UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA
Ingeniería de Telecomunicación

MEJORA DE LA ACCESIBILIDAD EN DISPOSITIVOS ANDROID CON EL SISTEMA JVOICEXML

J. MARCOS DEL SER BARTOLOMÉ

JULIO 2016

MEJORA DE LA ACCESIBILIDAD EN DISPOSITIVOS ANDROID CON EL SISTEMA JVOICEXML

AUTOR: J. Marcos del Ser Bartolomé
TUTORA: Pilar Rodríguez

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Julio de 2016

RESUMEN

Este proyecto consiste en crear una implementación Android del intérprete VoiceXML de código abierto JVoiceXML (Java VoiceXML) permitiendo así la ejecución de aplicaciones basadas en el estándar VoiceXML en la plataforma Android. Como demostrador se crea una aplicación llamada Help Desk que imita un centro de ayuda de una aplicación de música en streaming tipo Spotify.

El fin del proyecto es permitir que los desarrolladores puedan portar sus aplicaciones VoiceXML existentes a Android sin esfuerzo o también crear otras sólo para Android aprovechando la potencia del estándar VoiceXML. Los desarrolladores pueden entonces de una manera sencilla mejorar la accesibilidad de sus aplicaciones para las personas con déficit de visión pronunciado utilizando interfaces de voz para el control de las mismas.

Palabras clave

Reconocimiento de voz, VoiceXML, JVoiceXML, IVR, Código abierto, Android, Síntesis de voz, Interfaz de voz

ABSTRACT

This project consists of creating an Android implementation of the open source VoiceXML interpreter JVoiceXML thus allowing for the execution of VoiceXML based applications on the Android platform. To demonstrate it an application called Help Desk is created. This application mimics a help desk of a music streaming application such as Spotify.

The goal of the project is to allow developers to port their existing VoiceXML applications to Android with no effort or create Android specific apps enjoying the power of the VoiceXML standard. Therefore developers will be able to easily improve their apps accessibility for people with severe vision problems by means of voice interfaces.

Keywords

Voice recognition, VoiceXML, JVoiceXML, IVR, Open source, Android, Voice synthesis, Voice interface

Agradecimientos

Quiero agradecer en primer lugar a mi familia por su constante apoyo todos estos años, sin ellos no hubiera podido llegar hasta aquí. También agradecer a los buenos amigos que he hecho en la escuela por los buenos ratos pasados y por su apoyo.
Por último, agradecer a mi tutora Pilar Rodríguez por sus buenos consejos y el tiempo dedicado a que pudiera acabar en forma y tiempo este proyecto.
A todos ellos, muchas gracias.

J. Marcos del Ser

Julio 2016.

Índice de Contenido

Índice de Figuras

1 Introducción

En este capítulo se describen la motivación y los objetivos del proyecto en las secciones 1.1 y 1.2 respectivamente y se hace una descripción resumida de cómo está organizado el resto de la memoria en la sección 1.3.

1.1 Motivación

La motivación de este proyecto es añadir Android a la lista de plataformas que son capaces de procesar aplicaciones VoiceXML (VXML). Con esta capacidad los desarrolladores de aplicaciones Android podrán tanto portar fácilmente aplicaciones VoiceXML existentes a Android como desarrollar de manera sencilla nuevas aplicaciones que hagan uso de la voz como medio de interacción con sus usuarios. Gracias a estas dos ventajas este proyecto ayuda a cerrar la gran brecha tecnológica que hay en el uso de smartphones entre las personas con déficit de visión pronunciado y el usuario medio.

Utilizando el estándar VoiceXML se pueden definir de manera muy sencilla diálogos complejos entre humano y máquina, compartirlos o reutilizarlos en diferentes plataformas liberando a los desarrolladores de trabajo de programación de bajo nivel y gestión de recursos.

Según la recomendación del W3C VoiceXML 2.0 [1] el estándar entre otras ventajas minimiza las interacciones cliente-servidor especificando múltiples interacciones por cada documento, hace transparente a los creadores de aplicaciones VoiceXML de los detalles de bajo nivel y específicos de cada plataforma, facilita la portabilidad del servicio entre distintas plataformas y es fácil de usar para interacciones simples a la vez que facilita la creación de diálogos complejos.

Para conseguir los objetivos de este proyecto se hace una portabilidad del intérprete de código abierto JVoiceXML (Java Voice eXtensible Markup Language) y sus elementos

circundantes al entorno de Android utilizando las APIs de voz que éste proporciona.

