activación simultánea de todas ellas con el arranque del sistema. Esta etapa inicial siempre

se representa como una etapa normal con un recuadro exterior añadido y el número 0 o 1 asociado a la misma.

Es importante destacar que se debe entender un diagrama *GRAFCET* como un ciclo, por lo que siempre tiene que existir una etapa final que conecte de nuevo con la etapa inicial, haciendo así referencia al apagado de la estación automatizada.

4.3.2. TRANSICIONES

"Una transición representa la condición por la que el sistema evoluciona de las etapas que la preceden a las etapas que la suceden" **[7]**. Su representación gráfica es bastante simple, basada en "una barra horizontal que corta transversalmente al enlace entre etapas denominado arco" **[7]**. Al igual que las etapas llevan asociadas acciones, las transiciones quedarán ligadas a condiciones booleanas, es decir, que darán como respuesta valores 0 o 1 (por ejemplo, en el caso de sensores, activo o inactivo), que definirán el paso de una etapa a la siguiente. Las condiciones asociadas a las transiciones del diagrama son denominadas como ***receptividad***.

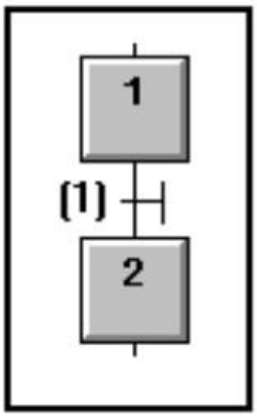

Figura 3-3. Representación de una transición asociada a una etapa [4].

Una transición quedará ***validada*** cuando se activan las etapas que la anteceden. Si la transición es receptiva, es decir, se cumplen las condiciones que tiene asociadas, entonces se activará, permitiendo el avance en la simulación del diagrama a las etapas posteriores, lo que implica la desactivación de las etapas previas.

4.3.3. ARCOS

"Los arcos vinculan etapas con transiciones, pero nunca etapas con etapas, o transiciones con transiciones. Son enlaces orientados que definen una relación de orden entre etapas y transiciones" **[7]**. Por lo general, siempre se lee el diagrama *GRAFCET* de forma descendente. Esto implica que en caso de que el arco tenga sentido ascendente su sentido deberá ser indicado obligatoriamente con una flecha.

4.3.4. ACCIONES

Las etapas pueden tener asociadas varias acciones, o, en algunos casos, ninguna. En caso de que una etapa no tenga ninguna acción asociada, será denominada como ***etapa de espera***, empleada "para representar una ausencia de evolución mientras que la transición que sucede a la etapa no sea receptiva" **[7]**.

Cuando la etapa tiene múltiples acciones asociadas, la interpretación que debe hacerse es que todas estas acciones deben de ejecutarse de forma concurrente cuando la etapa esté activa.

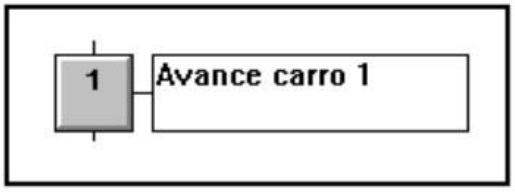

Figura 3-4. Representación de una única acción asociada a una etapa [4].

En el caso de las etapas con múltiples acciones, existen numerosas formas válidas de representarlas.

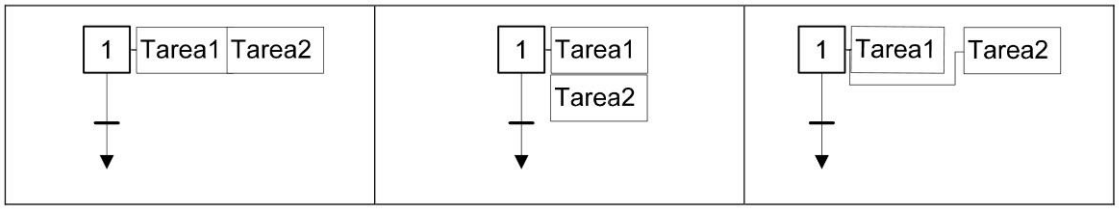

Figura 3-5. Representaciones de múltiples acciones asociadas a una misma etapa [4].

Hasta ahora, únicamente hemos hablado de la ***acción continua***, es decir, una acción que comienza junto a la activación de la etapa y termina cuando se desactiva la misma.

Aparte de este tipo de acciones, existen otros tipos de relaciones etapa-acción:

- **Acciones retardadas**

Este tipo de acciones comienzan un tiempo después de la activación de la etapa.

- **Acciones limitadas en el tiempo**

Al contrario de las acciones retardadas, estas acciones comienzan con la activación de la etapa y finalizan tras un tiempo definido, aunque la etapa continúe estando activa.

- **Acciones impulsionales**

"La acción dura el ciclo de operación de la activación de la etapa. Se emplea para acciones de control endógenas" **[7]**.

- **Acciones memorizadas**

"La acción se enclava tras la activación de la etapa y perdura tras su desactivación. Será necesario una etapa posterior para desenclavarla" **[7]**.

	Símbolo	Descripción
1	ninguno	acción continua mientras dura la etapa
2	N	acción continua mientras dura la etapa
3	R	desenclavamiento de la acción
4	S	enclavamiento de la acción
5	L	acción limitada tras la activación de la etapa
6	D	acción retardada tras la activación de la etapa
7	P	flanco de activación de la etapa
8	SD	acción memorizada y retardada
9	DS	acción retardada y memorizada
10	SL	acción memorizada y limitada en el tiempo
11	P1	flanco de activación de la etapa
12	P0	flanco de desactivación de la etapa

Tabla 3-1. Calificadores de acciones previstos en la norma [7].

4.4. ESTRUCTURAS DE UN DIAGRAMA GRAFCET

La versión más simple de un diagrama *GRAFCET* se define como un esquema lineal de etapas conectadas mediante transiciones. Sin embargo, debido a la complejidad de los procesos a automatizar, muchas veces necesitaremos otro tipo de estructuras que se adapten a las necesidades del programador.

- **Divergencia concurrente (Y)**

"Expresa un punto de sincronismo y el inicio simultáneo de un número de estructuras serie" **[7]**. Este tipo de estructura es representada con una doble línea horizontal de la que nacerán las distintas etapas y estructuras correspondientes.

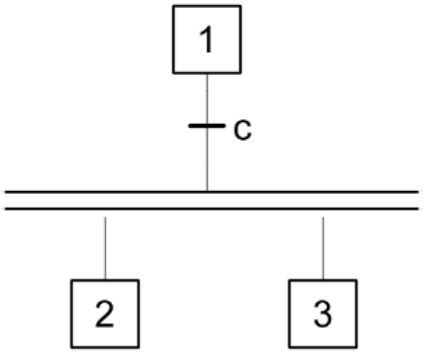

Figura 3-6. Representación de una Divergencia Y [7].

- **Divergencia alternativa (O)**

"Expresa selección de acciones de control alternativas en función de eventos" **[7]**. A diferencia de una divergencia concurrente, en este tipo de estructura nos encontraremos con una división en múltiples transiciones en lugar de etapas.

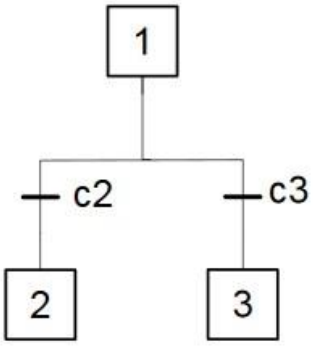

Figura 3-7. Representación de una Divergencia O [7].

- **Convergencia concurrente (Y)**

La convergencia concurrente define la finalización y el sincronismo de todas las estructuras en las que se había ramificado el diagrama al inicio de la divergencia concurrente. Por ello se representará como una divergencia Y invertida.

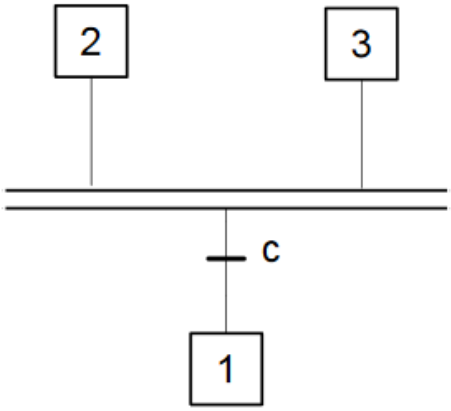

Figura 3-8. Representación de una Convergencia Y [7].

- **Convergencia alternativa (O)**

Al igual que ocurre con las convergencias concurrentes, una convergencia alternativa representa el fin de los procesos en los que se había divido el diagrama tras una divergencia alternativa.

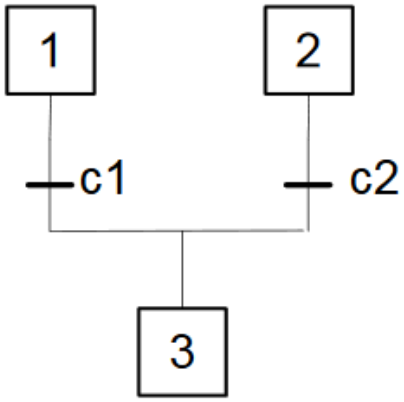

Figura 3-9. Representación de una Convergencia O [7].

- **Saltos condicionales**

Podemos definir un salto condicional como un caso especial de divergencia alternativa. En este caso, si se valida un evento concreto, la acción de control puede saltarse varias etapas conectadas entre sí. Por ello, los arcos que representan estos saltos no tienen etapas asociadas de forma clara.

Debemos destacar que los saltos condicionales pueden producirse en ambos sentidos, tanto hacia etapas posteriores como previas. Por ello, siempre es importante marcar con una flecha el sentido del salto condicional.

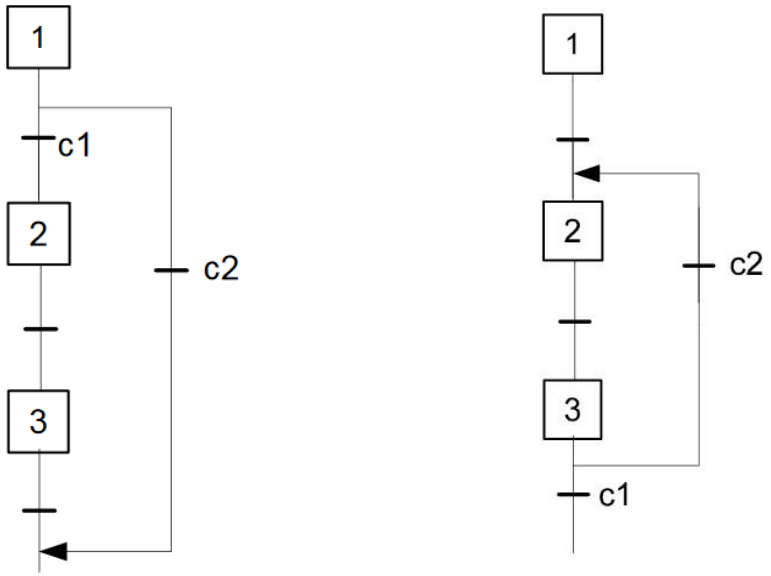

Figura 3-10. Representación de distintos tipos de saltos condicionales en GRAFCET [7].

4.5. REGLAS DE UN DIAGRAMA GRAFCET

GRAFCET tiene una serie de reglas que se utilizan para "describir la dinámica del automatismo modelado" **[7]**.

- **Regla de incio**

"El arranque del sistema supone la activación de todas las etapas iniciales y solamente éstas" **[7]**. Por ello, el estado inicial de cualquier diagrama GRAFCET modela el inicio del sistema a y su control. El estado inicial también suele corresponder con el estado de reposo o parada segura, definiendo así el estado en que debe encontrarse el sistema para

su puesta en marcha. Una de las medidas de seguridad que suele tomarse, es incluir una acción en la etapa inicial que se encargue de comprobar que el sistema se encuentra en su estado inicial.

- **Regla de evolución de una transición**

"Una transición franqueable deber ser inmediatamente franqueada" **[7]**. Una transición quedará validada cuando todas las etapas que la preceden estén activas.

- **Regla de evolución de las etapas activas**

"el franqueo de una transición supone la activación simultánea de todas las etapas inmediatamente posteriores y la desactivación simultánea de todas las etapas inmediatamente anteriores" **[7]**.

- **Regla de franqueamiento simultáneo**

"Todas las transiciones franqueables se franquearán inmediata y simultáneamente". Esta regla es de gran importancia gracias a que "permite definir la evolución de *GRAFCETs* estructurados complejos compuestos por otros *GRAFCETs*, macroetapas, etc" **[7]**.

- **Regla de prioridad de etapa activa**

"Si la evolución de un *GRAFCET* (debido a las reglas anteriores) implica la activación y desactivación simultánea de una etapa, esta deberá permanecer activa" **[7]**.

Esta regla se define para solventar algunos problemas que pueden surgir durante una simulación del diagrama. Un ejemplo lo podríamos encontrar en un esquema lineal, donde dos etapas consecutivas permanecen activas debido a que sus transiciones se verifican de manera simultánea. Sin aplicar esta regla podríamos ser conducidos a errores, como activar y desactivar de forma simultánea la segunda etapa. Por ello, para evitar estos conflictos, la segunda etapa deberá permanecer activa.

Todas estas reglas vienen descritas en la norma *IEC60848,* donde también se añaden otras consideraciones:

- "Cuando el franqueo de una transición conlleva la activación simultánea de varias etapas, las secuencias a las que pertenecen evolucionan posteriormente de manera independiente" **[7]**.
- "En el plano operativo, el tiempo que se tarda en franquear una transición se puede considerar tan corto como se quiera, pero nunca es cero" **[7]**.

5. ANDROID

5.1. ¿QUÉ ES ANDROID?

Android es, actualmente, el sistema operativo más utilizado del mundo en dispositivos móviles. Por sistema operativo (SO) entendemos "un programa que cuando arrancamos o iniciamos el ordenador se encarga de gestionar todos los recursos del sistema informático, tanto del hardware (partes físicas, disco duro, pantalla, etc.) como del software (programas e instrucciones), permitiendo así la comunicación entre el usuario y el ordenador" **[7]**. El sistema operativo permite al usuario ejecutar otras aplicaciones en el dispositivo, por lo que será uno de los pilares centrales de nuestro proyecto.

Android surgió durante el año 2003 como una empresa centrada en el desarrollo de software para teléfonos móviles. Esta empresa fue creada por *Andy Rubin*, *Rich Miner*, *Nick Sears* y *Chris White*. Durante sus primeros años continuaron con las mismas tareas, hasta que Google comienza a integrarse en las tecnologías móviles.

Unos años después de la creación de *Android Inc.*, concretamente el 5 de noviembre de 2007, se funda una asociación de numerosas empresas especializadas en tecnologías móviles llamada *OHA* (*Open Handset Alliance*) y liderada por *Google*. Este mismo día se dio a conocer de forma pública el sistema operativo de dispositivos móviles más popular de nuestro tiempo, *Android*, "una plataforma de código abierto para móviles que se presentaba con la garantía de estar basada en el sistema operativo *Linux*" **[9]**.

Figura 4-1. Logo oficial de Android [10]

Tras su anunciamiento, tuvimos que esperar hasta octubre de 2008 para poder ver de forma oficial en el mercado un dispositivo usando la primera versión de Android. Estos

primeros móviles eran muy distintos del tipo de terminales que utilizamos actualmente. Aunque ahora pueden parecer mecánicas desfasadas y obsoletas, esta primera versión supuso un cambio de paradigma para este tipo de tecnologías, incorporando numerosas novedades. Entre todas ellas, hay algunas que vale la pena destacar:

- **Ventana de notificación desplegable**

Una de las principales que incorpora *Android* es la posibilidad de tener un sistema de notificación que permita tener toda la información a la vista.

- **Widgets en la pantalla de inicio**

Como herencia de los ordenadores personales y su escritorio, encontramos en *Android* una pantalla de inicio fija definida, donde poder anclar los accesos directos a las aplicaciones más usadas. En esta característica podemos encontrar uno de los antecedentes al cambio de concepto respecto al uso de los dispositivos móviles, dejando de ser un terminal utilizado únicamente para funciones básicas, como las llamadas y la mensajería, para convertirse en un pequeño ordenador de bolsillo.

- **Integración de Gmail**

Gracias al trabajo con Google y la implementación de una aplicación dedicada a Gmail, Android 1.0 aportó el mejor soporte para correos electrónicos en dispositivos móviles del mercado.

- **Android Market**

En sus inicios, la oferta de aplicaciones que existía en la ***store*** propia de Android era muy limitada, algo que ha cambiado con el tiempo, llegando a tener actualmente más de dos millones de aplicaciones para elegir **[10]**.

5.2. LA EVOLUCIÓN DE ANDROID

La mejor manera de observar la evolución que ha sufrido este sistema operativo es a través de sus versiones y actualizaciones más importantes.

5.2.1. ANDROID 1.5 CUPCAKE

Esta es la primera versión en la que comienzan a aparecer cambios más allá de la mera corrección de errores. Entre estas primeras modificaciones destacan algunos de los cambios referidos a la interfaz, como la sustitución del logo del navegador u otros cambios en las barras de menú y buscadores.

Quizás el cambio más significativo sea la incorporación de teclados virtuales. Los primeros dispositivos con Android 1.0 poseían teclado físico, por lo que esta función nunca fue necesaria. Una de las diferencias que encontramos aquí con sus principales competidores, iOS y Windows pone, es que Android siempre ha permitido el desarrollo de teclados virtuales por parte de terceros, algo que aun a día de hoy continúa suponiendo una diferencia con ellos **[9]**.

Con esta versión de Android también comienzan a estar disponibles Widgets de terceros para su sistema operativo.

Por último, de esta actualización destacamos la posibilidad de grabar y reproducir vídeos, función que no estuvo disponible hasta esta versión y por la que los fabricantes comienzan a crear software específico para las cámaras móviles, añadiendo escenas y filtros.

5.2.2. ANDROID 2.0

Esta versión de *Android* supuso un gran cambio en el sistema, además de un aumento en su popularidad. Agregaron el apoyo a varias cuentas, permitiendo la posibilidad de gestionar múltiples correos electrónicos y listas de contactos en el mismo dispositivo. Este es el momento en el que comienza también la sincronización de contactos gracias a la comunicación entre distintos tipos de cuenta **[9]**.

Una de las funciones más utilizada por algunos usuarios también fue implementada en esta versión de *Android*. Nos referimos a la transcripción de voz a texto. A partir de esta

actualización los usuarios podían dictar mensajes a su teléfono para que fueran transcritos de forma inmediata.

- **Google Maps**

Google Maps se presentó de forma oficial junto a *Android 2.0* y supuso el primer paso en la implementación de sistemas de navegación en dispositivos móviles.

Por último, también hubo cambios en su interfaz, permitiendo añadir fondos de pantalla animados y añadiendo una pantalla de bloqueo al dispositivo.

5.2.3. ANDROID 3.0 HONEYCOMB

Esta versión de *Android* es la primera que se implementa no solo en móviles, sino también en tablets. Aparte de los típicos cambios y rediseños de interfaz, dio lugar a cambios realmente importantes en la industria móvil.

El principal cambio que destacaremos de esta actualización es la sustitución de botones dedicados en el propio terminal por botones virtuales, situados en la parte inferior de la pantalla. A partir de este punto comienzan a perder importancia los botones físicos y comienza el crecimiento exponencial de dispositivos con frontales enteramente táctiles.

5.2.4. ANDROID 4.0 ICE CREAM SANDWICH

Por último, destacaremos esta versión, pues con ella se potencia el uso de tecnología *NFC*. Aunque ya había sido promocionado con anterioridad (*Android 2.3*), es en esta actualización donde toma más protagonismo y se comienza a invertir en aplicaciones compatibles con dicha tecnología.

También implicó innovaciones como el desbloqueo por reconocimiento facial o una aplicación capaz de analizar y desglosar el consumo de datos móviles por parte del usuario.

5.3. ANDROID EN LA ACTUALIDAD

Después de todo, *Android* ha continuado con su evolución a través de múltiples actualizaciones, llegando ahora su última versión, *Android 10*, siendo lanzada de forma oficial a los primeros dispositivos el 3 de septiembre de 2019 **[11]**.

En esta nueva versión volveremos a encontrar cambios visuales, como la adición de nuevos temas para el diseño de la interfaz e iconos para las aplicaciones de mensajería.

Para los usuarios más entusiastas, llegan novedades en el apartado fotográfico, en el rendimiento de la memoria y sobre todo en la accesibilidad gracias a funciones *como **Live Caption***, que permite subtitular vídeos en tiempo real.

Actualmente, *Android* sigue siendo el sistema operativo más popular entre los dispositivos móviles.

5.3.1. ¿POR QUÉ ES ANDROID TAN POPULAR?

Que *Android* sea la elección principal tanto para fabricantes como para consumidores se debe a numerosas características, tanto referidas al funcionamiento del sistema operativo como a su servicio.

- **SOFTWARE LIBRE Y GRATUITO**

"Software libre es el software que respeta la libertad de los usuarios y la comunidad. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar, y mejorar el software" **[12]**. Esta es una característica de vital importancia, ya que abre el sistema a una cantidad ilimitada de desarrolladores. Combinado con esto, que sea gratuito lo convierte en la primera opción para la mayoría de empresas, pudiendo reducir de este modo los costos de producción.

Android ofrece un *SDK* (Kit de Desarrollo de Software) y un entorno de desarrollo propios, además de una página oficial con una extensa lista de guías y publicaciones para que todas aquellas personas interesadas en el desarrollo de aplicaciones para dispositivos móviles puedan comenzar con su programación en cualquier momento.

