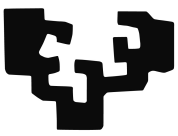
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

Ingeniería superior en informática

Arquitectura y Tecnología de Computadores

Proyecto de Fin de Carrera

Análisis del desarrollo de aplicaciones accesibles sobre el sistema operativo Android. Implementación de un sistema de barrido.

Nombre

Sergio Sáenz Romero

2013

Índice de contenido

1. Introducción

El presente proyecto se desarrolla tomando como referencia la aplicación *PiktoPlus*, desarrollada por la empresa *Límbika Assistive Technologies SL* para el sistema operativo Android. Esta empresa se dedica al desarrollo de tecnología y productos de apoyo para personas con diversidad funcional.

PiktopPus es un Comunicador Aumentativo y Alternativo para personas con dificultades en la emisión, comprensión o procesamiento del lenguaje oral. Está basado en un sistema de pictogramas 3D.

El ciclo de vida del desarrollo de *Piktoplus* priorizó la implementación de la funcionalidad básica de la aplicación. Quedando en segundo plano la universalidad en el uso de la aplicación.

Al estar dirigido tanto a niños como a adultos, especialmente a personas con autismo, parálisis cerebral, daño cerebral adquirido (DCA), síndrome de down y cualesquiera con dificultades en la comunicación oral, hizo necesario que *PiktoPlus* cupliese con los estándares de accesibilidad.

Desde la empresa Limbika, se planteó la adaptación del software para su compatibilidad con los servicios de accesibilidad y ayudas técnicas que el Sistema Operativo Android facilita.

Android provee diversos servicios de accesibilidad y ayudas técnicas para hacer de un dispositivo móvil una herramienta accesible para todos aquellos usuarios con diversidad funcional. No obstante, el buen funcionamiento de éstos sobre cualquier aplicación está sujeto a las buenas prácticas del desarrollador, a su formación en accesibilidad y a la versión de Android sobre la que se programe. Es necesario conocer las herramientas y servicios que el sistema operativo Android ofrece para la implementación de aplicaciones accesibles.

En este proyecto se ha analizado cómo el sistema operativo ha evolucionado en el ámbito de la accesibilidad a lo largo de sus diferentes versiones para ofrecer una guía, a modo de resumen, de los aspectos necesarios a tener en cuenta para el desarrollo y adaptación de aplicaciones accesibles. Finalmente, se propone la librería *AccessibilityHUD* (*Accessibility Head Up Display*) que proporciona un sistema de barrido como posible solución a las carencias de accesibilidad del sistema en sus versiones más tempranas.

El capítulo 2 está orientado a la descripción de objetivos del proyecto. En el capítulo 3 se expondrán algunas definiciones básicas para situar el proyecto dentro de las áreas de trabajo científicas con las que está relacionado. En el capítulo 4, se realiza una pequeña introducción al sistema operativo Android y su evolución. Se procede, en el capítulo 5, a indagar más específicamente sobre la SDK de Android y las herramientas que propone para la implementación de interfaces accesibles. En el capítulo 6 se aborda el desarrollo de la librería *AccessibilityHUD*,

que proporciona un sistema de barrido diseñado para funcionar como un servicio independiente de las aplicaciones en ejecución. En el capítulo 7 se hace un repaso del aquellos aspectos no abarcados en este proyecto, el trabajo futuro. Finalmente, el capítulo 8 muestra las conclusiones. tras estos apartados se ofrece una bibliografía. En los anexos se encuentra una guía para la implementación de aplicaciones accesibles sobre el sistema operativo Android y un manual de usuario para guiar al desarrollador en el uso de la librería AccessibilityHUD.

2. Descripción de objetivos del proyecto

2.1. Descripción del proyecto

El desarrollo del proyecto se divide en dos fases principales. En la primera se realiza un estudio del *framework* de desarrollo que Google ofrece para el desarrollo de aplicaciones sobre Android, profundizando especialmente en las herramientas dedicadas a la implementación de aplicaciones accesibles. Para ello se han analizado las diferentes herramientas orientadas a la implementación de aplicaciones así como las ayudas técnicas propias de Android para poder entender qué aspectos de la accesibilidad quedan cubiertos por el sistema de forma transparente al desarrollador.

En una segunda fase del proyecto se desarrolla una librería para la inclusión de un sistema de barrido en cualquier aplicación a modo de servicio de accesibilidad.

2.2. Objetivos del proyecto

El objetivo principal de este proyecto es facilitar a aquellos usuarios con diversidades motrices el uso de sus dispositivos móviles con sistema Android a través de un barrido sobre la pantalla. Este barrido se realiza con dos ejes cartesianos que el usuario controla mediante cualquier periférico adaptado que posea y le permite decidir las coordenadas de la pantalla sobre las que desea interactuar.

2.3. Alcance

El alcance inicial del proyecto, donde se delimita el trabajo que debe realizarse para una correcta entrega de la herramienta, se describe en los siguientes puntos:

- Análisis del sistema operativo Android en lo relativo a la implementación de aplicaciones accesibles y estudio de las ayudas técnicas que el sistema ofrece.
- Análisis y diseño de un sistema de barrido que permita al usuario interactuar con cualquier dispositivo Android. Adaptación del desarrollo como una librería que pueda ser importada en cualquier proyecto de aplicaciones Android.
- Elaboración de una guía de desarrollo para el correcto uso de la librería implementada y su fácil integración en cualquier aplicación Android.
- Escritura de la memoria del proyecto y defenderla.

2.4. Método de trabajo

Se ha abordado un método de trabajo de un proyecto de Ingeniería dividiendo el proyecto en distintas fases. Las fases que se han completado son la fase de gestión, donde se manejan los objetivos, planificación y seguimiento del desarrollo del proyecto, la fase de formación donde se observan las distintas herramientas que el proyectando debe conocer, la fase de análisis y diseño donde se contemplan las necesidades del sistema de barrido y cómo lograr que dicho sistema cumpla con los requisitos previos, la fase de implementación y pruebas del sistema, llamada fase "Aplicación" y finalmente la fase de cierre que trata sobre la creación de la memoria del proyecto y el manual del desarrollador para la librería del barrido.

A continuación se enumeran las tareas asociadas con cada fase del proyecto.

- **Gestión**
  - Objetivos: Relación de los objetivos a cumplir en este proyecto. Captura de requisitos y entendimiento general del proyecto.
  - Planificación: Lo relacionado con este capítulo.
  - Seguimiento: Control del desarrollo del proyecto para gestionar las posibles desviaciones y replanificación.
- **Formación**
  - Sistemas de barrido. Definiciones básicas y conocimiento general sobre las ayudas técnicas existentes para la accesibilidad de personas con diversidades motrices.
  - El *framework* de desarrollo de Android. Principales características y requerimientos para el desarrollo de aplicaciones sobre Android.
  - Herramientas y ayudas técnicas para la accesibilidad del sistema operativo Android.
- **Análisis y diseño**
  - Diseño de la interfaz gráfica del sistema de barrido. Los ejes cartesianos del barrido y su comportamiento.
  - Diseño del servicio según la especificación de Android.
  - Diseño de la conexión con el servicio siguiendo la especificación de la interfaz IBinder.
  - Diseño de la estrategia para la captura y generación de eventos.
- Aplicación
  - Implementación de la librería AccessibilityHUD
  - Pruebas de la librería AccessibilityHUD
- **Cierre**
  - Elaboración de la memoria.
  - Elaboración de un manual para desarrollo.

2.5. Gestión y planificación

2.5.1. Recursos

Se ha realizado una estimación de los recursos necesarios para desarrollar este proyecto, diferenciando entre recursos materiales y recursos humanos.

- **Recursos materiales**
  - Línea ADSL 10 Mb para documentación y consultas técnicas necesarias en el desarrollo del proyecto.
  - Ordenador *Sony Vaio* con 8 GB de memoria RAM y 500 GB de disco duro. En este ordenador se ha instalado el siguiente software para el desarrollo del proyecto:
  - Ubuntu 11.10. Sistema operativo gratuito y de libre distribución.
  - Eclipse 3.7 Indigo, entorno de desarrollo integrado de código abierto multiplataforma. Es la herramienta principal utilizada para la implementación del proyecto
  - SDK de Android. Plataforma de desarrollo de Google para el sistema operativo Android.
  - Android ADT para Eclipse. *Plugin* para el entorno de desarrollo Eclipse que integra las librerías de desarrollo del SDK (*Sofware Development Kit*) de Android Además de una máquina virtual de Android para la ejecución de aplicaciones y las herramientas de desarrollo de Android, entre las que se encuentran herramientas para compilar proyectos o para la comunicación con dispositivos, y diversos proyectos de ejemplo de aplicaciones Android.
- **Recursos humanos**
  - Desarrollador del proyecto, alumno de la Ingeniería Superior en Informática que cumpliendo los requisitos y normativa de la Facultad de Informática de la UPV/EHU para la finalización de la carrera, desarrolla el presente proyecto. Su función será la de elaborar el proyecto en todas sus fases; captura de requisitos, análisis, diseño, implementación, pruebas, integración y documentación. Para conseguir estos objetivos deberá adquirir los conocimientos necesarios previa documentación y estudio, siendo necesaria su actualización en todas las fases del proyecto.
  - Tutor del proyecto, persona perteneciente a la facultad del desarrollador, que coordina al mismo durante el desarrollo del proyecto haciendo uso de sus conocimientos y experiencia para que el proyecto finalice de forma satisfactoria.
  - Tutor de la empresa, persona perteneciente a la empresa Limbika Tecnologías Asistivas, que revisa y valida el trabajo realizado por el proyectando.

2.5.2. Entregables

- Librería *AccessibilityHUD* que implementa un servicio de accesibilidad para el sistema operativo Android realizando un barrido sobre la pantalla del dispositivo mediante el dibujado de dos ejes cartesianos controlados por un periférico que permite generar diversos eventos sobre las coordenadas de intersección de ambos ejes.

- Memoria, donde se muestra la información derivada del proyecto y el ciclo de vida de su desarrollo. Se refiere a este documento.

- Guía de uso de la librería *AccessibilityHUD* en aplicaciones Android (capítulo 8 del presente documento). En ella se orienta al desarrollador sobre la incorporación a sus desarrollos de esta librería para la ejecución del servicio de barrido sobre las interfaces de su aplicación.

2.5.3. Fases y tareas

Los puntos descritos en el alcance se desarrollarán en orden secuencial dependiendo de las características de cada uno, se le asignará un ciclo de vida diferente para adaptarlo a las particularidades de cada uno de ellos. También se prevé con la implementación de cada entregable, la modificación de los entregables desarrollados con anterioridad para mejorar el conjunto del proyecto y evitar posibles fallos descubiertos durante el desarrollo del proyecto o posibles carencias anteriormente no detectadas.

- Ciclo de vida del análisis del sistema operativo Android.
  - Estudio de las herramientas para el desarrollo de aplicaciones.
  - Estudio del *framework* de Android para la implementación de servicios de accesibilidad.
  - Análisis de la evolución del sistema Android en lo relativo a la accesibilidad.
  - Análisis de las ayudas técnicas existentes en el sistema Android.
- Ciclo de vida del desarrollo del sistema de barrido.
  - Análisis de los requisitos del sistema de barrido y posibles dificultades inherentes a la plataforma de desarrollo.
  - Diseño del sistema de barrido.
  - Implementación del sistema de barrido.
  - Pruebas del sistema de barrido.
- Ciclo de vida de la asistencia al desarrollador.
  - Desarrollo del *JavaDoc* de la librería *AccessibilityHUD*.
  - Diseño de la guía de uso de la librería *AccessibilityHUD*.
  - Desarrollo de la guía de uso de la librería *AccessibilityHUD*.
- Ciclo de vida de la memoria y defensa del proyecto.
  - Diseño de la memoria del proyecto.
  - Diseño de la defensa del proyecto.

- Desarrollo de la memoria, elaboración de la misma.
- Elaboración de la presentación del proyecto ante el tribunal.
- Entrega de la memoria del proyecto.
- Defensa del proyecto.

2.5.4. Estructura de desglose de trabajo

Tomando las fases indicadas en el anterior apartado, se obtiene la estructura de la *figura 1*, donde se observan las distintas tareas que se deben desarrollar para lograr el resultado esperado.

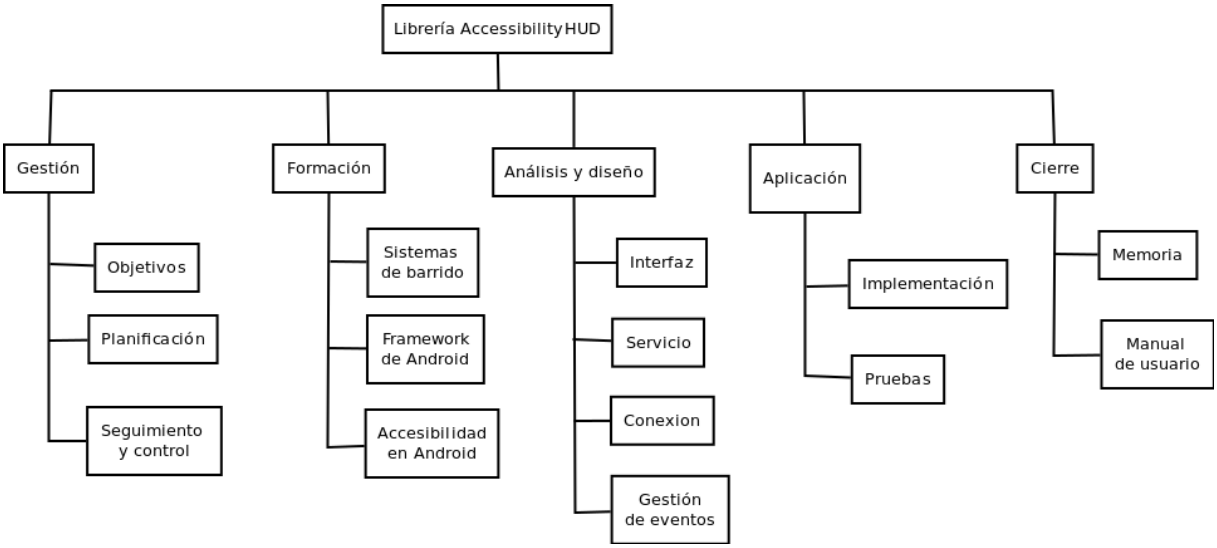


Figura 1: EDT

2.5.5. Estimación de horas por tarea

Este apartado muestra las horas estimadas que se asignan a cada tarea. Dado que el proyecto es un proyecto de la carrera de Ingeniería Superior equiparada a 15 créditos, resultan un total de 450 horas de trabajo. Sin embargo, dada la complejidad del proyecto se llegó a un acuerdo entre el proyectando, la empresa y el tutor para poder incrementar el número de horas. Esto explica el por qué el número de horas inicialmente planificadas era superior a 470 horas.

Las tablas que se presentan a continuación muestran el reparto de horas que se han estimado para el desarrollo del proyecto:

Tarea	**Horas**
Análisis de objetivos	20
Planificación	20
Seguimiento y control	40
Total:	80

Tabla 1: Estimación de tiempo para la gestión del proyecto.

Tarea	**Horas**
Sistema de barrido	40
Framework de Android	60
Accesibilidad en Android	60
Total:	160

Tabla 2: Estimación de tiempo para la fase de formación.

Tarea	**Horas**
Interfaz	40
Servicio	30
Conexión	20
Gestión de eventos	50
Total:	140

Tabla 3: Estimación de tiempo para la fase de análisis y diseño.

Tarea	**Horas**
Implementación	10
Pruebas	10
Total:	20

Tabla 4: Estimación de tiempo para el desarrollo de la aplicación.

Tarea	Horas
Memoria	40
Manual de desarrollador	10
Defensa del proyecto	20
Total:	70

Tabla 5: Estimación de tiempo para el cierre del proyecto.

- La tabla 1 expone el desglose de horas estimadas para los aspectos relativos a la gestión del proyecto.
- La tabla 2 muestra las horas estimadas para la formación del proyectando
- La tabla 3 muestra las horas estimadas para la fase de análisis y diseño.
- La tabla 4 expone la estimación realizada para el desarrollo de la librería.
- La tabla 5 muestra las horas estimadas para la fase de cierre del proyecto.

2.5.6. Calendario

Al tratarse de un proyecto realizado en el marco de la empresa Limbika, el calendario laboral fue impuesto por el horario laboral de la empresa. Este calendario contempla trabajar cuatro horas diarias los días laborales, teniendo en cuenta excepciones para los días festivos y por motivos personales.

La fecha de inicio del proyecto fue el 1 de Octubre de 2012 y la finalización del mismo fue el día 10 de Julio de 2013. La diferencia entre fechas no es representativa del coste temporal del proyecto ya que es debida al retraso, por parte del proyectando, en la elaboración de esta memoria. Las partes relevantes para la empresa se concluyeron el 21 de Diciembre, faltando tan solo la elaboración de la documentación que se paralizó debido a requerimientos laborales de la empresa hacia el proyectando. No obstante, a lo largo del desarrollo del sistema, se fue recopilando información, sobre todo en la fase de formación, para poder elaborar posteriormente la memoria, además, la redacción de la guía de uso fue realizada en el momento de la programación a través de *JavaDoc*, herramienta para documentación de código propia del lenguaje Java. Por este motivo no se computan las horas relativas a la recopilación de documentación final y redacción del presente documento.

En la *figura 2* se muestra un diagrama de *Gantt* con la estimación previa realizada para las tareas definidas en el apartado anterior.

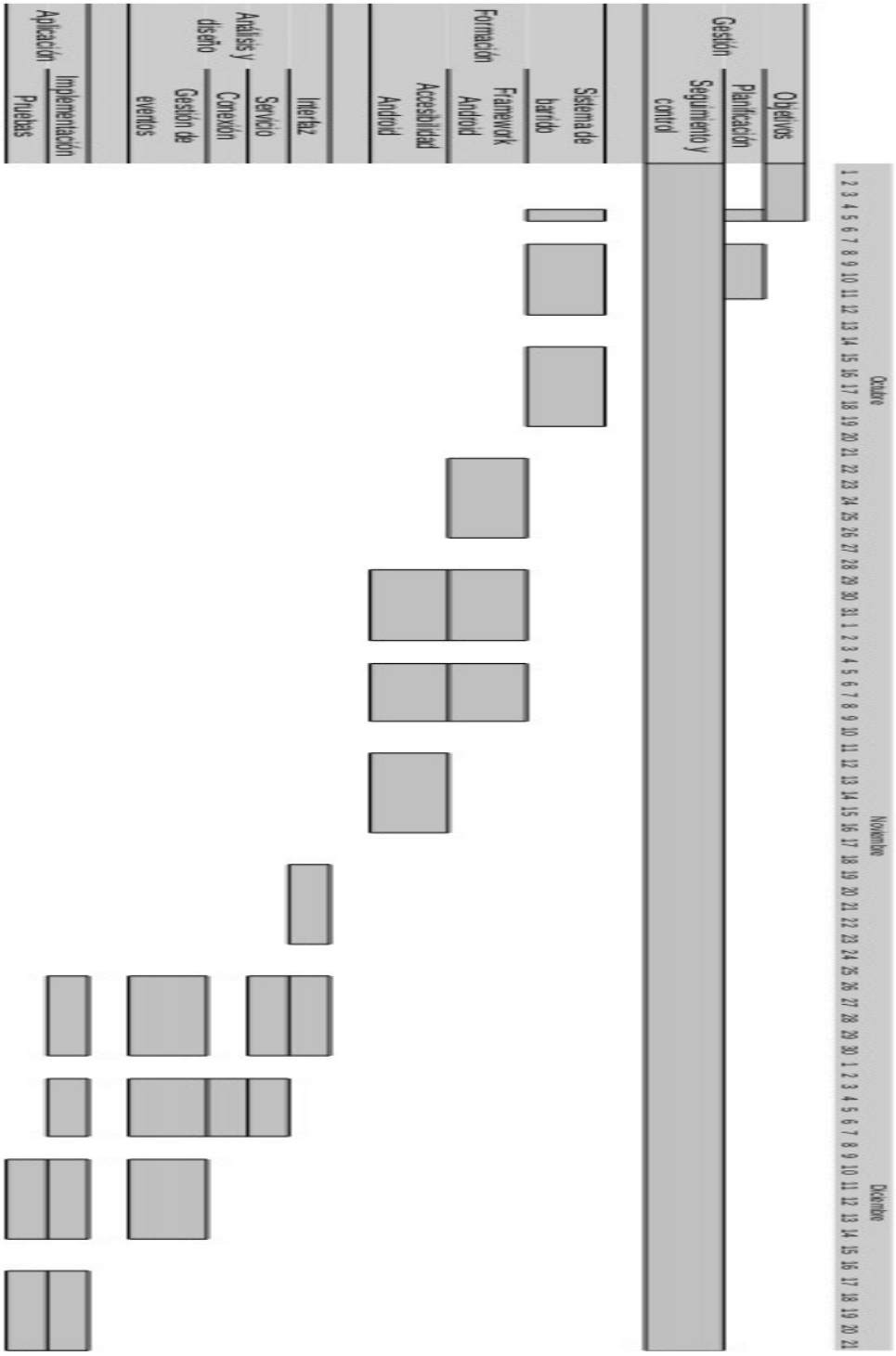


Figura 2: Planificación inicial.