Se elige VoiceXML porque es un estándar desarrollado por uno de los más importantes equipos de trabajo para la estandarización en Internet, el W3C (World Wide Web Consortium), y se basa en la interpretación de documentos que definen la interacción humano-máquina que son fáciles de escribir, accesibles y potentes. Estos documentos pueden cargar a su vez otros documentos VoiceXML utilizando su URI (Uniform Resource Identifier) y posibilitando así la creación de aplicaciones complejas, modulares y versátiles.

Gracias a que la capa superior del sistema Android también está escrita en Java esta portabilidad resulta eficiente y fluida.

1.2 Objetivos

El nuevo objetivo de este PFC es integrar el intérprete JVoiceXML en la plataforma Android. De esta manera los dispositivos basados en Android podrán soportar aplicaciones que establezcan interacción por voz con sus usuarios de una forma más sencilla y potente facilitando la mejora de la accesibilidad para personas con déficit de visión severo.

Este objetivo se puede descomponer en los siguientes subobjetivos:

1) Crear las clases necesarias y resolver los conflictos que aparezcan para iniciar el intérprete JVoiceXML en un dispositivo Android
2) Generar el código de la Plataforma de Implementación (en adelante PI) que haga uso de los motores TTS y ASR para soportar la ejecución de aplicaciones VoiceXML
3) Implementar los mecanismos que permitan comunicar el intérprete JVoiceXML con la actividad demostradora para la inicialización de los motores TTS y ASR y el envío de los mensajes a sintetizar y los reconocidos
4) Crear una aplicación VoiceXML (un documento VoiceXML) que se pueda ejecutar en un dispositivo Android real con la aplicación del proyecto instalada y

que demuestre que los tres objetivos anteriores se han cumplido.

1.3 Organización de la memoria

El resto de la memoria está organizada de la siguiente forma:

- **El segundo capítulo** describe el estado del arte de las diferentes tecnologías utilizadas en el proyecto.
- **En el tercer capítulo** se presenta una descripción de la fase de diseño del proyecto. En este capítulo también se encuentra una corta descripción de los diferentes elementos sobre los que se basa el proyecto y también algunas de las decisiones de diseño más relevantes y las razones que llevaron a tomarlas.
- **El cuarto capítulo** habla sobre la fase de implementación y desarrollo tanto de la portabilidad Android de JVoiceXML como de la aplicación demostradora Help Desk.
- **En el quinto capítulo** se presentan las pruebas que se realizaron en entornos reales para comprobar que los requisitos del proyecto se habían cumplido.
- **En el sexto capítulo** se incluyen las conclusiones del proyecto así como las posibles mejoras al proyecto en trabajos futuros
- **El apéndice A** contiene la transcripción del código VXML que define la aplicación demostradora Help Desk
- **El apéndice B** describe el presupuesto aproximado de los costes de este proyecto
- **El apéndice C** contiene el pliego de condiciones del proyecto

2 Estado del arte

2.1 Introducción

Uno de los paradigmas no resueltos de la computación es la capacidad de conversar de forma inteligente con los humanos. Después de que se formulara la Ley de Moore en 1965 muchos investigadores creyeron que para el final del siglo XX este paradigma se convertiría en realidad. A día de hoy sigue siendo ciencia ficción aunque se han hecho grandes avances con los asistentes personales tipo Siri, y con la nueva tecnología de los chat bots.

De cualquier forma, la necesidad de automatizar y reducir costes en la relación entre clientes y empresas sigue estando presente y es cada vez más importante debido al gran auge de teléfonos inteligentes, Internet, la imparable informatización de las empresas y el gran aumento mundial de usuarios que demandan una atención personalizada. Esto está poniendo cada vez más en valor la capacidad de interactuar entre los clientes y las empresas con los mínimos recursos posibles. Y una de las tecnologías más utilizadas para ello son los Interactive Voice Response (IVR) que permiten interactuar al usuario con un ordenador a través de la voz y los tonos DTMF (Dual-tone multi-frequency signaling).

Para facilitar el desarrollo de esa capacidad se han creado una serie de estándares, en particular por el W3C. VoiceXML es uno de esos estándares, desarrollado por el The Voice Browser Working Group.

Por otro lado hay un grupo de personas que se ve especialmente beneficiado por la mejora de los IVR, y son las personas con déficit de visión severo que tienen mayores dificultades para interactuar de forma táctil con los smartphones, tabletas u ordenadores.

En este capítulo se describe de forma sucinta la evolución de los IVR, cuáles son los sistemas más populares a día de hoy, en qué consiste el estándar VoiceXML y la distribución utilizada en el proyecto, Java VoiceXML (JVoiceXML), así como una

breve revisión de la situación actual de la accesibilidad en entornos móviles.