- **VARIEDAD DE HARDWARE**

Gracias a lo difundido que se encuentra Android entre los fabricantes, podemos encontrar terminales de cualquier gama, capaces de adaptarse al gusto de todos los consumidores y consumidoras.

- **ALTO NIVEL DE PERSONALIZACIÓN**

A diferencia de lo que ocurre con otros sistemas operativos, con *Android* los desarrolladores son libres de modificar la versión básica del software, pudiendo añadir infinitas capas de personalización y ofreciendo experiencias completamente distintas en función de la marca y el terminal.

- **INTEGRACIÓN CON SERVICIOS DE GOOGLE**

Al ser propiedad de *Google*, algunos de los servicios más consumidos por los usuarios generales vienen incorporados de serie en los dispositivos. Además, tener el respaldo de una empresa tecnológica tan grande implica contar con algunas de las novedades más vanguardistas del momento.

5.4. PROGRAMANDO PARA ANDROID: KOTLIN

Una de las ventajas con las que partía *Android* en sus inicios es que su lenguaje de programación nativo es *Java*, un leguaje muy extendido entre la comunidad de programadores, lo que facilita al público la posibilidad de programar para la plataforma.

Sin embargo, desde hace aproximadamente dos años, *Android* ha comenzado a promover el uso de un nuevo lenguaje llamado *Kotlin*, estableciéndolo como lenguaje oficial para su plataforma al mismo nivel que *Java*.

5.4.1. BREVE INTRODUCCIÓN A KOTLIN

Kotlin comenzó a desarrollarse durante 2011 por *JetBrains*, una empresa de desarrollo de software orientada a ayudar a los desarrolladores a trabajar de una forma más eficiente mediante la automatización de tareas repetitivas, permitiéndoles centrarse en el diseño de sus códigos **[15]**.

Acompañando a la filosofía de *Android*, *Kotlin* también nació como un proyecto de software libre, bajo licencia *Apache 2.0*.

Este lenguaje presenta algunas ventajas:

- **Es un lenguaje sencillo y pragmático**
- **Es totalmente interpolable con *Java***, por lo que cualquier código escrito en *Java* puede utilizarse directamente en *Kotlin*.
- **Se trata de un lenguaje ligero, preparado para ser ejecutado en todo tipo de dispositivos**

Figura 4-2 Logo oficial del lenguaje *Kotlin*.

Como se puede observar en la página oficial para desarrolladores de *Android*, el apoyo que da la empresa a este nuevo lenguaje es bastante firme, definiéndolo como un lenguaje "moderno y expresivo", que permite el desarrollo de aplicaciones de mayor calidad y más seguras.

Por último, debemos aclarar que, a pesar de lo interesante que hubiera sido programar nuestro editor con este novedoso lenguaje, hemos decidido mantener *Java*, el lenguaje utilizado en el programa original, para aprovechar al máximo el código desarrollado previamente, centrando el *port* de la aplicación en los conflictos fundamentales que surgen al moverse de una plataforma a otra.

6. XML

6.1. INTRODUCCIÓN A XML

XML (*Extensive Markup Language*) es un lenguaje de etiquetado utilizado para describir datos. Este lenguaje ofrece un estándar para representar datos de texto, como por ejemplo documentos o listas de cálculos **[13]** Es importante destacar que los datos recopilados en un fichero *XML* no realizan ninguna función por sí solos, por lo que se necesita acompañar su uso de sofwtare especializado en el procesamiento de datos.

La *World Wide Web Consortium* (*W3C*) estableció en 1998 *XML* como estándar dentro de los lenguajes de etiquetado, algo que influenció de forma fundamental la aceptación de este nuevo formato entre los desarrolladores.

El uso de *XML* supone numerosas ventajas respecto a otros lenguajes:

- ***XML* es un estándar en el entorno industrial**

Al no estar desarrollado por una empresa específica para un medio concreto, la difusión de *XML* fue mucho mayor que la de otras alternativas pertenecientes a empresas.

- ***XML* es ampliable**

A diferencia de otros lenguajes de etiquetado como *HTML*, *XML* permite al usuario definir sus propias etiquetas, evitando poner límites al desarrollador.

- ***XML* es fácil de procesar**

Anteriormente se utilizaban otros formatos para representar y enviar la información. El problema que presentaban dichos formatos reside en la dificultad para procesar los datos. Sin embargo, como comentamos antes, gracias a su sistema de etiquetado y a que puede ser leído fácilmente por software específico de procesado de datos, la información compartida en formato *XML* es muy fácil de difundir.

6.2. LIBRERÍA JDOM

JDOM es una librería dedicada a desarrollar una representación para *Java* de un documento *XML*. Ofrece una forma de representar estos documentos para proporcionar una lectura, escritura y manipulación fácil y eficiente.

El trabajo del que parte el presente proyecto tiene como base el uso de esta librería para gestionar de una forma más sencilla la parte *XML* del programa. Destacamos desde ahora su importancia y la compatibilidad con *Android*, la plataforma en la que desarrollaremos nuestro editor.

6.3. APLICACIONES DE XML EN ANDROID

Una prueba de la importancia que mantiene hoy en día el estándar *XML* es su relación con *Android*.

Dentro de esta plataforma, su uso es fundamental. El diseño todo el apartado visual, desde los menús hasta los botones queda definido por el sistema de etiquetado *XML*.

Como podemos observar en la **Figura 5-1**, el estándar *XML* permite que generar una estructura bastante clara, con una estricta jerarquía, permitiendo distinguir cada uno de los elementos definidos en el documento con facilidad.

Fundamentalmente debemos destacar la importancia de establecer un elemento padre, comúnmente llamado **nodo raíz,** "que formará el cuerpo del documento, y que englobará al resto de elementos hijos declarados" **[16]**. En el ejemplo proporcionado, como se puede observar, nuestro nodo raíz sería el propio ***relative layout***, el cual pararemos a definir más adelante.

A continuación, dentro de este nodo raíz quedarán contenidos todos los elementos hijos que queramos declarar dentro del documento. Estos serán llamados elementos hijos o ***nodos rama***. Los atributos asociados a cada uno de los componentes de la interfaz estarán definidos dentro del nodo, tal y como se observa en la figura adjunta.

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

    <include
        android:id="@+id/toolbar_dialog"
        layout="@layout/toolbar_dialog">

    </include>

    <TextView
        android:id="@+id/selectsimulation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginTop="80dp"
        android:fontFamily="@font/roboto_light"
        android:text="Seleccione un modo de simulación antes de continuar."
        android:textSize="16dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/toolbar_dialog">

    </TextView>

    <ImageView
        android:layout_width="24dp"
        android:layout_height="24dp"
        android:layout_alignParentEnd="true"
        android:layout_alignParentTop="true"
        android:layout_marginEnd="16dp"
        android:layout_marginTop="20dp"
        app:srcCompat="@drawable/baseline_warning_white_24">

    </ImageView>


</android.widget.RelativeLayout>
```

Figura 5-1. Ejemplo de estructura de un layout desarrollado en Android.

Es importante destacar que *XML*, como cualquier otro lenguaje de programación actual, permite el uso de comentarios en sus líneas de código para clarificar aún más el significado del documento.

7. ULLSIMGRAF

A continuación, adjuntaremos un breve resumen de *ULLSIMGRAF*, la aplicación en la que se basa el presente proyecto, para posteriormente hacer un análisis de los cambios necesarios para que este editor sea compatible con dispositivos móviles.

Tal y como comentan los desarrolladores, la aplicación queda estructurada en dos grandes partes claramente diferenciadas. Estas dos partes son, en esencia, la estructura de datos y la interfaz gráfica.

7.1. ESTRUCTURA DE DATOS

Esta parte corresponde a la base de la aplicación. "Consistirá en todo lo relacionado con el manejo de los datos, así como todas las operaciones que se requieran para el mantenimiento de esta" **[4]**. En esta sección del trabajo es donde mis antiguos compañeros utilizaron el estándar *XML* para estructurar de forma ordenada toda la información.

Algo que caracteriza a la estructura de datos, es que "no será visible para el usuario final de la aplicación, pero será fundamental para el buen funcionamiento de esta". Por ello, otra función de esta fracción del programa será comunicarse con la interfaz, y ya, a través de ella, con el usuario final.

Para mantener y modificar una estructura de datos, existe una serie de operaciones que es importante destacar:

- **Búsqueda de elementos en la estructura desarrollada.**
- **Inserción de elementos nuevos.**
- **Modificación o eliminación de elementos ya existentes.**

El uso de *XML* aporta numerosas ventajas en esta parte del proyecto. Gracias a representar toda la información de una forma bien estructurada y al uso de etiquetas, el acceso a partes concretas del documento queda claramente simplificado.

Tal y como comentan, en este trabajo aprovechamos la capacidad de *XML* de ser extensible pudiendo "implementar nuestros propios métodos y funciones en base a los ya creados en librerías, reduciendo notablemente el tiempo en la programación" **[4]**.

Otra ventaja importante a destacar, es que, gracias a *XML*, "podemos editar y cambiar la información guardada en un fichero de la aplicación, tan solo utilizando un editor de texto" **[4]**.

```
<Raiz>
        <Elemento Tipo=" " Id=" " />
        <Elemento Tipo=" " Id=" " />
        <Elemento_Grafico Tipo="Inicio" Id=" " >
            <Rama Tipo="Tipo" Id=" " >
                <Elemento Tipo=" " Id=" " />
                <Elemento Tipo=" " Id=" " />
                <Elemento_Grafico Tipo="Linea" />
            </Rama>
            <Rama Tipo="Tipo" Id=" " >
              ..........
            </Rama>
        <Elemento_Grafico Tipo="Final" Id=" " >
        <Elemento Tipo=" " Id=" " />
        <Elemento Tipo=" " Id=" " />
          ..........
        <Elemento_Grafico Tipo="End" Id=" " />
```

Figura 6-1. Ejemplo de estructura generada XML [4].

En el ejemplo que acompaña a este apartado (**Figura 6-1**), el nodo raíz corresponde a la propia raíz del documento generado, estableciéndola como padre de todo el diagrama *GRAFCET* que representaremos con el editor. A continuación, observamos distintos elementos que equivaldrían a etapas y transiciones. Al añadir ramas se complica el esquema generado, pero gracias al estándar *XML* continúa siendo muy intuitivo. Con las nuevas ramas, se definen nuevos padres, que tendrían una función similar al nodo raíz, esto es, contener nuevos elementos que se incluyan dentro de las distintas ramas creadas.

Por último, destacaremos la presencia en el esquema de otro tipo de etiquetas. Nos referimos concretamente a "Elemento_Gráfico". La utilidad que posee esta etiqueta es indicar el inicio y fin de una estructura concurrente (estructura Y) o alternativa (estructura O).

7.2. INTERFAZ GRÁFICA

"Comprende el conjunto de ventanas gráficas y programación orientada a la interacción del usuario con la Aplicación" **[4]**. Como esta parte es la encargada de comunicarse directamente con el usuario, debe reunir todas las opciones y herramientas utilizadas en el programa para editar o modificar el futuro diagrama *GRAFCET*, además de los posibles mensajes de error o precaución que puedan llegar a producirse.

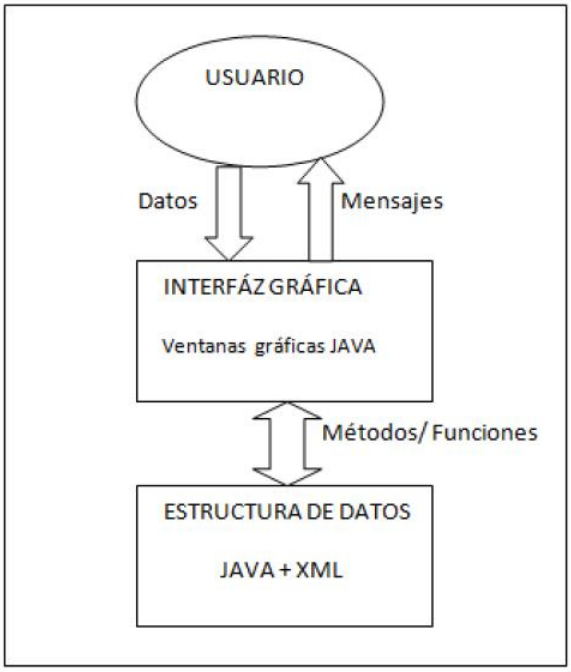

Figura 6-2. Estructura básica de ULLSIMGRAF

Como ellos mismos comentan, dividir la aplicación tal y como se observa en la imagen "permitiría, en el caso de ser necesario, cambiar por completo una de las partes sin que la otra se viera afectada en su funcionamiento, cambiando los métodos y funciones que comunican ambas partes" **[4]**.

7.3. FUNCIONES DE LA APP

7.3.1. CLASE ELEMENTO

Esta clase es fundamental para el desarrollo del diagrama, y, por ello, contiene tanto información de la parte gráfica como de los datos propios de la estructura interna del programa.

```
import org.jdom2.Element;

public class Elemento extends Element {

    public Elemento( String Tipo, int Id, int x, int y, String Etiq1, String Etiq2, int Act, String Salto_cond){

        super( name: "Elemento");
// Tenemos tres tipos de elementos: estados, transiciones y saltos condicionales
        setAttribute( name: "Tipo", Tipo);
        setAttribute( name: "Id", Integer.toString(Id));
// Con las coordenadas asociadas a cada elemento, podremos identificarlo en la pantalla del dispositivo
        setAttribute( name: "x", Integer.toString(x));
        setAttribute( name: "y", Integer.toString(y));
        setAttribute( name: "Etiq1", Etiq1);
        setAttribute( name: "Etiq2", Etiq2);
// Variable que indica si el elemento se encuentra seleccionado
        setAttribute( name: "Activacion", Integer.toString(Act));
        setAttribute( name: "Condicional", Salto_cond);   // Indica en una transición o estado  si existe un salto condicional
        setAttribute( name: "Destino_cond", Salto_cond);
/* Se usa en un salto condicional de la siguiente manera:
        1) En un Salto_condic: se guarda el "Id" del Estado al que se salta
        2) En un Estado, como podemos tener varios saltos condicionales que van a un mismo Estado,
        meteremos en un String los identificadores de las transiciones de la manera siguiente:
        "Id_tran1/Id_tran2/.../Id_tranN/" */
        setAttribute( name: "Compo_entradas",  value: ""); // En una Transición, crea la composición de entradas con las Etiquetas Nivel2
        setAttribute( name: "Contadores",  value: "");
/* Aqui insertaremos los Contadores que pudiera tener el Estado asociado para ello:
        podemos tener las siguientes acciones y si tenemos varias estarán separadas por una barra '/':
        1) Reseteo de un contador     R(CT"Id")
        2) Incremento de un contador  CT"Id"+
        3) Decrementar un contador    CT"Id"-
        4) Crear un contador          CT"Id"=Valor */
        setAttribute( name: "Temporizadores",  value: "");

    }
}
```

Figura 6-3. Clase ELEMENTO del programa.

Por lo que podemos observar en la **Figura 6-3**, en esta clase podemos definir tres tipos de elemento según el valor que tome la primera variable, llamada "***Tipo***". Estos pueden ser, etapas, transiciones o saltos condicionales. El resto de variables primarias hacen referencia a las coordenadas y etiquetas asociadas a cada elemento, permitiendo así su identificación en la pantalla de nuestro dispositivo.

También destacaremos la utilidad de otras variables como ***"Activacion"*** o ***"Destino_cond"***, las cuales nos permiten saber si hay algún salto condicional asociado a la etapa o si el elemento ha sido pulsado en pantalla.

Por último, recordamos que existe otra clase, llamada ***"Elemento_Grafico"***, que únicamente se utiliza para identificar el inicio y fin de estructuras divergentes o alternativas, por lo que no dedicaremos un apartado específico a su explicación. Queremos destacar, eso sí, que, al igual que la presente clase, también posee una variable llamada ***"Tipo"***, utilizada principalmente para distinguir las estructuras. Los valores que puede tomar dicha variable varían entre cuatro: ***InicioO***, ***FinalO***, ***InicioY*** y ***finalY***.

Para diagramas más complejos que contienen varias de estas estructuras combinadas, se incluye una variable que contiene un valor entero para identificar en qué nivel de profundidad nos encontramos. Como ejemplo, diremos que la línea principal del diagrama, es decir, la que nace directamente del nodo raíz, tiene asignada el valor 0.

7.3.2. BÚSQUEDA DE ELEMENTOS POR POSICIÓN

Con esta función podemos observar la necesidad de comunicación de datos entre las dos grandes partes que definimos previamente (parte de datos y parte gráfica). Será la forma que tendremos de acceder al contenido de los elementos definidos en el diagrama *GRAFCET*.

El proceso que se lleva a cabo es el siguiente:

1. El usuario cliquea sobre el área de trabajo. En este momento, el programa captura las coordenadas del punto donde se produjo el clic.
2. Tras conocer las coordenadas, se envían a la estructura de datos para ejecutar el método correspondiente. Esta acción comprobará si las coordenadas coinciden con las de algún elemento del diagrama. El procedimiento en este momento es muy sencillo. Como se observa en la **Figura6-4**, el programa "dibujará" un círculo de radio predefinido centrado en el punto donde el usuario hizo clic. A continuación, se comprobará si el círculo dibujado contiene coordenadas correspondientes a alguno de los elementos del diagrama. Para ello, el programa analizará y comparará todas las coordenadas de los elementos de la estructura de datos. Para evitar errores en este procedimiento, es importante escoger

correctamente el radio del circulo, así evitaremos que pueda contener coordenadas de varios elementos a la vez.

```
public boolean ecuacionDeCirculo(Point punto, int Element_x, int Element_y){

    int RADIO = (Dist_y/2);
    return (((punto.x - Element_x) * (punto.x - Element_x) + (punto.y - Element_y) * (punto.y - Element_y)) <= (RADIO * RADIO));
}

public boolean Buscar_Elemento(Element Padre, Point punto){

    boolean Encontrado = false;

    List Elementos=Padre.getChildren();
    Iterator i = Elementos.iterator();
    while ((i.hasNext())&&(!Encontrado)){
        Element e= (Element)i.next();
        String Tipo_Element =e.getAttributeValue( attname: "Tipo");
        if ((Tipo_Element.contains("Estado")) || (Tipo_Element.contains("Transicion")) || Tipo_Element.contains("Salto_condic")) {
            int Pos_x =Integer.parseInt(e.getAttributeValue( attname: "x"));
            int Pos_y =Integer.parseInt(e.getAttributeValue( attname: "y"));
            Encontrado = ecuacionDeCirculo(punto, Pos_x, Pos_y);
            if (Encontrado){
                Element_Encont = e; // Si se encuentra, devuelve ese Elemento en la variable Global "Element_Encont"
            }
        }
        List Hijos =e.getChildren();
        if (Hijos.size() != 0){
            Encontrado = Buscar_Elemento(e, punto); //Busqueda recursiva si tiene hijos
        }

    }
    return(Encontrado);
}
```

Figura 6-4. Definición de las clases encargadas de buscar un elemento al clicar.

3. Llegados a este punto, tras el análisis de la estructura de datos, el programa puede devolvernos dos valores. Por un lado, no ha coincidido ninguno de los elementos del diagrama, por lo que no ejecutará ninguna otra acción. Por otro lado, si el círculo dibujado contiene las coordenadas de algún elemento, se asignará dicho elemento a la variable llamada "***Element_Encont***", tal y como observamos en la **Figura6-4**.
4. Por último, vuelve a producirse la comunicación entre la estructura de datos y la parte gráfica. Una vez almacenado el elemento en la variable correspondiente, se ejecutará una acción encargada de cambiar el valor de la variable ***"Activacion"*** de dicho elemento de 0 a 1. Con este cambio el programa ya interpreta que la etapa, transición o salto condicional (dependiendo del tipo de elemento) ha sido

seleccionado. En este punto la parte gráfica se encargará de actualizar el área de trabajo para repintar el esquema con el elemento seleccionado dibujado del color que corresponda.

Para terminar con esta función, es importante destacar lo que ocurre después de que haya un elemento seleccionado si el usuario vuelve a cliquear en el área de trabajo. Por un lado, si no se detecta ningún nuevo elemento, igual que como comentamos anteriormente, no se ejecuta ninguna acción. Lo importante en este punto es que se mantiene activo el último elemento que había sido activado. Por otro lado, en caso de que se detecte otro elemento simplemente se actualizará la pantalla, al igual que con el primer elemento encontrado, cambiando simplemente el elemento que está actualmente activo y, por tanto, pintado en otro color.

7.4. OPCIONES DEL MODO EDICIÓN

Mientras permanezcamos en este modo, el usuario podrá añadir, borrar o modificar elementos del diagrama *GRAFCET*. Antes de pasar a comentar las distintas opciones de este modo, describiremos brevemente algunas funciones básicas de la app.

Antes de comenzar, es importante saber que, una vez creado un nuevo proyecto, automáticamente el programa insertará un estado inicial y la primera transición, tal y como observamos en la **Figura 6-5**.