2.5.7. Resultado obtenido

En este apartado se compara el tiempo planificado que se puede observar en las tablas del apartado 2.5.5 con el tiempo consumido realmente. La planificación inicial contemplaba el solapamiento de las fases de formación referidas al *framework* y al estudio sobre la accesibilidad en Android. Sin embargo, la realidad demostró que no fue posible y la fase de formación se prolongó durante casi todo el mes de Noviembre.

Comenzando el desarrollo del proyecto al principio del mes de Octubre de 2012, durante el mes de Noviembre 2012 se replanificó el proyecto tras comprobar que el tiempo necesario para la formación del proyectando en las técnicas necesarias para la implementación de un servicio de accesibilidad era bastante superior al inicialmente estimado.

La situación en el momento de la replanificación indicó un desfase con respecto a lo planeado, debido al retraso en el análisis y el diseño que debían estar realizados. Este retraso fue debido fundamentalmente a lo relatado en el anterior párrafo, no obstante, desde la empresa Límbika y debido a estrategias de márqueting, se requirió la finalización del proyecto en fechas para el lanzamiento del producto en la temporada estival. En la replanificación no se pudieron mover los hitos de entrega por lo que fue la jornada laboral del proyectando sufrió cambios. Se computaron más horas diarias hasta duplicar la jornada laboral, lo que supuso un aumento de la productividad que pudo traducirse en la entrega de la implementación a tiempo para navidad, tal y como requirió la empresa. En la *figura 3* se muestra un diagrama de *Gantt* con la replanificación realizada.

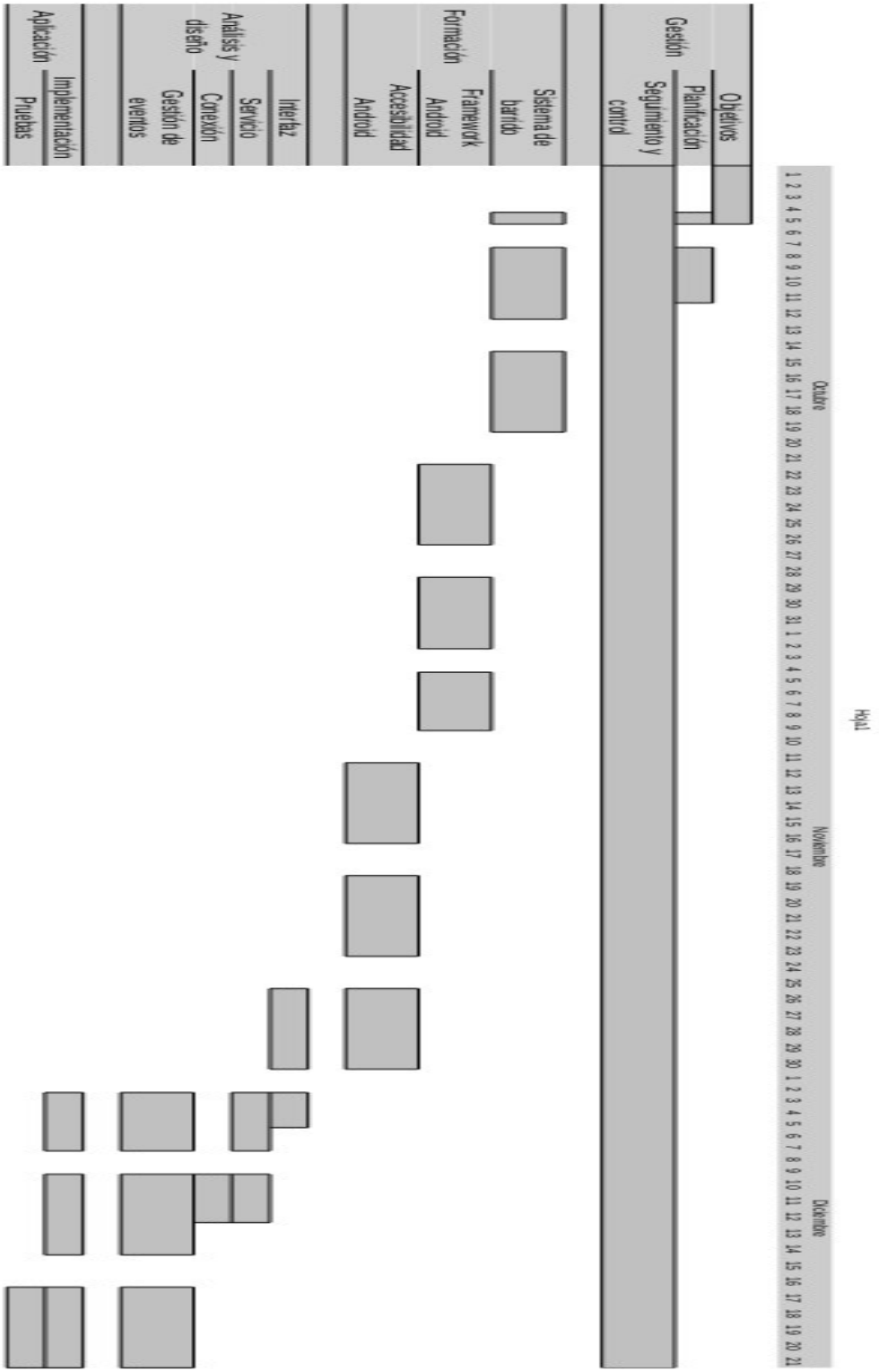


Figura 3: replanificación.

Considerando los datos recogidos durante el seguimiento del proyecto y los cambios que ha supuesto la replanificación del mismo, se puede deducir el porcentaje de tiempo dedicado a cada uno de los aspectos más relevantes de la creación del sistema de barrido. En la *figura 4* se destacan los porcentajes de tiempo dedicados a la gestión del proyecto, la formación en las herramientas necesarias, al análisis y diseño del sistema, a la impementación de la aplicación y al cierre del proyecto.

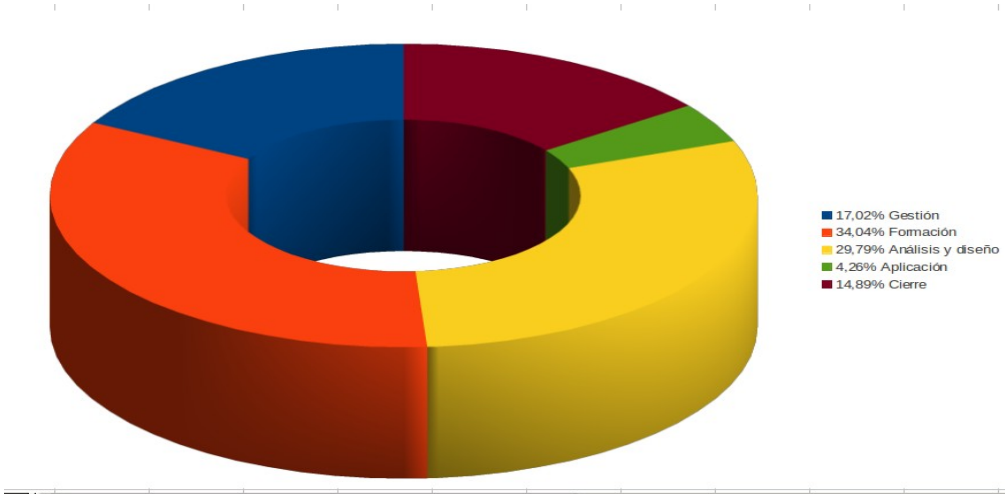


Figura 4: Porcentaje de tiempo empleado.

2.5.8. Concordancia entre planificación y resultados

Analizando los datos anteriormente descritos, se puede observar que la mayor cantidad de tiempo se ha invertido en aspectos relativos al estudio de las herramientas necesarias para la implementación del sistema. Concretamente en la formación relativa al *framework* de Android y las herramientas para la implementación de aplicaciones accesibles. Para la recopilación de datos, estudio y aprendizaje de estas herramientas se ha encontrado el inconveniente de la falta de información sobre los mecanismos necesarios para adaptación de aplicaciones desarrolladas sobre versiones tempranas de la plataforma. Esto ha hecho que haya sido necesario un esfuerzo no previsto que ha aplazado el inicio del desarrollo.

Otro aspecto a destacar, es el uso tan peculiar que se hace de la gestión de la interfaz del barrido. Android no proporciona mecanismos para la gestión de una interfaz visual por parte de un servicio. Además de la gran potencia requerida por el sistema en la gestión de eventos para su compatibilidad con la mayor cantidad posible de periféricos. Estos aspectos han hecho que la fase de análisis y diseño ocupara gran parte del tiempo.

2.6. Riesgos

Riesgo: Hacer una estimación errónea de la planificación temporal.
Plan de contingencia: Aumentar las horas de dedicación diarias asignadas.

Riesgo: Pérdida de información. Mala gestión de las copias de seguridad y del archivo de información.
Plan de contingencia: Uso de un sistema de versiones tanto para el código como para la documentación para poder recuperar la información.

Riesgo: No finalizar la aplicación en el plazo acordado con lo que la empresa pierde la oportunidad de ventas para la fechas acordadas además de la consiguiente desmotivación del autor y del equipo que le apoya.

Riesgo: Falta de comunicación y descoordinación entre el autor del proyecto y el equipo que dirige el proyecto.
Plan de contingencia: Establecer reuniones periódicas para la coordinación entre los interesados así como un plan de envío de notificaciones urgentes para posibles situaciones inesperadas.

2.7. Oportunidades

- Aprendizaje de la arquitectura del sistema operativo Android y de las herramientas que ofrece para la implementación de aplicaciones accesibles.
- Aprendizaje de las ayudas técnicas propias del sistema operativo Android.
- Aprendizaje de los requerimientos funcionales y de interfaz que personas con diversidad funcional demandan en sus dispositivos.

2.8. Personas y entidades interesadas en el proyecto

En este apartado se identifican las personas o colectivos a las que puede afectar de algún modo la elaboración de este proyecto:

- Desarrollador del proyecto.
- Tutor del proyecto.
- Empresa Limbika Tecnologías Asistivas.
- Desarrollador de aplicaciones Android.

2.9. Expectativas de los interesados

- Desarrollador del proyecto
  - Finalizar el proyecto en el plazo establecido para poder defenderlo y terminar su trayectoria universitaria.
  - Realizar un producto consistente, robusto y de calidad.
  - Ofrecer un producto que sea del agrado del resto de implicados.
  - Ofrecer una imagen de implicación y profesionalidad.
  - Adquirir experiencia en el desarrollo de aplicaciones Android.
- Tutor
  - Guiar al alumno en el desarrollo de un proyecto de software, estimulándolo para organizarse y superar por sí mismo los obstáculos que pueda encontrar durante el progreso del proyecto.
  - Orientar al alumno para el buen fin de su trabajo.
  - Aumentar el nivel de conocimiento del alumno sobre el desarrollo de un producto
  - Aumentar el nivel de conocimiento del alumno sobre el desarrollo de un producto software.
  - Conseguir que el trabajo de tutorización no suponga una carga extra de trabajo que dificulte el normal desarrollo de sus tareas habituales.
- Empresa Límbika Tecnologías Asistivas
  - Elaboración de un producto de calidad.
  - Aumento de las ventas de sus productos gracias al valor añadido por este proyecto.
  - Colaboración con otras empresas para la integración de este proyecto en sus productos.
- Desarrollador de aplicaciones Android
  - Obtener un servicio que permita aumentar la accesibilidad de sus aplicaciones.
  - Disponer de una librería sencilla de integrar en sus aplicaciones.

2.10. Costes

En este apartado se desarrolla el plan de costes del proyecto. Se ha realizado una estimación de costes según el juicio del autor, valorando a este como programador junior, cobrando 15.000€/año brutos, lo que supone 21€/hora. Al tutor de proyecto se le ha estimado una tarifa de 50€/hora. El sueldo del empleado de la empresa Límbika encargado de dirigir al proyectando se ha estimado en 50€/hora.

El software utilizado para el desarrollo del sistema de barrido, así como para la documentación, es de licencia gratuita, por lo que el coste asociado es nulo.

El IDE Eclipse y los *plugins* proporcionados por la empresa Google para el desarrollo y depuración de aplicaciones, son herramientas de código abierto y su uso es libre, lo que no supone un aumento del coste del proyecto.

Así pues, el coste total del proyecto se limita al asociado a los recursos humanos implicados en el proyecto.

Recurso	Coste	Unidades	Coste Total
Desarrollador	21 €/hora	480 h	10080
Tutor	50 €/hora	30 h	1500
Responsable de Limbika	50 €/hora	50 h	2500
Línea ADSL 10 Mb	25 €/mes	4 meses	100
Ordenador Sony Vaio	900 €/unidad	1 unidad	900
Ubuntu 11.10	0 €	1 unidad	0 €
Eclipse 3.7 Indigo	0 €	1 unidad	0 €
Android SDK	0 €	1 unidad	0 €
Android ADT	0 €	1 unidad	0 €
		TOTAL:	15080€

Tabla 6: Resumen de costes del proyecto.

2.11. Análisis de factibilidad

Tras establecer los objetivos del proyecto, puede realizarse una valoración global sobre las posibilidades de realizar el trabajo asignado de forma satisfactoria.

Valorando inicialmente los componentes tecnológicos necesarios para la correcta implementación de la librería AccessibilityHUD se hace un análisis de las tecnologías requeridas que se han presentado en el apartado 2.5.1 sobre las tecnologías utilizadas y se llega a la conclusión de que estas cubren las necesidades para poder desarrollar una aplicación estable, cubriendo el alcance fijado.

Al estudiar los componentes tecnológicos seleccionados finalmente y otros que posteriormente se han descartado, se ha podido comprobar que existen varios de ellos que permiten desarrollar la aplicación que se pretende conseguir e incluso ofrecen funcionalidades adicionales a las necesarias. Por ello, se concluye que, tras conocer el trabajo a desarrollar y habiendo analizado y escogido los componentes tecnológicos necesarios para realizar dicho trabajo, el presente proyecto es factible.

3. Definiciones básicas

En este capítulo se procede a definir varios conceptos que serán de utilidad a lo largo de la memoria, así como a situar este proyecto en su contexto dentro del ámbito científico-técnico.

Se denomina **dispositivo móvil** a aquellos aparatos de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para realizar una función determinada, pero que pueden llevar a cabo otras funciones más generales [1].

Dentro de los dispositivos móviles, el tipo más extendido es el teléfono inteligente (*smartphone* en inglés). Un teléfono inteligente es un teléfono móvil construido sobre una plataforma informática móvil, con una mayor capacidad de almacenar datos y realizar actividades semejantes a una mini-computadora y conectividad que un teléfono móvil convencional. El término “inteligente“ hace referencia a la capacidad de usarse como un ordenador de bolsillo, llegando incluso a poder reemplazar, en algunos casos, a un ordenador personal [2].

Llamamos **comunicador** al conjunto máquina-programa que, de una manera u otra, permite la comunicación con otras personas. Esta comunicación se entiende como la generación de una serie de mensajes que puedan ser recibidos por los demás sin dificultad. Para ello se vale, fundamentalmente, de la escritura de frases en la pantalla del comunicador o del ordenador, que posteriormente podrán ser escritas a través de una impresora o leídas en voz alta por un sintetizador [3].

Sistemas de comunicación alternativa. Son aquellas formas de comunicación distinta del habla empleadas por una persona en contextos de comunicación “cara a cara”. La comunicación alternativa incluye todos los métodos de comunicación utilizados por las personas que, por diferentes causas (discapacidad, enfermedad, envejecimiento, accidente, etc.), carecen de lenguaje oral. Existe un amplio abanico de sistemas de comunicación alternativa que van desde el lenguaje de signos a los lenguajes pictográficos, pasando por las ayudas a la producción de texto escrito.

Los sistemas de comunicación alternativa basados en dispositivos móviles centran sus objetivos principalmente en la interfaz de usuario y en el software de comunicación. La interfaz de usuario debe estar definida de modo que sea accesible para la persona que utiliza el sistema. Dado que el abanico de características de los posibles usuarios es muy amplio, no existe un única interfaz que satisfaga todas las necesidades. Por ello, es necesario utilizar técnicas que permitan adaptarla a sus diferentes características físicas y cognitivas.

Sistemas de comunicación aumentativa. Están orientados a personas que carecen de comunicación oral pero que pueden aprender a hablar o expresarse, o aquellas que mantienen parte de su capacidad de usar el lenguaje oral pero su inteligibilidad es baja. Los sistemas de

comunicación aumentativa tienen el doble objetivo de promover y apoyar el habla y garantizar una forma de comunicación alternativa.

3.1. Métodos de acceso

La **interfaz de control** es el hardware mediante el cual el usuario controla u opera un dispositivo. El **conjunto de entrada** o dominio de entrada es el rango de valores que puede tomar una variable que representa a la entrada, y puede ser discreto o continuo. Si la entrada es discreta, ésta toma un número finito de valores fijos sin pasos intermedios entre ellos, como por ejemplo un teclado. En cambio, la entrada continua toma un rango de valores virtualmente infinito (por ejemplo, las posiciones del ratón).

Se denomina **conjunto de selección** al grupo de los ítems disponibles entre los que el usuario escoge. Puede contener símbolos ortográficos (letras, palabras, frases) o símbolos que representan ideas o ideogramas (iconos o imágenes). El tamaño del conjunto de selección, la modalidad y el tipo de selección dependen de las necesidades y capacidades del usuario y de la actividad de salida a la que se destinan: Comunicación, movilidad, control del entorno, etc.

El **método de selección** es el procedimiento usado para escoger un elemento del conjunto de selección, y normalmente supone una acción sobre la interfaz de control. Según Vanderheinden, los métodos de selección se pueden clasificar en tres categorías: Selección directa, barrido y codificación [4].

La **selección directa** permite escoger de manera aleatoria cualquier ítem del conjunto de selección. La forma más habitual es la selección espacial realizada sobre los teclados estándar. En este caso, cada opción se selecciona pulsando la tecla asignada a ella, que se encuentra situada en una posición concreta en el espacio. El tiempo necesario para ello es independiente del ítem que se seleccione pero depende de la destreza del usuario. Desde el punto de vista cognitivo, la selección directa es muy intuitiva ya que la salida generada es el resultado directo de la acción de seleccionar. Sin embargo, exige que la persona disponga de la movilidad, fuerza y precisión necesarias para poder pulsar cualquier tecla.