2.2 Accesibilidad en entornos móviles

La telefonía móvil ha generado un potencial enorme en las comunicaciones, así como en la enseñanza, en la capacidad de autoaprendizaje y ha creado nuevos modelos de negocio y formas de expresarse y relacionarse. Con el abaratamiento tanto del hardware como del software y de la conexión a redes digitales la penetración del móvil en el mundo ha llegado a un punto impensable hace algunos años, como demuestra este gráfico del informe anual de la empresa Ditrendia "Informe mobile en España y en el mundo" [2], llegando a superar en algunas regiones el 100% de penetración de líneas móviles activas (esto es, hay más líneas activas que habitantes en esa región) tal y como muestra la siguiente figura:

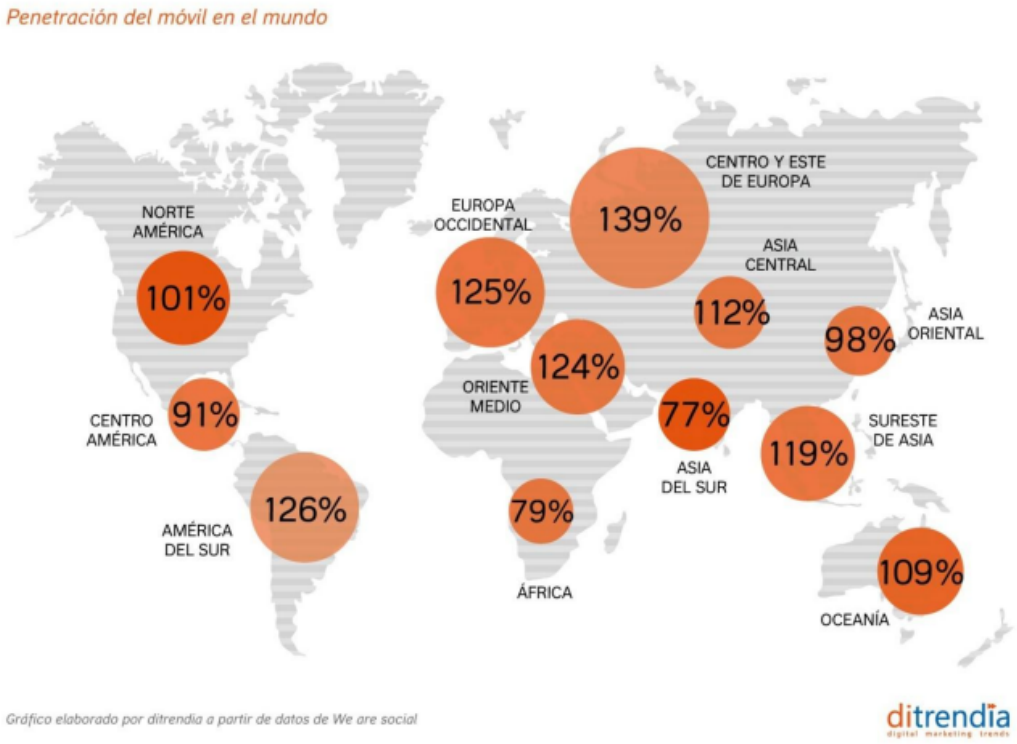

Figura 2-1: Penetración de líneas móviles en el mundo según el informe "Informe mobile en España y en el mundo" de Ditrendia [2]

Sin embargo, cuando hablamos del colectivo de personas con algún tipo de discapacidad, que en España y según datos del INE de 1999 suponen el 9% de la población española (3.528.221 personas) el panorama no es tan alentador.

Según el informe "Acceso y uso de las TIC por las personas con discapacidad" [3] de la Fundación Vodafone España centrado en las personas con discapacidad visual, auditiva y de movilidad la brecha en el acceso a las TIC es muy grande. En particular, utiliza el teléfono móvil el 91,8%, el ordenador el 42,8% e Internet el 32,5% frente al 95.5%, 72,7% y 75,1% respectivamente por término medio en el seno del conjunto de población española. Esto significa que aunque en el uso de Internet y el ordenador hay una brecha flagrante, en el caso del móvil el porcentaje de uso es muy similar al del conjunto de la población.

El estudio destaca también que la accesibilidad de los dispositivos y de las aplicaciones está considerada como herramienta y desarrollo imprescindible para mejorar la calidad de vida y la autonomía personal de las personas con discapacidad.