```
public Documento_XML(){

    root = new Element( name: "Raiz");
    Doc = new Document(root); // Se crea el documento
    fichero = "";

    //Inicializamos todos las variables
    Pila = new Class_Pila_Elementos();
    Pila_enteros = new Class_Pila_int();
    Elemento Elemento_inic = new Elemento( Tipo: "Estado",Cont_est++, x: 0, y: 0, Etiq1: "", Etiq2: "", Act: 1, Salto_cond: "");
    root.addContent(Elemento_inic);
    Element Transic_inic = new Elemento( Tipo: "Transicion",Cont_tran++, x: 0, y: 0, Etiq1: "", Etiq2: "", Act: 0, Salto_cond: "");
    root.addContent(Transic_inic);
    Elemento_Grafico Fin = new Elemento_Grafico( Tipo: "End", Id: 0, x: 0, y: 0, nivel: 0);
    root.addContent(Fin);
    Element_Encont = Elemento_inic;
}
```

Figura 6-5(A). Código para la creación de un nuevo documento.

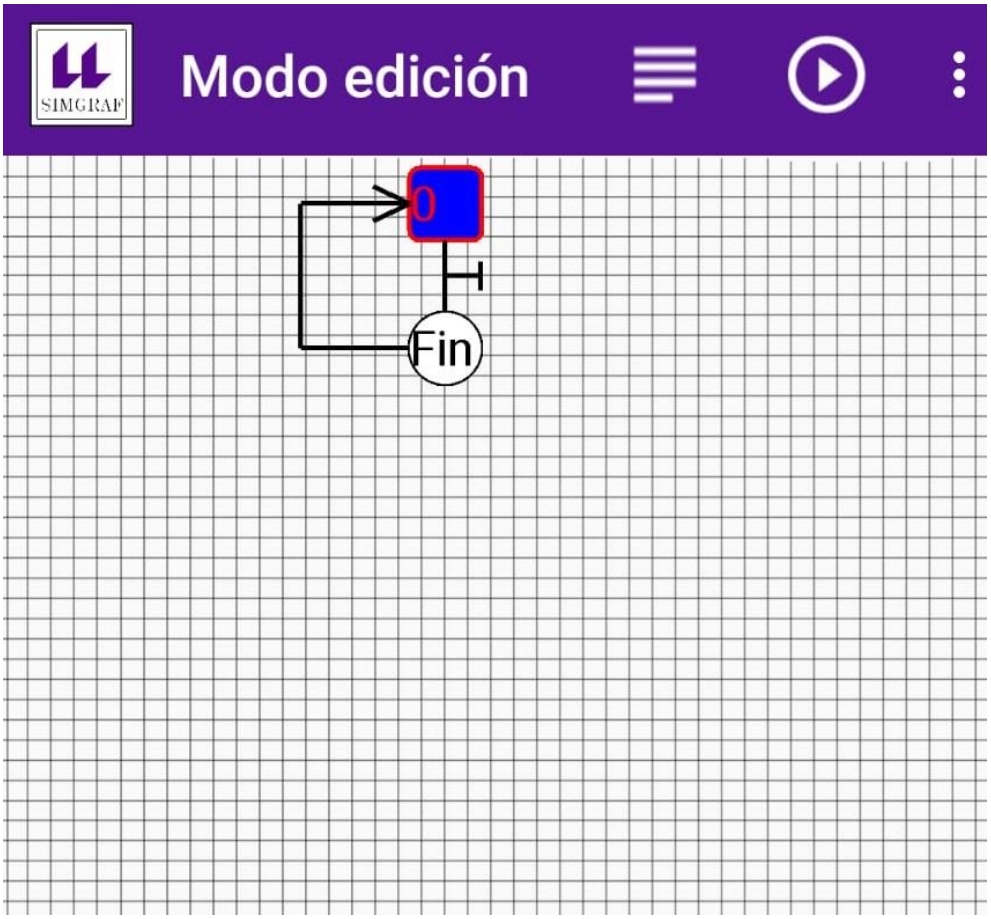

Figura 6-5 (B). Elemento inicial creado junto al nuevo documento.

Por último, antes de comenzar a describir las opciones del modo edición, debemos destacar que, para poder añadir nuevos elementos al diagrama, antes el usuario debe haber seleccionado un elemento, para añadir las nuevas etapas y transiciones de forma consecutiva al elemento activo.

7.4.1. AÑADIR ESTADO

Al seleccionar esta opción en nuestro menú, se añadirá al diagrama *GRAFCET* automáticamente una nueva etapa y una transición. Dependiendo de qué elemento tengamos seleccionado, variará el orden en que se añaden. Lo que queremos decir con esto, es que el programa está preparado para añadir los nuevos elementos sin errores, independientemente de si el usuario ha seleccionado un estado o una transición. La diferencia radica en que, si el elemento seleccionado es una etapa, se añadirá una transición y una etapa a continuación de este. Sin embargo, si el elemento seleccionado es una transición, serán añadidos en la posición previa, es decir, antes de la transición seleccionada.

```
public void Insertar_Estado_Transicion(){

    String Tipo_encont =Element_Encont.getAttributeValue( attname: "Tipo");
    String Id_encontrado = Element_Encont.getAttributeValue( attname: "Id");
    Element Padre = Element_Encont.getParentElement();
    List Elementos=Padre.getChildren();
    Iterator i = Elementos.iterator();
    int posicion = Buscar_por_Posicion(Padre, Id_encontrado, Tipo_encont);

    if (Tipo_encont.contains("Estado")){
        Elemento Estado_inic=new Elemento( Tipo: "Estado", Cont_est++, x: 0, y: 0, Etiq1: "", Etiq2: "", Act: 0, Salto_cond: "");
        Padre.addContent( index: posicion+1, Estado_inic);
        Elemento Transic_inic=new Elemento( Tipo: "Transicion", Cont_tran++, x: 0, y: 0, Etiq1: "", Etiq2: "", Act: 0, Salto_cond: "no");
        Padre.addContent( index: posicion+1,Transic_inic);
    }
    if (Tipo_encont.contains("Transicion")){
        Elemento Estado_inic=new Elemento( Tipo: "Estado", Cont_est++, x: 0, y: 0, Etiq1: "", Etiq2: "", Act: 0, Salto_cond: "");
        Padre.addContent(posicion, Estado_inic);
        Elemento Transic_inic=new Elemento( Tipo: "Transicion", Cont_tran++, x: 0, y: 0, Etiq1: "", Etiq2: "", Act: 0, Salto_cond: "no");
        Padre.addContent(posicion,Transic_inic);
    }
    // Si seleccionamos un Salto Condicional, no nos dejará meter un Estado+Transición ya que quedaría algo
    // inconsistente
    if (Tipo_encont.contains("Salto_condic")){
          Frame_errorNewEstado errorNewEstado = new Frame_errorNewEstado(25);
          errorNewEstado.show();
    }
}
```

Figura 6-6. Código utilizado para insertar estados y transiciones en nuestro GRAFCET.

7.4.2. AÑADIR DISYUNCIONES

Al seleccionar una de estas opciones, *Añadir una Estructura Concurrente o Alternativa*, se abrirá un cuadro emergente (*Alert Dialog* en *Android*) donde deberemos introducir el número de ramas que tendrá la nueva estructura que crearemos.

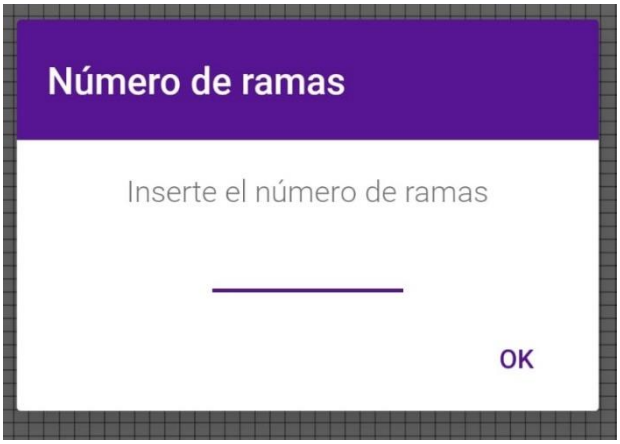

Figura 6-7. *Alert Dialog* utilizado para introducir el número de ramas de las disyunciones.

Tras insertar un número (de valor mínimo 2), se generará la disyunción a partir del elemento que estuviera seleccionado.

Aprovechamos para destacar que otra opción disponible es *Añadir Rama*, pero no le dedicaremos un punto propio porque, a pesar de tener su código propio, como podremos observar en los anexos, utiliza el mismo cuadro de diálogo (**Figura6-7**) que las otras estructuras. Dependiendo del tipo de disyunción en el que vayamos a añadir las nuevas ramas, el programa incluirá en ellas, una etapa o una transición.

Una vez seleccionada la disyunción y su número de ramas, el programa se encargará de crearla, asignando a cada estructura el nivel de profundidad (variable *nivel* descrita en el apartado 6.3.1) y a cada rama el tipo de estructura al que pertenece (Tipo Y o tipo O).

En estas estructuras es donde entra en uso la clase *Elemento_Grafico* que, como comentamos previamente, se encarga de almacenar los atributos correspondientes a las disyunciones.

7.4.3. INSERTAR SALTO CONDICIONAL

Para añadir un salto condicional el funcionamiento de la aplicación es ligeramente distinto que con el resto de funciones. Antes de poder añadirlo, el usuario debe seleccionar una transición. Completado este paso, se seleccionará en el menú la opción de añadir un salto condicional y, por último, se seleccionará el estado marcado como destino del salto.

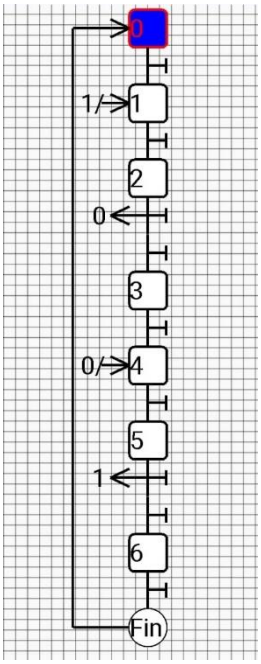

Figura 6-8. Ejemplos de saltos condicionales.

Tal y como se observa en la **Figura 6-8**, la forma de representar los saltos condicionales es bastante intuitiva. Por un lado, en la transición donde se asocia el salto, lo representaremos como una transición unida a una flecha que sale del esquema. Por otro lado, en la etapa a la que nos redirecciona el salto condicional se representa el mismo con una flecha apuntando hacia el propio estado. Para poder distinguirlos y relacionar los elementos sin problema, cada pareja de flechas viene acompañada por un número a modo de identificador del salto condicional.

Para que todo esto funcione correctamente, definimos una serie de variables en nuestro programa. Primero, para poder añadir elementos, será necesario que el valor booleano de la variable específica para el modo edición sea ***true***. En el caso de tener seleccionada una transición y seleccionar en el menú la opción de insertar un salto condicional, esta variable pasará a ***false***, imposibilitando así la adición de nuevos elementos. En este momento también pasará a ***true*** la variable de selección, utilizada específicamente para los saltos condicionales. Una vez seleccionado el estado destino del salto condicional, la variable de selección volverá a anularse y se activará de nuevo la variable que permite seguir editando el diagrama *GRAFCET*.

7.4.4. BORRAR UN ELEMENTO

Dentro de todas las funciones que se programan en este modo de edición, el borrado es la más compleja de todas debido a la cantidad que casos inconsistentes que se pueden generar. Definimos como casos inconsistentes todos aquellos que concluyen con modelos erróneos de *GRAFCET*. Un ejemplo de caso inconsistente lo encontramos a la hora de borrar una transición, quedando dos estados unidos únicamente por arcos, algo que no está permitido en un diagrama *GRAFCET*.

Es por esto que comentamos, que existen una serie de casos en la que no se permitirá borrar un elemento:

- **No podremos borrar la etapa inicial.**
- **Una etapa que quede entre el inicio de una disyunción concurrente y una disyunción alternativa.**

Esta etapa no se puede borrar porque queda definida como la unión entre ambas estructuras. Borrarla implica eliminar el estado de espera entre las disyunciones, lo que

provocaría además la unión de dos transiciones, conduciéndonos, como nombramos anteriormente, a un estado inconsistente. En el caso del final de dos de estas estructuras ocurre lo mismo. Al quedar unidas por una etapa de espera, la misma no podrá ser borrada.

- **La transición final del diagrama GRAFCET.**
- **Transición entre el inicio de una disyunción O y una disyunción Y.**

Ocurre lo mismo, pero a la inversa, que en el caso anterior. Para avanzar hacia una divergencia concurrente es necesaria la activación de una transición. Sin esta transición, el avance del diagrama sería inviable.

Como comentamos, el proceso de borrado es realmente complejo y varía en función del elemento que queramos borrar.

Lo primero que debe hacer el usuario es seleccionar el elemento en la interfaz. Una vez hecho esto, ya puede proceder a seleccionar la opción de borrado. Si la acción de borrado deriva en una inconsistencia, el programa mostrará el mensaje de error correspondiente. Si el elemento a borrar es válido para este proceso, será eliminado y el diagrama *GRAFCET* se reestructurará sin problema.

Sin embargo, el trabajo reside en la parte de estructura de datos. Un método se encargará de analizar y comprobar que el tipo de elemento seleccionado es válido, y, comparándolo con los elementos colindantes, decidirá si se genera un caso inconsistente o no. Esta parte del programa es realmente compleja, pues hay que preparar a la aplicación para ser capaz de gestionar los distintos tipos de inconsistencia. Para mostrar una estructura del funcionamiento del borrado, compartimos la **Figura6-9**, un esquema diseñado por *Adrián David Estévez Corona* y *José Manuel Royo Hardisson* para su proyecto, en el cual nos hemos basado para el desarrollo de esta *App*.

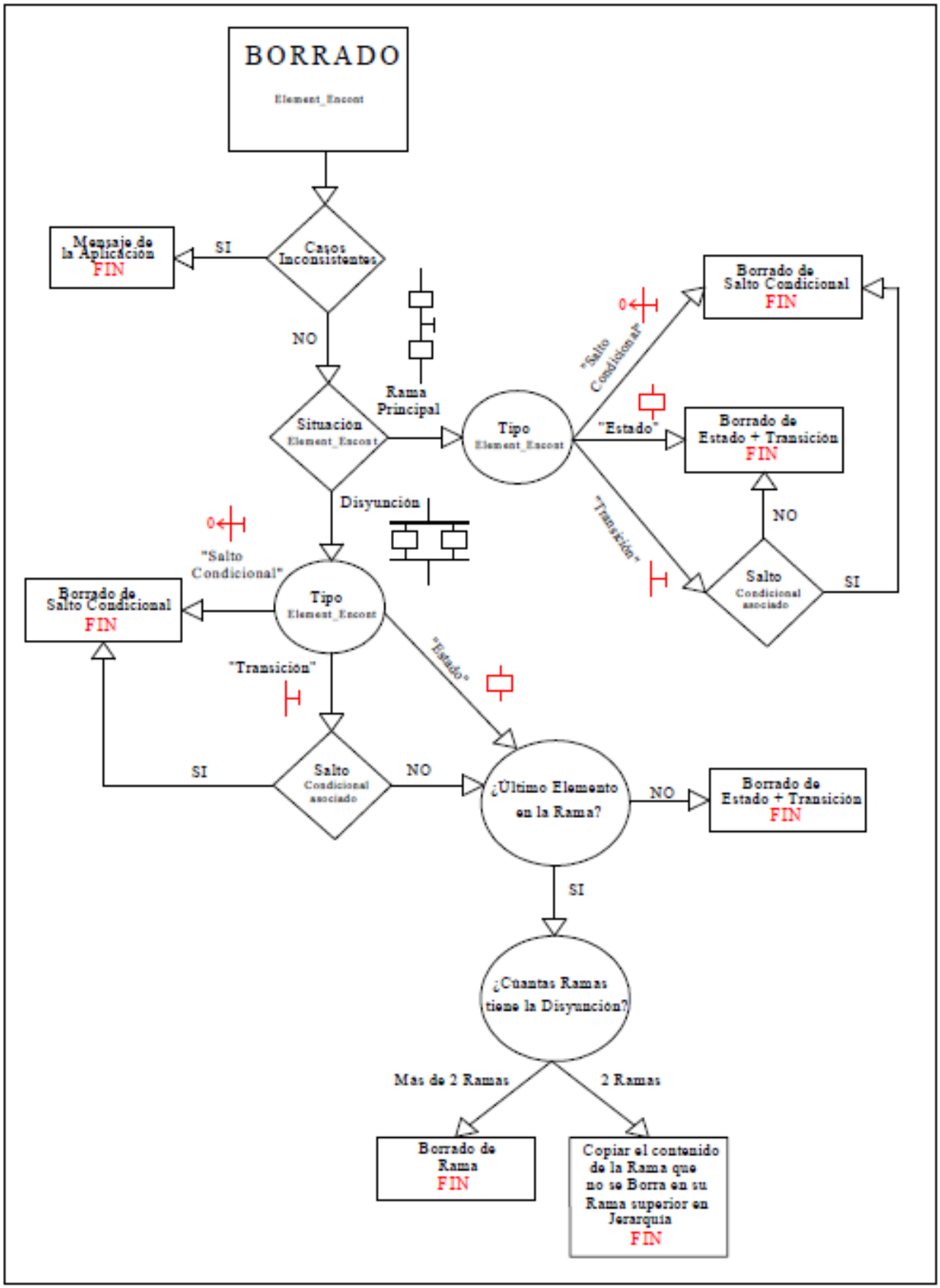

Figura 6-9. Esquema del funcionamiento del borrado [4].

Un caso a destacar, es el borrado de una rama. En el caso de que la estructura posea tres o más ramas, tan solo se borrará la rama y se redibujará el esquema con las ramas restantes. Sin embargo, en el caso de que solamente tenga dos ramas, si borramos una de ellas, el diagrama *GRAFCET* quedará como una estructura lineal, insertando las etapas y transiciones correspondientes a la rama restante entre los elementos que abrían y cerraban la disyunción. También se borrarán los elementos gráficos que marcaban el inicio y fin de esta estructura.

8. ADAPTANDO ULLSIMGRAF A DISPOSITIVOS ANDROID

Una vez descrito el funcionamiento básico del editor, pasaremos a comentar los principales cambios que se producen en el programa al adaptarlo a una nueva plataforma.

8.1. INTERFAZ DE LA *APP*

Debemos tener en cuenta que *Android* es un sistema operativo utilizado principalmente en móviles y tablets, dispositivos con pantalla táctil. Este factor implica un cambio radical a la hora de rediseñar la interfaz, ofreciendo estos terminales nuevas posibilidades para los desarrolladores.

Antes de comenzar a comentar directamente la interfaz de nuestra *app*, describiremos los elementos básicos que pueden conformar la interfaz de una aplicación desarrollada para estos dispositivos.

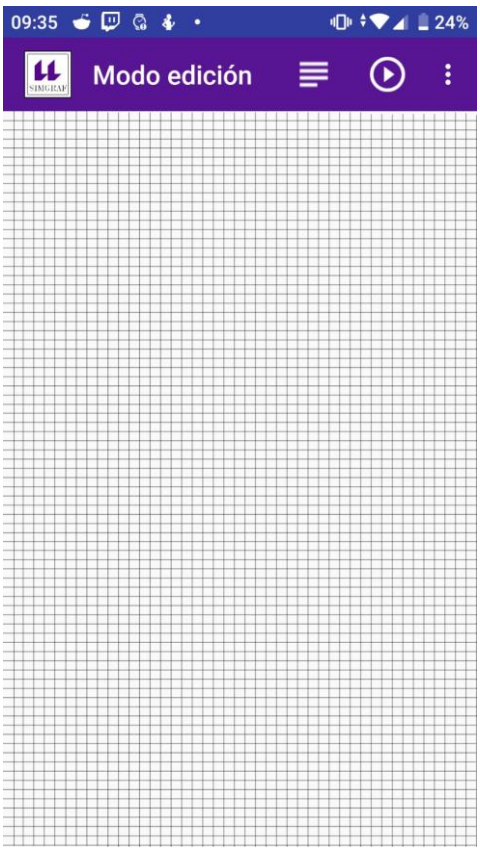

Figura 7-1. Interfaz principal de la *app*.

8.1.1. LAYOUT

En Android, entendemos por *layout* el diseño de la propia interfaz, su estructura y el conjunto constituido por todos los elementos que contiene.

Para comenzar a definir una interfaz en *Android*, crearemos un documento *XML*. Dentro de este documento, estableceremos como nodo raíz el propio *layout*, especificando qué tipo es el que hemos decidido utilizar. Como es propio del estándar *XML*, comenzaremos a desarrollar la estructura jerárquica propia de este tipo de documentos. Es en este punto donde se asocian a la interfaz todos los elementos que serán necesarios para el correcto funcionamiento de la *app*.

Es importante destacar que tan solo se definen los elementos y sus atributos, pero no su funcionamiento.

Existen principalmente dos tipos de *layouts* predefinidos por *Android*:

- ***Linear Layout***

"El diseño Lineal (*Linear layout*) es un grupo de vista que alinea todos los campos secundarios en una única dirección, de manera vertical u horizontal" **[15]**. Con los atributos que podemos definir en el documento *XML* especificaremos la dirección deseada para la alineación.

- ***Relative Layout***

Con este tipo de diseño, asignamos posiciones relativas a los elementos de la interfaz. Esto significa que podemos asociar cada uno de los componentes de la interfaz a la posición de un elemento que fijemos como referencia.

Aparte de estos, también podremos utilizar *layouts* personalizadas en función del objetivo de la *app*, como diseños específicos para visualizar páginas webs.

Por último, es importante destacar que todos los elementos gráficos en *Android* deben tener un *layout* definido, desde las propias interfaces hasta los menús.

8.2. TOOLBAR

Llamamos *toolbar* a la barra superior (**Figura 7-1**) definida en la interfaz de la aplicación. Comúnmente en ella aparece el nombre de la *app* o, en su lugar, un título que ayude a clarificar la función de la actividad en la que nos encontramos. Asociado a la *toolbar*, encontraremos el menú principal de la aplicación, representado habitualmente con tres botones dispuestos en vertical, icono que debe ser pulsado para desplegar dicho menú. También podremos fijar las opciones más importantes de este menú a la propia *toolbar*, tal y como observamos en la **Figura 7-1**, quedando distribuidas por la barra y representadas por los iconos que queramos asociarles.

8.3. TIPOS DE MENÚ

Android ofrece varios tipos de menú a los desarrolladores:

- **Menú de opciones**

Este menú se asocia a la barra principal de la *app*, tal y como comentamos en el apartado anterior. Al igual que cualquier otro elemento de la interfaz, debe definirse en un documento XML.

Figura 7-2. Menú de opciones de la *app*.

Los elementos de este menú pueden tener sus propios submenús, como ocurre con la opción *"Simulación"* del menú adjunto en la **Figura 7-2**. Para generar estos submenús es necesario definirlos dentro de la jerarquía del documento *XML* propio del menú de opciones, tal y como se observa en la **Figura 7-3**.