El **barrido** es un método de selección indirecta que permite al usuario escoger mediante una acción voluntaria y detectable una opción de las que se le presentan secuencialmente. La acción para escoger depende de las características físicas del usuario. Por ejemplo, puede activar un sensor mediante una parte del cuerpo sobre la que disponga de control (mano, pie, soplo, párpado...). El barrido también puede ser considerado como un teclado temporal, ya que una sola tecla va tomando diferentes valores en el tiempo y, dependiendo del momento de la pulsación, ofrece diferentes resultados. Este método requiere poca movilidad, precisión y fuerza, ya que se pueden diseñar pulsadores del tamaño y sensibilidad adecuados para la capacidad de una persona concreta. Sin embargo, sí requiere capacidad visual, atención y coordinación temporal, al tener que seleccionar el carácter adecuado en el momento que se le ofrece. No obstante, el

periodo de tiempo durante el cual un carácter es seleccionable puede ajustarse al usuario. Desde el punto de vista cognitivo, resulta más complejo que el caso anterior, ya que la asociación entre la acción de seleccionar y el resultado no es tan directa.

El **barrido dirigido** consiste en una solución híbrida entre la selección directa y el barrido. Los usuarios con suficiente movilidad pueden recorrer la matriz que contiene el conjunto de selección mediante un joystick (o su matriz de emulación, compuesta por entre 3 y 9 opciones), lo que permite una selección más rápida que el barrido y requiere menos precisión motora que la selección directa.

La **codificación** permite generar diferentes códigos asociados a diferentes significantes mediante un número restringido de teclas. Cada carácter tiene un código asociado que se compone, mediante la pulsación, de la secuencia de teclas adecuada. La ventaja de este sistema es que el número de teclas necesario es pequeño. Comparado con los teclados de selección directa, la movilidad requerida es menor y la complejidad cognitiva puede ser mayor. Los teclados reducidos empleados en sistemas codificados pueden ser igualmente usados mediante barrido, con lo que se combinan las ventajas de ambos métodos de acceso.

Cuando los elementos a codificar tienen distinta frecuencia de aparición, resultan más rentables los códigos de longitud variable, en los que se asignan las codificaciones más cortas a los elementos más frecuentes. Uno de los métodos de codificación de longitud variable más conocido es el código Morse, que ha sido empleado con éxito en ciertos casos que requieren sistemas de comunicación alternativa, adaptándolo a su uso mediante uno, dos o tres pulsadores.

4. El sistema operativo Android

4.1. Introducción

Este proyecto se ha desarrollado completamente bajo la tecnología de Android. Con el fin de situarlo en su contexto, en este capítulo se van a exponer las principales características del sistema operativo Android. Inicialmente se analizará brevemente la historia y evolución del mismo para proceder a explicar la arquitectura general de su diseño. Se hará un estudio más en detalle de la última capa de esta arquitectura, que es sobre la que los desarrolladores implementan las aplicaciones. Finalmente se profundizará en los aspectos relativos a la accesibilidad para acabar redactando una guía sobre la creación de aplicaciones accesibles.

4.2. Características generales

Android es un Sistema Operativo basado en Linux, diseñado principalmente para dispositivos móviles con pantalla táctil como teléfonos inteligentes (*smartphones*) o tabletas inicialmente desarrollados por Android inc., que Google respaldó económicamente y más tarde compró en 2005. Android fue presentado en 2007 junto con la fundación del *Open Handset Alliance;* un consorcio de compañías de *hardware*, *software* y telecomunicaciones para avanzar en los estándares abiertos de los dispositivos móviles. Google liberó la mayoría del código de Android bajo la licencia Apache, caracterizada por ser libre y de código abierto. El primer móvil con el Sistema Operativo Android se vendió en octubre de 2008 [5].

Android ha sufrido numerosas actualizaciones desde su liberación inicial. Estas actualizaciones típicamente arreglan errores del sistema operativo base y agregan nuevas funciones. Generalmente cada actualización del sistema operativo Android se presenta bajo un nombre en código de un elemento relacionado con postres, en orden alfabético. Android diferencia entre versión de la plataforma y nivel de API (*Application programmer interface*). El nivel de API es un número entero que identifica de forma exclusiva la revisión del *framework* de desarrollo. Una misma versión de la plataforma, puede mantener, hacia los desarrolladores, niveles de API anteriores. Además, cada nuevo nivel de API mantiene compatibilidad con niveles más bajos. En la web oficial para desarrolladores de Android se detallan las equivalencias entre las diferentes versiones de la plataforma con sus correspondientes niveles de API [6].

Según la compañía de investigación de mercado Canalys, Android alcanzó en España el 92% en ventas de nuevos *smartphones* para el trimestre comprendido entre diciembre 2012 y febrero 2013, seguido de iOS, el sistema operativo para dispositivos móviles de Apple, con un 4.4% [7].

4.3. Arquitectura Android

La arquitectura de Android está formada por varias capas que facilitan al desarrollador la creación de aplicaciones. Además, esta distribución permite acceder a las capas más bajas mediante el uso de bibliotecas para abstraer al desarrollador de las funcionalidades de más bajo nivel que hacen uso de los componentes hardware de los dispositivos. Cada una de las capas, utiliza elementos de la capa inferior para realizar sus funciones, es por ello que a este tipo de arquitectura se le conoce también como pila. La *figura 5* muestra un diagrama de la arquitectura de Android tomada del sitio oficial para los desarrolladores de Android [10].

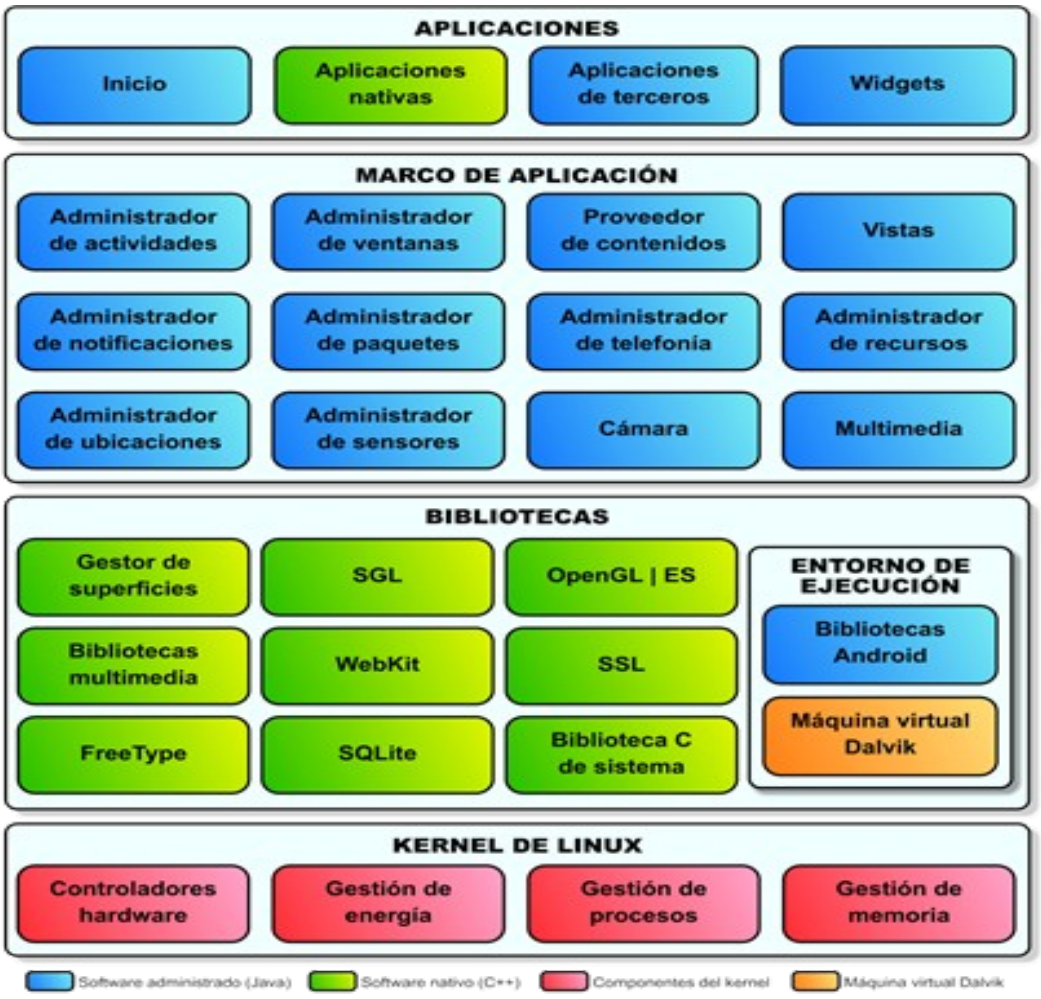


Figura 5: Arquitectura del Sistema Operativo Android

4.3.1. Kernel de Linux

El núcleo del sistema operativo Android es un *kernel Linux* versión 2.6, similar al que puede incluir cualquier distribución de Linux, como Ubuntu, sólo que adaptado a las características del hardware en el que se ejecutará Android (normalmente, arquitecturas ARM).

Proporciona una capa de abstracción para los elementos hardware a los que tienen que acceder las aplicaciones. Esto permite hacer uso de estos componentes sin necesidad de conocer sus características específicas gracias a la existencia de controladores (o *drivers*) dentro del *kernel* que permiten su utilización desde la capa software. Estos drivers, son la parte del sistema operativo no registrada bajo licencia apache, ya que son los fabricantes de dispositivos los que implementan el *firmware* de sus productos [8].

Además de proporcionar controladores *hardware*, el *kernel* se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, …) y del sistema operativo en sí: Procesos, elementos de comunicación, etc.

4.3.2. Bibliotecas

La capa que se sitúa sobre el *kernel* la componen las bibliotecas nativas de Android. Estas bibliotecas están escritas en C o C++ y compiladas para la arquitectura hardware específica del teléfono, tarea que normalmente realiza el fabricante, que también se encarga de instalarlas en el terminal antes de ponerlo a la venta. Su cometido proporciona funcionalidad a las aplicaciones, para tareas que se repiten con frecuencia, evitando tener que codificarlas cada vez y garantizando que se lleven a cabo de la forma más eficiente.

Estas son algunas de las bibliotecas que se incluyen habitualmente [9]:

- **Gestor de superficies (***Surface Manager***)**: Se encarga de componer las imágenes que se muestran en la pantalla a partir de capas gráficas 2D y 3D. Cada vez que la aplicación pretende *dibujar* algo en la pantalla, la biblioteca no lo hace directamente sobre ella. En vez de eso, realiza los cambios en imágenes (mapas de bits) que almacena en memoria y que después combina para formar la imagen final que se envía a pantalla. Esto permite realizar con facilidad diversos efectos: superposición de elementos, transparencias, transiciones, animaciones, etc.

- **SGL (Scalable Graphics Library)**: Desarrollada por Skia (empresa adquirida por Google en 2005) y utilizada tanto en Android como en Chrome (navegador web de Google), se encarga de representar elementos en dos dimensiones. Es el motor gráfico 2D de Android.

- **OpenGL | ES (OpenGL for Embedded Systems)**: Motor gráfico 3D basado en las APIs (*Application Program Interface*) de OpenGL ES 1.0, 1.1 (desde la versión 1.6 de Android) y 2.0 (desde la versión 2.2 de Android). Utiliza aceleración *hardware* ;si el teléfono la proporciona; o un motor *software,* cuando no la hay.

- **Bibliotecas multimedia**: Basadas en OpenCORE, permiten visualizar, reproducir e incluso grabar numerosos formatos de imagen, vídeo y audio como JPG, GIF, PNG, MPEG4, AVC (H.264), MP3, AAC o AMR.

- **WebKit**: Motor web utilizado por el navegador (tanto como aplicación independiente como embebido en otras aplicaciones). Es el mismo motor que utilizan Google Chrome y Safari (el navegador de Apple, tanto en Mac como en el iPhone).

- **SSL (Secure Sockets Layer)**: Proporciona seguridad al acceder a Internet por medio de criptografía.

- **FreeType**: Permite mostrar fuentes tipográficas, tanto basadas en mapas de bits como vectoriales.

- **SQLite**: Motor de bases de datos relacionales, disponible para todas las aplicaciones.

- **Biblioteca C de sistema (libc)**: Está basada en la implementación de *Berkeley Software Distribution* (BSD), pero optimizada para sistemas Linux embebidos. Proporciona funcionalidad básica para la ejecución de las aplicaciones.

4.3.3. Entorno de ejecución

El entorno de ejecución de Android, aunque se apoya en las bibliotecas enumeradas anteriormente, no se considera una capa en sí mismo, dado que también está formado por bibliotecas. En concreto, las bibliotecas esenciales de Android, incluyen la mayoría de la funcionalidad de las bibliotecas habituales de Java así como otras específicas de Android.

El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik, componente que ejecuta todas y cada una de las aplicaciones no nativas de Android. Las aplicaciones se codifican normalmente en Java y son compiladas, pero no generan un ejecutable binario compatible con la arquitectura hardware específica del dispositivo Android. En lugar de eso, se compilan en código intermedio para la máquina virtual *Dalvik*, que es la que las ejecuta. Esto permite compilar una única vez las aplicaciones y distribuir las ya compiladas teniendo la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera cada aplicación [8].

Aunque las aplicaciones se escriben en Java, Dalvik no es realmente una máquina virtual Java. Es decir, no es compatible con el *bytecode* Java (instrucciones ejecutables independientes de la arquitectura hardware) que ejecutan las máquinas virtuales Java normales. Java se usa únicamente como lenguaje de programación, pero los ejecutables que se generan con el SDK de Android no son ejecutables Java convencionales y, por lo tanto, no se pueden ejecutar en máquinas virtuales Java convencionales. Durante el proceso de compilación de los programas Java (normalmente archivos .java) sí que se genera, de forma intermedia, el *bytecode* habitual (archivos .class). Pero esos archivos son convertidos al formato específico de Dalvik en el proceso final (.dex, de *Dalvik executable*).

Google hace esto por una cuestión de optimización. Los archivos .dex son mucho más compactos que los .class equivalentes, lo que permite ahorrar memoria en el teléfono y acelerar el proceso de carga. Además, a diferencia de las máquinas virtuales tradicionales, Dalvik se basa en registros en lugar de una pila para almacenar los datos, requiriendo menos instrucciones. Todo ello permite ejecuciones más rápidas en un entorno con menos recursos.

Las aplicaciones Android se ejecutan cada una en su propia instancia de la máquina virtual Dalvik, evitando así interferencias entre ellas, y tienen acceso a todas las bibliotecas mencionadas antes y, a través de ellas, al hardware y al resto de recursos gestionados por el *kernel* [10].

4.3.4. *Framework* de aplicaciones

Es la capa formada por todas las clases y servicios que utilizan directamente las aplicaciones para realizar sus funciones y que se apoyan en las bibliotecas y en el entorno de ejecución detallado en el apartado anterior. La mayoría de los componentes de esta capa son bibliotecas Java que acceden a los recursos a través de la máquina virtual Dalvik. El siguiente apartado de este capítulo se dedica a hacer un repaso de las principales herramientas de esta capa.

4.3.5. Aplicaciones

La capa superior de esta pila software la forman todas las aplicaciones instaladas en el dispositivo, tanto las que tienen interfaz de usuario como las que no, tanto las nativas (programadas en C o C++), como las administradas (programadas en Java), tanto las que vienen de serie con el dispositivo, como las instaladas por el usuario.

Esta capa incluye la aplicación principal del sistema: **Inicio** (*Home*), llamada a veces lanzador (*launcher*), porque es la que permite ejecutar otras aplicaciones proporcionando la lista de aplicaciones instaladas y mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones o incluso pequeñas aplicaciones incrustadas o *widgets*, que son a su vez aplicaciones de esta capa.

Lo principal, a tener en cuenta, de esta arquitectura es que todas las aplicaciones, ya sean las nativas de Android, las que proporciona Google, las que incluye de serie el fabricante del teléfono o las que instala después el usuario, utilizan el mismo marco de aplicación para acceder a los servicios que proporciona el sistema operativo. Esto implica dos cosas. Que podemos crear aplicaciones que usen los mismos recursos que usan las aplicaciones nativas (nada está reservado o inaccesible) y que podemos reemplazar cualquiera de las aplicaciones del teléfono por otra de nuestra elección. Esto permite al usuario tener un control total sobre el software que se ejecuta en su dispositivo [10].

4.4. *Framework* de aplicaciones

Como ya se ha comentado en el apartado anterior, esta capa de la arquitectura del sistema Android la forman todas las clases y servicios que utilizan directamente las aplicaciones. A continuación se describen las más relevantes para el desarrollo de aplicaciones:

- **Administrador de actividades (***Activity Manager***)**: Se encarga de controlar el ciclo de vida de las actividades y la propia pila de actividades. Las actividades definen cómo se muestra la interfaz de usuario sobre la pantalla del dispositivo Android [11].
- **Administrador de ventanas (***Windows Manager***)**: Se encarga de organizar lo que se muestra en pantalla, creando superficies que pueden ser *rellenadas* por las actividades [12].
- **Proveedor de contenidos (***Content Provider***)**: Permite encapsular un conjunto de datos que va a ser compartido entre aplicaciones creando una capa de abstracción que hace accesible dichos datos sin perder el control sobre cómo se accede a la información. Por ejemplo, uno de los proveedores de contenido existentes permite a las aplicaciones acceder a los contactos almacenados en el teléfono. Esta biblioteca nos permite crear también nuestros propios proveedores para permitir que otras aplicaciones accedan a información que gestiona la nuestra [13].
- **Vistas (***Views***)**: Las vistas son cualquier elemento visual que se añade a una ventana. Android proporciona numerosas vistas con las que construir las interfaces de usuario: botones, cuadros de texto, listas, etc. También proporciona otras más sofisticadas, como un navegador web o un visor de Google Maps [14].
- **Administrador de notificaciones (***Notification Manager***)**: Proporciona servicios para notificar al usuario cuando algo requiere su atención. Normalmente las notificaciones se realizan mostrando una alerta en la barra de estado, pero esta biblioteca también permite sonidos, activar el vibrador o hacer parpadear los LEDs del dispositivo (si los tiene) [15].
- **Administrador de paquetes (***Package Manager***)**: Las aplicaciones Android se distribuyen en paquetes (archivos .apk) que contienen tanto los archivos *".dex"* como todos los recursos y archivos adicionales que necesite la aplicación para facilitar su descarga e instalación. Esta biblioteca permite obtener información sobre los paquetes actualmente instalados en el dispositivo Android, además de gestionar la instalación de nuevos paquetes [16].

- **Administrador de telefonía (***Telephony Manager***)**: Proporciona acceso a la pila hardware de telefonía del dispositivo Android, si la tiene. Permite realizar llamadas o enviar y recibir SMS/MMS, aunque no permite reemplazar o eliminar la actividad que se muestra cuando una llamada está en curso (por motivos de seguridad) [17].
- **Administrador de recursos (***Resource Manager***)**: Proporciona acceso a todos los elementos propios de una aplicación que se incluyen directamente en el código como cadenas de texto traducidas a diferentes idiomas, imágenes, sonidos e incluso disposiciones de las vistas dentro de una actividad (*layouts*). Permite gestionar esos elementos fuera del código de la aplicación y proporcionar diferentes versiones en función del idioma del dispositivo o la resolución de pantalla que tenga, por ejemplo [18].
- **Administrador de ubicaciones (***Location Manager***)**: Permite determinar la posición geográfica del dispositivo Android (usando el GPS o las redes disponibles) y trabajar con mapas [19].
- **Administrador de sensores (***Sensor Manager***)**: Gestiona los sensores hardware disponibles en el dispositivo Android: Acelerómetro, giroscopio, luminosidad, campo magnético, brújula, presión, proximidad, temperatura, etc [20].
- **Cámara**: Proporciona acceso a las cámaras del dispositivo Android, tanto para tomar fotografías como para grabar vídeo [21].
- **Multimedia**: Conjunto de bibliotecas que reproducen y visualizan audio, vídeo e imágenes en el dispositivo [22].

4.5. Introducción al desarrollo de aplicaciones en Android

Para poder entender cómo desarrollar aplicaciones accesibles sobre Android y cual es el verdadero potencial que nos ofrece su entorno de desarrollo, considero necesarias unas nociones básicas sobre la creación de aplicaciones para este sistema operativo. A continuación, se procede a explicar brevemente las principales herramientas del *framework* para este cometido.