Hoy en día existen multitud de dispositivos móviles adaptados a las personas con discapacidad, pero como muestra la figura siguiente, su alto precio es un impedimento para un porcentaje muy importante de este colectivo por lo que deben buscarse otras soluciones

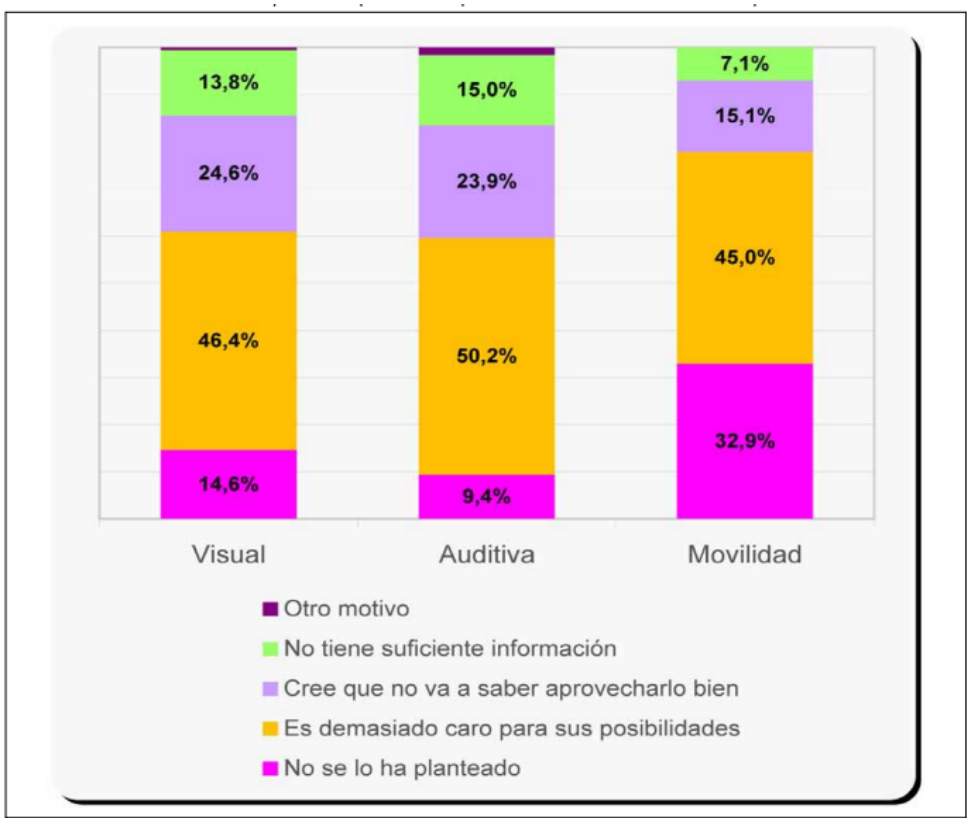

Figura 2-2: Población con discapacidad visual, auditiva o de movilidad usuaria de teléfono móvil, motivos por los que no dispone de teléfono móvil adaptado (%) según el informe "Acceso y uso de las TIC por las personas con discapacidad" [3] de la Fundación Vodafone España

2.3 Evolución de los IVR

Lo que sigue es la traducción libre del artículo de Wikipedia sobre los IVR [4]:

> Antes incluso del advenimiento de la computación digital ya se estaba investigando la tecnología de síntesis de voz. El primer proyecto tuvo lugar en los Laboratorios Bell en 1936 lo que resultó en un dispositivo llamado "The Voder". Este dispositivo intentaba imitar los efectos del tracto vocal humano. Era muy complejo tanto en su construcción como en su manejo pero permitió por primera vez generar voz reconocible con una máquina.
>
> En 1961 Bell System desarrolló una nueva metodología de marcado por tonos lo

que dio lugar al primer teléfono que podía marcar códigos de región utilizando tecnología DTMF presentado en sociedad en 1962.

Por otro lado al principio de la década de los 70 Leonard E. Baum y Lloyd R. Welch dieron un gran paso adelante con el desarrollo de un concepto estadístico llamado Hidden Markov Model que permitía un acercamiento matemático a la tarea del reconocimiento de voz. Se habían puesto ya los cimientos para el desarrollo de IVRs.

Ya en esa década se registró un avance importante en la implementación de la tecnología IVR para automatizar tareas en los call centers, pero todavía la tecnología era compleja y cara. No será hasta principios de los 80 cuando se empiece a generalizar el uso de los IVR gracias a la reducción del coste de los discos duros lo que permitió a Leon Ferber sustituir la síntesis de voz producida por tecnología DSP (digital signal processing) por grabaciones de voz. De esta manera el sistema contiene grabada la voz en el disco duro, reproduce el mensaje requerido y procesa la respuesta humana en forma de tonos DTMF.