```
<item
    android:id="@+id/simulacion"
    android:title="Simulación">

    <menu>

        <item
            android:id="@+id/interactive"
            android:title="Simulación interactiva">
        </item>

        <item
            android:id="@+id/step"
            android:title="Simulación paso a paso">
        </item>

    </menu>
</item>
```

Figura 7-3. Definiendo un submenú en *Android*.

- **Menú deslizante**

Para utilizar este tipo de menú, primero debemos definir en nuestra interfaz un *Drawer Layout*. En resumen, este tipo de diseño permite que vistas interactivas puedan ser sacadas desde los bordes de la ventana.

Para aprovechar todas las herramientas que ofrece *Android* y diferenciarnos de una aplicación típica de ordenadores, hemos utilizado este tipo de menú para ubicar las opciones básicas del modo edición (**Figura 7-4.**).

Estos menús también pueden tener, como se observa en la **Figura7-4**, una *toolbar* propia asociada.

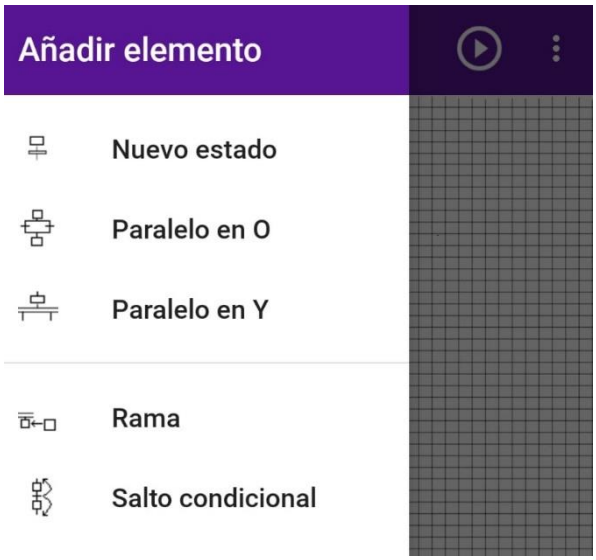

Figura 7-4. Menú deslizante de la *app*.

- **Menú contextual**

Podemos definir el menú contextual como "un menú flotante que aparece cuando el usuario hace un clic largo en un elemento. Proporciona acciones que afectan el contenido seleccionado o el marco contextual" **[15]**.

Pese a la utilidad de este tipo de menú, no será utilizado en el desarrollo de la *app* debido a la dificultad de implementarlo de una forma útil en el programa.

8.4. OTROS ELEMENTOS

- **Edit Text**

Elemento de la interfaz, utilizado para que el usuario pueda introducir texto o datos modificables. En esta aplicación los utilizaremos en los cuadros emergentes, para introducir, por ejemplo, el número de ramas de una disyunción alternativa. Durante el diseño de las interfaces, al trabajar con un *Edit Text*, podemos predefinir el tipo de datos que recibirá dicho elemento.

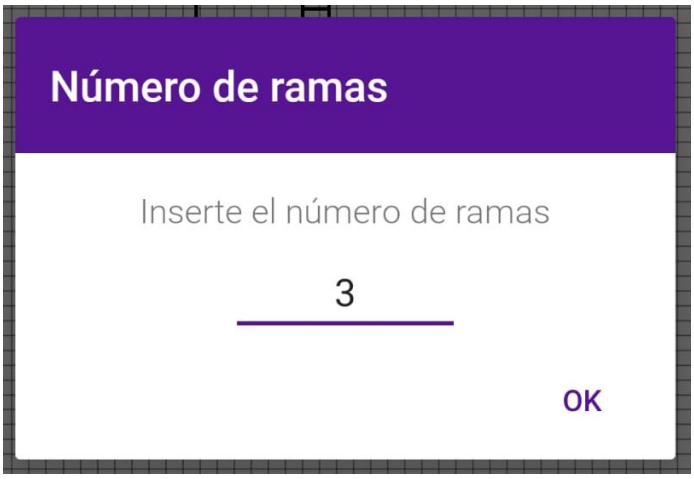

Figura 7-5. Ejemplo de *Alert Dialog* con un *Edit Text*.

- **Alert Dialog**

Podemos definir estos diálogos de alerta como ventanas emergentes que se generan para aportar información importante en casos concretos. Uno de estos casos, es el aviso de posibles errores. Nuestra aplicación está programada de forma que surjan estos diálogos para informar al usuario de fallos que se cometen en el desarrollo del diagrama *GRAFCET*.

Otro uso que le damos a esta función, es la mostrada en la **Figura 7-5**, aprovechándola para preguntarle al usuario por el número de ramas que desea introducir en una disyunción del esquema.

Estos cuadros tienen también su propio diseño, pudiendo poseer su propia *toolbar* y hasta tres botones en la zona inferior de la ventana emergente.

Tras definir su interfaz en otro documento *XML*, se creará una clase, programada en *Java*, en la que definiremos la función de cada uno de los botones, en caso de que los tenga, pudiendo añadir, como en la **Figura 7-6**, código específico para la comunicación de datos con otras clases.

```
import ...

public class Dialog_Paralelos extends DialogFragment {

    private EditText numero_ramas;
    private Dialog_ParalelosListener listener;
    public Dialog onCreateDialog(Bundle savedInstanceState){
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        View view = inflater.inflate(R.layout.dialog_paralelos, root: null);
        builder.setView(view)
                .setPositiveButton("OK", (dialog, id) → {
                        String num_ramas = numero_ramas.getText().toString();
                        listener.sendramas(num_ramas);
        });
        numero_ramas = view.findViewById(R.id.file1);
        return builder.create();
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try {
            listener = (Dialog_ParalelosListener)context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString() + "Implementar Listener");
        }
    }

    public interface Dialog_ParalelosListener{
        public void sendramas(String num_ramas);
    }

}
```

Figura 7-6. Programación de un *Alert Dialog*.

8.5. COMUNICACIÓN INTERNA DE LA *APP*

Como se mostraba en la **Figura 6-2**, en la versión de ordenador, la aplicación se había dividido en dos bloques principales, quedando por un lado la estructura de datos, y por otro lado la parte gráfica, que se encarga de la comunicación con el usuario.

Sin embargo, debido al funcionamiento de *Android*, en nuestra *app* esta estructura cambia ligeramente. Por ello, comentaremos previamente algunas de las características de *Android* para comprender mejor la nueva estructura, que explicaremos posteriormente.

8.5.1. ACTIVITIES

"Una actividad es un componente de la aplicación que contiene una pantalla con la que los usuarios pueden interactuar para realizar una acción, como marcar un número de telefónico, tomar una foto, etc." **[15]**.

Las actividades son fundamentales para el correcto funcionamiento de *Android*. En ellas, asociamos a la propia actividad la interfaz para que pueda ser creada con el arranque de la aplicación.

En el código de la actividad programaremos las funciones a realizar por cada uno de los elementos del menú de opciones principal y del menú deslizante que contiene las opciones del modo edición.

La actividad en *Android* tiene un ciclo de vida (**Figura 7-7**).

En resumen, con el arranque de la aplicación se crea la actividad por primera vez y se ejecuta el método *onCreate()*. Por ello, es en este método donde se realiza la configuración básica de algunos elementos, como la creación de menús, definición de vistas, etc.

Si en algún momento cambiamos de actividad, como observamos en el ciclo, la actividad actual se destruirá. Esto significa que cuando volvamos a la actividad previa habremos perdido todos los datos que no hayamos programado para ser guardados en el momento del cambio. De hecho, esto no ocurre solamente al cambiar de actividad. Al girar la pantalla del dispositivo, la actividad se destruye y se vuelve a crear de nuevo, como si acabara de arrancar la aplicación.

Para guardar datos de interés, debemos utilizar la combinación de dos métodos: ***onSaveInstanceState()*** y ***onRestoreInstanceState()***.

El primero de estos métodos será ejecutado por *Android* justo antes de destruir la *app*, guardando los datos que recogidos en él. El segundo método se ejecutará con la nueva creación de la actividad.

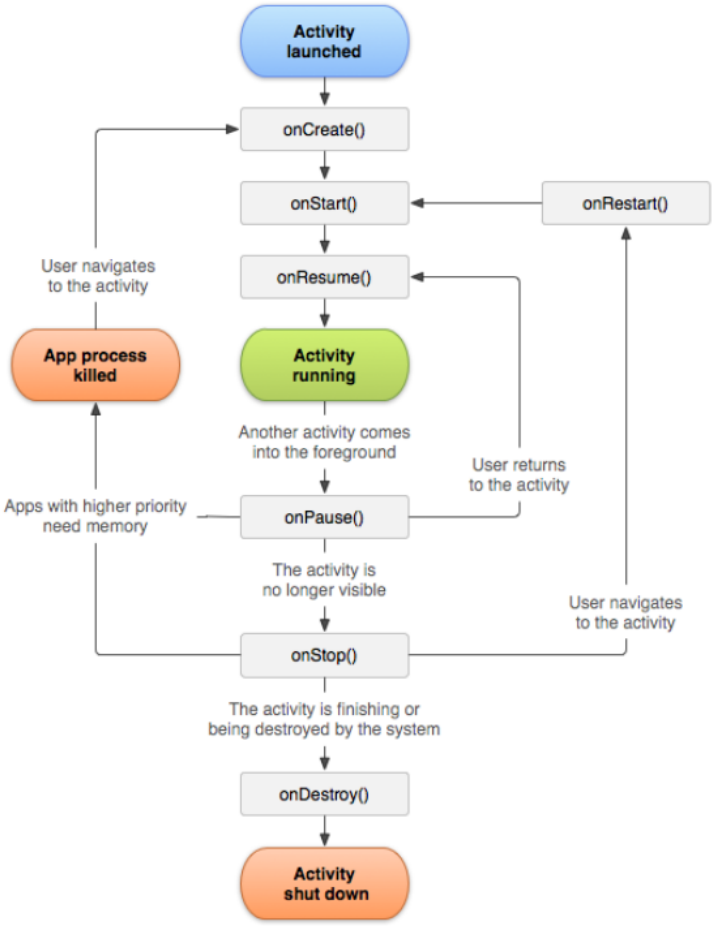

Figura 7-7. Ciclo de vida de una actividad [15].

8.5.2. CLASE VIEW

La herencia de esta clase es fundamental en una aplicación *Android*. Muchas de las acciones relacionadas con la interacción del usuario con la interfaz deben estar definidas en una clase hija de la clase *View*. Estas acciones son muy amplias, yendo desde el dibujado en el contenido de la interfaz hasta la detección de pulsaciones en la pantalla táctil del dispositivo.

Esto implica centrar una de las grandes partes de nuestro proyecto en esta clase. Es importante destacar que, para actualizar lo dibujado en pantalla, se debe ejecutar la acción ***invalidate()***. Este método solo puede ser ejecutado en una clase con herencia de la clase *View*.

Al igual que nos encontramos con métodos específicos y necesarios para el correcto funcionamiento del programa en la clase con herencia de *Activity*, en este apartado vuelven a aparecer métodos similares.

Para el dibujado en pantalla, no solo hemos tenido que utilizar las librerías propias de *Android*, sino que todas las acciones de pintado tienen que estar contenidas en el método ***onDraw()*** de esta clase.

Para capturar los eventos táctiles ocurre lo mismo. El método que se ejecutará cuando tratemos de seleccionar etapas o transiciones dibujadas en pantalla será ***onTouchEvent()***. Habitualmente, la variable de entrada que se pasa a este método es un evento. Este evento detectará el tipo de pulsación que hace el usuario sobre la pantalla.

Tras comentar estas características específicas del funcionamiento de *Android*, pasamos a comentar la estructura escogida para la aplicación.

A diferencia de la división en dos grandes partes que hicieron nuestros antiguos compañeros, en el presente proyecto dividiremos la estructura de la *app* en tres módulos principales:

- **Main Activity**

Primero definimos la actividad principal, donde se definen y programan los elementos básicos de la interfaz. Esta parte se encargará de la comunicación básica entre el usuario y la parte gráfica. Respecto a la programación, en esta clase se define las acciones que se deben llevar a cabo al pulsar cualquiera de las opciones de los menús.

- **La clase *Draw***

Esta clase corresponde a la parte gráfica. Se corresponde principalmente al *"FramePrincipal"* de la versión para ordenadores. Sin embargo, por las necesidades específicas comentadas previamente, hay ciertos cambios que se deben realizar, tal y como se observa en el código adjunto en los anexos.

Esta parte se encarga de gestionar el dibujado por pantalla, detectar los eventos táctiles y, por último, de comunicarse con la estructura de datos.

- **Estructura de datos**

Esta parte sí coincide con la programada en la versión de computadores. En ella nos encargaremos de gestionar la estructura del documento XML que se va generando con el desarrollo del diagrama GRAFCET y de aportarle a la parte gráfica los datos que sean requeridos.

8.6. DIFICULTADES DEL *PORT*

Por último, nos parece importante citar brevemente algunas de las dificultades con las que nos hemos topado a lo largo del desarrollo de la *app*.

Principalmente, el cambio en una estructura tan bien definida como era este proyecto, ha supuesto que el código tuviera que ser reescrito en numerosas ocasiones. Esto, junto a la adición de nuevos elementos propios de la plataforma, como las actividades o una nueva forma de definir los elementos de la interfaz, ha supuesto el problema central de este proyecto.

En menor grado, también ha influido en este *port* la dificultad de trabajar con distintas librerías. A pesar de contener acciones similares, las librerías de *Android* interpretan los datos introducidos de forma distinta, lo que implica readaptar las fórmulas y algunos de los elementos definidos en el código.

8.7. SIMULACIÓN

Por último, queremos comentar brevemente el modo simulación, añadido para complementar al editor.

El funcionamiento es bastante simple. El simulador mostrará un elemento activo. Mediante la pulsación de las transiciones, lo cual interpretamos como la activación de la condición asociada a dicha transición, podremos ir avanzando a lo largo del esquema GRAFCET y sus distintas etapas.

La aplicación está preparada para permitir el paso a la siguiente etapa únicamente cuando se pulsa la transición correcta.

9. PRESUPUESTO

El presupuesto de un proyecto como este es bastante sencillo de elaborar, pues solo requerimos la adquisición de un dispositivo móvil y pagar el salario del desarrollador/a.

Escogiendo como referencia un terminal de gama media, como, por ejemplo, el *Xiaomi Mi A2 6GB/128GB 4G*, con un precio de **237.95€**.

A esto le sumaremos el sueldo de la persona encargada de la programación de la *app*. Como el rango de salarios en el sector es muy amplio, estableceremos un sueldo de **2.000€** mensuales.

Por ello, el proyecto tendrá un coste inicial de 2.237,95€, viéndose incrementado por cada mes de desarrollo.

10. ANEXOS

En los anexos, incluimos las clases principales que definen la estructura de este proyecto.

10.1. CLASE *"Documento_XML"*

Esta clase corresponde a la parte que mis antiguos compañeros definieron como *"estructura de datos"*.

```
package e.brian.ullsimgraf;


import android.graphics.Point;
import android.support.v7.app.AppCompatActivity;

import java.io.*;
import java.util.*;
import org.jdom2.*;
import org.jdom2.output.*;
import org.jdom2.input.SAXBuilder;


public class Documento_XML extends AppCompatActivity{

    // Variables globales
    public int Cont_est = 0;
    public int Cont_tran = 0;
    public int Cont_tranY = 0;
    public int Cont_tranO = 0;
    public int Cont_saltos = 0;
    public int Cont_tiempo = 0;

    // Variables de la parte gráfica
    public static int Dist_x = 150;
    public static int Dist_y = Dist_x/2;
    public static int Dist_prof = Dist_y/2;
    public static int PosX_inic = 480;
    public static int PosY_inic = 50;
    public static int Max_nivel = 5;
    public static int PosX_min = PosX_inic;
    public static int Max_Caracteres = 8;
    public static int Tam_font = 14;

    // Variables de la parte XML
    public Class_Pila_Elementos Pila;
    public Class_Pila_int Pila_enteros;
    private Element root;
    private String fichero;
    private Document Doc;
    public Element Element_Encont;
```

```
    // CONSTRUCTOR QUE CREA UN NUEVO DOCUMENTO XML

    public Documento_XML(){

        root = new Element("Raiz");
        Doc = new Document(root); // Se crea el documento
        fichero = "";

        //Inicializamos todos las variables
        Pila = new Class_Pila_Elementos();
        Pila_enteros = new Class_Pila_int();
        Elemento Elemento_inic = new
Elemento("Estado",Cont_est++,0,0,"","",1,"");
        root.addContent(Elemento_inic);
        Element Transic_inic = new
Elemento("Transicion",Cont_tran++,0,0,"","",0,"");
        root.addContent(Transic_inic);
        Elemento_Grafico Fin = new Elemento_Grafico("End",0,0,0,0);
        root.addContent(Fin);
        Element_Encont = Elemento_inic;
    }

    // CONSTRUCTOR QUE INICIALIZA EL DOCUMENTO DESDE UN DOCUMENTO XML
YA EXISTENTE
    public Documento_XML(String Fichero_XML){

        fichero = Fichero_XML;

        try{
            File f =new File(fichero);
            SAXBuilder builder = new SAXBuilder(false);
            Doc = builder.build(f);
            Cambiar_root(Doc.getRootElement());
            XMLOutputter out=new XMLOutputter();
            out.output(Doc,System.out);
        }
        catch (Exception e){
            //error al intentar abrir el fichero seleccionado
//            Dialog_errorNewEstado errorNewEstado = new
Dialog_errorNewEstado("19");
//            errorNewEstado.show(getFragmentManager(),
"errornewestado");
        }

        Element_Encont = null; // Variable utilizada para buscar
elementos
        // Iniciamos los contadores
        Pila = new Class_Pila_Elementos();
        Pila_enteros = new Class_Pila_int();
        Inicializar_contadores(root);
        // Cuando abrimos un fichero existente buscamos el último ID
        Cont_est++;
        Cont_tran++;
        Cont_tranY++;
        Cont_tranO++;
        Cont_saltos++;
    }

    public void Insertar_Contador(){
```

```
        int existe = -1;

        if(existe == 0){
//            Dialog_errorNewEstado errorNewEstado = new
Dialog_errorNewEstado("32");
//            errorNewEstado.show(getFragmentManager(),
"errornewestado");
        }
        else{
            // Añadir contador Nuevo
        }
    }

    public void Inicializar_contadores(Element Padre){

        List Elementos = Padre.getChildren();
        Iterator i = Elementos.iterator();
        while(i.hasNext()){
            Element e = (Element)i.next();
            String Tipo_Element = e.getAttributeValue("Tipo");
            int Id = Integer.parseInt(e.getAttributeValue("Id"));
            if(Tipo_Element.contains("Estado")){
                if(Id > Cont_est)
                    Cont_est = Id;
            }
            if(Tipo_Element.contains("Transicion")){
                if(Id > Cont_tran)
                    Cont_tran = Id;
            }
            if(Tipo_Element.contains("Salto_condic")){
                if(Id > Cont_saltos)
                    Cont_saltos = Id;
            }
            if (Tipo_Element.contains("InicioY")){
                if (Id > Cont_tranY)
                    Cont_tranY=Id;
            }
            if (Tipo_Element.contains("InicioO")){
                if (Id > Cont_tranO)
                    Cont_tranO=Id;
            }

            List Hijos =e.getChildren();
            if (Hijos.size() != 0){
                Inicializar_contadores(e);
            }
        }
    }

// Accede al nombre del fichero seleccionado
    public String Devolver_fichero(){

        return(fichero);
    }
// Accede a la variable Root que es el padre del Documento XML

    public Element Devolver_root(){

        return(root);
    }
```

```
    public void Cambiar_root(Element New_root){
        root = New_root;
    }

    public void Guardar_Documento_XML(String NomFichero){

        fichero = NomFichero;
        try {
            XMLOutputter out = new XMLOutputter();
            FileWriter writer = new FileWriter(new File(fichero));
            out.output(Doc, writer);
            out.output(Doc, System.out);
        }
        catch(Exception e){

        }
    }

    public boolean Comparar_numeros(String cadena1, String cadena2){
        int num1 = Integer.parseInt(cadena1);
        int num2 = Integer.parseInt(cadena2);
        return(num1 == num2);
    }

    public void Limpiar_marcas_rama(Element Rama){

        List Elementos = Rama.getChildren();
        Iterator i = Elementos.iterator();
        while(i.hasNext()){
            Element e = (Element)i.next();
            String Tipo_Element = e.getAttributeValue("Tipo");
            if(Tipo_Element.contains("Estado") ||
Tipo_Element.contains("Transicion") ||
Tipo_Element.contains("Salto_condic")){
                e.setAttribute("Activacion", Integer.toString(0));//
Se desactiva la marca
            }
        }
    }

    public void Eliminar_Marcas_en_Y_O(Element FinalY_O){

        String Tipo = FinalY_O.getAttributeValue("Tipo");
        String Tipo_buscado ="";
        if (Tipo.contains("FinalY"))
            Tipo_buscado = "InicioY";
        if (Tipo.contains("FinalO"))
            Tipo_buscado = "InicioO";