4.5.1. Activity

Una Activity es un componente de la aplicación que proporciona una pantalla con la que los usuarios interactúan y que provee la funcionalidad de la aplicación. En términos de ingeniería del software, una Activity es el nexo de unión entre la vista y la lógica de negocio. Cada Activity recibe una ventana en la que dibuja la interfaz de usuario [23].

Una aplicación se compone, generalmente, de varias actividades. Una de ellas se declara como la principal, que será la que primero se dibuja cuando se ejecuta la aplicación. Cada Activity puede lanzar otras con el fin de realizar diferentes acciones. Cada vez que se inicia una nueva Activity, la anterior se detiene, pero el sistema conserva su referencia en una pila (“Back stack”). El usuario puede acceder a las actividades anteriores a través del botón atrás, que recupera la Activity de la cima de la pila.

Para la gestión de las actividades dentro de una aplicación, el desarrollador debe conocer lo que la documentación de Android llama “el ciclo de vida” de las actividades. A continuación se muestra un gráfico con dicho ciclo de vida [25].

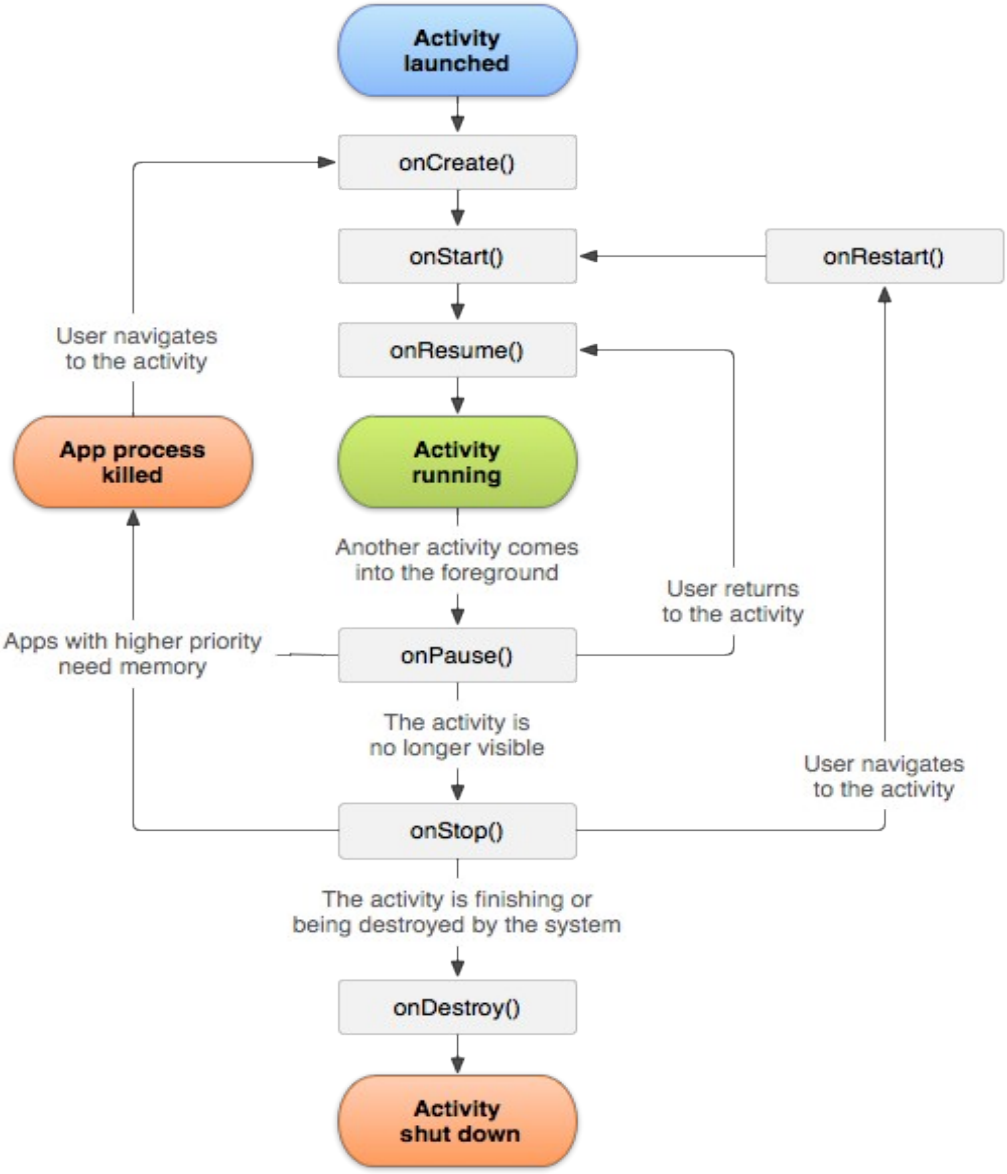


Figura 6: Ciclo de vida de una Acitivity

4.5.2. Service

Un Service es un componente de una aplicación, entendiendo como componentes aquellos elementos que forman parte de la aplicación, que pude realizar operaciones de larga duración en segundo plano y no proporciona una interfaz de usuario. Otro componente de la aplicación, como una Activity, puede iniciar un servicio y que continuará funcionando de fondo incluso si el usuario cambia a otra aplicación. Además, un componente puede unirse a un servicio para interactuar con él e incluso realizar una comunicación entre procesos. Por ejemplo, un servicio puede manejar las transacciones de red, reproducir música o interactuar con un proveedor de contenidos (se explica con más detalle este componente en el siguiente apartado), todo en segundo plano [24].

Es necesario destacar que un servicio no se ejecuta en un proceso aparte del componente que lo ejecuta. Si se requiere una operación de larga duración que pueda ser bloqueante, es necesario que el servicio cree un hilo aparte para esta tarea.

Un servicio se puede ejecutar de dos formas:

- **Desatendida.** Se inicia el servicio a través del método *startService*() propio de la clase *Context*, de la que hereda la clase *Activity*. Una vez iniciado, un servicio puede ejecutarse en segundo plano de forma indefinida, incluso si el componente que lo inició se destruye. Esta forma de ejecución es típica para tareas que no requieren una respuesta por parte del servicio, por ejemplo, descargar o cargar un fichero en la red. Una vez realizada la operación, el servicio puede pararse por sí mismo, pero también puede ser la propia *Activity* la que lo detenga a través del método *stopService() [27]*.
- **Con conexión.** Este modo de ejecución establece una conexión directa con el servicio ejecutado. Un componente de aplicación se une al servicio llamando al método *bindService()*. Un servicio con conexión ofrece una interfaz cliente-servidor que permite interactuar con él, enviar solicitudes, obtener resultados, etc. Un servicio ejecutado de esta forma está activo mientras el componente que lo ha lanzado esté conectado a él. El servicio se destruye cuando finaliza el componente o cuando se le indica expresamente la desconexión [26].

Aunque se diferencian estas dos formas de uso de los servicios de Android, no son excluyentes. Un servicio se puede ejecutar de forma desatendida y establecer una conexión con él. Es cuestión de qué métodos de la clase se implementen. El método *onStartCommand()* es el encargado de ejecutar el servicio de forma desatendida mientras que para iniciar una conexión con él, se ha de implementar el método *onBind() [24]*.

4.5.3. Intent

Un objeto Intent es una estructura de datos pasiva que contiene una descripción abstracta de una operación a realizar [28]. Es el mecanismo utilizado en Android para la llamada a componentes de una aplicación, por ejemplo, para ejecutar servicios o actividades tanto propias de la aplicación como de otras aplicaciones instaladas en el dispositivo. Los métodos startActivity() y startService() de la clase Context reciben como parámetro un objeto Intent con la información necesaria para que el sistema ejecute un Service o una Activity en tiempo de ejecución.

4.5.4. ContentProvider

La clase *ContentProvider* es la encargada de gestionar el acceso a los datos de una aplicación. Permite la abstracción del acceso a datos por parte de las aplicaciones y provee una interfaz de comunicación para la compartición de los datos propios de una aplicación. Android facilita implementaciones de esta clase para acceso a datos existentes en el dispositivo como el contenido multimedia (música, vídeos e imágenes) o la lista de contactos [29].

5. Accesibilidad en Android

En un principio, Android no incorporó ningún tipo de accesibilidad en sus primeras versiones, no obstante, se creó el grupo de trabajo *Eyesfree Project [30].* Para el desarrollo y mejora constante de herramientas para conseguir la accesibilidad en un dispositivo Android. Algunos de sus proyectos más importantes son el motor de *Text-To-Speech picoTTS*, el lector de pantallas *Talkback* o la aplicación *WalkyTalky*, todos ellos de código abierto.

En el *ANEXO C* se muestra una tabla con la evolución de la accesibilidad en el sistema operativo Android. Como se puede ver en dicha tabla, hasta la versión de Android 1.6 no se introdujo una API de accesibilidad, en la que se incluyen las clases relacionadas con servicios y componentes para la accesibilidad [33].

A partir de esta versión, todos los componentes de interfaz en Android (*Views*) incluyen un manejador de eventos de accesibilidad, de esta forma, si se desarrolla utilizando los componentes propios de Android, la aplicación implementará accesibilidad de forma intrínseca.

No obstante, no todos los componentes proveen accesibilidad completa o es necesario la declaración de un nuevo componente personalizado. En estos casos es necesario definir cierta funcionalidad de accesibilidad, en versiones anteriores a Android 4.0 (ICS) se debe sobrescribir el método *dispatchPopulateAccessibilityEvent* [31] y a partir de ICS existe la clase *AccesibilityDelegate*, que se debe sobrescribir para gestionar los eventos de accesibilidad. En el siguiente apartado se incluye una guía completa para la implementación de aplicaciones accesibles en Android en la que se describe paso a paso los aspectos a tener en cuenta para poder usar todo el potencial que Android ofrece en el ámbito de la accesibilidad.

Además de estas soluciones accesibles es necesario proporcionar ayuda a los desarrolladores para que los futuros desarrollos sean cada vez más accesibles. Para ello, el proyecto ACCESSIBLE, de la fundación Vodafone, proporciona un entorno de validación de accesibilidad y simulación de discapacidad basados en código abierto [32]. Este entorno permite que desarrolladores de páginas *Web*, *Web Services* y aplicaciones móviles, sean conscientes del nivel de accesibilidad de sus aplicaciones.

5.1. Características de Accesibilidad en Android

Las características propias del sistema operativo que permiten la creación y utilización de diferentes Ayudas Técnicas para mejorar la accesibilidad en Android, éstas han ido apareciendo en diferentes versiones.

Text-To-Speech (TTS), también conocido como síntesis de voz, permite al terminal reproducir texto convirtiéndolo a voz.

La síntesis de voz se implementó por primera vez en la versión 1.5, pero hasta la versión 2.2 no se empezó a incorporar la posibilidad de cambiar el motor de voz (*TTS Engine*). Por defecto en las versiones anteriores a ICS (Android 4.0) se incorporan el *TTSEngine PicoTTS*. A partir de esta versión se implementa *Google.TTS*, ambos desarrollados en el marco de *Eyesfree project* [34]. En algunas ocasiones, como en el caso de los terminales Samsung, es el propio fabricante el que proporciona el motor de voz. Aparte de los que incorpora el smartphone de fábrica, existen multitud de *TTS Engine* que se pueden instalar en el terminal. Pueden utilizar el servicio en la nube o en local, de código abierto, gratuitos o de pago, o un sin fin de posibilidades.

Estos motores de síntesis de voz tienen varios parámetros de configuración. Los más comunes, que suelen compartir todos, son el idioma, el tipo de voz (generalmente masculina o femenina), el *Rate* (velocidad de lectura) y el *Pitch* (frecuencia del tono de la voz).

Cuando se quiere desarrollar usando el TTS se debe instanciar la clase *Text-to-Speech*, abstrayéndose del *TTSEngine* que esté seleccionado, por defecto, en ese momento [35] [36].

Explore by Touch es una funcionalidad de todo el sistema Android que actúa en todas las aplicaciones sin necesidad de la intervención de ningún desarrollador. Esta funcionalidad, si está activa, permite al sistema generar eventos de accesibilidad según se vaya colocando el dedo en diferentes partes de la pantalla. Estos eventos pueden ser escuchados por un lector de pantalla que se encargue de reproducir la descripción y/o el contenido del elemento que se esté explorando en ese momento. El *Explore by Touch* de Android se incorporó por primera vez en la versión Android 4.0 (ICS).

A partir de la versión 4.1 (*Jelly Bean*) se ha mejorado el *Explore by Touch* permitiendo la navegación lineal mediante gestos. Esto consiste en que en vez de tener que mover el dedo hacia donde se encuentre el elemento, éste es accesible mediante una serie de gestos de navegación. Existen dieciséis gestos diferentes, los cuatro básicos para realizar la navegación (derecha, arriba, izquierda y abajo). Los otros doce restantes son las combinaciones posibles de dos gestos básicos. Cada uno de estos gestos tiene una funcionalidad diferente, como volver atrás, ir al *Home*, abrir aplicaciones recientes, etc.

En ciertas ocasiones, cuando se desarrolla una aplicación, si se desea que sea completamente accesible mediante la navegación por gestos, es necesario añadir ciertos eventos

de entrada en la declaración de la accesibilidad del componente. También es posible personalizar la navegación de la aplicación indicando que componentes pueden ser explorados por este método e indicar cual es el siguiente componente en obtener el foco al navegar en cierta dirección.

5.2. Ayudas técnicas

Para desarrollar aplicaciones accesibles para Android es necesario conocer las ayudas técnicas (ATs) que posibilitan la accesibilidad del dispositivo. A continuación se realiza un breve recorrido sobre el estado del arte de las ayudas técnicas más utilizadas y reconocidas por usuarios en la plataforma Android.

Las ATs en Android se pueden dividir en servicios de accesibilidad o métodos de entrada.

Un Servicio de Accesibilidad es un proceso que corre en segundo plano y gestiona los eventos de accesibilidad generados por el sistema. Los eventos de accesibilidad se generan cuando el usuario interacciona con la interfaz de usuario, como cuando se hace click en un botón o cuando un *View* tiene el foco. El objetivo de un Servicio de Accesibilidad es ofrecer un feedback alternativo al usuario. El más utilizado en Android es el lector de pantallas *Talkback*.

Un Método de Entrada *(Input Method, IME)* es una herramienta software que permite a los usuarios introducir texto y navegar por la interfaz de usuario. Se instala en el sistema Android como una aplicación más, pero ésta debe contener una clase que extienda de *InputMethodService*. Por lo general, estas ATs son herramientas de software que facilitan el acceso a dispositivos móviles para las personas con problemas de movilidad. Un claro ejemplo serían los IME *Dasher* o *Tecla*, desarrollados en el marco del proyecto AEGIS [37].

A continuación se van a describir algunas de las ayudas técnicas más importantes y utilizadas en Android:

- *Talkback*:

Talkback es un Servicio de Accesibilidad oficial de Google, es el lector de pantalla más utilizado en Android [38]. Existen dos aplicaciones de *Talkback* diferentes:

Anterior a Android 4.0 (ICS). También conocido como *Talkback preICS*, éste es un lector de pantalla bastante simple, no tiene opciones de configuración y únicamente se limita a leer el elemento que contiene el foco.

Desde Android 4.0 (ICS) A partir de ICS todos los dispositivos incorporan el *Talkback* de fábrica, aquí el *Talkback* no es sólo un lector de pantalla, sino un servicio de accesibilidad que proporciona un feedback de todo tipo. Además, escucha eventos de *onHover* y *offHover*

generados al tener activo el modo de exploración táctil. En este *Talkback* se permiten varios parámetros de configuración, como poder *cambiar* el volumen o configurar otros tipos de feedback como la vibración.

Cabe destacar que en *Talkback* la versión utilizada también importa en ciertos casos. Se recomienda utilizar siempre la última versión disponible, no obstante a la hora de desarrollar, debemos tener en cuenta las versiones anteriores. Un ejemplo de estas diferencias se encuentra a partir de la versión 3.3, donde ya se incluye por defecto, en los botones, el tipo de elemento que son, mientras que en las versiones anteriores, si el desarrollador no lo indicaba, sólo reproducía el nombre de la etiqueta.

- *KickBack*

Es uno de los servicios de accesibilidad oficiales de Google. Provee un feedback de vibración cuando sucede un evento, como desbloquear la pantalla o cambiar el foco del elemento. A partir de Android 4.0 (ICS) estas funcionalidades están integradas en el *Talkback* [39].

- *SoundBack*

Es uno de los servicios de accesibilidad oficiales de Google. Genera un breve sonido cuando se realiza una acción sobre el teléfono como hacer click en un elemento o mover la selección con el manejador de selección. A partir de Android 4.0 (ICS) estas funcionalidades están integradas en el *Talkback* [40].

- *Tecla Access*

Es un método de entrada alternativo diseñado para facilitar el manejo del dispositivo para personas con movilidad reducida. Tecla soporta el control del móvil mediante dispositivos inalámbricos, como un teclado físico, un pulsador externo o la propia silla de ruedas del usuario. A parte del teclado normal para la introducción de texto, Tecla proporciona un teclado táctil para navegar en el móvil. También dispone de un modo en pantalla completa que permite controlar el dispositivo e introducir texto tan solo pulsando en la pantalla mediante un sistema de barrido [41].

- *Dasher*

Dasher es un método de entrada diseñado para la introducción de texto mediante una serie de suaves movimiento de un puntero, de esta forma se puede escribir arrastrando suavemente el dedo por la pantalla, girando el móvil a través de la información que proporciona el acelerómetro o mediante dispositivos externos como *joysticks*, *trackballs*, pulsadores, pero también punteros controlados por los pies, la cabeza, con los ojos o incluso con el *Wii-mote*. Dasher está planteado

principalmente para usuarios con movilidad reducida, pero también está destinado a personas que simplemente utilizan este teclado por la rapidez que se puede alcanzar escribiendo [42].

- *EyesFree Keyboard*

Es un método de entrada de texto que además proporciona una interfaz para navegar por la pantalla mediante gestos, consiste en un Controlador Direccional (D-pad) que permite mover el foco en las cuatro direcciones. Se debe utilizar junto con un lector de pantalla, recomendablemente *Talkback*. El teclado de texto se puede explorar con el dedo mientras se reproduce letra a letra y dando la posibilidad de escribir a personas con discapacidad visual.

No todas las interfaces se pueden explorar con el D-pad y la exploración del teclado no funciona correctamente en todos los modelos de dispositivos [43].

- *BrailleBack*

Es una App desarrollada por *Eyes-Free project*, que permite conectar un dispositivo externo de braille permitiendo navegar y escribir texto mediante este método [44].

5.3. Conclusión

El sistema operativo Android ofrece un gran número de herramientas y ayudas técnicas que lo hacen accesible cada vez a más usuarios. No obstante, como puede deducirse de la guía presentada en el ANEXO A, la adaptación de aplicaciones Android a la accesibilidad, está sujeta al nivel de API usado en su desarrollo. Hasta los niveles 9 y 10 (Android 2.3.3 Gingerbread) Android implantó dichas herramientas y el desarrollo de aplicaciones accesibles para niveles inferiores se complica si no están totalmente basadas en los componentes proporcionados por el *framework.* Sin embargo, existe una gran variedad de dispositivos en el mercado con versiones del sistema operativo anteriores a Android 2.3.3 (Gingerbread, API nivel 9 y 10). La *figura 7* muestra que más del 60% de los dispositivos móviles integran dicha versión [46].