Cuando los call centers empezaron a hacer la migración hacia el multimedia a finales de los 90, las empresas empezaron a invertir en CTI (computer telephony integration) que engloba cualquier tecnología que permita la integración y coordinación de las interacciones entre un teléfono y un ordenador. Esto produjo que los IVR se convirtieran en parte fundamental de los call centers actuando como un agente que recogía datos del usuario y permitía tomar decisiones inteligentes de redireccionamiento de la llamada. También en esta década empezaron a utilizarse sistemas que podían reconocer la voz dentro de un pequeño vocabulario de interés en vez de requerir que el usuario utilizara los tonos DTMF.

Ya en la primera década del siglo XX los IVR se hicieron más y más comunes y mucho más baratos de implementar. Esto fue debido al incremento en la potencia de las CPUs y en la migración masiva de las aplicaciones de voz desde código propietario al estándar VXML.

En los últimos años ha habido grandes mejoras tanto en la calidad de la síntesis de voz,

como en la capacidad de reconocimiento de voz, con aplicaciones en la industria del automóvil o la creación de los asistentes Siri de Apple, Cortana de Microsoft y Google Now de Google que permiten abrir el abanico de aplicaciones de esta tecnología.

2.4 Estándar VoiceXML

A continuación un extracto del artículo de Wikipedia sobre el estándar VoiceXML [5]:

> VoiceXML es un estándar que especifica diálogos interactivos por voz entre un humano y un ordenador. Se utiliza para desarrollar aplicaciones interactivas de audio y voz como sistemas de banca y portales de atención al cliente automatizados. Las aplicaciones VoiceXML se desarrollan e implementan de una manera análoga a como un navegador web interpreta el lenguaje HTML que recibe de un servidor web. Los documentos VoiceXML son interpretados por un navegador de voz y en la arquitectura más común los usuarios interactúan con el navegador a través de la red telefónica pública conmutada (PSTN, Public Switched Telephone Network).
>
> El formato de los documentos VoiceXML está basado en el Extensible Markup Language (XML).
>
> En el año 1999 AT&T Corporation, IBM, Lucent y Motorola crearon el Foro VoiceXML para desarrollar un lenguaje de marcado estándar para especificar diálogos de voz. En septiembre de ese año el Foro presentó públicamente su VoiceXML 0.9 para recibir feedback y comentarios. Pocos meses después, en marzo del año 2000 publicaron VoiceXML 1.0, y poco tiempo después el Foro entregó el control del proyecto al W3C. Ya en sus manos el W3C produjo varias versiones intermedias del VoiceXML 2.0, llegando a la fase final de "Recomendación" en marzo del año 2004. Basado en el feedback de las diferentes implementaciones de VoiceXML 2.0 el grupo generó la versión VoiceXML 2.1 que añadía un conjunto relativamente pequeño de funcionalidades extra al 2.0. Esta versión recibió el estatus de "Recomendación" en junio de 2007.

Desde entonces se está trabajando en la versión 3.0 que no acaba de terminarse y parece que el W3C ha aparcado de momento ese proyecto ya que en la página de "The Voice Browser Working Group" [6] dicen haber cerrado oficialmente ese grupo el día 12 de octubre del año 2015. Se ha buscado sin éxito alguna noticia que clarifique los planes del W3C al respecto.

2.4.1 Ejemplos de sistemas compatibles con VoiceXML

Según escribe en su blog [7] el programador experto en tecnologías de comunicación Kevin Junghans el cambio más grande de los últimos tiempos en el desarrollo de las aplicaciones de reconocimiento de voz ha sido el desplazamiento del desarrollo desde entornos propietarios a plataformas basados en estándares como VoiceXML y CCXML (Call Control eXtensible Markup Language) que están controlados por el W3C.

Estos estándares abiertos cambiaron radicalmente la manera en que las aplicaciones de voz se desarrollan, moviéndose a herramientas más centradas en la web y facilitando la transición de los desarrolladores ya que les permitió utilizar sus conocimientos en áreas como JavaScript, XML, servidores web y otras tecnologías de desarrollo web.

A continuación se describen algunas de las plataformas de código libre basadas en los estándares mencionados y sus características más importantes.