        String Id_Element =FinalY_O.getAttributeValue("Id");
        Element Padre = FinalY_O.getParentElement();
        List Hermanos = Padre.getChildren();
        Iterator i = Hermanos.iterator();
        int posicion = Buscar_por_Posicion(Padre, Id_Element,
Tipo_buscado);
        Element buscado = (Element) Padre.getContent(posicion);
        // CAMBIAR Buscar_por_Posicion(Padre, Id_Element,
Tipo_buscado);
        // Buscado = "InicioY" o "InicioO"
        List Hijos = buscado.getChildren();
        i = Hijos.iterator();
```

```
        int Num_Ramas = Hijos.size();
        for (int j=0;j<Num_Ramas; j++){
            Element Rama= (Element)i.next();
            Limpiar_marcas_rama(Rama);
        }
    }
    public boolean ecuacionDeCirculo(Point punto, int Element_x, int
Element_y){

        int RADIO = (Dist_y/2);
        return (((punto.x - Element_x) * (punto.x - Element_x) +
(punto.y - Element_y) * (punto.y - Element_y)) <= (RADIO * RADIO));
    }

    public boolean Buscar_Elemento(Element Padre, Point punto){

        boolean Encontrado = false;

        List Elementos=Padre.getChildren();
        Iterator i = Elementos.iterator();
        while ((i.hasNext())&&(!Encontrado)){
            Element e= (Element)i.next();
            String Tipo_Element =e.getAttributeValue("Tipo");
            if ((Tipo_Element.contains("Estado")) ||
(Tipo_Element.contains("Transicion")) ||
Tipo_Element.contains("Salto_condic")) {
                int Pos_x =Integer.parseInt(e.getAttributeValue("x"));
                int Pos_y =Integer.parseInt(e.getAttributeValue("y"));
                Encontrado = ecuacionDeCirculo(punto, Pos_x, Pos_y);
                if (Encontrado){
                    Element_Encont = e; // Si se encuentra, devuelve
ese Elemento en la variable Global "Element_Encont"
                }
            }
            List Hijos =e.getChildren();
            if (Hijos.size() != 0){
                Encontrado = Buscar_Elemento(e, punto); //Busqueda
recursiva si tiene hijos
            }

        }
        return(Encontrado);
    }

    public void Insertar_Estado_Transicion(){

        String Tipo_encont =Element_Encont.getAttributeValue("Tipo");
        String Id_encontrado = Element_Encont.getAttributeValue("Id");
        Element Padre = Element_Encont.getParentElement();
        List Elementos=Padre.getChildren();
        Iterator i = Elementos.iterator();
        int posicion = Buscar_por_Posicion(Padre, Id_encontrado,
Tipo_encont);

        if (Tipo_encont.contains("Estado")){
            Elemento Estado_inic=new Elemento("Estado", Cont_est++, 0,
0, "","", 0, "");
            Padre.addContent(posicion+1, Estado_inic);
            Elemento Transic_inic=new Elemento("Transicion",
Cont_tran++, 0, 0,"","", 0, "no");
            Padre.addContent(posicion+1,Transic_inic);
```

```
        }
        if (Tipo_encont.contains("Transicion")){
            Elemento Estado_inic=new Elemento("Estado", Cont_est++, 0,
0, "","", 0, "");
            Padre.addContent(posicion, Estado_inic);
            Elemento Transic_inic=new Elemento("Transicion",
Cont_tran++, 0, 0,"","", 0, "no");
            Padre.addContent(posicion,Transic_inic);
        }
        // Si seleccionamos un Salto Condicional, no nos dejará meter
un Estado+Transición ya que quedaría algo
        // inconsistente
        if (Tipo_encont.contains("Salto_condic")){
 //           Frame_errorNewEstado errorNewEstado = new
Frame_errorNewEstado(25);
//            errorNewEstado.show();
        }
    }

    public int Buscar_por_Posicion(Element Padre, String Id_buscado,
String Tipo_buscado){

        int posicion = 0;

        List Elementos=Padre.getChildren();
        Iterator i = Elementos.iterator();
        Boolean Encontrado = false;
        while ((i.hasNext())&&(!Encontrado)){
            Element e= (Element)i.next();
            String Id_Element =e.getAttributeValue("Id");
            String Tipo_element = e.getAttributeValue("Tipo");
            if ((Comparar_numeros(Id_buscado,Id_Element)) &&
(Tipo_element.contains(Tipo_buscado))){
                Encontrado = true;
            }
            else
                posicion++;
        }
        return (posicion);
    }

    public void Calcular_Posiciones(Element Padre){

        String Tipo_element = null;
        List Elementos=Padre.getChildren();
        Iterator i = Elementos.iterator();
        Element Anterior = null;
        while (i.hasNext()){
            Element e= (Element)i.next();
            Tipo_element = e.getAttributeValue("Tipo");
            Asignar_Posicion(Padre, Anterior, e);
            Anterior = e;
            List Hijos =e.getChildren();
            if (Hijos.size() != 0){ //Cuando tenemos un "InicioY" o
"InicioO" miramos sus hijos llamando recursivamente
                Calcular_Posiciones(e);
            }
        }
    }

    public void Calcular_Anchos(Element Actual){
```

```
        List Elementos=Actual.getChildren();
        Iterator i = Elementos.iterator();

        while (i.hasNext()){
            Element Anterior = Actual;
            Actual= (Element)i.next();
            String Tipo_element = Actual.getAttributeValue("Tipo");
            // Cuando encontramos una rama metemos un '1' en la pila
            if ((Tipo_element.contains("Tipo_Y")) ||
(Tipo_element.contains("Tipo_O"))){
                Pila_enteros.Push(1);
            }
            // Cuando encontramos una "Linea" copiamos el valor del
Top de la pila y lo guardamos en el campo
            // "Ancho" de la rama correspondiente
            if (Tipo_element.contains("Linea")){
                int Ancho = Pila_enteros.Devolver_Valor_Top();
                Element Padre = Actual.getParentElement();
                Padre.setAttribute("Ancho", Integer.toString(Ancho));
            }
            // Extraemos tantos elementos de la pila como hijos
tuviera su correspondiente "Inicio", lo sumamos
            // y comparamos con el anterior Ancho que tenemos en la
pila, si es mayor, lo reemplazamos si es menor
            // se mantiene el anterior
            if ((Tipo_element.contains("FinalY")) ||
(Tipo_element.contains("FinalO"))){
                List Hijos=Anterior.getChildren();
                int Num_Ramas = Hijos.size();
                int suma = 0;
                for (int j=0;j< Num_Ramas; j++){
                    int extraido = Pila_enteros.Pop();
                    suma = suma + extraido;
                }
                suma++;
                Element Padre = Actual.getParentElement();
                // Comprobamos que el padre no es root porque si no no
habría elemento con el que comparar
                if (Padre != root){
                    int Ancho = Pila_enteros.Devolver_Valor_Top();
                    if (suma > Ancho){
                        Pila_enteros.Pop();
                        Pila_enteros.Push(suma);
                    }
                }
            }
            List Hijos =Actual.getChildren();
            if ((Hijos.size() != 0)){ //Cuando tenemos un "InicioY" o
"InicioO" miramos sus hijos llamando recursivamente
                Calcular_Anchos(Actual);
            }
        }
    }

    int Calcular_distancia(Element Inicio_Y_O, Element Rama_actual ){
        int dist_ramas = 0;


        List Ramas = Inicio_Y_O.getChildren();// Todas las ramas
        int nivel
```

```
=Integer.parseInt(Inicio_Y_O.getAttributeValue("nivel"));

        //VARIABLES DE LA RAMA ACTUAL
        int Ancho_Rama =
Integer.parseInt(Rama_actual.getAttributeValue("Ancho"));
        int Id_Rama =
Integer.parseInt(Rama_actual.getAttributeValue("Id"));

        //VARIABLES DE LA RAMA ANTERIOR
        Element Rama_anterior = (Element)
Inicio_Y_O.getContent(Id_Rama-1);
        int Ancho_anterior
=Integer.parseInt(Rama_anterior.getAttributeValue("Ancho"));
        Element Primer_hijo = (Element) Rama_anterior.getContent(0);
        int x_primer
=Integer.parseInt(Primer_hijo.getAttributeValue("x"));

        if ((Ancho_anterior == 1) && (Ancho_Rama == 1))
            dist_ramas = 2;
        else
        if (Ancho_anterior < Ancho_Rama)
            dist_ramas = ((Ancho_anterior/(1+nivel) ) + Ancho_Rama);
        else
            dist_ramas = ((Ancho_Rama/(1+nivel)  ) + Ancho_anterior);

        int Pos_x = x_primer + dist_ramas*Dist_x;
        return(Pos_x);
    }

    public void Asignar_Posicion(Element Padre, Element Anterior,
Element Actual){

        int x_ant, y_ant;
        int Pos_x = 0, Pos_y = 0;

        if (Anterior == null){
            if (Padre == root){ //Significa que es un Estado o
Transicion en la posicion inicial del Grafcet
                Pos_x = PosX_inic;
                Pos_y = PosY_inic;
                PosX_min = PosX_inic;
            }
            else{//Estado o Transicion, primer elemento de una rama,
no tiene anterior
                //Actual   => Elemento Estado o Transicion primera de
una rama, al que le asignaremos posicion
                //Anterior => NULL porque es el primer elemento de la
rama
                //Padre    => RAMA donde esta el elemento Actual
                //Abuelo   => ELEMENTO GRAFICO(InicioY, InicioO) donde
esta la rama "Padre" y dentro de esta "Actual"
                String Tipo_act = Actual.getAttributeValue("Tipo");
                String Id_act = Actual.getAttributeValue("Id");
                if ((!(Tipo_act.contains("Tipo_O")))&&
(!(Tipo_act.contains("Tipo_Y")))) { //Si es Rama, no se hace nada ya
que no llevan posicion
                    Element Abuelo = Padre.getParentElement();
                    List Ramas = Abuelo.getChildren();
                    Iterator i = Ramas.iterator();
                    int Num_Ramas = Ramas.size();
                    int Id_Rama =
```

```
Integer.parseInt(Padre.getAttributeValue("Id"));
                    x_ant
=Integer.parseInt(Abuelo.getAttributeValue("x"));
                    y_ant
=Integer.parseInt(Abuelo.getAttributeValue("y"));
                    if (Id_Rama == 0){
                        Pos_x = x_ant;//Primera rama la coloca en el
extremo izquierdo
                        if (PosX_min > Pos_x)  // Comparamos para
irnos quedando siempre con el elemento mas a la izq.
                            PosX_min = Pos_x;  // para saber ubicar
luego la linea que une el final con el principio.
                    }
                    else{ //Sabemos que tiene anterior,accedemos a
este mediante el identificador de rama actual(Id-1)
                        //Procedimiento que calcula la separacion que
habrá entre las ramas en función de sus anchos
                        Pos_x = Calcular_distancia(Abuelo, Padre);
                    }
                    Pos_y = y_ant + Dist_y;
                }
            }
        }
        else{ //Se puede calcular su posicion basandose en el elemento
anterior

            String Tipo_ant = Anterior.getAttributeValue("Tipo");
            String Id_ant = Anterior.getAttributeValue("Id");
            String Id_element = Actual.getAttributeValue("Id");
            String Tipo_element = Actual.getAttributeValue("Tipo");
            x_ant = 0;
            y_ant = 0;
            if (!(Tipo_ant.contains("Tipo_Y")) &&
(!(Tipo_ant.contains("Tipo_O")))){ // si el elemento anterior no es
una Rama
                x_ant
=Integer.parseInt(Anterior.getAttributeValue("x"));
                y_ant
=Integer.parseInt(Anterior.getAttributeValue("y"));
            }
            if ((Tipo_element.contains("InicioY")) ||
(Tipo_element.contains("InicioO"))){
                int nivel
=Integer.parseInt(Actual.getAttributeValue("nivel"));
                List Hijos = Actual.getChildren();
                int Num_Ramas = Hijos.size();
                int ancho_linea = 0;
                Iterator i = Hijos.iterator();
                Element Rama_anterior = (Element)i.next();
                int ancho_anterior
=Integer.parseInt(Rama_anterior.getAttributeValue("Ancho"));
                for (int j=0;j<Num_Ramas-1; j++){
                    Element Hijo= (Element)i.next();
                    int ancho
=Integer.parseInt(Hijo.getAttributeValue("Ancho"));
                    if ((ancho_anterior == 1) && (ancho == 1))
                        ancho_linea = ancho_linea + 2;
                    else
                    if (ancho_anterior < ancho)
                        ancho_linea = ancho_linea +
((ancho_anterior/(1+nivel) ) + ancho);
```

```
                    else
                        ancho_linea = ancho_linea + ((ancho/(1+nivel)
) + ancho_anterior);
                    ancho_anterior = ancho;
                }
                Pos_x = x_ant - (((ancho_linea*Dist_x))/2);
                Pos_y = y_ant + Dist_y;
            }
            else{
                if ((Tipo_ant.contains("FinalY")) ||
(Tipo_ant.contains("FinalO"))){ //Proceso de calcular la distancia
media
                    // Cuando el elemento anterior es un "FinalY" o
"FinalO" como para pintar nos quedamos con el extremo
                    // izquierdo de la linea, habrá que ver cuantos
hijos tiene, y como la posicion del ultimo hijo coincide
                    // con su extremo opuesto, coger la distancia de
linea media que será donde pintaremos el elemento siguiente
                    // en la posicion X. Eso es lo que haremos en lo
que sigue a continuación:
                    List Hermanos = Padre.getChildren();
                    int posicion =  Buscar_por_Posicion(Padre, Id_ant,
Tipo_ant);
                    Element Inicio_Elem_Graf = (Element)
Hermanos.get(posicion-1); //Nos quedamos con el Inico del Elemento
                    List Ramas_Elem_Graf =
Inicio_Elem_Graf.getChildren(); //Grafico, miramos el numero de ramas
que tiene
                    int Num_Ramas = Ramas_Elem_Graf.size(); //para
saber la longitud de la linea y quedarnos con la mitad
                    int nivel
=Integer.parseInt(Inicio_Elem_Graf.getAttributeValue("nivel"));
                    int ancho_linea = 0;
                    Iterator i = Ramas_Elem_Graf.iterator();
                    Element Rama_anterior = (Element)i.next();
                    int ancho_anterior
=Integer.parseInt(Rama_anterior.getAttributeValue("Ancho"));
                    for (int j=0;j<Num_Ramas-1; j++){
                        Element Hijo= (Element)i.next();
                        int ancho
=Integer.parseInt(Hijo.getAttributeValue("Ancho"));
                        //ancho_linea = ancho_linea
+(ancho_anterior/2)+ ancho;
                        if ((ancho_anterior == 1) && (ancho == 1))
                            ancho_linea = ancho_linea + 2;
                        else
                        if (ancho_anterior < ancho)
                            ancho_linea = ancho_linea +
((ancho_anterior/(1+nivel) ) + ancho);
                        else
                            ancho_linea = ancho_linea +
((ancho/(1+nivel)  ) + ancho_anterior);
                        ancho_anterior = ancho;
                    }
                    Pos_x = x_ant + (((ancho_linea*Dist_x))/2);
                    Pos_y = y_ant + Dist_y;
                }
                else{
                    if (Tipo_element.contains("Linea")){
                        Pos_x = x_ant;
                        Pos_y = y_ant + Dist_y;
```

```
                        Cambiar_Posicion(Actual,Pos_x,Pos_y);
                        Pila.Push(Actual);
                    }
                    else{
                        if ((Tipo_element.contains("FinalY")) ||
(Tipo_element.contains("FinalO"))){
                            List Hijos = Anterior.getChildren();
                            int Num_Hijos = Hijos.size();
                            int Max_prof = 0; //Esta variable nos dira
la profundidad maxima de la rama que tiene
                            //mas elementos.
                            // NOTA: Ahora iremos extrayendo tantos
elementos de la pila como hijos tiene el Elemento_Grafico
                            // y iremos quedandonos con su posicion en
Y e compararemos, de manera que nos quedaremos
                            // con la posicion Y mayor que será la que
nos dara la profundidad de la rama mas larga
                            for (int j=0; j<Num_Hijos; j++){
                                Element Extraido = Pila.Pop();
                                Pos_y =
Integer.parseInt(Extraido.getAttributeValue("y"));
                                if (Pos_y>Max_prof){
                                    Max_prof = Pos_y;
                                }
                            }
                            //Con la Profundidad maxima ya podemos
calcular la posicion que llevara el Elemento Grafico
                            //FinalY o FinalO que sera:
                            //1)En X: la misma que su elemento
anterior InicioY o InicioO
                            //2)En Y: Maxima profundidad + 10
                            x_ant
=Integer.parseInt(Anterior.getAttributeValue("x"));
                            Pos_x = x_ant;
                            Pos_y = Max_prof + Dist_prof;
                            //Asignamos las longitudes de lineas
correspondientes una vez calculada la Profundidad maxima
                            Iterator i = Hijos.iterator();
                            while (i.hasNext()){
                                Element e= (Element)i.next();
                                List Hijos_rama = e.getChildren();
                                Element Elem_linea = (Element)
Hijos_rama.get(Hijos_rama.size()-1);
                                int PosY_linea
=Integer.parseInt(Elem_linea.getAttributeValue("y"));
                                int longitud = Pos_y - PosY_linea;
                                Elem_linea.setAttribute("Id",
Integer.toString(longitud));
                            }
                        }
                        else{
                            Pos_x = x_ant;
                            Pos_y = y_ant + Dist_y;
                        }
                    }
                }
            }
        }
        String Tipo_act = Actual.getAttributeValue("Tipo");
        if (!(Tipo_act.contains("Tipo_Y")) &&
(!(Tipo_act.contains("Tipo_O"))) && (!(Tipo_act.contains("Linea")))){
```

```
            //No podemos asignar posicion cuando se hace la llamada
con rama

            Cambiar_Posicion(Actual,Pos_x,Pos_y);
        }
    }

    public void Cambiar_Posicion(Element Actual, int x, int y){

        Actual.setAttribute("x", Integer.toString(x));
        Actual.setAttribute("y", Integer.toString(y));
    }

    public void Cambiar_nivel(Element Actual, int nivel){

        Actual.setAttribute("nivel", Integer.toString(nivel));

    }
    public void Cambiar_nivel_rama(Element Rama){

        List Elementos_rama = Rama.getChildren();
        Iterator i = Elementos_rama.iterator();
        while ((i.hasNext())){
            Element e= (Element)i.next();
            String Tipo_Element =e.getAttributeValue("Tipo");
            List Hijos =e.getChildren();
            if ((Tipo_Element.contains("FinalY")) ||
(Tipo_Element.contains("FinalO")) ||
(Tipo_Element.contains("Linea"))){
                int nivel
=Integer.parseInt(e.getAttributeValue("nivel"));
                nivel--;
                Cambiar_nivel(e, nivel);
            }
            if (Hijos.size() != 0){
                if ((!(Tipo_Element.contains("Tipo_Y"))) &&
(!(Tipo_Element.contains("Tipo_O")))){
                    int nivel
=Integer.parseInt(e.getAttributeValue("nivel"));
                    nivel--;
                    Cambiar_nivel(e, nivel);
                }
                Cambiar_nivel_rama(e);
            }
        }
    }
    public boolean Guardar_max_nivel(int nivel){

        boolean superado = false;

        if (nivel == Max_nivel)
            superado = true;
        return(superado);
    }

    public void Limpiar_marcas_Act (Element Padre){
        List Elementos=Padre.getChildren();
        Iterator i = Elementos.iterator();
        while (i.hasNext()){
            Element e= (Element)i.next();
            String Tipo_Element =e.getAttributeValue("Tipo");
```

```
            if (Tipo_Element.contains("Estado") ||
Tipo_Element.contains("Transicion") ||
Tipo_Element.contains("Salto_condic")){
                e.setAttribute("Activacion", Integer.toString(0));
//Desactivamos la marca
            }
            List Hijos =e.getChildren();
            if (Hijos.size() != 0) {
                Limpiar_marcas_Act(e);
            }
        }
    }

    public int Anadir_Rama(){

        int codigo = 0; //Se puede añadir una rama

        Element Padre = Element_Encont.getParentElement();
        List Elementos=Padre.getChildren();
        if (Padre == root){
            codigo = 1; //NO esta dentro de una estructura 'Y' u 'O'
no se puede añadir
        }
        else{ // Ya sabemos que esta dentro de una Estructura 'Y' u
'O' y se puede añadir ramas
            Element Abuelo = Padre.getParentElement();
            String Tipo_estructura = Abuelo.getAttributeValue("Tipo");
            List Ramas=Abuelo.getChildren();
            int nivel
=Integer.parseInt(Abuelo.getAttributeValue("nivel"));
            int Num_Ramas = Ramas.size();
            boolean superado = Guardar_max_nivel(nivel+1);
            if (!(superado)){
                if (Tipo_estructura.contains("InicioY")){
                    Rama New_rama=new Rama("Tipo_Y", Num_Ramas, 0, 0);
                    Abuelo.addContent(Num_Ramas, New_rama);
                    Elemento New_estado=new Elemento("Estado",
Cont_est++, 0, 0, "","", 0,"");
                    New_rama.addContent(New_estado);
                    Elemento_Grafico Linea=new
Elemento_Grafico("Linea", 10, 0, 0, nivel);
                    New_rama.addContent(Linea);