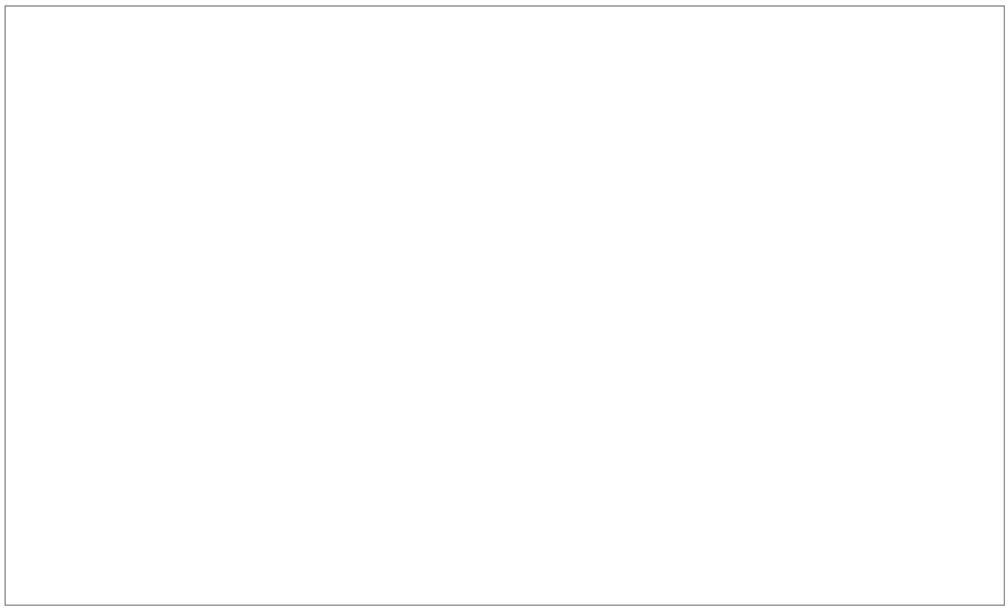


Figura 7: Fragmentación de las versiones de Android en los dispositivos móviles vendidos hasta mayo de 2012

Desde el mercado de las aplicaciones Android, se requiere la mayor compatibilidad posible con todas las distribuciones del sistema operativo, por lo que, en la práctica, la implementación de aplicaciones accesibles queda limitada a las librerías de soporte y a la habilidad del desarrollador en la implementación de interfaces gráficas, así como en la comprensión de la arquitectura de Android.

Desde la empresa Limbika, se planteó una solución para la accesibilidad de personas con diversidad motriz al sistema operativo en sus versiones iniciales. Un barrido que pueda funcionar sobre cualquier aplicación y que sea independiente de la plataforma Android sobre la que se ejecuta. En el siguiente capítulo se desarrolla ampliamente el diseño e implementación de la librería AccessibilityHUD, que proporciona un sistema de barrido.

6. Sistema de barrido para accesibilidad.

En este capítulo se explican los principales problemas encontrados a la hora de desarrollar un servicio de barrido sobre el sistema operativo Android. Se analiza la captura de requisitos proporcionada por la empresa Limbika y se procede a resolver los aspectos más relevantes de la implementación.

Al ser un desarrollo realizado en un ámbito empresarial, los detalles del código no pueden ser expuestos en esta memoria ya que están sujetos a una licencia de software propietaria. Así mismo, las pruebas unitarias realizadas no pueden ser expuestas por razones análogas. No obstante, está a nuestro alcance dar una explicación detallada de las soluciones empleadas ante las limitaciones del *framework* de Android para la implementación de un sistema de barrido.

6.1. Captura de requisitos:

Se trata de implementar un barrido de ejes cartesianos que permite generar un evento determinado sobre las coordenadas de intersección de los ejes. Inicialmente el sistema dibuja una barra horizontal o vertical, que se desplaza realizando un barrido sobre toda la pantalla. Cuando el usuario lo desea, detiene el desplazamiento para ese eje. Seguidamente se dibuja en la pantalla el otro eje, con un comportamiento análogo, hasta que el usuario lo detenga. En ese momento, el sistema genera un evento con las coordenadas de la intersección de los dos ejes sobre la ventana actual. Dicho evento puede ser configurable permitiendo al desarrollador elegir el tipo de comportamiento. El ejemplo típico es la generación de un evento de click, con un comportamiento equivalente a posicionar el ratón en ese punto y hacer click con el botón izquierdo.

Además, la interfaz visual del barrido se puede configurar de forma independiente para cada uno de los ejes:

- **Posición inicial**. La coordenada tanto en el eje *x* como en el *y* desde la que ese eje empieza a moverse.
- **Tiempo de espera** antes de empezar el movimiento. Define cuánto tiempo espera la barra desde que se dibuja hasta que empieza a moverse.
- **Sentido de movimiento** al inicio. Esto es, cuando se desplaza el eje, en qué sentido lo hace dependiendo de qué eje sea, este u oeste, norte o sur.
- **Anchura del eje**. Es una configuración meramente visual que no influye con la precisión de las coordenadas del evento, sino que ayuda al usuario a identificar el eje sobre el resto de la interfaz.
- **Velocidad**. Establece la velocidad a la que se desplaza el eje.

También permite elegir **con qué eje comienza** el sistema, el vertical o el horizontal, para que la primera barra que se dibuja sea la elegida por el usuario.

Además dispone de la opción de **restaurar la posición inicial** de las barras a la definida en la configuración o a la posición que quedaron en la intersección anterior, estableciendo la posición previa como la posición de inicio de los ejes.

El tipo de evento generado por el sistema en la intersección de las dos barras puede ser configurable. Permitiendo decidir qué tipo de comportamiento se desea dentro de los eventos proporcionados por Android.

En cuanto a los requisitos técnicos, el sistema puede ser utilizado por cualquier dispositivo Android del mercado en cualquiera de sus versiones. Así mismo, es independiente de la aplicación, esto es, funciona como un servicio propio del sistema sin tener que integrar el código en la aplicación. Esto permite ser utilizado sobre cualquier aplicación. A la vez que puede formar parte de la misma en caso necesario. Su control se realiza a través de cualquier elemento hardware que se pueda conectar al móvil mediante cualquier interfaz (teclados, *trackballs*, ratones, etc). La conexión de dispositivos inalámbricos al dispositivo se realiza mediante los controladores *wifi* o *bluetooth* existentes en el dispositivo, que identifican el tipo de periférico. El *framework* de Android permiten al desarrollador abstraerse del tipo de conexión del elemento de entrada con lo que el sistema es intrínsecamente compatible con este tipo de *hardware*.

6.2. Estudio sobre las posibles estrategias a seguir.

Haciendo un análisis funcional del sistema, se trata de controlar una interfaz visual (los dos ejes) desde aplicaciones externas, por lo que es necesaria la comunicación constante con las aplicaciones sobre las que se está usando. A su vez, funciona estando asociado a una aplicación concreta que requiere usar el sistema en un contexto determinado. Como puede deducirse, el comportamiento de los ejes es independiente del contexto de ejecución del sistema de barrido. Los ejes siempre van a tener el mismo comportamiento, lo que permite hacer un análisis por separado de ambos problemas haciendo un diseño Modelo-Vista-Controlador. La interfaz visual de los ejes es la vista, la configuración de los ejes mencionada en la captura de requisitos hará las veces de modelo y el controlador es la parte encargada de asignar a los ejes los parámetros de configuración y ofrecer al desarrollador una interfaz de control para ejecutar el sistema, pararlo o modificar su comportamiento.

6.2.1. Interfaz visual de los ejes. Componente nativo o propio.

Para la interfaz visual de los ejes cartesianos, Android proporciona varios componentes de dibujado como pueden ser botones, textos, imágenes, ventanas emergentes, etc. El comportamiento del sistema requiere de un elemento visual que podamos mover por la pantalla y que pueda capturar eventos de entrada/salida. En un análisis inicial, se intentó usar uno de los componentes ya citados, como un botón, que dibuja un elemento visual que captura eventos. No obstante, nuestro sistema requiere que se capturen todos los eventos que le lleguen al dispositivo,

y cualquier componente visual de Android sólo capturará aquellos cuyas coordenadas están contenidas en el interior de sus márgenes. Para poder capturar todos los eventos del sistema, el componente tiene que pintarse a lo largo y ancho de toda la pantalla. Esto implica tener que recurrir a clases más genéricas de comportamiento y dibujado.

La clase básica de la que heredan todos los elementos visuales de Android es *View*. Esta clase representa el elemento básico para los componentes de la interfaz de usuario. Una *View* ocupa un área rectangular en la pantalla, y es responsable del dibujado y la gestión de eventos. La subclase *ViewGroup* es la base para los diseños, que son contenedores invisibles que tienen otras *Views* (u otros *ViewGroups*) y definen sus propiedades de diseño y comportamiento.

El problema planteado es fácilmente solucionable a través de la clase *View*, ya que desde ella tenemos acceso al lienzo sobre el que dibujar lo que nosotros queramos. Además, el comportamiento de los ejes nos lleva a pensar en una máquina de estados para su implementación. Los estados podrían ser “esperar”, “dibujar la primera barra”, “dibujar la segunda barra” y “ejecutar evento”.

6.2.2. Controlador de la interfaz. Activity o Service

Al requerir que el barrido pueda funcionar de forma autónoma, como una aplicación independiente, las opciones de implementación son limitadas. Android provee dos vías de desarrollo para resolver este problema. Desarrollar la aplicación a través de una Actividad (*Activity*), o como un Servicio (*Service*).

Una Actividad, como el propio SDK la describe, “*es una única acción enfocada que el usuario puede hacer*”. Es el camino más usual para interactuar con el usuario, por lo que esta clase se encarga de dibujar una ventana para proporcionar la interfaz de usuario.

Un Servicio “*es un componente de una aplicación destinado a realizar operaciones de larga duración en segundo plano y no proporciona una interfaz de usuario.*”

En este punto, es necesario tomar una decisión sobre una de las dos vías de acción. A priori ambas soluciones pueden cubrir, en parte, nuestras necesidades, pero no totalmente. Si implementamos nuestro sistema como una Actividad, tendríamos un acceso sencillo al dibujado de la interfaz (los ejes) y al control de eventos de entrada. No obstante, sería difícil la comunicación con otras aplicaciones, ya que Android solo mantiene una aplicación enfocada, la que está en la cima de la pila de aplicaciones en ejecución. Esto hace imposible que el sistema funcione independiente de las aplicaciones que estén en ejecución. Tendríamos el problema de que si queremos lanzar una aplicación a través de nuestro sistema de barrido, esta se añadiría a la cima de la pila y nuestro servicio perdería el foco de la ventana y pasaría a segundo plano, por lo que no tendríamos capacidad para volver a pintar los ejes ya que nuestra Activity a sido pausada.

Parece ser que la única solución es la clase *Service.* Si implementamos el sistema como un Servicio, nos sería bastante fácil la comunicación con cualquier aplicación, independientemente de su posición en la pila. Además también podríamos manejar de forma fácil los eventos que vengan tanto de la pantalla como de un ratón, pero tendríamos problemas a la hora de dibujar la interfaz y no podríamos capturar los eventos de teclado.

6.2.3. Capturando eventos de teclado desde un Service.

Según la documentación oficial de Android, la forma adecuada de capturar eventos de teclado desde una aplicación es a través de los métodos onKeyUp() y onKeyDown(), propios de la clase Activity. No obstante, es deseable que nuestra aplicación pueda delegar el evento a su interfaz visual para que sea ésta la que lo gestione. Es el caso de la clase Button, que por defecto consume los eventos de teclado que tengan los códigos KEYKODE_DPAD_CENTER y KEYDOCE_ENTER. Esto es posible gracias a que la clase View publica una interfaz para el manejo de eventos de teclado llamada OnKeyListener. Implementando esta interfaz y asociando la instancia de la misma a una *View* determinada, a través del método setOnKeyListener(), se puede controlar los eventos de teclado que le llegan. En las secciones siguientes se hace un análisis más general del manejo de eventos por parte del sistema operativo.

6.2.4. Añadiendo interfaz visual a un Service

Según la especificación de Android, un *Service* no provee una interfaz visual dentro de la especificación de la clase como tal. Esta funcionalidad es propia de una *Activity*, que maneja el control de la ventana. Desde el SDK tenemos varias herramientas para el control de la ventana visual en curso y fijándonos en cómo Android implementa esa gestión en las Actividades y de qué tipo de clases hace uso, podemos hacernos una idea de cómo hacer un control análogo en nuestro sistema para darle una interfaz.

La interfaz *WindowManager* provee varios métodos para controlar el aspecto y comportamiento de la ventana actual a través de la clase *Window*. Las Actividades, cuando pintan su interfaz, llaman a esta clase para añadir su vista cuando se ejecutan y para eliminarla cuando se terminan. Podemos tener acceso a la instancia del sistema de la clase *WindowManager* a través de un método llamado *getSystemService()* que recibe como parámetro un *String* que identifica el servicio del sistema al que se quiere acceder. Este método es propio de la clase *Context*. Un *Context* en Android es una interfaz que provee información global acerca del entorno de la aplicación. Tanto las Actividades como los Servicios implementan esta interfaz. Gracias a esto, podemos hacer uso del método anteriormente mencionado para usar el *WindowManager* y pintar una vista para nuestro servicio.

6.3. Gestión de eventos.

Nuestro sistema debe poder gestionar los eventos que le lleguen desde la pantalla o un ratón que se conecte por un puerto. Además, debe poder generar sus propios eventos para pasarlos a la Actividad sobre la que se ejecuta.

6.3.1. Generación de eventos

En el caso de generar eventos, Android proporciona una clase llamada *Instrumentation*. Esta clase es la encargada de monitorizar toda la interacción entre el sistema y una aplicación. Provee herramientas para realizar llamadas a métodos concretos de clases implementadas en la aplicación. Su uso típico es en los proyectos de pruebas unitarias de código. En nuestro caso nos va a ser útil para generar eventos de cualquier tipo. Los métodos de la clase que nos permiten ésto son:

- *sendKeySync(KeyEvent event)*

  Envía un evento de tecla a la ventana o vista que tenga el foco y espera a que sea procesado

- *sendPointerSync(MotionEvent event)*

  Envía un evento de tipo puntero. Similar al evento que se genera al tocar la pantalla o al hacer click con un ratón.

Es importante tener en cuenta que el manejador de eventos de Android, cuando le llega un evento, busca entre las vistas la que tiene el foco y le pasa el evento, si éste es consumido, acaba, si no, sigue en la jerarquía de vistas de forma descendente hasta que no quedan. Es necesario eliminar la vista del sistema de barrido antes de generar un click para que los elementos que están debajo, los que realmente nos interesan, reciban el evento.

Por otro lado, una aplicación que utilice esta clase, sólo puede enviar eventos a elementos de la propia aplicación. No podemos enviar eventos generados dinámicamente sobre aplicaciones ajenas. Según Android, por seguridad, ninguna aplicación tiene permitido interactuar con otras aplicaciones a través de eventos generados. Parece lógico, ya que con esta clase sería relativamente sencillo implementar software malicioso que hiciese uso de otras aplicaciones para, por ejemplo, acceder a cuentas privadas. La única forma de poder inyectar eventos es que los genere una aplicación que forme parte del sistema. Para ello, nuestra aplicación ha de ir firmada con la misma firma que el sistema operativo. A priori, este aspecto puede parecer negativo, ya que no vamos a poder instalar la aplicación en sistemas de los que no conozcamos las firmas, que son prácticamente todos. No obstante, a la empresa Limbika ésto le ofreció la oportunidad de comerciar con fabricantes de dispositivos a los que les brindó la oportunidad de lanzar dispositivos con un sistema de barrido para accesibilidad integrado en el sistema operativo.

6.3.2. Captura de eventos

La gestión de eventos de entrada en Android se realiza a través de la clase View. Esta clase publica varias interfaces para cada uno de los tipos de evento que el sistema soporta. Cuando el sistema detecta una interrupción de la pantalla táctil, del teclado, del ratón, o de cualquier otro dispositivo, ejecuta el método de la interfaz correspondiente sobre la vista que en ese momento tiene el foco.

La interfaz asociada al manejo de interrupciones de la pantalla se llama *OnTouchEventListener*. Que define un método llamado OnTouch que recibe como parámetro la vista sobre la que se ha generado el evento y una instancia de la clase MotionEvent, que guarda todas las características del evento como coordenadas, duración, tipo, etc.

Para las interrupciones del ratón, la interfaz tiene el nombre de *OnClickEventListener*. Su método se llama *OnClick()* que tan solo recibe como parámetro la vista sobre la que se ha hecho un click. Realmente, el evento de click Android lo traduce como dos eventos de touch, el de pulsar (ACTION_DOWN) y el de despulsar (ACTION_UP). Esto tiene relevancia ya que cuando se asigna un *OnTouchEventListener* a una vista y se hace click sobre ella, se ejecutan dos eventos de touch, el de bajar y el de subir, con lo cual, si se consumen estos eventos, nunca se llamará a la interfaz *OnClickEventListener*. Aspecto que habrá que tener muy en cuenta a la hora de implementar el sistema de barrido.

Los eventos de teclado se gestionan a través de la interfaz *OnKeyListener* y el método *onKey()* que recibe tres parámetros: La vista a la que se delega el evento, un entero que corresponde con el código de la tecla pulsada y una instancia de la clase KeyEvent, que tiene información relativa al evento del teclado como la duración de la pulsación, el tipo de pulsación (pulsar o soltar), etc.

Con esto quedan abarcados todos los aspectos conflictivos en cuanto a la toma de decisión. A modo de resumen, nuestro sistema va a ser implementado como un Servicio que añadirá una vista en la parte más alta de la jerarquía ocupando toda la pantalla. Esta vista será gestionada a través de eventos de ratón, pantalla o teclado. Además generará eventos táctiles (*MotionEvent*) y de teclado (*KeyEvent*) sobre las ventanas que haya debajo.

6.4. Diseño y modelo de clases.

Una vez terminada la fase de formación inicial, en la que hemos analizado y dado respuesta todos aquellos problemas que hemos podido prever en la aplicación. Llega el momento de hacer un diseño que cumpla las características requeridas. La gestión del sistema de barrido, se ha decidido implementarla a través de la clase Service, que controlará una vista con el comportamiento definido en la captura de requisitos. No obstante, nuestro sistema debe poder funcionar de forma autónoma, como Servicio del sistema, o formar parte de una aplicación y ser

usado exclusivamente por ésta. Por todo ello, el diseño ha de respetar todas las funcionalidades de base que la clase Service implementa y así poder mantener la dualidad de uso de esta clase. Se ha propuesto el siguiente diseño:

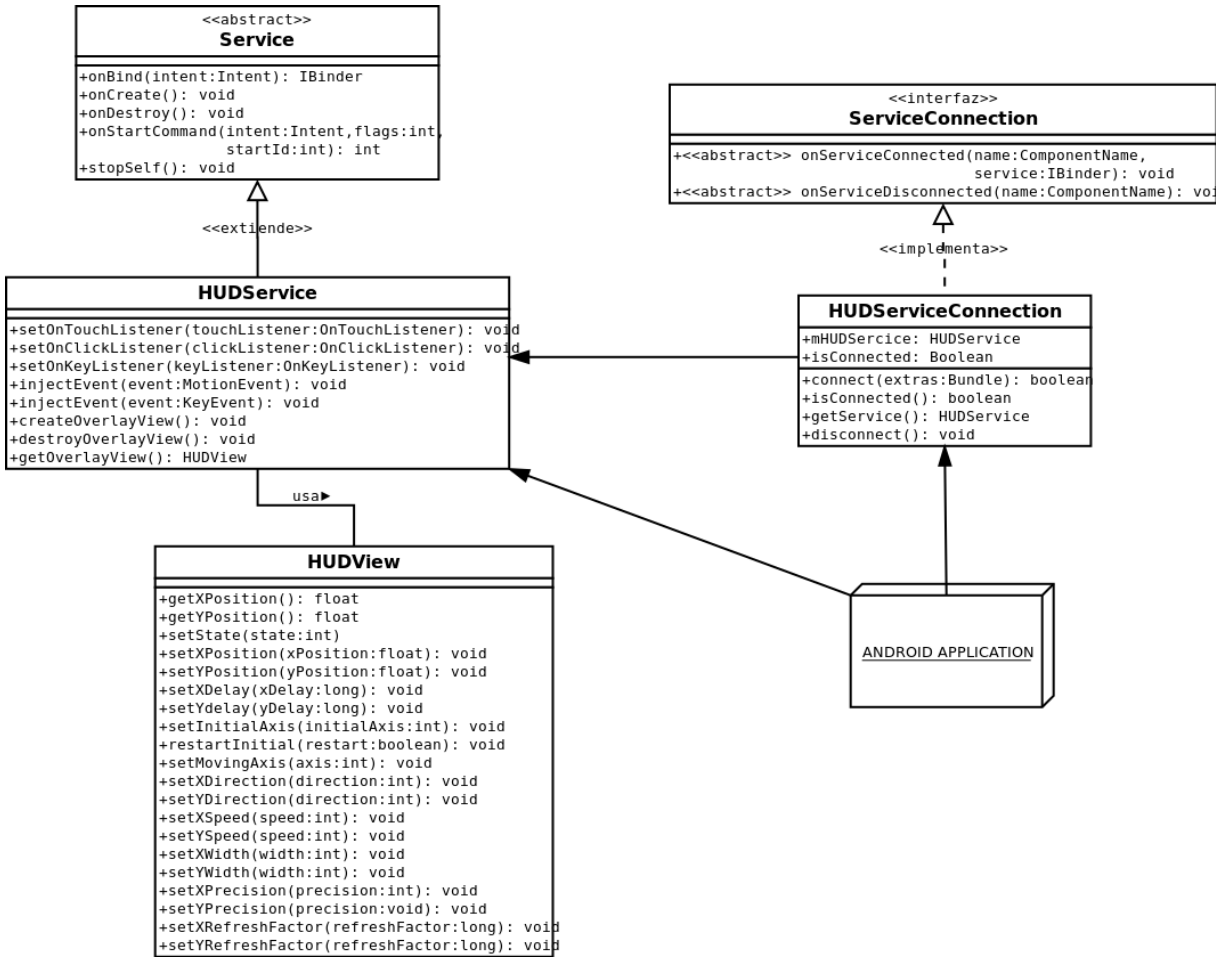


Figura 8: Modelo de clases

Como puede observarse en la *figura 8*, la mayor parte de la funcionalidad relativa a la ejecución nos la proporciona el sistema operativo. La clase *HUDService* ha sido implementada para poder trabajar con ella, tanto siguiendo un modelo cliente-servidor como una tarea en segundo plano.