VoiceModel

La descripción que se hace de este sistema en su página web oficial [8] se podría resumir como sigue:

> VoiceModel es un proyecto que facilita el desarrollo de aplicaciones de voz VoiceXML y de la plataforma Tropo utilizando el entorno de desarrollo ASP.NET MVC 4 y Razor.
>
> VoiceModel utiliza el patrón de diseño MVVM (Model-view-VoiceModel) para conseguir un nivel de abstracción alto en las aplicaciones de voz y separa la interfaz de voz del dominio de la aplicación y el flujo de llamadas. VoiceModel también

soporta el uso de Reusable Dialog Components (RDC) lo que permite la construcción de módulos que pueden reutilizarse en otras aplicaciones de voz aumentando así la productividad. También incluye una máquina de estados de fácil uso que permite definir fácilmente el flujo de llamadas en las aplicaciones de voz. Con VoiceModel es posible desarrollar las aplicaciones de voz una vez y desplegarlas después en cualquier plataforma compatible con VoiceXML o con Tropo.

VoiceModel se distribuye bajo la licencia de código abierto Apache License 2.0.

Asterix

Así se describe Asterix en su página oficial [9]:

> Asterix es un entorno de desarrollo de código libre para construir aplicaciones de comunicaciones. Convierte un ordenador ordinario en un servidor de comunicaciones. La tecnología de Asterix permite crear sistemas IP PBX, pasarelas VoIP, servidores de conferencia y otras soluciones a la medida. Asterix es utilizado por pequeños negocios, grandes empresas, call centers, operadoras y agencias de gobierno de todo el mundo.

Asterix es gratuito y se distribuye bajo la licencia de código libre GNU Public License version 2.

VoiceGlue

Este proyecto está alojado en GitHub y se pueden descargar todas sus clases. En la wiki del proyecto en GitHub [10] encontramos la descripción del mismo:

> El proyecto VoiceGlue ofrece un intérprete VXML para Asterix utilizando el intérprete OpenVXI. El proyecto integra diversas iniciativas de código libre y contiene módulos originales de código abierto para crear una solución VXML de código libre para los usuarios de Asterix.

VoiceGlue permite a los usuarios de Asterix desplegar una plataforma VoiceXML 100% de código libre para construir IVRs y otras aplicaciones útiles. Está bien documentada y tiene una activa comunidad de usuarios que comparten su conocimiento ofreciendo consejos y ayudando a solucionar los problemas con los que se encuentren

los desarrolladores.

2.5 Proyecto de código libre JVoiceXML

JVoiceXML es un intérprete VXML escrito en Java con una arquitectura abierta que permite crear ampliaciones a la medida y es compatible con el estándar VoiceXML 2.1.

Tal y como explica el administrador de JVoiceXML Dirk Schnelle-Walka en la entrevista [11] que concedió al equipo de la Universidad de Carnegie Mellon que desarrolla Sphinx (uno de los reconocedores de voz de código libre más importantes y uno de los que soporta JVoiceXML) el proyecto nace en 2005 cuando Dirk estaba haciendo su tesis doctoral en el Telecooperation Lab de la Universidad Técnica de Darmstadt. El objetivo de la tesis era utilizar interfaces de voz en entornos inteligentes.

Como doctorando Dirk era consciente de que los institutos de investigación encuentran problemas para utilizar los sistemas comerciales por su alto coste. En aquel momento existían implementaciones gratuitas de VoiceXML pero no se podían utilizar sin un teléfono. Por todo ello Dirk decidió iniciar su propio proyecto de código libre para crear un intérprete VoiceXML. Al poco de iniciar el proyecto Steve Doyle se unió y desarrolló la librería XML. En 2006 se unieron otros dos miembros, Torben Hardt y Christoph Bünte. Torben creó el scripting engine y Christoph hizo del procesador de gramáticas como su tesis de máster. Así surgió el núcleo del proyecto al que se unirían después otros programadores para hacer compatible el sistema con el estándar VoiceXML 2.1 soportando la totalidad de las etiquetas del lenguaje VXML.

La principal ventaja de JVoiceXML es que se puede utilizar de forma gratuita, incluso en sistemas comerciales. Otras ventajas son su capacidad de ampliación, interoperabilidad y soporte para interfaces estándar. Los usuarios principales de JVoiceXML son las universidades y algunas start-ups pequeñas de India, China y Estados Unidos sobretodo.

En el curso 2010-2011 se añade a propuesta de Dirk Scnhelle-Walka y Germán Montoro el proyecto "Interfaces de interacción mediante dispositivos móviles" dentro de la oferta de PFCs de la Escuela Politécnica Superior de la UAM, proyecto que sería el germen de éste y que tenía como objetivo realizar una portabilidad Android del intérprete JVoiceXML.