                }
                else{
                    if (Tipo_estructura.contains("InicioO")){
                        Rama New_rama=new Rama("Tipo_O", Num_Ramas, 0,
0);
                        Abuelo.addContent(Num_Ramas, New_rama);
                        Elemento New_trans=new Elemento("Transicion",
Cont_tran++, 0, 0, "","", 0, "no");
                        New_rama.addContent(New_trans);
                        Elemento_Grafico Linea=new
Elemento_Grafico("Linea", 10, 0, 0, nivel);
                        New_rama.addContent(Linea);
                    }
                }
            }
            /*else{
                Frame_errorNewEstado errorNewEstado = new
Frame_errorNewEstado(29);
```

```
                errorNewEstado.show();
            }*/
        }
        return(codigo);
    }

    public void Insertar_TransicionO(int num_ramas){

        int posicion_insertar = 0;
        int nivel=0;

        String Tipo_encont =Element_Encont.getAttributeValue("Tipo");
        String Id_encontrado = Element_Encont.getAttributeValue("Id");
        Element Padre = Element_Encont.getParentElement();
        if (Padre != root){
            Element Abuelo = Padre.getParentElement();
            nivel
=Integer.parseInt(Abuelo.getAttributeValue("nivel"));
            nivel++;
        }
        //Miramos si se ha superado el máximo nivel de profundidad del
Grafcet, entonces no deja meter mas niveles
        boolean superado = Guardar_max_nivel(nivel);
        if (!(superado)) {
            List Elementos = Padre.getChildren();
            Iterator i = Elementos.iterator();
            int posicion = Buscar_por_Posicion(Padre, Id_encontrado,
Tipo_encont);
            if (Tipo_encont.contains("Estado")) {
                posicion_insertar = posicion + 1;
            }
            if (Tipo_encont.contains("Transicion")) {
                posicion_insertar = posicion;
            }
            Elemento Transic_inic = new Elemento("Transicion",
Cont_tran++, 0, 0, "", "", 0, "no");
            Padre.addContent(posicion_insertar, Transic_inic);
            Elemento Estado_inic = new Elemento("Estado", Cont_est++,
0, 0, "", "", 0, "");
            Padre.addContent(posicion_insertar + 1, Estado_inic);

            Elemento_Grafico Inicio_O = new
Elemento_Grafico("InicioO", Cont_tranO, 0, 0, nivel);

            Padre.addContent(posicion_insertar + 2, Inicio_O);
            Elemento_Grafico Final_O = new Elemento_Grafico("FinalO",
Cont_tranO++, 0, 0, nivel);
            Padre.addContent(posicion_insertar + 3, Final_O);
            Elemento Estado = new Elemento("Estado", Cont_est++, 0, 0,
"", "", 0, "");
            Padre.addContent(posicion_insertar + 4, Estado);
            for (int cont = 0; cont < num_ramas; cont++) {
                Rama New_rama = new Rama("Tipo_O", cont, 0, 0);
                Inicio_O.addContent(cont, New_rama);
                Elemento Transic_rama = new Elemento("Transicion",
Cont_tran++, 0, 0, "", "", 0, "no");
                New_rama.addContent(Transic_rama);
                Elemento_Grafico Linea = new Elemento_Grafico("Linea",
10, 0, 0, nivel);
                //El tamaño mínimo de Linea sera de 10 y después lo
recalculamos cuando crezca la estructura
```

```
                //utilizamos el Campo "Id" de la clase
Elemento_Grafico
                New_rama.addContent(Linea);

            }
        }
        /*else{
            Frame_errorNewEstado errorNewEstado = new
Frame_errorNewEstado(24);
            errorNewEstado.show();
        }*/
    }

    public void Insertar_TransicionY(int num_ramas){

        int posicion_insertar = 0;
        int nivel = 0;

        String Tipo_encont =Element_Encont.getAttributeValue("Tipo");
        String Id_encontrado = Element_Encont.getAttributeValue("Id");
        Element Padre = Element_Encont.getParentElement();
        if (Padre != root){ //Comprobamos el nivel que tiene su Abuelo
"InicioY"
            Element Abuelo = Padre.getParentElement();
            nivel
=Integer.parseInt(Abuelo.getAttributeValue("nivel"));
            nivel++;
        }
        //Miramos si se ha superado el máximo nivel de profundidad del
Grafcet, entonces no deja meter mas niveles
        boolean superado = Guardar_max_nivel(nivel);
        if (!(superado)){
            List Elementos=Padre.getChildren();
            Iterator i = Elementos.iterator();
            int posicion = Buscar_por_Posicion(Padre, Id_encontrado,
Tipo_encont);
            if (Tipo_encont.contains("Estado")){
                posicion_insertar = posicion+1;
            }
            if (Tipo_encont.contains("Transicion")){
                posicion_insertar = posicion;
            }
            Elemento Transic_inic = new Elemento("Transicion",
Cont_tran++, 0, 0,"","", 0, "no");
            Padre.addContent(posicion_insertar,Transic_inic);

            Elemento_Grafico Inicio_Y = new
Elemento_Grafico("InicioY", Cont_tranY, 0, 0,nivel);
           Padre.addContent(posicion_insertar+1, Inicio_Y);
            Elemento_Grafico Final_Y = new Elemento_Grafico("FinalY",
Cont_tranY++, 0, 0,nivel);
            Padre.addContent(posicion_insertar+2, Final_Y);
            Elemento Transic_final = new Elemento("Transicion",
Cont_tran++, 0, 0,"","", 0, "no");
            Padre.addContent(posicion_insertar+3, Transic_final);
            Elemento Estado_final=new Elemento("Estado", Cont_est++,
0, 0, "","", 0, "");
            Padre.addContent(posicion_insertar+4, Estado_final);
            for (int cont=0; cont<num_ramas; cont++){
                Rama New_rama=new Rama("Tipo_Y",cont, 0, 0);
                Inicio_Y.addContent(cont, New_rama);
```

```
                Elemento Estado_rama=new Elemento("Estado",
Cont_est++, 0, 0, "","", 0, "");
                New_rama.addContent(Estado_rama);
                Elemento_Grafico Linea=new Elemento_Grafico("Linea",
10, 0, 0, nivel);
                //El tamaño mínimo de Linea sera de 10 y después lo
recalculamos cuando crezca la estructura
                //utilizamos el Campo "Id" de la clase
Elemento_Grafico
                New_rama.addContent(Linea);
            }
        }
        /*else{
            Frame_errorNewEstado errorNewEstado = new
Frame_errorNewEstado(24);
            errorNewEstado.show();
        }*/
    }

    public void Insertar_Salto_Condicional(Element Transicion_origen,
Element Estado_destino)  {

        String Id_Estado = Estado_destino.getAttributeValue("Id");
        Elemento Salto= new Elemento("Salto_condic", Cont_saltos++, 0,
0, "","", 0, "si");
        Salto.setAttribute("Destino_cond", Id_Estado);
        //Buscamos la posicion donde esta la Transicion ya que el
Salto Condicional se insertará encima de esta
        String Id_transicion =
Transicion_origen.getAttributeValue("Id");
        Element Padre = Transicion_origen.getParentElement();
        List Elementos=Padre.getChildren();
        Iterator i = Elementos.iterator();
        int posicion = Buscar_por_Posicion(Padre, Id_transicion,
"Transicion");
        // posicion = posicion de la Transicion, luego insertaremos en
esa misma
        Padre.addContent(posicion, Salto);
    }
    //* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    // QUE HACE: Método que cambia el valor del atributo "Explorada"
de una rama. Esto sirve para cuando estamos en la
    //           simulación y estamos dentro de una transición 'Y'
para seguir en la simulación tras el "FinalY" debemos
    //           comprobar que ya se han explorado todas sus ramas
(Explorada = '1')
    // ENTRADA: * rama -> Element rama del que se cambiará la marca de
Exploracion a lo que le pasemos en Act
    //          * Act -> valor nuevo que tomará el campo "Explorada"
de la rama pasada por cabecera
    public void Cambiar_exploracion_rama(Element rama, int Act){

        rama.setAttribute("Explorada", Integer.toString(Act));
    }

    public void Recalcular_Id_Ramas(Element Padre){

        List Elementos=Padre.getChildren();
        Iterator i = Elementos.iterator();
        int cont_ramas = 0;
```

```
        while (i.hasNext()){
            Element e= (Element)i.next();
            e.setAttribute("Id", Integer.toString(cont_ramas++));
        }

    }

    public void Anadir_Etiquetas(Element Elemento_cambiar, String
Etiqueta1, String Etiqueta2){
        // Cogemos las etiquetas que ya tenia el elemento y les
añadimos un campo nuevo y el separador '/'
        String Etiq1_Actual =
Elemento_cambiar.getAttributeValue("Etiq1");
        Etiq1_Actual = Etiq1_Actual + Etiqueta1 + '/';
        String Etiq2_Actual =
Elemento_cambiar.getAttributeValue("Etiq2");
        Etiq2_Actual = Etiq2_Actual + Etiqueta2 + '/';
        // Metemos las etiquetas con los campos nuevos insertados
        Elemento_cambiar.setAttribute("Etiq1", Etiq1_Actual);
        Elemento_cambiar.setAttribute("Etiq2", Etiq2_Actual);
    }

    public int Comprobar_Etiqueta2(String Etiqueta2){
        int valida = 0;

        if (Etiqueta2.length() > 12)
            valida = -1;
        else{
            Character caracter = Etiqueta2.charAt(0);
            if (!(Character.isLetter(caracter)) && (!(caracter ==
'_')))
                valida = -1;
            if ((caracter != 'R') && (caracter != 'S'))
                valida = -1;
            else{
                int contador = 1;
                if ((caracter == 'R') || (caracter == 'S')){
                    if (Etiqueta2.length() > 1){
                        caracter = Etiqueta2.charAt(contador);
                        contador++;
                        if (caracter == '-'){
                            if (Etiqueta2.length() > 2){
                                caracter = Etiqueta2.charAt(contador);
                                if (caracter == '>'){
                                    contador++;
                                    if ((Etiqueta2.length() -
contador) == 0)
                                        valida = -1;
                                    else{
                                        caracter =
Etiqueta2.charAt(contador);
                                        if
(!(Character.isLetterOrDigit(caracter)))
                                            valida = -1;
                                    }
                                }
                                else
                                    valida = -1;
                            }
                            else
                                valida = -1;
```

```
                        }
                    }
                }
                while ((contador < Etiqueta2.length()) && (valida ==
0)){
                    caracter = Etiqueta2.charAt(contador);
                    if (!(Character.isLetterOrDigit(caracter)) &&
(!(caracter == '_')) && (!(caracter == '.'))){
                        valida = -1;
                    }
                    if (caracter == '.'){
                        Character anterior =
Etiqueta2.charAt(contador-1);
                        Character siguiente =
Etiqueta2.charAt(contador+1);
                        if (!(Character.isDigit(anterior)) ||
(!(Character.isDigit(siguiente)))){
                            valida = -1;
                        }
                        else{
                            int restante = Etiqueta2.length() -
contador;
                            System.out.println(restante);
                            if (restante > 3){
                                valida = -1;
                            }
                            if (restante == 3){
                                Character sig_siguiente =
Etiqueta2.charAt(contador+2);
                                if
(!(Character.isDigit(sig_siguiente)))
                                    valida = -1;
                            }
                        }
                    }
                    contador++;
                }
            }
        }
        return(valida);
    }

}
```

10.2. CLASE “DRAW”

Esta clase se encarga de la gestión de la interfaz y de la comunicación de todos sus elementos con la estructura de datos.

```
package e.brian.ullsimgraf;

import android.app.FragmentManager;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.text.Selection;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

import org.jdom2.Element;

import java.util.Iterator;
import java.util.List;
class Draw extends View{

    public Documento_XML Nuestro;
    int Activacion;
    int x;
    int y;
    int Cont_Visual_Est;
    String c;
    String Salto_condic;
    Element Elemento_salto;
    Element Element_simulacion = null;
    Element Padre;
    private Context s;
    boolean draw = false;
    boolean doc_created = false;
    boolean editar = false;
    boolean select = false;
    boolean Simular = false;
    boolean Encontrado;
    boolean b_ = true;

    public Draw(Context context, AttributeSet attrs){

        super(context, attrs);
        s = context;
    }

    public void newProject(){

        if(doc_created == false){
            draw = true;
```

```
            editar = true;
            Nuestro = new Documento_XML();
            Nuestro.Calcular_Posiciones(Nuestro.Devolver_root());
            doc_created = true;
            Padre = Nuestro.Devolver_root();
            invalidate();
        }
    }

    public void addState(){

        Nuestro.Insertar_Estado_Transicion();
        Nuestro.Calcular_Anchos(Nuestro.Devolver_root());
        Nuestro.Calcular_Posiciones(Nuestro.Devolver_root());
        invalidate();
    }

    public void addEtiquetas(String et1, String et2){

    }

    public void addRama(){

        int rama = 0;

        if(Nuestro.Element_Encont!= null) {
            rama = Nuestro.Anadir_Rama();
            if (rama == 0) {     //No se produce error al añadir la
rama
                Nuestro.Calcular_Anchos(Nuestro.Devolver_root());
                Nuestro.Calcular_Posiciones(Nuestro.Devolver_root());
            }
        }
        invalidate();
    }

    public void addOR(int num_ramas){

        if(Nuestro.Element_Encont != null){

            Nuestro.Insertar_TransicionO(num_ramas);
            Nuestro.Calcular_Anchos(Nuestro.Devolver_root());
            Nuestro.Calcular_Posiciones(Nuestro.Devolver_root());
        }

        invalidate();
    }

    public void addY(int num_ramas){

        if(Nuestro.Element_Encont != null){

            Nuestro.Insertar_TransicionY(num_ramas);
            Nuestro.Calcular_Anchos(Nuestro.Devolver_root());
            Nuestro.Calcular_Posiciones(Nuestro.Devolver_root());
        }

        invalidate();
    }

    public void addCondit(){
```

```
        if(Nuestro.Element_Encont != null){
            String Tipo_selec =
Nuestro.Element_Encont.getAttributeValue("Tipo");
            if(Tipo_selec.contains("Estado")){
                //Error, se debe seleccionar una transición
            }
            else{ // Se ha seleccionado una transición
                Salto_condic =
Nuestro.Element_Encont.getAttributeValue("Condicional");
                if(Salto_condic.contains("si")){
                    //Añadir Dialog Error
                }
                else{ // La transición no tiene ningún Salto
condicional asignado
                    Elemento_salto = Nuestro.Element_Encont;
                    select = true;
                }

            }

        }
    }
    private void MeterId_Salto_en_Estado(){

        String Id_Estado =
Nuestro.Element_Encont.getAttributeValue("Destino_cond");
        String Id_Trans = Integer.toString(Nuestro.Cont_saltos-1);
        String New_Id_Estado = Id_Estado + Id_Trans + '/';
        Nuestro.Element_Encont.setAttribute("Destino_cond",
New_Id_Estado);
        Nuestro.Element_Encont.setAttribute("Condicional", "si");

    }


    @Override
    protected void onSizeChanged (int ancho, int alto, int
ancho_anterior, int alto_anterior){

        super.onSizeChanged(ancho, alto, ancho_anterior,
alto_anterior);

    }
    public void PlaySim(){
        Simular = true;
        editar = false;
        Nuestro.Limpiar_marcas_Act(Nuestro.Devolver_root());
        Element Raiz = Nuestro.Devolver_root();
        Element_simulacion = (Element) Raiz.getContent(0);
        Element_simulacion.setAttribute("Activacion",
Integer.toString(1));
        Element_simulacion = (Element) Raiz.getContent(1);
        String Tipo_simulacion =
Element_simulacion.getAttributeValue("Tipo");
        Element_simulacion.setAttribute("Activacion",
Integer.toString(1));
        if (Tipo_simulacion.contains("Salto_condic")){
            Element_simulacion = (Element) Raiz.getContent(2);
            Element_simulacion.setAttribute("Activacion",
Integer.toString(1));
        }
```

```
        if (Tipo_simulacion.contains("InicioO")){
            List Ramas=Element_simulacion.getChildren();
            Iterator i = Ramas.iterator();
            while (i.hasNext()){
                Element Rama= (Element)i.next();
                Element Transicion = (Element) Rama.getContent(0);
                Transicion.setAttribute("Activacion",
Integer.toString(1));
            }
        }
        Cont_Visual_Est = 0;
        invalidate();
    }
    public void Simulacion(){

        Element Padre = Element_simulacion.getParentElement();
        List Hermanos = Padre.getChildren();
        String Tipo_seleccionado
=Element_simulacion.getAttributeValue("Tipo");
        String Id_seleccionado =
Element_simulacion.getAttributeValue("Id");
        Iterator i = Hermanos.iterator();
        //Buscamos el elemento que hemos pinchado
        Boolean Encontrado = false;
        Element e = null;
        String Tipo_element = null;
        String Id_element = null;
        Element Anterior = null;
        int posicion = 0;
        while ((i.hasNext())&&(!Encontrado)){
            Anterior = e;
            e= (Element)i.next();
            Tipo_element = e.getAttributeValue("Tipo");
            Id_element = e.getAttributeValue("Id");
            if ((Nuestro.Comparar_numeros(Id_seleccionado,Id_element))
&& (Tipo_element.contains(Tipo_seleccionado))){
                Encontrado = true;
            }
            else
                posicion++;
        }
        // * 1ª PARTE: COGEMOS EL ELEMENTO ANTERIOR AL SELECCIONADO
CON EL RATÓN

        // Salimos del bucle => e = TRANSICION o SALTO_CONDIC
seleccionado con el ratón
        //Hay que mirar el elemento anterior para borrar las marcas
que quedan del paso anterior
        //     1)    Anterior = elemento anterior a la "transicion" o
"salto_condic"
        // 1.1) Si Anterior = "FinalY" => habría que borrar las marcas
que quedaron en los estados al final de
        //      cada rama de la "Y"
        // 1.2) Si Anterior = "Estado" => ponemos su marca de
activacion a '0'
        // 1.3) Si Anterior = null => significa que la transicion esta
en el primer elemento de una rama de una
        //      transicion 'O', por tanto habrá que quitar la marca
del estado que esta antes del "InicioO"
        // 1.4) Si Anterior = "Salto_condic" => Quitar la marca de
activación del Salto condicional y además habrá
```

```
        //      que quitarla del Estado anterior a este

        String Tipo_anterior = "";
        if (Anterior != null)
            Tipo_anterior = Anterior.getAttributeValue("Tipo");
        if (Tipo_anterior.contains("FinalY")){    // 1.1)
            Nuestro.Eliminar_Marcas_en_Y_O(Anterior);
        }
        if (Tipo_anterior.contains("Estado")){  // 1.2)
            Anterior.setAttribute("Activacion", Integer.toString(0));
        }
        if (Anterior == null){                  // 1.3)
            Element Abuelo = Padre.getParentElement();          //Padre
: Rama donde está la transicion seleccionada
            Element Padre_Abuelo = Abuelo.getParentElement();    //
Abuelo: Elemento grafico "InicioO"
            List Hermanos_InicioO = Padre_Abuelo.getChildren();//
PAdre_Abuelo: Padre del "InicioO"
            String Id_InicioO =Abuelo.getAttributeValue("Id");
            Iterator j = Hermanos_InicioO.iterator();
            posicion = 0;
            Encontrado = false;
            while ((j.hasNext())&&(!Encontrado)){
                Element buscado= (Element)j.next();
                String Id_encontrado =buscado.getAttributeValue("Id");
                String Tipo_encont =
buscado.getAttributeValue("Tipo");
                if
((Nuestro.Comparar_numeros(Id_encontrado,Id_InicioO)) &&
(Tipo_encont.contains("InicioO"))){
                    Encontrado = true;
                }
                else
                    posicion++;
            }
            // En posicion tenemos el elemento grafico "InicioO",
entonces para ir al Estado anterior nos situamos
            // en posicion-1
            Element Estado = (Element) Hermanos_InicioO.get(posicion-
1);
            Estado.setAttribute("Activacion", Integer.toString(0));

        }
        if (Tipo_anterior.contains("Salto_condic")){  // 1.4)
            Anterior.setAttribute("Activacion", Integer.toString(0));
            //Cogemos el elemento que está en dos posiciones por
detras de "posicion" que es donde
            // esta la transicion seleccionada con el ratón, este
puede ser:
            // 1) Un ESTADO
            // 2) Un FINALY
            Element Elemento = (Element) Hermanos.get(posicion-2);
            String Tipo_Elemento = Elemento.getAttributeValue("Tipo");
            if (Tipo_Elemento.contains("Estado")){
                Elemento.setAttribute("Activacion",
Integer.toString(0));
            }
            if (Tipo_Elemento.contains("FinalY")){
                Nuestro.Eliminar_Marcas_en_Y_O(Elemento);
            }
        }
```

```
        // * 2ª PARTE: COGEMOS EL ELEMENTO SIGUIENTE AL SELECCIONADO
CON EL RATÓN
        // Siguiente = Elemento que precede a la Transicion
seleccionada con el raton