Tal y como se describe en el capítulo 4, para la ejecución de servicios existen dos caminos posibles. Cuando no se necesita el uso directo del servicio, sino que éste tan solo realiza una tarea asíncrona, se puede ejecutar con el método *startService()*, con el que se le informa al sistema de

que se quiere ejecutar un servicio determinado. El sistema arranca el servicio hasta que cualquier otra aplicación indica explícitamente que quiere detenerlo a través del método *stopService()*.

Otra forma de ejecutar un Servicio, es uniendo éste a una aplicación que lo va a tener en uso a través del método *bindService()*. Android provee la interfaz *ServiceConnection* que se invoca cuando se establece la conexión con el servicio y nos proporciona una interfaz de comunicación llamada *IBinder*. Esta interfaz hace las veces de puente entre el servicio y la conexión, de modo que podemos publicar en ella tantos métodos y funcionalidades como deseemos para interactuar con el servicio, o, como en nuestro caso, hasta la propia instancia del servicio.

El diseño planteado mantiene estas formas de uso al extender las propiedades de Service. Esto permite programar una aplicación que haga uso del Sistema de Barrido según el primer método y luego finalice. El servicio seguirá corriendo hasta que le indiquemos que pare mediante un botón, o con algún gesto sobre la pantalla, la elección del evento a tratar dependerá del desarrollador.

También se puede hacer uso de él desde una aplicación que necesite una comunicación constante con el servicio como una respuesta del estado del sistema, o para calcular las tasas de acierto y error del usuario, etc.

Para facilitar el uso del Sistema de Barrido y abstraer un poco más de la propia implementación de los Servicios de Android, se ha programado la clase *HUDServiceConnection*. Ésta permite arrancar el sistema de forma “desatendida” y pararlo. También simplifica la unión con el servicio gracias a los métodos *connect()*, *isConnected()* y *disconnect()*, con los que se gestiona la conexión, y el método *getService()*, que devuelve una instancia de *HUDService* que es la que se encarga de manejar la vista del barrido y controlar la inyección de eventos.

6.5. Configuración de la velocidad de desplazamiento de los ejes.

Un aspecto de gran relevancia en este diseño es la forma de asignar la velocidad de desplazamiento de los ejes del sistema de barrido. Antes de proceder a explicar que métodos de la clase HUIDView permiten el ajuste de la velocidad, se hace necesario entender con qué frecuencia el sistema operativo refresca la pantalla y cuales son los aspectos que pueden modificar el tiempo de refresco.

La base de funcionamiento del sistema operativo es un bucle repetitivo que realiza comprobaciones y operaciones periódicas sobre sus componentes y herramientas. Una de esas operaciones es la actualización de los elementos de la interfaz de usuario. Cuando un elemento de la interfaz requiere que se actualice su forma de visualización, normalmente por un cambio de estado que invalida su aspecto actual, ejecuta el método *invalidate*(), que avisa al sistema de que ha de volver a calcular sus parámetros visuales. La frecuencia máxima con la que esta operación

puede realizarse es la que va a limitar la velocidad máxima del desplazamiento de los ejes. Es decir, este parámetro está intrínsecamente ligado al rendimiento del conjunto formado por el dispositivo y el sistema operativo.

Desde la empresa Limbika, las comprobaciones que se requirieron fueron para comprobar si los dispositivos que actualmente están en el mercado, son capaces de ejecutar el sistema sin que la velocidad de los ejes se vea afectada, o al menos, que la velocidad máxima fuera suficiente para un usuario con un alto rendimiento en el control del sistema.

Las pruebas realizadas sobre el modelo de tablet *Edison* de la empresa *BQ*, demostraron que el sistema era capaz de desplazarse por toda la pantalla mostrando ambos ejes secuencialmente, primero un eje en ambos sentidos y luego el otro, en menos de un segundo si el sistema no tenía carga y en apenas dos segundos cuando el sistema operativo estaba reproduciendo un vídeo y haciendo un acceso a datos de mucho volumen. Para la Limbika esta prueba fue suficiente para lanzar el producto al mercado con garantías de la velocidad de respuesta del sistema.

No obstante, como se puede deducir, el problema en este tipo de dispositivos de última generación no es la velocidad máxima, sino cómo permitir al desarrollador configurar la velocidad del barrido para cualquier frecuencia de refresco.

Se pensó en un diseño de tres parámetros. Precisión, velocidad y factor de refresco. Por un lado, la precisión con la que se desplaza el eje por la pantalla se define en píxeles y establece el cambio de posición del eje en cada actualización. Otro aspecto es el tiempo de espera para la actualización de la posición. Para ello se definen los otros dos atributos. La velocidad se define de 0 a 100. Siendo 100 la velocidad máxima, una velocidad de 0 implica que el eje permanece estático. El factor de refresco es el tiempo, en milisegundos, que multiplica a una función de la velocidad para calcular el tiempo de espera para refrescar la posición de los ejes. El proceso se ilustra con un algoritmo para una mejor comprensión:

```
Si el eje está en movimiento y la velocidad es mayor que 0.
        posición_del_eje = posición_del_eje + desplazamiento.
        esperar_para_actualizar_posición((100 - velocidad) * factor_de_refresco) milisegundos
```

Como puede observarse, se emplea la velocidad para reducir o aumentar el tiempo de refresco de los ejes. Y gracias al factor de refresco, podemos tener mayor rango de velocidades para una mejor adaptación a cada dispositivo.

En caso de que las características del dispositivo sean muy reducidas, se puede establecer un factor de refresco de 1 milisegundo y la velocidad a 100, que resultaría en una actualización constante de la vista en cada iteración del bucle del sistema operativo. Además, en caso de no ser

suficiente, puede alterarse el factor de precisión para, en detrimento del número de coordenadas susceptibles de ser pulsadas, aumentar la velocidad de desplazamiento de los ejes.

Con este último análisis y la guía para desarrolladores, se da por concluido el desarrollo de la librería. No obstante, sería interesante invertir más tiempo en realizar pruebas de estrés diferentes dispositivo para comprobar la velocidad de respuesta del sistema. Así mismo, falta también por hacer un análisis más exhaustivo del tiempo de respuesta del sistema ante los eventos hardware, quedando este cometido como trabajo futuro del presente proyecto.

7.Trabajo futuro

Pese a que el desarrollo de la librería AccessibilityHUD ha concluido con éxito, el trabajo sobre la misma no se puede dar por terminado. La empresa Limbika, que inicialmente ofertó el proyecto, ya ha integrado esta librería en dos de sus productos estrella, PiktoPlus, que ha sido descrito en la introducción de este proyecto, y PiktoPop, el primer juego educativo adaptado para usuarios con diversidades motrices gracias al sistema de barrido que integra.

No obstante, quedan numerosos aspectos que pueden enriquecer la funcionalidad del sistema de barrido y mejorar su diseño inicial. A partir de la experiencia y los resultados que los productos de Limbika y otras empresas y desarrolladores irán obteniendo, se podrán enfocar esfuerzos en pro de la evolución y mejora de esta herramienta para la accesibilidad. En este capítulo, se hace una reflexión sobre estos aspectos de trabajo futuro para el presente proyecto.

Una de las características más deseables que se pueden requerir de esta librería, es la adaptación de su interfaz visual para que los límites de dibujado de los ejes del barrido ocupen una porción de la pantalla. Esto permitiría a las aplicaciones, por ejemplo, mostrar un menú con componentes controlados con un sistema de barrido.

La configuración de la interfaz del sistema de barrido puede ser modificada con diferentes estilos para los ejes, tanto por motivos estéticos, como por motivos de comprensión cognitiva. Por ejemplo, ofrecer la posibilidad de dibujar una diana en medio de la primera barra, en vez de dos barras, y que sea la diana la que se mueva sobre la primera barra. Para niños con déficit de atención puede ser un refuerzo positivo que les haga entender mejor los elementos con los que están interactuando.

La creación de una aplicación que sea parte del sistema y lance la clase HUDService aumentaría la accesibilidad del sistema operativo Android instalado en el dispositivo del usuario. Además, esto haría posible la configuración del barrido a través de la aplicación de ajustes del sistema.

Un aspecto interesante de implementar, es la inclusión de un botón o una zona en la vista del barrido a modo de escape para poder pararlo o como control modificar algún parámetro del sistema. Aumentaría la potencia de la librería y la haría más adaptable.

Se hace necesario un estudio más exhaustivo sobre la capacidad de respuesta de los diferentes dispositivos a las interrupciones hardware, para comprobar cómo éstas afectan al tiempo de respuesta del sistema de barrido. Así mismo, el calculo de la velocidad máxima a la que puede operar el sistema, también está sujeta a la carga del sistema operativo y a las características hardware del dispositivo sobre el que se ejecuta. Queda como trabajo futuro analizar estos aspectos para poder ajustar más adecuadamente la velocidad del sistema de barrido.

Otras funcionalidades que enriquecen la librería se enumeran a continuación:

- Implementar un *Input Method Editor* que haga uso del sistema de barrido para escribir.

- Implementar una interfaz que permita a las aplicaciones validar o rechazar un evento generado por el servicio.
- Implementar una estrategia de ejecución del barrido integrada en la propia librería que permita adaptar la velocidad de los ejes.

8. Conclusiones.

La aportación de un sistema de barrido a dispositivos Android permite que un mayor número de usuarios puedan interactuar con este sistema operativo, consiguiendo más universalidad con él. Por ello, el sistema propuesto en el presente proyecto permite que desarrolladores de aplicaciones Android integren de forma sencilla esta librería en sus desarrollos para ofrecer a usuarios con diversidades motrices una mayor usabilidad de sus dispositivos.

Así mismo, ofrece a fabricantes de dispositivos la oportunidad de integrar en sus productos este sistema, lo que los revaloriza en el mercado al ampliar el marco de usuarios a los que puede ir destinado.

Al tratarse de una implementación orientada a desarrolladores capaz de integrarse en cualquier aplicación, permite que éstas puedan configurar los parámetros del sistema de barrido en tiempo de ejecución para adaptarlo a las necesidades del usuario en cada momento.

Este sistema de barrido permite que los dispositivos móviles puedan ser usados por un gran número de personas, independientemente de sus conocimientos o capacidades personales y de las características técnicas del equipo utilizado para el manejo de la tecnología. Acerca la tecnología a usuarios con diversidades motrices y les permite hacer uso de todas las capacidades de sus dispositivos. Permitiendo una mayor integración social y acceso a la tecnología.

9. Bibliografía

[1] http://es.wikipedia.org/wiki/Dispositivo_móvil

[2] http://es.wikipedia.org/wiki/Teléfono_inteligente

[3] Luis Galdeazabal Montón, “Aplicaciones de la tecnología de computadores a la mejora de la velocidad de comunicación en sistemas de comunicación aumentativa y alternativa”, pp 6-18

[4] Vanderheiden G.C., “Augmentative Modes ofCommunication for the Severely Speech- and Motor-Imaired”, Clinical Orthopaedics and Related Research, no 148 pp-70-86, 1980.

[5] http://es.wikipedia.org/wiki/Android

[6] http://developer.android.com/guide/guide/topics/manifes/uses-sdk-element.html#ApliLevels

[7] http://es.wikipedia.org/wiki/Android]

[8] http://androideity.com/2011/07/04/arquitectura-de-android/

[9] http://columna80.wordpress.com/2011/02/17/arquitectura-de-android/

[10] http://developer.android.com/index.html

[11] http://developer.android.com/reference/android/app/ActivityManager.html

[12] http://developer.android.com/reference/android/view/WindowManager.html

[13] http://developer.android.com/reference/android/content/ContentProvider.html

[14] http://developer.android.com/reference/android/view/View.html

[15] http://developer.android.com/reference/android/app/NotificationManager.html

[16] http://developer.android.com/reference/android/content/pm/PackageManager.html

[17] http://developer.android.com/reference/android/telephony/TelephonyManager.html

[18] http://developer.android.com/reference/android/content/res/Resources.html

[19] http://developer.android.com/reference/android/location/LocationManager.html

[20] http://developer.android.com/reference/android/hardware/SensorManager.html

[21] http://developer.android.com/reference/android/hardware/Camera.html

[22] http://developer.android.com/reference/android/provider/MediaStore.html

[23] http://developer.android.com/guide/components/activities.html

[24] http://developer.android.com/reference/android/app/Service.html

[25] http://developer.android.com/reference/android/app/Activity.html

[26] http://developer.android.com/guide/components/bound-services.html

[27] http://developer.android.com/guide/components/services.html

[28] http://developer.android.com/reference/android/content/Intent.html

[29] http://developer.android.com/reference/android/content/ContentProvider.html

[30] https://code.google.com/p/eyes-free/

[31] http://developer.android.com/guide/topics/ui/accessibility/apps.html#populate-events

[32] http://www.vodafone.es/fundacion/es/proyecto-accessible/

[33] http://developer.android.com/guide/topics/ui/accessibility/index.html

[34] http://eyes-free.googlecode.com/svn/trunk/documentation/android_access/developers.html

[35] http://android-developers.blogspot.com.es/2009/09/introduction-to-text-to-speech-in.html

[36] http://eyes-free.googlecode.com/svn/trunk/documentation/tutorial/tutorial.html

[37] http://www.aegis-project.eu/

[38] https://play.google.com/store/apps/details?id=com.google.android.marvin.talkback

[39] https://play.google.com/store/apps/details?id=com.google.android.marvin.kickback&hl=es

[40] https://play.google.com/store/apps/details?id=com.google.android.marvin.soundback&hl=es

[41] http://mobile-accessibility.idrc.ocad.ca/projects/tekla/demos

[42] http://www.inference.phy.cam.ac.uk/dasher/

[43] https://play.google.com/store/apps/details?id=com.googlecode.eyesfree.inputmethod.latin&hl=es

[44] https://play.google.com/store/apps/details?id=com.googlecode.eyesfree.brailleback

[45] http://www.inteco.es/Accesibilidad/difusion/Manuales_y_Guias/guia_moviles/

[46] http://www.tuandroid.com/nuevos-datos-sobre-la-fragmentacion-en-android/

ANEXO A - Guía para implementar aplicaciones Accesibles sobre el sistema operativo Android

I. Introducción

El objetivo de esta guía es hacer un resumen de los aspectos que nuestra aplicación debe cumplir para garantizar la accesibilidad de la misma. En ella, se hace un repaso de las herramientas proporcionadas por el SDK de Android en sus diferentes niveles de API. Cabe destacar que en niveles bajos del API, la mayor parte de la responsabilidad para la accesibilidad recae sobre el desarrollador, que deberá adecuar su aplicación a los requerimientos del sistema.

II. Accesibilidad en Android

Las aplicaciones desarrolladas para Android son más accesibles para usuarios con limitaciones visuales, físicas o relacionadas con la edad cuando éstos activan los servicios y funcionalidades de accesibilidad en sus dispositivos, sin necesidad de cambio alguno en el código de la aplicación para adecuarla a sus necesidades. Sin embargo, es muy conveniente completar una serie de pasos para optimizar la accesibilidad de la aplicación y asegurar una experiencia agradable al usuario.

Asegurarse de que nuestra aplicación es accesible para todos los usuarios requiere tan sólo de unos pocos pasos, particularmente cuando creamos las interfaces de usuario con componentes proporcionados por el *framework* de Android. Si usamos exclusivamente los componentes estándar, los pasos a seguir serían los siguientes:

- Añadir textos descriptivos a las interfaces de control de la aplicación usando el atributo "`android:contentDescription`". En especial en los elementos de interfaz *ImageButton*, *ImageView* y *CheckBox*.
- Asegurarse de que todos los elementos de la interfaz que aceptan un método de entrada (touch o click) pueden ser seleccionados mediante un control direccional como un trackball, D-pad, o gestos de navegación.
- Estar seguros de que los mensajes de audio van siempre acompañados por otro indicador visual o notificación, para ayudar a los usuarios sordos o con problemas de audición.
- Probar la aplicación utilizando únicamente los servicios de navegación y características de accesibilidad. Activar el servicio TalkBack y explorar con el tacto, y luego tratar de usar la aplicación utilizando sólo los controles direccionales.
- En caso de implementar nuestros propios controles personalizados que amplían la clase View, se debe realizar un trabajo adicional que asegure que sus componentes son accesibles. A continuación se describe el procedimiento a seguir para desarrollar componentes personalizados compatibles con los servicios de accesibilidad de Android.

III. Etiquetando los elementos de la interfaz.

Muchos controles de la interfaz de usuario dependen de señales visuales que indican su significado y uso. Por ejemplo, una aplicación para la toma de notas puede usar un *ImageButton* con una foto de un signo '+' para indicar al usuario que puede añadir una nueva nota. Un componente *EditText* puede tener una etiqueta que informa acerca de su propósito. Un usuario con problemas de visión no puede apreciar estas señales lo suficientemente claro como para seguirlo, lo que los hace inútiles.

Estos controles se pueden hacer más accesibles utilizando el atributo "android:`contentDescription`" en la definición del control. El texto especificado en este atributo no aparece en la pantalla, pero es leído por los servicios de accesibilidad que proporcionan mensajes auditivos. Este atributo está soportado por todos los componentes estándar de Android, ya que es propio de la clase *View*, de la que extienden todos los elementos visuales de Android. No obstante, en el caso de los *EditText*, controles que sirven para la introducción de texto, se proporciona el atributo "`android:hint`" al que se le asigna el texto que se desea que aparezca en el cuadro de texto cuando está vacío. Este texto será leído por los servicios de accesibilidad como *TalkBack*.

Android permite la navegación por los diferentes controles de la aplicación usando un controlador direccional, ya sea físico, como un trackball, un pad direccional (D-pad) o las techas de flecha, o virtual, como el *Eyes-Free Keyboard* o la navegación mediante gestos disponible desde Android 4.1. Todos los controladores direccionales permiten al usuario navegar por los diferentes elementos de la interfaz.