2.5.1 Otras portabilidades Android

Cuando se inició el proyecto original propuesto por Dirk Schnelle-Walka en 2011 no había ninguna portabilidad Android de JVoiceXML. Desde entonces ha habido varias, todas realizadas por estudiantes de Ingeniería Informática, y con diferente nivel de éxito. A continuación se describe cada una de ellas, ordenadas por año:

2012, Matteo Bruneti

Este estudiante italiano de la Universitá Politecnica delle Marche realizó su proyecto de fin de carrera teniendo como objetivo la portabilidad de JVoiceXML a Android. El título de su trabajo es "Porting JVoiceXML su Android" [12]. Debido a las dificultades que encontró en su desarrollo desistió del objetivo inicial y su desarrollo consistió en desplegar el proyecto JVoiceXML en un PC, iniciar el intérprete e interpretar un documento, aunque no queda claro si con éxito o no.

Matteo trabajó de forma independiente sin colaborar directamente con Dirk Schnelle-Walka.

2013, Eros Federici

Este estudiante italiano de la misma universidad que Matteo Bruneti realizó el proyecto de fin de carrera titulado "Human-Computer vocal interaction: un portiere robotico" [13]. El objetivo del mismo era, al igual que en el de Matteo Bruneti, realizar la portabilidad Android de JVoiceXML y demostrar su correcto funcionamiento creando una aplicación VXML que permitiera dar información al usuario sobre los profesores de la universidad y sus despachos mediante peticiones realizadas por voz.

Eros Federici utilizó el trabajo no concluido de según sus propias palabras "un estudiante español" para desarrollar su proyecto.

Eros Federici aparentemente sí que consiguió el objetivo que perseguía (como demuestra un vídeo de demostración subido a Youtube [14]). El código del su proyecto no está disponible en Internet y en la memoria de su tesis no queda claro cómo se realizó la portabilidad.

2014, Martin Nekula

Estudiante checo de la Universidad Masarykova de Brno, realizó una portabilidad Android del intérprete JVoiceXML como proyecto de fin de carrera con título "Cliente de texto JVoiceXML en la plataforma Android" [15]. Actualmente ésta es la portabilidad Android oficial del proyecto JVoiceXML.

Martin Nekula consiguió portar con éxito JVoiceXML a Android pero generando una interfaz de texto en vez de voz. Es decir, con el proyecto de Martin es posible ejecutar documentos VXML en un teléfono Android pero tanto los mensajes del sistema al usuario como las respuestas de éste tienen una representación textual en la pantalla del móvil.

2.6 Conclusiones

En las secciones anteriores se ha explicado cómo las tecnologías IVR han mejorado cualitativamente los últimos años gracias a la estandarización y a la creación de plataformas de código libre como JVoiceXML. También se ha visto cómo la penetración de los móviles es casi universal y cómo entre las personas con alguna discapacidad el porcentaje de uso es sólo ligeramente inferior.

Android es de lejos la plataforma móvil con más usuarios y cuando se inició este proyecto no había ninguna portabilidad Android del intérprete de código libre JVoiceXML que permitiera la ejecución de aplicaciones VXML en esa plataforma.

Otra de las claves es que la mejora de la accesibilidad es parte fundamental para la inclusión de este colectivo. La tecnología en general y la telefonía móvil en particular han supuesto una gran ayuda para apoyar y hacer más independientes a las personas con todo tipo de discapacidades. Pero todavía queda un largo camino por recorrer, y un porcentaje muy alto de personas con discapacidad no puede permitirse la compra de un dispositivo especialmente adaptado a sus necesidades por razones económicas.

Por todas estas razones este proyecto pretende facilitar que la brecha digital se reduzca y que sea más sencillo desarrollar aplicaciones accesibles para avanzar en la inclusión digital del colectivo de personas que sufren déficit de visión severo.

3 Diseño

En esta sección se describen las herramientas utilizadas en el proyecto, así como una visión general de la estructura del mismo y los requisitos a los que ceñirse.

3.1 Requisitos

El proyecto surge de la oportunidad de aprovechar la gran penetración de los móviles en general y del ecosistema Android en particular para aumentar en gran medida el potencial del sistema JVoiceXML haciéndolo compatible con Android. Por ello los requisitos están muy ligados a que el proyecto JVoiceXML pueda aprovecharse de esta portabilidad y que su mantenimiento sea mínimo ya que debido a que es un proyecto de código libre y sin ánimo de lucro no cuenta con grandes recursos y depende de los colaboradores para su evolución.