        Element Siguiente = (Element) i.next();
        Tipo_element = Siguiente.getAttributeValue("Tipo");
        //      2)  Si hemos seleccionado una TRANSICION solo podrian
venir los elementos siguientes
        //          2.1) Un 'ESTADO':
        //          2.2) Un 'INICIOY':
        //          2.3) Una 'LINEA':
        //          2.4) El elemento 'FIN':
        //      Y en el caso en que hayamos seleccionado con el ratón
un SALTO_CONDIC, solo podrá venir después
        //          2.5) Una 'TRANSICIÓN':
        if (Tipo_element.contains("Estado")) { // 2.1)
            Comprobar_Estado(Siguiente);
        }
        if (Tipo_element.contains("InicioY")){ // 2.2)
            List Ramas = Siguiente.getChildren();
            int Num_Ramas = Ramas.size();
            i = Ramas.iterator();
            for (int j=0;j<Num_Ramas; j++){
                Element Rama= (Element)i.next();
                Element Primer_rama = (Element)
Rama.getContent(0);//Coge el primer ESTADO de cada rama
                Comprobar_Estado(Primer_rama);
            }
        }
        if (Tipo_element.contains("Linea")){  // 2.3)
            //Primero borramos todas las marcas que quedan activas en
las ramas
            Element Abuelo = Padre.getParentElement(); //Abuelo =
"InicioO"
            List Ramas = Abuelo.getChildren();
            int Num_Ramas = Ramas.size();
            Nuestro.Limpiar_marcas_Act(Abuelo);
            //Ahora tenemos que ACTIVAR el ESTADO que va después del
"FinalO"
            Element Padre_Abuelo = Abuelo.getParentElement();
            String Id_InicioO =Abuelo.getAttributeValue("Id");
            List Hermanos_InicioO = Padre_Abuelo.getChildren();
            Iterator j = Hermanos_InicioO.iterator();
            Encontrado = false;
            while ((j.hasNext())&&(!Encontrado)){
                e= (Element)j.next();
                String Id_encontrado =e.getAttributeValue("Id");
                String Tipo_encont = e.getAttributeValue("Tipo");
                if
((Nuestro.Comparar_numeros(Id_encontrado,Id_InicioO)) &&
(Tipo_encont.contains("InicioO"))){
                    Encontrado = true;
                }
            }
            e= (Element)j.next(); //Aqui tendriamos el elemento
"FinalO"
            e= (Element)j.next(); //Aqui tendriamos el elemento
"Estado" que va despues de la 'O'
            Comprobar_Estado(e);
```

```
        }
       /* if (Tipo_element.contains("End")){        // 2.4)
            this.botonSTOPMousePressed(null);
        }*/
        if (Tipo_element.contains("Transicion")){ // 2.5)
            //Buscamos el Estado hacia el que va el Salto Condicional
y quitamos la marca de la transicion que le
            //precede
            String Id_salto =
Element_simulacion.getAttributeValue("Destino_cond");
            int Pos_destino = Nuestro.Buscar_por_Posicion(Padre,
Id_salto , "Estado");
            Element Estado_destino = (Element)
Hermanos.get(Pos_destino);
            Siguiente.setAttribute("Activacion", Integer.toString(0));
            Comprobar_Estado(Estado_destino);

        }
        //if (!((Tipo_seleccionado.contains("Salto_condic"))))
        Element_simulacion.setAttribute("Activacion",
Integer.toString(0));
      //  LimpiarAreaTrabajo(AreaTrabajo.getGraphics());
        Cont_Visual_Est = 0;
        invalidate();
    }

    // * * * * * * * * * * * * * * * * * PROCEDIMIENTOS DE SIMULACION
GUIADA * * * * * * * * * * * * * * * * * * * * *
    //* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * *
    // QUE HACE: Procedimiento que es llamado desde el Procedimiento
"Simulacion()"
    public void Comprobar_Estado(Element Estado){

        Estado.setAttribute("Activacion", Integer.toString(1));
        //Buscamos el elemento dentro de la lista de hijos del padre
para poder quedarnos con el elemento siguiente
        Element Padre = Estado.getParentElement();
        List Hermanos = Padre.getChildren();
        String Tipo_seleccionado =Estado.getAttributeValue("Tipo");
        String Id_seleccionado = Estado.getAttributeValue("Id");
        Iterator i = Hermanos.iterator();
        //Buscamos el elemento que hemos pasado por la cabecera
        Boolean Encontrado = false;
        Element e = null;
        String Tipo_element = null;
        String Id_element = null;
        int posicion = 0;
        while ((i.hasNext())&&(!Encontrado)){
            e= (Element)i.next();
            Tipo_element = e.getAttributeValue("Tipo");
            Id_element = e.getAttributeValue("Id");
            if ((Nuestro.Comparar_numeros(Id_seleccionado,Id_element))
&& (Tipo_element.contains(Tipo_seleccionado))){
                Encontrado = true;
            }
            else
                posicion++;
        }
        //Cuando salimos del bucle, en 'e' tenemos el elemento pasado
por la cabecera pero cogido de la lista de
```

```
        //hijos, de manera que ahora podemos ver el elemento que le
precede asi como demas hermanos
        Element Siguiente = (Element) i.next();
        Tipo_element = Siguiente.getAttributeValue("Tipo");
        //Despues de un ESTADO solo podrian venir los elementos
siguientes
        // A) Una 'TRANSICION': Se activa la transicion
        // B) Una 'LINEA': significa que estamos dentro de una
estructura tipo 'Y' ya que es la única manera de
        //    que despues de un estado venga una linea.Ha llegado al
final de esa rama pero para saber si se sale
        //    de la estructura 'Y' debemos de mirar si todas las ramas
han sido marcadas, esto es, que en todas las
        //    ramas se ha alcanzado el elemento linea.
        //    -> Si se han finalizado todas las ramas: se deja la
marca en el ultimo estado y se marca la siguiente transicion
        //       que va despues del "FINALY"
        //    -> Si no se han finalizado todas las ramas: se marca la
rama como terminada y se espera
        // C) Un 'INICIOO':
        // D) Un 'SALTO_CONDIC': Activamos la transición condicional y
también la Transición que precede a esta
        if (Tipo_element.contains("Transicion")){ // A)
            Siguiente.setAttribute("Activacion", Integer.toString(1));
        }
        if (Tipo_element.contains("Linea")){      // B)
            Nuestro.Cambiar_exploracion_rama(Padre, 1); //Activamos la
rama como EXPLORADA
            Element Abuelo = Padre.getParentElement(); //Elemento
grafico "InicioY"
            List Ramas = Abuelo.getChildren(); //Lista de todas las
ramas de la 'Y'
            i = Ramas.iterator();
            boolean Acabado = true;
            while ((i.hasNext())&&(Acabado)){
                Element Rama= (Element)i.next();
                int Explo
=Integer.parseInt(Rama.getAttributeValue("Explorada"));
                if (Explo == 0)
                    Acabado = false;
            }
            if (Acabado){
                // HAY QUE QUITAR LA EXPLORACION A TODAS LAS RAMAS:
                i = Ramas.iterator();
                while (i.hasNext()){
                    Element Rama= (Element)i.next();
                    Rama.setAttribute("Explorada",
Integer.toString(0));
                }
                // Tenemos que encontrar el elemento "FinalY" y
quedarnos con la TRANSICION siguiente para activarla
                // Para ello y teniendo el "InicioY"(Abuelo) buscamos
dentro de una lista hasta que coincida el Id
                Element Padre_Abuelo = Abuelo.getParentElement();
                List Hermanos_abuelo = Padre_Abuelo.getChildren();
                String Id_Buscado =Abuelo.getAttributeValue("Id");
                Iterator h = Hermanos_abuelo.iterator();
                posicion = 0;
                Encontrado = false;
                while ((h.hasNext())&&(!Encontrado)){
                    Element buscado= (Element)h.next();
```

```
                    String Id_encontrado
=buscado.getAttributeValue("Id");
                    String Tipo_encont =
buscado.getAttributeValue("Tipo");
                    if
((Nuestro.Comparar_numeros(Id_encontrado,Id_Buscado)) &&
(Tipo_encont.contains("InicioY"))){
                        Encontrado = true;
                    }
                    else
                        posicion++;
                }
                // Hermanos_abuelo(posicion) = "InicioY"
Hermanos_abuelo(posicion+1) = "FinalY"  Entonces en
                // posicion+2 tenemos la transicion que buscamos
                Element buscado = (Element)
Hermanos_abuelo.get(posicion+2);
                buscado.setAttribute("Activacion",
Integer.toString(1));//Activamos la "Transicion" o "Salto_condic"
                String Tipo_buscado =
buscado.getAttributeValue("Tipo");
                //SI despues del "FinalY" viene un Salto_condic ademas
habrá que activar la Transicion que está por
                //debajo de este (posicion+3)
                if (Tipo_buscado.contains("Salto_condic")){
                    buscado = (Element)
Hermanos_abuelo.get(posicion+3);
                    buscado.setAttribute("Activacion",
Integer.toString(1));
                }

            }
        }
        if (Tipo_element.contains("InicioO")){     // C)
            List Ramas = Siguiente.getChildren();
            int Num_Ramas = Ramas.size();
            i = Ramas.iterator();
            for (int j=0;j<Num_Ramas; j++){
                Element Rama= (Element)i.next();
                Element Primer_rama = (Element)
Rama.getContent(0);//Coge la primera TRANSICION de cada rama
                Primer_rama.setAttribute("Activacion",
Integer.toString(1));// y la pone a ACTIVA
            }
        }
        if (Tipo_element.contains("Salto_condic")){     // D)
            Siguiente.setAttribute("Activacion", Integer.toString(1));
            Siguiente =  (Element) Padre.getContent(posicion+2);
            Siguiente.setAttribute("Activacion", Integer.toString(1));
        }
    }

    public void onDraw (Canvas s) {
        int Tamaño = Nuestro.Dist_y;
        super.onDraw(s);
        Paint fillPaint = new Paint();
        fillPaint.setStyle(Paint.Style.FILL);
        fillPaint.setColor(Color.WHITE);

        Paint fillSelect = new Paint();
        fillSelect.setStyle(Paint.Style.FILL);
```

```
        fillSelect.setColor(Color.BLUE);

        Paint strokePaint = new Paint();
        strokePaint.setStyle(Paint.Style.STROKE);
        strokePaint.setStrokeWidth(5);
        strokePaint.setColor(Color.BLACK);

        Paint strokeSelect = new Paint();
        strokeSelect.setStyle(Paint.Style.STROKE);
        strokeSelect.setStrokeWidth(5);
        strokeSelect.setColor(Color.RED);

        Paint textPaint = new Paint();
        textPaint.setTextSize(50);
        textPaint.setTypeface(Typeface.DEFAULT);

        Paint textSelect = new Paint();
        textSelect.setTextSize(50);
        textSelect.setTypeface(Typeface.DEFAULT);
        textSelect.setColor(Color.RED);
            if (draw) {
                List Elementos = Padre.getChildren();
                Iterator i = Elementos.iterator();
                while (i.hasNext()){
                    Element e = (Element) i.next();
                    x = Integer.parseInt(e.getAttributeValue("x"));
                    y = Integer.parseInt(e.getAttributeValue("y"));
                    String Tipo_Element = e.getAttributeValue("Tipo");
                    String Etiqueta = "";
                    //Guardamos la Etiqueta que se va a mostrar según
la opción de Etiqueta que este seleccionada
                    /* if ((Tipo_Element.contains("Estado"))
||(Tipo_Element.contains("Transicion")) ||
                        (Tipo_Element.contains("Salto_condic"))) {
                            if (EtiquetaNivel == 1)
                                Etiqueta =
e.getAttributeValue("Etiq1");
                            if (EtiquetaNivel == 2)
                                Etiqueta =
e.getAttributeValue("Etiq2");
                        }*/

                    if (Tipo_Element.contains("Estado")) {
                        String c =(e.getAttributeValue("Id"));
                        int Act =
Integer.parseInt(e.getAttributeValue("Activacion"));
                        String Condicional =
e.getAttributeValue("Condicional");
                        String Id_Salto =
e.getAttributeValue("Destino_cond");
                        RectF rectF = new RectF();
                        if (Act == 0) {
                            rectF.set(x - (Tamaño/2), y -(Tamaño/2)
,(x - (Tamaño/2))
                                    + Tamaño,(y -(Tamaño/2)) +
Tamaño);
                            s.drawRoundRect(rectF, 10, 10, fillPaint);
                            s.drawRoundRect(rectF, 10, 10,
strokePaint);
                            s.drawText(c, x- (Tamaño/2), y
+(Tamaño/4), textPaint);
```

```
                        }
                        if(Act == 1) {
                            rectF.set(x - (Tamaño/2), y -(Tamaño/2)
,(x - (Tamaño/2))
                                    + Tamaño,(y -(Tamaño/2)) +
Tamaño);
                            s.drawRoundRect(rectF, 10, 10,
fillSelect);
                            s.drawRoundRect(rectF, 10, 10,
strokeSelect);
                            s.drawText(c, x- (Tamaño/2), y
+(Tamaño/4), textSelect);
                        }
                        if(Simular && Act ==1){
                            rectF.set(x - (Tamaño/2), y -(Tamaño/2)
,(x - (Tamaño/2))
                                    + Tamaño,(y -(Tamaño/2)) +
Tamaño);
                            s.drawRoundRect(rectF, 10, 10, fillPaint);
                            s.drawRoundRect(rectF, 10, 10,
strokePaint);
                            s.drawCircle(x, y, Tamaño/3,strokeSelect);
                            s.drawCircle(x, y , Tamaño/3,fillSelect);
                        }
                        if (Integer.parseInt(c) == 0) {
                            rectF.set(x - (Tamaño/2), y -(Tamaño/2)
,(x - (Tamaño/2))
                                    + Tamaño,(y -(Tamaño/2)) +
Tamaño);
                            rectF.set(x - (Tamaño/2), y -(Tamaño/2)
,(x - (Tamaño/2))
                                    + Tamaño, (y -(Tamaño/2))+
Tamaño);
                            s.drawRoundRect(rectF, 10, 10,
strokeSelect);
                        }
                        if (Condicional.contains("si")){
                            //Dibujamos la flecha que indica que es
salto condicional
                            s.drawLine(x -(Tamaño/2) , y , x -
(Tamaño+Tamaño/4), y, strokePaint);
                            s.drawLine(x - (Tamaño/2) , y , x -
(Tamaño), y
                                    + (Tamaño/4), strokePaint);//
Primera linea de la flecha
                            s.drawLine(x - (Tamaño/2) , y , x -
(Tamaño), y
                                    - (Tamaño/4), strokePaint);//
Segunda linea de la flecha
                            //Dibujamos el identificador del salto
condicional al lado izquierdo de la flecha entrante
                            s.drawText(Id_Salto, x -(Tamaño +
Id_Salto.length()*30), y + (Tamaño/4),textPaint);
                            invalidate();
                        }

                    }

                    if (Tipo_Element.contains("Transicion")) {
                        int Act =
Integer.parseInt(e.getAttributeValue("Activacion"));
```

```
                        String Condicional =
e.getAttributeValue("Condicional");
                        String Id_Salto =
e.getAttributeValue("Destino_cond");
                        if (Act == 0) {
                            s.drawLine(x, y - (Tamaño/2), x , y +
(Tamaño/2), strokePaint);
                            s.drawLine(x , y , x + (Tamaño/2), y,
strokePaint);
                            s.drawLine(x + (Tamaño/2) , y -(Tamaño/5)
, x +
                                    (Tamaño/2), y +(Tamaño/5),
strokePaint);
                        }
                        if (Act == 1) {
                            s.drawLine(x, y - (Tamaño/2), x , y +
(Tamaño/2), strokeSelect);
                            s.drawLine(x , y , x + (Tamaño/2), y,
strokeSelect);
                            s.drawLine(x + (Tamaño/2) , y -(Tamaño/5)
, x +
                                    (Tamaño/2), y +(Tamaño/5),
strokeSelect);
                        }
                    }

                    if (Tipo_Element.contains("Salto_condic")) {
                        int Act =
Integer.parseInt(e.getAttributeValue("Activacion"));
                        String Id_Salto = e.getAttributeValue("Id");
                        if (Act == 0) {
                            s.drawLine(x, y - (Tamaño/2), x , y +
(Tamaño/2), strokePaint);
                            s.drawLine(x , y , x + (Tamaño/2), y,
strokePaint);
                            s.drawLine(x + (Tamaño/2) , y -(Tamaño/5)
, x +
                                    (Tamaño/2), y +(Tamaño/5),
strokePaint);
                            s.drawLine(x , y , x - (Tamaño), y,
strokePaint);
                            s.drawLine(x - (Tamaño) , y , x -
(Tamaño/2), y
                                    + (Tamaño/4),strokePaint);//
Primera linea de la flecha
                            s.drawLine(x - (Tamaño) , y , x -
(Tamaño/2), y
                                    - (Tamaño/4), strokePaint);//
Segunda linea de la flecha
                            //Dibujamos el identificador a un lado
                            s.drawText(Id_Salto, x- (Tamaño)-
(Tamaño/2) , y + (Tamaño/4), textPaint);
                        }
                        if (Act == 1) {
                            s.drawLine(x, y - (Tamaño/2), x , y +
(Tamaño/2), strokeSelect);
                            s.drawLine(x , y , x + (Tamaño/2), y,
strokeSelect);
                            s.drawLine(x + (Tamaño/2) , y -(Tamaño/5)
, x +
                                    (Tamaño/2), y +(Tamaño/5),
```

```
strokeSelect);
                            s.drawLine(x , y , x - (Tamaño), y,
strokeSelect);
                            s.drawLine(x - (Tamaño) , y , x -
(Tamaño/2), y
                                    + (Tamaño/4),strokeSelect);//
Primera linea de la flecha
                            s.drawLine(x - (Tamaño) , y , x -
(Tamaño/2), y
                                    - (Tamaño/4), strokeSelect);//
Segunda linea de la flecha
                            //Dibujamos el identificador a un lado
                            s.drawText(Id_Salto, x- (Tamaño)-
(Tamaño/2) , y + (Tamaño/4), textSelect);
                        }
                    }

                    if (Tipo_Element.contains("InicioY")) {
                        int nivel =
Integer.parseInt(e.getAttributeValue("nivel"));
                        List Hijos = e.getChildren();
                        int ancho_linea = 0;
                        int Tamañolinea = Documento_XML.Dist_x;
                        int Num_Ramas = Hijos.size();
                        Iterator k = Hijos.iterator();
                        Element Rama_anterior = (Element)k.next();
                        int ancho_anterior
=Integer.parseInt(Rama_anterior.getAttributeValue("Ancho"));
                        for (int j=0;j<Num_Ramas-1; j++){
                            Element Hijo= (Element)k.next();
                            int ancho
=Integer.parseInt(Hijo.getAttributeValue("Ancho"));
                            //Formula para calcular el ancho de linea
adecuado segun su situacion y el de su hermano por la izqda.
                            if ((ancho_anterior == 1) && (ancho == 1))
{
                                ancho_linea = ancho_linea + 2;
                            }
                            else {
                                if (ancho_anterior < ancho)
                                    ancho_linea = ancho_linea +
((ancho_anterior / (1 + nivel)) + ancho);
                                else {
                                    ancho_linea = ancho_linea +
((ancho / (1 + nivel)) + ancho_anterior);
                                }
                                ancho_anterior = ancho;
                            }
                        }
                        s.drawLine(x , (y - (Tamaño/2)), x +
(Tamañolinea * ancho_linea), (y - (Tamaño/2)), strokePaint);
                        s.drawLine(x , (y - (Tamaño/4)), x +
(Tamañolinea * ancho_linea), (y - (Tamaño/4)), strokePaint);
                        Iterator h = Hijos.iterator();
                        while (h.hasNext()){
                            Element f = (Element)h.next();
                            List Hijos_rama = f.getChildren();
                            Element Primer_element = (Element)
Hijos_rama.get(0);
                            int Pos_X
=Integer.parseInt(Primer_element.getAttributeValue("x"));
```

```
                            s.drawLine(Pos_X, (y - (Tamaño/4)), Pos_X,
(y + (Tamaño/2)), strokePaint);
                        }
                    }
                    if (Tipo_Element.contains("InicioO")) {
                        int nivel =
Integer.parseInt(e.getAttributeValue("nivel"));
                        List Hijos = e.getChildren();
                        int Num_Ramas = Hijos.size();
                        int Tamañolinea = Nuestro.Dist_x;
                        int ancho_linea = 0;
                        Iterator k = Hijos.iterator();
                        Element Rama_anterior = (Element)k.next();
                        int ancho_anterior
=Integer.parseInt(Rama_anterior.getAttributeValue("Ancho"));
                        for (int j=0;j<Num_Ramas-1; j++){
                            Element Hijo= (Element)k.next();
                            int ancho
=Integer.parseInt(Hijo.getAttributeValue("Ancho"));
                            //Formula para calcular el ancho de linea
adecuado segun su situacion y el de su hermano por la izqda.
                            if ((ancho_anterior == 1) && (ancho == 1))
{
                                ancho_linea = ancho_linea + 2;
                            }
                            else {
                                if (ancho_anterior < ancho) {
                                    ancho_linea = ancho_linea +
((ancho_anterior / (1 + nivel)) + ancho);
                                }
                                else {
                                    ancho_linea = ancho_linea +
((ancho / (1 + nivel)) + ancho_anterior);
                                }
                                ancho_anterior = ancho;

                            }
                        }
                        s.drawLine(x, y, x +
(Tamañolinea*(ancho_linea)), y, strokePaint);
                        s.drawLine(x +
((((ancho_linea)*Tamañolinea))/2), y, x +
                                (((ancho_linea)*Tamañolinea)/2), (y -
(Tamaño/2)), strokePaint);
                        Iterator h = Hijos.iterator();
                        while (h.hasNext()){
                            Element m= (Element)h.next();
                            List Hijos_rama = m.getChildren();
                            Element Primer_element = (Element)
Hijos_rama.get(0);
                            int Pos_X
=Integer.parseInt(Primer_element.getAttributeValue("x"));
                            s.drawLine(Pos_X, (y), Pos_X, (y +
(Tamaño/2)), strokePaint);