IV. Habilitando la selección de elementos en la interfaz.

Los elementos de la interfaz de usuario son accesibles mediante los controles direccionales cuando se establece el atributo "`android:focusable`" a verdadero. Esto permite a Android interpretar que ese elemento puede ser seleccionado, o, para aproximarnos a la semántica del atributo, enfocado. Todos los componentes visuales estándar que proporciona Android tienen dicho atributo por defecto.

Android proporciona varias APIs que permiten controlar si un elemento de la interfaz de usuario está enfocado, e incluso provocar su enfoque mediante los siguientes métodos propios de la clase *View*:

setFocusable()
isFocusable()
requestFocus()

Si una vista no es enfocable por defecto, se le puede dar esta propiedad desde el archivo de diseño (en el directorio layout del proyecto) estableciendo el atributo "`android: focusable`" a verdadero o llamando al método *setFocusable*().

IV.I. Control del orden del foco.

Cuando el usuario navega en cualquier dirección utilizando los controles de dirección, el foco pasa desde un elemento de la interfaz de usuario (*View*) a otro, según esté determinado el orden del foco. Este orden se basa en un algoritmo que encuentra el vecino más cercano en una dirección dada. En algunos casos, puede que el orden calculado por el algoritmo no sea el que queremos para nuestra interfaz, o puede no ser lógico para los usuarios. En estas situaciones, se pueden proporcionar órdenes alternativas usando los siguientes atributos XML en el archivo de diseño:

`android:nextFocusDown`

Define el siguiente elemento que recibirá el foco cuando el usuario navegue hacia abajo.

`android:nextFocusLeft`

Define el siguiente elemento que recibirá el foco cuando el usuario navegue hacia la izquierda.

`android:nextFocusRight`

Define el siguiente elemento que recibirá el foco cuando el usuario navegue hacia la derecha.

`android:nextFocusUp`

Define el siguiente elemento que recibirá el foco cuando el usuario navegue hacia arriba.

El siguiente ejemplo, muestra dos elementos de la interfaz de usuario donde los atributos "`android:nextFocusDown`" y "`android:nextFocusUp`" han sido proporcionados explícitamente. El *TextView* se encuentra a la derecha del *EditText*. Sin embargo, al haberse establecido las propiedades mencionadas anteriormente, el elemento *TextView* puede ahora ser enfocado pulsando la flecha hacia abajo cuando el foco está en el elemento *EditText*.

```
 <LinearLayout android:orientation="horizontal"
 ... >
<EditText android:id="@+id/edit"
android:nextFocusDown=”@+id/text”
... />
<TextView android:id="@+id/text"
android:focusable=”true”
android:text="Hello, I am a focusable TextView"
android:nextFocusUp=”@id/edit”
... />
 </LinearLayout>
```

Cuando modificamos el orden del foco, es necesario asegurarse de que la navegación funciona como esperamos en todas las direcciones posibles para cada control de la interfaz de usuario.

Se puede modificar el orden del foco de los componentes de la interfaz de usuario en tiempo de ejecución, usando los métodos *setNextFocusDownId*(), *setNextFocusRightId*(), *setNextFocusUpId*() y *setNextFocusLeftId*().

V. Construyendo vistas personalizadas de forma accesible.

Si nuestra aplicación requiere la implementación de componentes personalizados, elementos visuales que extienden las funcionalidades de los componentes del sistema Android, se necesita hacer un trabajo adicional para asegurarse de que estos son accesibles. A continuación se detallan aspectos que garantizan la accesibilidad de los componentes personalizados:

- Controlar los eventos de las teclas direccionales.
- Implementar los métodos de la API para la accesibilidad.
- Rellenar los eventos de accesibilidad y la información para accesibilidad de la vista.

V.I. Control de eventos de las teclas direccionales

En la mayoría de los dispositivos, la pulsación de una tecla usando los controles direccionales, envía un evento *KeyEvent* sobre la vista que tiene el foco en ese momento. Todas las vistas estándar de Android manejan estos eventos apropiadamente. Cuando construimos componentes personalizados, hay que asegurarse que tengan el mismo efecto que al tocar la pantalla. Los códigos de las teclas que, al menos, se aconseja controlar son los siguientes:

KEYCODE_DPAD_LEFT - flecha izquierda -
KEYCODE_DPAD_RIGHT - flecha derecha -
KEYCODE_DPAD_UP - flecha arriba -
KEYCODE_DPAD_DOWN - flecha abajo -
KEYCODE_DPAD_CENTER - En controles direccionales (D-Pad) botón ok, equivale a enter-
KEYCODE_ENTER - tecla enter-
KEYCODE_BACK - En D-pads botón cancelar, equivale a escape -
KEYCODE_ESC - tecla escape-

V.II. Implementando los métodos de accesibilidad proporcionados por la API de Android.

Los eventos de accesibilidad son mensajes acerca de la interacción del usuario con los componentes de la interfaz de nuestra aplicación. Estos mensajes son controlados por Servicios de Accesibilidad, que usan esa información de los eventos para producir respuestas suplementarias y avisos. A partir de la versión 4.0 de la plataforma Android, los métodos para generar eventos de accesibilidad se han ampliado proporcionando información más detallada que la interfaz *AccessibilityEventSource* introducida en Android 1.6. Estos nuevos métodos son parte de la clase *View* así como de la clase *View.AccessibilityDelegate*. Se enumeran a continuación:

- *sendAccessibilityEvent*() (API nivel 4)

A este método se le llama cuando un usuario realiza una acción sobre una vista. El evento es clasificado con el tipo de acción del usuario, como, por ejemplo TYPE_VIEW_CLICKED. Como se ha mencionado anteriormente, no es necesario implementar este método a no ser que estemos creando nuestras propias vistas.

- *sendAccessibilityEventUnchecked*() (API nivel 4)

Este método se usa en contextos de código en los que se ha comprobado si la accesibilidad está activada en el dispositivo. Al implementar este método, debemos hacerlo como si el modo accesibilidad estuviera habilitado, independientemente de la configuración del sistema.

- *dispatchPopulateAccessibilityEvent*() (API nivel 4)

El sistema llama a este método cuando una vista personalizada genera un evento de accesibilidad. Al igual que en niveles más altos del API, la implementación por defecto de este método llama a *onPopulateAccessibilityEvent*() de esa vista y luego a *dispatchPopulateAccessibilityEvent*() para cada uno de los hijos de la misma. Con el fin de soportar servicios de accesibilidad de revisiones de Android anteriores a la 4.0, se debe sobreescribir este método, así como el método *getText*() para hacer que devuelva el texto descriptivo para nuestra vista personalizada, que es el método que usan los servicios de accesibilidad como TalkBack.

- *onPopulateAccessibilityEvent*() (API nivel 14)

Este método establece el texto de aviso que será reproducido por el evento de accesibilidad de nuestra vista. También es invocado si esta vista es hija de otra que genera un evento de accesibilidad. Pese a que este método permite modificar diversos atributos del evento de accesibilidad, es aconsejable limitar las modificaciones al contenido del texto y usar el método *onInitializeAccessibilityEvent*() para alterar otras propiedades del evento.

- *onInitializeAccessibilityEvent*() (API Level 14)

El sistema llama a este método para obtener información adicional acerca del estado de la vista, además del texto descriptivo. Si nuestros componentes personalizados proporcionan un control interactivo más allá de un simple texto o botón, se debe sobrescribir este método y establecer toda la información adicional acerca del mismo.

- *onInitializeAccessibilityNodeInfo*() (API nivel 14)

Este método proporciona servicios de accesibilidad con información acerca del estado de una vista. La implementación por defecto proporciona un conjunto estándar de propiedades. En caso de que nuestra vista personalizada necesite ampliar este conjunto, se debe sobrescribir este método y añadir la información adicional en una clase del tipo *AccessibilityNodeInfo*.

- onRequestSendAccessibilityEvent() (API nivel14)

El sistema llama a este método cuando el hijo de una vista genera un evento de accesibilidad. Este paso permite al padre completar el evento con información adicional.

Para poder soportar estos métodos de accesibilidad de Android en nuestros propios componentes, se debe seguir una de las siguientes opciones:

- Si la aplicación se implementa para la versión 4.0 de la plataforma Android (API nivel 14) o superior, se deben sobrescribir e implementar los métodos descritos anteriormente.
- Si nuestra vista personalizada requiere compatibilidad con niveles de API 4 en adelante, se debe añadir la biblioteca de soporte en su revisión 5 o superior. Estas bibliotecas fueron desarrolladas con el objetivo de mantener compatibilidad con versiones anteriores de Android.

En ambos casos se deben implementar los siguientes métodos de accesibilidad para vistas personalizadas:

dispatchPopulateAccessibilityEvent()
onPopulateAccessibilityEvent()
onInitializeAccessibilityEvent()
onInitializeAccessibilityNodeInfo()

La clase View proporciona implementaciones por defecto para los siguientes tipos de eventos de accesibilidad:

API nivel 4:

TYPE_VIEW_CLICKED: Evento de click. Puede ser generado con un ratón o tocando la pantalla.

TYPE_VIEW_LONG_CLICKED: Evento de click con un tiempo mínimo entre la pulsación y la liberación.

TYPE_VIEW_FOCUSED: Evento de adquisición de foco. Generado por el sistema cuando se selecciona un elemento de la interfaz.

API nivel 14:

TYPE_VIEW_SCROLLED: Evento de desplazamiento de una vista. Propio de elementos de la interfaz cuyo contenido no puede ser mostrado completamente y se proporciona un scroll lateral para desplazar el contenido. Este scroll puede ser horizontal o vertical.

TYPE_VIEW_HOVER_ENTER: Evento ampliación o zoom de un componente de la interfaz. Ciertas ayudas técnicas, que permiten la navegación direccional por los elementos de la interfaz, proporcionan un feedback del componente seleccionado aumentando su tamaño en un plano por encima del resto de la interfaz.

TYPE_VIEW_HOVER_EXIT: Evento contrario al anterior, generado al restaurar el componente a su tamaño y plano original.

V.III. Completando eventos de accesibilidad

Cada evento de accesibilidad tiene un conjunto de propiedades requeridas que describen el estado actual de la vista. Estas propiedades incluyen aspectos como el nombre de la clase de la vista, la descripción del contenido y estado de activación. La implementación *View* proporciona valores predeterminados para estas propiedades. Muchos de estos valores, incluyendo el nombre de la clase y la hora del evento, se proporcionan automáticamente. Si se va a crear un componente personalizado, se debe proporcionar alguna información sobre el contenido y las características de la vista. Esta información puede ser tan simple como una etiqueta de botón, pero también puede incluir información de estado adicional que se desee agregar al evento.

El requisito mínimo para el suministro de información a los servicios de accesibilidad con una vista personalizada es implementar *dispatchPopulateAccessibilityEvent*(). A este método lo llama el sistema para solicitar información de un evento de accesibilidad y hace que nuestra vista personalizada sea compatible con los servicios de accesibilidad de Android 1.6 (API Nivel 4) y superior. El siguiente código de ejemplo ha sido extraído de la documentación de Android y muestra una implementación básica de este método.

```
 @Override
 public void dispatchPopulateAccessibilityEvent
  (AccessibilityEvent event) {
super.dispatchPopulateAccessibilityEvent(event);
// Call the super implementation to populate
// its text to the event, which
// calls onPopulateAccessibilityEvent() on API Level 14 and up.

// In case this is running on
// a API revision earlier that 14, check
// the text content of the event and add an appropriate text
// description for this custom view:
CharSequence text = getText();
if (!TextUtils.isEmpty(text)) {
     event.getText().add(text);
}
 }
```

Para Android 4.0 (API de nivel 14) y superior, se debe utilizar *onPopulateAccessibilityEvent*() y *onInitializeAccessibilityEvent*() para rellenar o modificar la información en un evento de accesibilidad. El método *onPopulateAccessibilityEvent* () se usa para agregar o modificar el contenido de texto del evento, que se convierte en avisos audibles por los servicios de accesibilidad como *TalkBack*. Mientras que el método *onInitializeAccessibilityEvent* () se utiliza para rellenar la información adicional sobre el evento, como el estado de la selección de la vista.

Además, es necesario implementar el método *onInitializeAccessibilityNodeInfo*(). Los objetos *AccessibilityNodeInfo* devueltos por este método son utilizados por los servicios de accesibilidad para investigar la jerarquía de vistas que genera un evento de acceso después de recibirlo, para obtener la información de contexto más detallada y proporcionar información adecuada a los usuarios.

El código de ejemplo siguiente muestra cómo sobrescribir estos tres métodos a través de la biblioteca de compatibilidad en su versión 5 o superior mediante el método de accesibilidad V*iewCompat.setAccessibilityDelegate*().

```
 ViewCompat.setAccessibilityDelegate(
                  new AccessibilityDelegateCompat() {
@Override
public void onPopulateAccessibilityEvent(
                      View host, AccessibilityEvent event) {
     super.onPopulateAccessibilityEvent(host, event);
```

```
        // We call the super implementation to populate its text
        //for the event. Then we add our text not present in a
        //super class. Very often you only need to add the text
        //for the custom view.
        CharSequence text = getText();
        if (!TextUtils.isEmpty(text)) {
            event.getText().add(text);
        }
    }

    @Override
    public void onInitializeAccessibilityEvent(
                        View host, AccessibilityEvent event) {
        super.onInitializeAccessibilityEvent(host, event);
        // We call the super implementation to let super classes
        // set appropriate event properties. Then we add the
        //new property (checked) which is not supported by
        //a super class.
        event.setChecked(isChecked());
    }

    @Override
    public void onInitializeAccessibilityNodeInfo(
                    View host, AccessibilityNodeInfoCompat info) {
        super.onInitializeAccessibilityNodeInfo(host, info);
        // We call the super implementation to let super classes set
        // appropriate info properties. Then we add our properties
        // (checkable and checked) which are not supported by
        //a super class.
        info.setCheckable(true);
        info.setChecked(isChecked());
        // Quite often you only need to add the text for
        //the custom view.
        CharSequence text = getText();
        if (!TextUtils.isEmpty(text)) {
            info.setText(text);
        }
    }
     }
```

Para aplicaciones compiladas sobre Android 4.0 o superior, estos métodos se implementan directamente sobre la clase del componente propio.

ANEXO B - Manual de uso para desarrolladores Android de la librería AccessibilityHUD.

1. Introducción.

La presente guía pretende ilustrar al desarrollador en el uso e integración de la librería AccessibilityHUD en aplicaciones Android de un sistema de barrido. Este sistema facilita a usuarios con diversidades motrices el uso de la aplicación aumentando la universalidad de la misma.

El sistema de barrido publicado en esta librería consiste en el dibujado sobre la interfaz de usuario de dos ejes perpendiculares, uno horizontal y otro vertical, que se desplazan sobre la pantalla permitiendo al usuario detenerlas para así generar un evento con las coordenadas de la intersección de las mismas.

El desarrollador puede asignar el tipo de evento generado sobre la interfaz de usuario. También permite configurar el tipo de interrupción con el que se controla la detención de los ejes. La librería publica un API para la configuración de diversos parámetros de los ejes como su velocidad, anchura y aspecto de los ejes, el eje con el que se inicia el barrido, y otros aspectos que se detallan a lo largo del presente manual.

Esta guía está orientada a aplicaciones desarrolladas sobre la plataforma Eclipse usando el plugin ADT (*Android Development Tool*) de Android. Se requiere por parte del desarrollador un amplio conocimiento del *framework* de Android y, más concretamente, sobre el uso de servicios y diferentes componentes proporcionados. Se entiende que las siguientes clases y sus diferentes funcionalidades son conocidas por el lector y se remite a la documentación oficial de Android para su entendimiento Intent ServiceActivity, bundle, ServiceConnection, Ibinder, AndroidManifes, Context, View y todas las interfaces de control de eventos que publica como OnClickListener, OnTouchListener y OnKeyListener. Al final del presente manual se ofrece bibliografía sobre estas clases.

2. Integración de la librería en proyectos de desarrollo de aplicaciones Android.

Para poder trabajar con esta librería y hacer uso de las funcionalidades que proporciona, es necesaria la inclusión de la misma en las aplicaciones. La librería está compilada en formato .jar, que es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java, lo que permite su integración en aplicaciones Android de forma sencilla. Para importar la librería, hay que

copiar el fichero AccessibilityHUD.jar en el directorio /libs del proyecto Android y para hacer uso de ella desde el código hay que modificar la variable BuildPath del entorno de desarrollo Eclipse. Desde el entorno de desarrollo seleccionar el proyecto e ir a “project -> properties”, en la ventana que aparece elegir la opción “Java Build Path” y hacer click sobre el botón “Add JARs...”. Desplegar el directorio del proyecto y luego el directorio /libs. Hacer click sobre el fichero AccessibilityHUD.jar, previamente copiado, y aceptar.

Con esto queda la librería incluida en el proyecto y el desarrollador tendrá acceso al API de la misma desde cualquier punto del código de la aplicación.

3. Funcionamiento general del sistema.

La librería AccessibilityHUD proporciona un servicio que integra una interfaz visual en forma de dos ejes cartesianos que se desplazan por la pantalla a modo de barrido para accesibilidad.

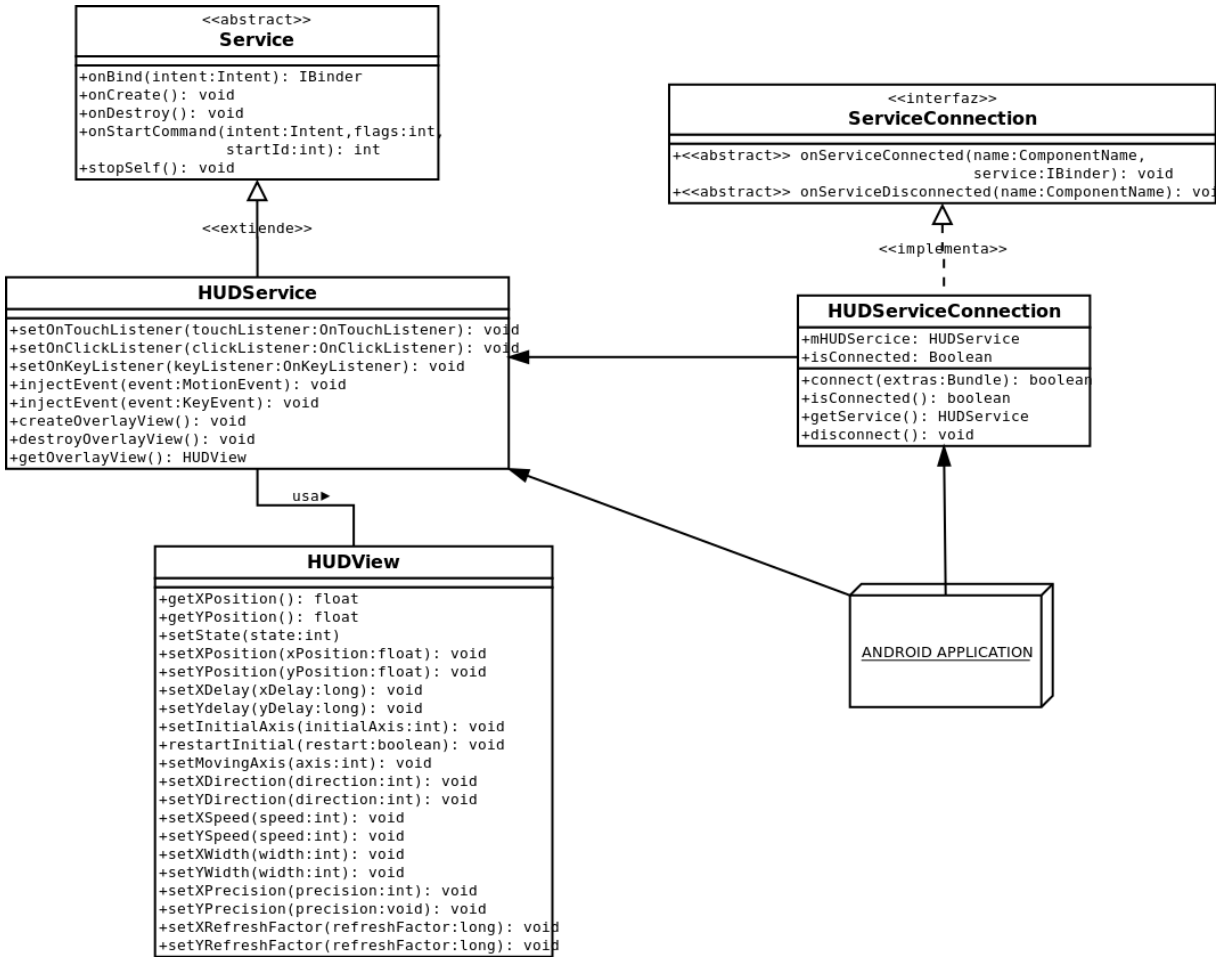
Inicialmente, el sistema dibuja, sobre la interfaz de usuario, uno de los ejes que comienza su desplazamiento sobre la pantalla y espera a que el usuario lo detenga, por ejemplo, con un pulsador. Tras la detención del primer eje, se muestra el segundo eje perpendicular con un comportamiento análogo. Cuando el usuario detiene el segundo eje, el sistema elimina su componente visual para generar un evento sobre la interfaz de usuario.