                        }
                    }
                    if (Tipo_Element.contains("FinalY")) {
                        String Id_Element = e.getAttributeValue("Id");
                        Iterator h = Elementos.iterator();
                        int posicion = 0;
```

```
                        Boolean Encontrado = false;
                        while ((h.hasNext()) && (!Encontrado)) {
                            Element buscado = (Element) h.next();
                            String Id_encontrado =
buscado.getAttributeValue("Id");
                            String Tipo_encont =
buscado.getAttributeValue("Tipo");
                            if
((Nuestro.Comparar_numeros(Id_encontrado, Id_Element)) &&
(Tipo_encont.contains("InicioY"))) {
                                Encontrado = true;
                            }
                            else {
                                posicion++;
                            }
                        }
                        Element buscado = (Element)
Elementos.get(posicion);
                        List Hijos = buscado.getChildren();
                        int nivel =
Integer.parseInt(buscado.getAttributeValue("nivel"));
                        int Tamañolinea = Documento_XML.Dist_x;
                        int Num_Ramas = Hijos.size();
                        int ancho_linea = 0;
                        Iterator k = Hijos.iterator();
                        Element Rama_anterior = (Element)k.next();
                        int ancho_anterior
=Integer.parseInt(Rama_anterior.getAttributeValue("Ancho"));
                        for (int j=0;j<Num_Ramas-1; j++){
                            Element Hijo= (Element)k.next();
                            int ancho
=Integer.parseInt(Hijo.getAttributeValue("Ancho"));
                            //Formula para calcular el ancho de linea
adecuado segun su situacion y el de su hermano por la izqda.
                            if ((ancho_anterior == 1) && (ancho == 1))
{
                                ancho_linea = ancho_linea + 2;
                            }
                            else {
                                if (ancho_anterior < ancho) {
                                    ancho_linea = ancho_linea +
((ancho_anterior / (1 + nivel)) + ancho);
                                }
                                else {
                                    ancho_linea = ancho_linea +
((ancho / (1 + nivel)) + ancho_anterior);
                                }
                                ancho_anterior = ancho;

                            }
                        }
                        s.drawLine(x , y, x + (Tamañolinea *
ancho_linea), y, strokePaint);
                        s.drawLine(x , (y + (Tamaño/4)), x +
(Tamañolinea * ancho_linea), (y + (Tamaño/4)), strokePaint);
                        //Pintamos la rayita pequeña en mitad de las
lineas de "FinalY"
                        s.drawLine(x +
(((ancho_linea*Tamañolinea))/2), (y + (Tamaño/4)),
                                x + (((ancho_linea*Tamañolinea))/2),
(y + (Tamaño/2)), strokePaint);
```

```
                    }

                    if (Tipo_Element.contains("FinalO")) {
                        String Id_Element = e.getAttributeValue("Id");
                        Iterator h = Elementos.iterator();
                        int posicion = 0;
                        Boolean Encontrado = false;
                        while ((h.hasNext()) && (!Encontrado)) {
                            Element buscado = (Element) h.next();
                            String Id_encontrado =
buscado.getAttributeValue("Id");
                            String Tipo_encont =
buscado.getAttributeValue("Tipo");
                            if
((Nuestro.Comparar_numeros(Id_encontrado, Id_Element)) &&
(Tipo_encont.contains("InicioO"))) {
                                Encontrado = true;
                            }
                            else {
                                posicion++;
                            }
                        }
                        Element buscado = (Element)
Elementos.get(posicion);
                        List Hijos = buscado.getChildren();
                        int nivel =
Integer.parseInt(buscado.getAttributeValue("nivel"));
                        int Tamañolinea = Nuestro.Dist_x;
                        int Num_Ramas = Hijos.size();
                        int ancho_linea = 0;
                        Iterator k = Hijos.iterator();
                        Element Rama_anterior = (Element)k.next();
                        int ancho_anterior =
Integer.parseInt(Rama_anterior.getAttributeValue("Ancho"));
                        for (int j=0;j<Num_Ramas-1; j++) {
                            Element Hijo = (Element) k.next();
                            int ancho =
Integer.parseInt(Hijo.getAttributeValue("Ancho"));
                            //Formula para calcular el ancho de linea
adecuado segun su situacion y el de su hermano por la izqda.
                            if ((ancho_anterior == 1) && (ancho == 1))
{
                                ancho_linea = ancho_linea + 2;
                            }
                            else {
                                if (ancho_anterior < ancho) {
                                    ancho_linea = ancho_linea +
((ancho_anterior / (1 + nivel)) + ancho);
                                }
                                else {
                                    ancho_linea = ancho_linea +
((ancho / (1 + nivel)) + ancho_anterior);
                                }
                                ancho_anterior = ancho;
                            }
                        }
                        s.drawLine(x , y, x + (Tamañolinea *
ancho_linea), y, strokePaint);
                        s.drawLine(x , (y + (Tamaño/4)), x +
(Tamañolinea * ancho_linea), (y + (Tamaño/4)), strokePaint);
                        //Pintamos la rayita pequeña en mitad de las
```

```
lineas de "FinalY"
                        s.drawLine(x +
(((ancho_linea*Tamañolinea))/2), (y + (Tamaño/4)),
                                x + (((ancho_linea*Tamañolinea))/2),
(y + (Tamaño/2)), strokePaint);
                    }
                    if (Tipo_Element.contains("Linea")) {
                        int tamano =
Integer.parseInt(e.getAttributeValue("Id"));
                        int Tamañolinea = Nuestro.Dist_y;
                        s.drawLine(x, y - (Tamaño / 2), x, y +
Tamañolinea, strokeSelect);
                    }
                    if (Tipo_Element.contains("End")){
                        int Pos_iniX = Documento_XML.PosX_inic;
                        int Pos_iniY = Documento_XML.PosY_inic;
                        int Pos_Xmin = Documento_XML.PosX_min;
                        s.drawCircle(x , y , Tamaño/2, strokePaint);
                        s.drawCircle(x , y , Tamaño/2, fillPaint);
                        // Pintamos la linea de retorno al Estado
inicial
                        s.drawLine(x - (Tamaño/2) , y , Pos_Xmin -
(Tamaño * 2) , y, strokePaint );
                        s.drawLine(Pos_Xmin - (Tamaño * 2) , y ,
Pos_Xmin - (Tamaño * 2), Pos_iniY, strokePaint );
                        s.drawLine(Pos_Xmin - (Tamaño * 2) , Pos_iniY
, Pos_iniX - (Tamaño/2), Pos_iniY, strokePaint );
                        // Pintamos el extremo final de la linea de
retorno (la flecha)
                        s.drawLine( Pos_iniX - Tamaño , Pos_iniY -
(Tamaño/4), Pos_iniX - (Tamaño/2), Pos_iniY,strokePaint );
                        s.drawLine( Pos_iniX - Tamaño , Pos_iniY +
(Tamaño/4), Pos_iniX - (Tamaño/2), Pos_iniY,strokePaint );
                        s.drawText("Fin", x- (Tamaño/2), y
+(Tamaño/4), textPaint);
                    }
                   /* List Hijos =e.getChildren();
                    if (Hijos.size() != 0){
                        Padre = e;
                        invalidate();
                    }*/
                }
            }
    }

 //   @Override
    public boolean onTouchEvent(MotionEvent event) {

        if(doc_created == true) {

            if (event.getAction() == MotionEvent.ACTION_DOWN) {
                int x_touch = (int) event.getX();
                int y_touch = (int) event.getY();
                Point touch = new Point(x_touch, y_touch);
                Encontrado =
Nuestro.Buscar_Elemento(Nuestro.Devolver_root(), touch);

                if(Encontrado && editar) {

Nuestro.Limpiar_marcas_Act(Nuestro.Devolver_root());
                    Nuestro.Element_Encont.setAttribute("Activacion",
```

```
Integer.toString(1));
                }

                if(select){
                    String Tipo_encont =
Nuestro.Element_Encont.getAttributeValue("Tipo");
                    if(Tipo_encont.contains("Estado")) {
                        Element Padre_Encont =
Nuestro.Element_Encont.getParentElement();
                        Element Padre_Salto =
Elemento_salto.getParentElement();
                        if (Padre_Encont == Padre_Salto) {
                            //El estado seleccionado se encuentra en
la misma rama que la transición
                            Elemento_salto.setAttribute("Condicional",
"si");

Nuestro.Insertar_Salto_Condicional(Elemento_salto,
Nuestro.Element_Encont);
                            MeterId_Salto_en_Estado();
                            select = false;
                        }
                    }
                }
                if (Simular) {
                    String Tipo_encont =
Nuestro.Element_Encont.getAttributeValue("Tipo");
                    int Activacion =
Integer.parseInt(Nuestro.Element_Encont.getAttributeValue("Activacion"
));
                    if ((Activacion == 1) &&
((Tipo_encont.contains("Transicion")) ||
(Tipo_encont.contains("Salto_condic")))) {
                        //Nuestro elemento de simulacion será el
devuelto por la busqueda
                        Element_simulacion = Nuestro.Element_Encont;
                        Simulacion();
                    }

                }
            }
        }
        invalidate();
        return super.onTouchEvent(event);
    }

    public void Repintar(){
        invalidate();
    }
}
```

10.3. CLASE “MAIN ACTIVITY”

Esta clase corresponde a la actividad principal (y única) de nuestra *app*. Se encarga de instanciar los menús, la *toolbar* y cualquier otro elemento de la parte gráfica. Además, es la parte del proyecto que se encargará de la comunicación entre el usuario y la parte gráfica.

```
package e.brian.ullsimgraf;

import android.content.Context;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.AppCompatActivity;
import android.support.design.widget.NavigationView;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.TextView;
import org.jdom2.*;


public class MainActivity extends AppCompatActivity implements
Dialog_Paralelos.Dialog_ParalelosListener,
        Dialog_StateOptions.Dialog_StateOptionsListener{

    public boolean Primera = true;
    public Element Element_Encont2 = null;
    public Draw draw;
    public boolean drawO = false;
    public boolean drawY = false;
    public float touch_x;
    public float touch_y;
    public String accion;
    public int c_state = 0;
    public String Count_States;
    public EditText insert_ramas;
    private Toolbar toolbar;
    private NavigationView navigation_View;
    private DrawerLayout drawerLayout;
    public Dialog_Paralelos dialogParalelos = new Dialog_Paralelos();
    public Dialog_errorNewEstado errorNewEstado = new
Dialog_errorNewEstado();

    public Documento_XML documento = new Documento_XML();

// Variables para definir a qué tipo de simulación se accede
    boolean b_interactive = false;
```

```
    boolean b_step = false;
    // Método que crea la Actividad
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
        setContentView(R.layout.activity_main);
        draw = findViewById(R.id.Draw);


        insert_ramas = (EditText)findViewById(R.id.file1);

// Definimos la toolbar y la establecemos como la barra de app de la
actividad.
        toolbar = (Toolbar)findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

// Definimos nuestro Drawer Layout.
        drawerLayout = (DrawerLayout)
findViewById(R.id.drawer_layout);
        navigation_View =
(NavigationView)findViewById(R.id.navigation_view);

// Eliminamos el color asociado de serie a los iconos del Navigation
View.
        navigation_View.setItemIconTintList(null);
        navigation_View.setNavigationItemSelectedListener(
                new NavigationView.OnNavigationItemSelectedListener(){
                    @Override
                    public boolean onNavigationItemSelected(MenuItem
menuItem){

                        switch(menuItem.getItemId()){

                            case R.id.añadir_Estado:
                                if(documento.Element_Encont !=null) {
                                    draw.addState();
                                }
                                else{

errorNewEstado.show(getFragmentManager(), "error");
                                }
                                menuItem.setChecked(true);
                                drawerLayout.closeDrawers();
                                break;

                            case R.id.añadir_O:
                                menuItem.setChecked(true);
                                drawO = true;
                                drawY = false;
                                if(documento.Element_Encont !=null) {

dialogParalelos.show(getFragmentManager(), "ramasO");
                                }
                                else{

errorNewEstado.show(getFragmentManager(), "error");
                                }
                                drawerLayout.closeDrawers();
```

```
                                    break;

                                case R.id.añadir_Y:
                                    menuItem.setChecked(true);
                                    drawY = true;
                                    drawO = false;
                                    if(documento.Element_Encont !=null) {

dialogParalelos.show(getFragmentManager(), "ramasO");
                                    }
                                    else{

errorNewEstado.show(getFragmentManager(), "error");
                                    }
                                    drawerLayout.closeDrawers();
                                    break;

                                case R.id.añadir_Rama:
                                    menuItem.setChecked(true);
                                    drawerLayout.closeDrawers();
                                    break;

                                case R.id.añadir_Salto:
                                    if(documento.Element_Encont !=null) {
                                        draw.addCondit();
                                    }
                                    else{

errorNewEstado.show(getFragmentManager(), "error");
                                    }
                                    menuItem.setChecked(true);
                                    drawerLayout.closeDrawers();
                                    break;

                            };

                            return true;
                        }
                    }
            );

        }

    /* Método que se ejecuta antes de destruir la Actividad, por lo que se
utiliza para definir qué
        queremos guardar para continuar utilizando cuando se vuelve a
crear la Actividad*/
        @Override
        public void onSaveInstanceState (Bundle savedInstanceState){

            super.onSaveInstanceState(savedInstanceState);


        }

        @Override
        public void onRestoreInstanceState(Bundle savedInstanceState){

            super.onRestoreInstanceState(savedInstanceState);

            c_state = savedInstanceState.getInt(Count_States);
```

```
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu_main){

        getMenuInflater().inflate(R.menu.menu_main, menu_main);
        return true;
    }

//    Este método se utiliza para definir la función de cada una de
las opciones del menú.
    @Override
    public boolean onOptionsItemSelected(MenuItem opcion_menu){

        int id = opcion_menu.getItemId();

        if(id == R.id.opcioneselementos) {
            if (documento.Element_Encont != null) {
                StateOptions();
            }
            else {
                errorNewEstado.show(getFragmentManager(), "error");
            }
        }

        if(id == R.id.abrir_proyecto) {

            return true;
        }

        if(id == R.id.nuevo_proyecto) {

            draw.newProject();

            return true;
        }

        if(id==R.id.guardar) {

            Dialog_Save dialogsave = new Dialog_Save();
            dialogsave.show(getFragmentManager(),"dialogSave");

            return true;
        }

        if(id == R.id.interactive){
            if (documento.Element_Encont != null) {
                b_interactive = true;
                b_step = false;
               draw.PlaySim();
            }
            else {
                errorNewEstado.show(getFragmentManager(), "error");
            }
            return true;
        }

        if(id == R.id.step){
```

```
            if (documento.Element_Encont != null) {
                b_step = true;
                b_interactive = false;
            }
            else {
                errorNewEstado.show(getFragmentManager(), "error");
            }
            return true;
        }

        if(id== R.id.ayuda){
            //                Invocamos un AlertDialog
            Dialog_Help help_dialog = new Dialog_Help();
            help_dialog.show(getFragmentManager(),"helpDialog");

        }

        if(id == R.id.play){

            if(b_interactive == true && b_step == false){

            }

            if(b_step == true && b_interactive == false){

            }

            if(b_step == false && b_interactive == false){

//                Invocamos un AlertDialog
                Dialog_SelectSimulation select_simulation = new
Dialog_SelectSimulation();

select_simulation.show(getFragmentManager(),"selectsimulation");
            }

            return true;
        }

        return super.onOptionsItemSelected(opcion_menu);
    }
    public void sendramas(String num_ramas) {
        int int_ramas = Integer.parseInt(num_ramas);
        if (drawO){
            draw.addOR(int_ramas);
            drawO = false;
        }
        if(drawY){
            draw.addY(int_ramas);
            drawY = false;
        }
    }

    public void sendetiquetas(String et1, String et2){

    }

    public void StateOptions(){
        Dialog_StateOptions StateOptions = new Dialog_StateOptions();
        StateOptions.show(getFragmentManager(),"opciones");
    }
```

```
// Método ejecutado al crear un nuevo proyecto
      /* String Tipo_Element = e.getAttributeValue("Activacion");
       int int_Element = Integer.parseInt(Tipo_Element);
        if(int_Element == 1){
            final DrawActive active = new
DrawActive(mContext,c_state);
            frameLayout.addView(active);
        }
        else {
            final DrawState state = new DrawState(mContext, c_state);
            frameLayout.addView(state);
        }
*/


   /* public void Repintar (Element Padre){

        List Elementos = Padre.getChildren();
        Iterator i = Elementos.iterator();
        while (i.hasNext()) {
            Element e = (Element) i.next();
            String Tipo_Element = e.getAttributeValue("Tipo");
            String Etiqueta = "";

            if(Tipo_Element.contains("Estado")){
                int Act =
Integer.parseInt(e.getAttributeValue("Activacion"));
                if(Act == 1){
                    DrawActive act_state = new DrawActive(mContext,
Integer.parseInt(e.getAttributeValue("Id")));
                    frameLayout.addView(act_state);
                }
            }

        }


    }*/

}
```

10.4. CLASE *"DIALOG PARALELOS"*

Incluimos esta clase como ejemplo de la programación de *Alert Dialogs*

```
package e.brian.ullsimgraf;

import android.app.Dialog;
import android.app.DialogFragment;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.EditText;

public class Dialog_Paralelos extends DialogFragment {

    private EditText numero_ramas;
    private Dialog_ParalelosListener listener;
    public Dialog onCreateDialog(Bundle savedInstanceState){
        AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
        LayoutInflater inflater = getActivity().getLayoutInflater();
        View view = inflater.inflate(R.layout.dialog_paralelos,null);
        builder.setView(view)
                .setPositiveButton(R.string.ok, new
DialogInterface.OnClickListener(){
                    public void onClick(DialogInterface dialog, int
id){
                        String num_ramas =
numero_ramas.getText().toString();
                        listener.sendramas(num_ramas);
                    }

        });
        numero_ramas = view.findViewById(R.id.file1);
        return builder.create();
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try {
            listener = (Dialog_ParalelosListener)context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString() +
"Implementar Listener");
        }
    }

    public interface Dialog_ParalelosListener{
        public void sendramas(String num_ramas);
    }

}
```

10.5. DOCUMENTO *"activity_main"*

Por último, añadimos este documento XML, donde se define la interfaz de la actividad *main* y todos los elementos que la contienen.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:id="@+id/drawer_layout"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/linear_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <include
            android:id="@+id/toolbar"
            layout="@layout/toolbar">

        </include>

        <e.brian.ullsimgraf.Draw
            android:id="@+id/Draw"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@drawable/cuadricula"
            android:focusable="true" />


    </LinearLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/navigation_view"
        android:layout_width="250dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        app:menu="@menu/drawer"
        app:headerLayout="@layout/header"
        android:background="@color/white"
        android:theme="@style/navigation_theme"
        >

    </android.support.design.widget.NavigationView>



</android.support.v4.widget.DrawerLayout>
```

11. BIBLIOGRAFÍA

[1]M. Brei, T., (octubre 2013). What is industrial automation? Recuperado de: https://www.surecontrols.com/what-is-industrial-automation/

[2] Sy Corvo, H. Automatización industrial: historia, características y tipos. Recuperado de: https://www.lifeder.com/automatizacion-industrial/#Historia

[3] Autor desconocido. El PLC. Recuperado de: http://www.sc.ehu.es/sbweb/webcentro/automatica/WebCQMH1/PAGINA%20PRINCIPAL/PLC/plc.htm

[4] Estévez Corona, Adrián y Royo Hardisson José. (junio 2010). *Proyecto de diseño e implementación de un simulador de GRAFCET para autómatas programables PLC's*

[5] Autor desconocido. ¿Qué es la Industria 4.0? Recuperado de: https://www2.deloitte.com/es/es/pages/manufacturing/articles/que-es-la-industria-4.0.html

[6] Boisset, F., (mayo 2018). The History of Industrial Automation in Manufacturing. Recuperado de: https://kingstar.com/the-history-of-industrial-automation-in-manufacturing/

[7] Autor desconocido. Introducción al modelado GRAFCET. Recuperado de: http://www.elai.upm.es/moodle/pluginfile.php/1171/mod_resource/content/0/GrafcetAmpliacion.pdf

[8] Autor desconocido. Sistemas operativos. Recuperado de: https://www.areatecnologia.com/sistemas-operativos.htm

[9] Autor desconocido. (diciembre 2012). Introducción, ¿Qué es Android? Recuperado de: https://histinf.blogs.upv.es/2012/12/14/android/

[10] Autor desconocido. Página oficial de Android. Recuperado de: https://www.android.com/intl/es_es/

[11] Aguilar, Ricardo, (septiembre 2019). Android 10 ya está aquí: todas las novedades, móviles compatibles y cómo actualizar a la nueva versión de Android. Recuperado de:

https://www.xatakandroid.com/sistema-operativo/android-10-esta-aqui-todas-novedades-moviles-compatibles-como-actualizar-a-nueva-version-android

[12] Autor desconocido. ¿Qué es el software libre? Recuperado de: https://www.gnu.org/philosophy/free-sw.es.html

[13] Joshi, B., (2017). *Beginning XML with C#7*, India

[14] Autor desconocido. Extensible Markup Language (XML). Recuperado de: https://www.w3.org/TR/1998/REC-xml-19980210

[15] Autor desconocido. Página oficial para desarrolladores de Android. Recuperado de: https://developer.android.com/

[16] Autor desconocido. Tratamiento de XML en Android: Introducción. Recuperado de: https://academiaandroid.com/tratamiento-de-xml-en-android-introduccion/