Este sistema permite diversas configuraciones tanto para el comportamiento de los ejes (velocidad, posición inicial, tiempo de espera) como para la gestión de eventos.

El control del barrido se realiza a través de un servicio que mantiene la estructura y funcionalidades propias de la clase Service de Android. Ello nos permite lanzarlo de forma desatendida asignándole tan solo una configuración inicial a través del *Intent* que lo ejecuta. También permite mantener una comunicación con él para modificar su comportamiento en tiempo de ejecución o para consultar su estado haciendo uso de la ejecución del servicio en modo cliente-servidor a través de la interfaz IBinder.

4. Modelo de clases.

En este apartado se hace una descripción de las clases que componen la librería AccessibilityHUD con el objetivo de poner en contexto al desarrollador para su mayor comprensión de la estructura de la misma. Posteriormente se va a proceder a explicar de forma más extendida cada una de las clases y su cometido en el modelo. A continuación se muestra el modelo de clases de la librería.



Modelo de clases de la librería AccessigilityHUD

4.1. HUDVIew.

Esta clase se encargada de la gestión de la interfaz visual del sistema de barrido y es la responsable de dibujar los ejes y desplazarlos por la pantalla. Provee un API para la configuración de diversos parámetros que especifican el comportamiento de los ejes y su desplazamiento. A continuación se hace una descripción detallada de cada uno de sus métodos:

NOTA: La configuración de algunos parámetros de esta clase se realiza tanto para el eje horizontal (eje X) como para el vertical (eje Y). La documentación de un mismo parámetro para cada uno de los ejes es análoga. Con el fin de no duplicar información, se proporciona una documentación genérica para cada par de métodos que realizan la misma función pero en ejes distintos. Los métodos que no se aplican a cada eje por separado, se documentan de forma específica.

- void setInitialXPosition (int initialXPosition)
- void setInitialYPosition (int initialYPosition);

Establece la posición inicial que va a tener el eje. El sistema comprueba si el parámetro recibido está fuera de los límites de la pantalla, en cuyo caso, la posición inicial será 0. Esta será la posición sobre la que se dibuje el eje para empezar su desplazamiento cuando el atributo booleano *restartInitial* sea *true*, lo que significa que después de generar un evento sobre la interfaz de usuario, se volverá a dibujar el sistema según la configuración inicial. Para más información ver documentación del método *setRestartInitial(boolean restart)*.

- void setInitialXDelay (long delayInMilis)
- void setInitialYDelay (long delayInMilis);

Establece el tiempo en milisegundos que el eje está detenido desde que se dibuja sobre la interfaz, hasta que comienza a desplazarse.

- void setXSpeed (int xSpeed);
- void setYSpeed (int ySpeed);

El eje se desplaza un número determinado de píxeles sobre la pantalla en función de este atributo y de la velocidad de actualización de la pantalla. El número de píxeles de desplazamiento del eje se define con el método setPrecision(int pixels). El rango de valores para la velocidad es de 0 a 100, siendo 100 la velocidad máxima que se traduce en la frecuencia máxima con la que el el sistema operativo puede redibujar la pantalla. Una velocidad intermedia, por ejemplo 50, establece un tiempo de espera de (100 - 50)*REFRESH_FACTOR para desplazar el eje en función de la precisión. El atributo REFRESH_FACTOR se establece a través del método setRefreshFactor (long milis).

Para una velocidad de 0, el eje permanece estático. Esto se traduce en que la velocidad mínima ajustable desde este método es 1.

El desarrollador a de prestar especial cuidado en la configuración de los atributos REFRESH_FACTOR, PRECISION Y SPEED para el correcto establecimiento de la velocidad de desplazamiento de los ejes.

- void setXWidth (int widthInPixels)
- void setYWidth (int widthInPixels);

Define la anchura visual que tendrán los ejes. Siendo la anchura máxima permitida la mitad del ancho o alto de la pantalla.

- void setXDirection (int xDirection)
- void setYDirection (int yDirection);

Establece el sentido de movimiento del eje al comenzar a desplazarse. Los valores permitidos son 1, para un desplazamiento hacia abajo (eje x) o hacia la derecha (eje Y), o -1, para un desplazamiento hacia arriba (eje x) o hacia la izquierda (eje y).

- void setRestartInitial (boolean restart);

Este parámetro establece el comportamiento visual del sistema después de generar un evento sobre la interfaz de usuario, es decir, cuando se vuelve a dibujar sobre la pantalla el primer eje, dónde se va a posicionar, en la posición en la que se quedó al ser detenido por el usuario, o en la posición que se estableció en la configuración inicial a través del método setIntialPosition() del eje determinado. Para un valor *true*, el sistema vuelve a la configuración inicial, en caso contrario, redibuja los ejes en la posición que quedaron al ser detenidos por el usuario.

- void setInitialAxis (int intialAxis);

Establece cual va a ser el primer eje en dibujarse al inicio del sistema o tras reiniciarlo al generar un evento. Un valor de 1 establece el eje X como inicial, mientras que un valor de 2 hace que el sistema comience con el eje Y.

4.2. HUDServiceConnection

Esta clase se ha desarrollado con el objetivo de abstraer al desarrollador de la conexión con el servicio en modo cliente-servidor. Proporciona métodos para conectarse y desconectarse del servicio así como para acceder a la instancia de la clase HUDService en ejecución.

Sus métodos son los siguientes:

- void connect (Bundle extras);

Realiza una llamada al método Context.bindService() para establecer una conexión con el servicio. El atributo *extras* se utiliza para enviar parámetros de configuración al HUDService a través del Intent que recibe como parámetro el método bindService(), para más información sobre la conexión de un servicio a través de este método acudir a la documentación oficial de Android.

- void disconnect()

Detiene el servicio HUDService y rompe la conexión establecida previamente.

- boolean isConnected()

Devuelve true en caso de que ya exista una conexión con el servicio. False en caso contrario

- HUDService getHUDService()

Devuelve una instancia de la clase HUDService en caso de que exista una conexión con éste y null en caso contrario.

- void setOnConnectionListener (OnConnectionListener connectionListener);

Asigna una interfaz que define el método onConnection() que será lanzado cuando la conexión sea establecida.

4.3. HUDService

Esta clase controla la interfaz del barrido y proporciona métodos para la gestión de eventos. Cuando se ejecuta, se instancia la clase HUDView con los parámetros enviados a través del Intent que ejecutó el servicio.

- HUDView getOverlayView()

Devuelve una instancia de la clase HUDView. Este método se proporciona con la intención de asignar los parámetros de ejecución a la interfaz de los ejes del barrido

- void createOverlayView();

Es el método encargado de iniciar la interfaz visual del servicio. Tras la ejecución de este método, el sistema crea una nueva instancia de la clase HUDView y la añade a la ventana actual por encima de la interfaz de usuario. Esto provoca que se inicie el barrido de los ejes.

- void destroyOverlayView();

La invocación de este método provoca que se elimine de la pantalla la interfaz visual del barrido. Puede ser llamado tanto por la aplicación que ejecuta el servicio como por la propia clase HUDService para generar el evento deseado sobre la interfaz de usuario.

- void setOnClickListener(OnClickListener clickListener);

Asigna un controlador a la clase HUDView que responde a los eventos de click.

- void setOnTouchListener(OnTouchListener touchListener);

Asigna un controlador a la clase HUDView que responde a los eventos táctiles sobre la pantalla.

- void setOnKeyListener(OnKeyListener keyListener);

Asigna un controlador a la clase HUDView que responde a los eventos de teclado.

- void injectKeyEvent (KeyEvent keyEvent);

Genera el evento de teclado recibido como parámetro a la interfaz sobre la que se está ejecutando el barrido.

- void injectMotionEvent (MotionEvent motionEvent);

Genera el evento recibido como parámetro a la interfaz sobre la que se está ejecutando el barrido. Previo al envío del evento, se asignan a éste las coordenadas de intersección de los dos ejes del barrido.

5. Ejemplo de uso del sistema de barrido.

A continuación se muestran algunos fragmentos de código que, a modo de ejemplo, pueden ilustrar al desarrollador sobre cómo lanzar el sistema sobre sus aplicaciones.

Lo primero que el desarrollador ha de tener en cuenta es que debe añadir la especificación del servicio en el fichero AndroidManifes.xml de la aplicación, al igual que se hace con el resto de las Activities y Services de las que se hace uso durante la ejecución de la misma.

Para la ejecución del servicio en modo cliente-servidor, hay que instanciar la clase HUDServiceConnection. Esta nos proporciona el método connect() que es el encargado de lanzar el servicio y conectarse con el:

```
HUDService service = null;
HUDServiceConnection connection = new HUDServiceConnection();

connection.setOnConnectionListener = new
HUDServiceConnection.OnConnectionListener () {

@override
public void onConection () {
      HUDService = connection.getHUDService();
      //A partir de aquí ya se tiene una instancia
      //del servicio de barrido para poder asignarle los
      //parámetros de comportamiento deseados.
}
 };
```

Para la ejecución del sistema de barrido sin conexión directa con él, es necesario hacerlo desde una instancia de la clase Activity. Los pasos serían los siguientes:

```
Intent intent = new Intent ( this.getApplicationContext(),
                              HUDService.class );
this.startService (intent);

…//El sistema de barrido
…//está en ejecución

stopService(intent)    //Se detiene el barrido
```

La última línea de código del ejemplo anterior detiene el sistema de barrido, no obstante, no es necesario hacerlo desde la Activity que lo lanza, podría ser otra aplicación la que lo detenga, o la misma pero en otra Activity.

6. Documentación recomendada

http://developer.android.com/reference/android/content/Intent.html

http://developer.android.com/reference/android/app/Service.html

http://developer.android.com/reference/android/app/Activity.html

http://developer.android.com/reference/android/os/Bundle.html

http://developer.android.com/reference/android/content/ServiceConnection.html

http://developer.android.com/reference/android/os/IBinder.html

http://developer.android.com/guide/topics/manifest/manifest-intro.html

http://developer.android.com/reference/android/content/Context.html

http://developer.android.com/reference/android/view/View.html

http://es.wikipedia.org/wiki/Java_Archive

ANEXO C – Evolución de la accesibilidad en Android

Versión	Nueva funcionalidad	Comentario
Android 1.5 (Cupcake)	Se incluyó la API TTS	Se incluye el primer motor de síntesis de Texto a Voz, pero solo se puede utilizar el propio de Android (picoTTS).
Android 1.6 (Donuts)	Se incluyo la API de Accesibilidad	Permite la aparición de los primeros lectores de pantalla para Android como Talkback y Spiel.
Android 2.0-2.1 (Eclair)	Mejoras en la API de Accesibilidad	Permite por primera vez la conexión de métodos de entrada de texto externos.
Android 2.2 (FroYo)	Mejoras en la entrada de voz y en TTS	Permite cambiar el motor de síntesis de voz a otro que no sea el nativo de Android.
Android 2.3 (Gingerbread)	Calendario de android accesible. Eyes-free Keyboard, da por primera vez una opción accesible de introducir texto en Android	Aunque no cambie la API de Accesibilidad, mejora su implementación permitiendo utilizar características que previamente no se podían. Como acceder a la información previa en un cuadro de edición de texto mediante un Servicio de Accesibilidad.
Android 3.0 (Honeycomp)	Incluye web scripts para WebView	La funcionalidad de la navegación física sigue siendo necesaria. El WebView es tratado por el sistema como una unidad impidiendo acceder a los elementos definidos en el contenido del sitio web visualizado.
Android 4.0 (Ice Cream Sandwich)	Incluye el tamaño de fuente ajustable y la exploración táctil (Explore-by-Touch)	El TTS cambia automáticamente cuando un nuevo idioma es seleccionado. El teclado propio del teléfono es accesible mediante la exploración táctil (Ya no es necesario Eyes-free keyboard)

Android 4.1 (Jelly Beam)	Incorpora la navegación mediante gestos (Explore-by-Touch gestures). Se introduce BraileBack, permitiendo leer y escribir en braile con un dispositivo externo.	El Talkback incorpora el Soundback y Kickback. Las notificaciones son configurables, pudiendo ampliar el tamaño del texto, imagen, etc.
Android 4.2 (Jelly Beam)	Incorpora el magnificador de pantalla.	Permite ampliar cualquier contenido de la pantalla.

ANEXO D – Criterios de validación de la accesibilidad para dispositivos móviles.

En la primera columna se describe la característica requerida para garantizar la accesibilidad que ha de validarse en la aplicación.
La segunda columna se asocia cada caso a la discapacidad a la que está orientado.
En la tercera columna se describe la característica de accesibilidad.
La cuarta columna ofrece una sugerencia para la validación del caso concreto si es posible.
La quinta columna establece la prioridad para el cumplimiento de cada caso. OB para obligatoria, RE para recomendada y OP para opcional.

CASO	**Discapacidad**	**Descripción**	**Forma de validar**	**Prioridad**
Iconos fácil de entender	Cognitiva	Los elementos activos de la aplicación deben llevar un icono asociado respetando la normativa de la ISO/IEC TR 19766:2007		OB
Interfaz simple	Cognitiva	La interfaz no debe estar sobrecargada con multitud de controles y datos en pantalla.	La interfaz debe tener mecanismos de agrupación, tales como pestañas de selección o navegación estructurada por menús.	RE
Información multi-modo	Cognitiva-Auditiva	Los estados de la aplicación (Si los tuviera) deben ser claros con indicadores gráficos y de texto.		RE
Canal de ayuda	Cognitiva	Debe existir un canal de ayuda para poder resolver todos los problemas de usabilidad que puedan surgir.		OP

Mecanismo de configuración y personalización mediante asistente	Cognitiva	Debe existir un asistente que guíe al usuario durante el proceso de configuración.	Durante el proceso de configuración debe de acompañarse con cuadros de ayuda y/o notificaciones.	OP
Significado de los colores	Cognitiva	Se recomienda al mostrar información utilizar el color más adecuado a la naturaleza de la información.	En la Tabla del anexo E se puede observar el significado y el contexto de los principales colores utilizados.	OP
Estilo		La aplicación debe ser consecuente con su estilo, es decir, p.e. no puede tener la misma apariencia una etiqueta estática que un elemento raíz de una lista desplegable.		RE
Método de entrada alternativo	Movilidad	El método de entrada alternativo permite a los usuarios manejar el móvil táctil mediante pulsaciones en cualquier parte de la pantalla o mediante un pulsador.	La aplicación debe funcionar correctamente con el método pantalla completa del tecla. Todos los Elementos deben ser focusables.	OB
Método de entrada por pulsación.	Movilidad	Se debe incluir la recomendación para personas de movilidad reducida de instalar y utilizar un método de entrada por pulsación.		OP
Notificación audio	Auditiva	No debe existir ninguna notificación o salida sonora sin su correspondiente notificación luminosa y/o de vibración.		OB

Video	Auditiva	Todo archivo de vídeo debe contener subtítulos y/o una descripción del mismo.		OB
Alto Contraste	Baja visión	Debe tener suficiente contraste (4,5:1 AA y 7:1 AAA) o tener como opción de configuración de varios temas de alto contraste, como blanco sobre negro, amarillo sobre azul, etc.	Se comprobará mediante una simulación con diferentes filtros visuales que simulan diferentes problemas de visión.	RE
Tamaño Fuente	Baja visión	El tamaño de fuente debe ser lo suficientemente grande.	Se comprobará mediante una simulación con diferentes filtros visuales que simulan diferentes problemas de visión.	OB
Diferencias de colores	Baja visión	En la aplicación se deben poder diferenciar los elementos visuales para diferentes problemas de daltonismo.	Se probarán diferentes filtros que simulan los diferentes problemas de daltonismo	OP
Espaciado de la fuente	Baja visión	Deben existir espaciados entre caracteres, entre palabras y entre lineas lo suficientemente amplios. (Tener en cuenta si se va a utilizar algún tipo de fuente nuevo)	Entre caracteres: Un mínimo de un píxel Entre palabras: Un mínimo de la letra H o N Entre lineas: Un mínimo de un píxel, donde esta linea no debe contener partes de caracteres, símbolos diacríticos o caracteres de subrayado.	OB

Asignar descripción	Visión nula	Toda imagen debe llevar asociado un campo de texto que la describa	Cuando se explore la imagen o ésta tenga el foco el lector de pantalla (Talkback) debe reproducir el contenido de descripción de dicha imagen.	OB
Información del tipo de elemento (rol)	Visión nula	Si es un elemento activo, debe indicar que tipo de elemento es.	Cuando se explore dicho elemento o éste tenga el foco, el Talkback debe indicar que tipo de elemento es. Por ejemplo, cuando se explore un Boton, el Talkback deberá reproducir “Botón ….”.	OB
Información del valor del elemento	Visión nula	Se debe indicar el valor actual de todo elemento	Cuando se explore dicho elemento o éste tenga el foco, el Talkback debe indicar el valor de éste. Por ejemplo, cuando se explore un cuatro de texto se debe indicar que valor tiene escrito.	OB
Información de estado del elemento	Visión nula	Si es un elemento con estado, (checkbox, switch, etc),	Cuando se explore dicho elemento o éste tenga el foco, el Talkback debe indicar el estado de éste Por ejemplo, cuando se explore un CheckBox, el Talkback deberá reproducir “CheckBox seleccionado/no seleccionado”	OB

Cuadros edición texto	Visión nula	El cuadro de edición de texto debe estar asociado a una etiqueta.	Cuando se explore el cuadro de edición o éste tenga el foco, el Talkback debe indicar que etiqueta se refiere a dicho cuadro de edición.	OB
Navegación mediante foco	Visión nula	Todos los elementos, sean activos o no, deben tener la posibilidad de que sean seleccionados mediante un medio de navegación (joystick, Gesture).	A partir de la versión de Android 4.1, se puede navegar por la pantalla mediante gestos con el dedo. Se comprobará que todos los elementos son focusables con este método.	OB

ANEXO E – Propuesta de Google sobre el significado de los colores.

Color	**Significado**	**Contexto de uso**
Rojo	Activo, dinámico, peligro, pasión	Mensajes de error, advertencias, feedback negativo
Amarillo	Atención, inteligencia, organización	Feedback positivo
Azul	Silencio, aceptación, paciencia	Ayuda, información, descripción de error
Naranja	Confianza, amigable, salud, energía	Aplicaciones relacionadas con el deporte, salud
Morado	Religión, sabiduría	Información médica
Rosa	Saludable, infantil, femenino	Información especial para mujeres
Verde	Dinero, salud, relajación, naturaleza, armonía, inmaduro	Tareas abiertas.
Gris	Conservador, neutral	Segundo plano, elementos inactivos
Cyan	Relaciones de amigos, sociabilidad	Comunidad social, organizador personal