Debido a la incertidumbre inicial sobre la carga de trabajo que podían suponer se dividieron los requisitos en dos categorías, obligatorios y opcionales.

Los requisitos obligatorios son:

1. Poder ser utilizado y extendido por otras aplicaciones.
2. Soportar la mayoría de etiquetas básicas del estándar VoiceXML
3. Interactuar con el usuario por voz utilizando los motores TTS y ASR
4. Representar un posible caso real de aplicación comercial con la aplicación demostradora (Help Desk)
5. Poder cargar documentos VXML alojados en Internet
6. Funcionar con dispositivos con versión Android igual o superior a la 4.0

Y los requisitos opcionales son:

1. Ser un módulo independiente que no requiera cambios en el código de JVoiceXML. Así el desarrollo de nuevas versiones de JVoiceXML debería ser compatible con la portabilidad Android.
2. Permitir ejecutar documentos VXML en inglés y en español

3. Desarrollar el proyecto de acuerdo a las estructuras y filosofía de programación Android

3.2 Herramientas

En esta sección se hace una breve revisión de las herramientas utilizadas para el proyecto así como las razones que llevaron a adoptarlas.

3.2.1 Eclipse

Eclipse es un entorno de desarrollo integrado (IDE, Integrated Development Environment) multiplataforma compuesto por un conjunto de herramientas de programación de código abierto. Programado principalmente en Java, una de sus mayores ventajas es que consta de un sistema extensible de plug-ins que permite adaptar el entorno al lenguaje de programación deseado. Soporta multitud de lenguajes de programación diferentes y gracias a la activa comunidad que colabora en su desarrollo ofrece numerosas herramientas adicionales.

En los inicios del sistema operativo Android, Google escogió Eclipse como el entorno de desarrollo estándar para la programación de sus aplicaciones. Lanzó el Eclipse ADT (Android Development Toolkit) como herramienta oficial para facilitar el desarrollo de aplicaciones Android. En el punto 3.2.2 se describen algunas de sus características más importantes. Actualmente se pueden desarrollar aplicaciones Android en multitud de entornos de desarrollo y Google ha creado uno exclusivo para Android llamado Android Studio. Desde diciembre de 2014 Android Studio es el IDE oficial de Android y ofrece muchas facilidades para el desarrollo de aplicaciones con interfaces y lógica complejas y que requieran compatibilidad con el inmenso universo Android.

Este proyecto se ha desarrollado en su totalidad utilizando Eclipse. Las razones para escogerlo son múltiples, siendo las más importantes:

- El autor ya había utilizado previamente Eclipse

- El administrador de JVoiceXML utilizaba también Eclipse para sus desarrollos
- Cuando se empezó este proyecto aún no existía Android Studio y Eclipse era el IDE oficial de Android
- Es un IDE considerado robusto y con pocos fallos

Aparte de las razones anteriormente expuestas, el uso de Eclipse facilitó el desarrollo del proyecto gracias a varias funcionalidades como son:

- Autocompletado de código
- Autosangrado de las líneas de código

Herramientas de búsqueda y navegación dentro de las distintas clases del proyecto: esto fue clave debido a que el intérprete JVoiceXML está constituido por centenares de clases, muchas de las cuales heredan de otras clases, abstractas o no.

3.2.2 Servidor Web

Tal y como se describe en el estándar VoiceXML [1]:

> *A VoiceXML interpreter context needs to fetch VoiceXML documents, and other resources, such as audio files, grammars, scripts, and objects.*

Estos recursos se referencian mediante una URI, y pueden estar alojados en una red interna o en internet.

Durante el desarrollo del proyecto se utilizaron diferentes documentos VXML para ir probando la funcionalidad desarrollada (más sobre esto en la sección 6). Se probó a subir estos documentos a servicios de hosting habituales como Drive, Dropbox y otros, sin éxito. Estos servidores no permiten la descarga de documentos con el método GET del protocolo HTTP devolviendo un error 403 (ver traza del error en la figura 3-1) probablemente por seguridad. De hecho los links a documentos alojados en estos servicios comienzan con “https” por lo que están pensados para descargarse con el protocolo de seguridad HTTPS. El problema es que el código de JVoiceXML utilizado para el proyecto no soporta ese protocolo.

```
org.jvoicexml.event.error.BadFetchError: error.badfetch: Forbidden
(HTTP error code 403)
    at org.jvoicexml.documentserver.schemestrategy.HttpSchemeStrategy.
getInputStream(HttpSchemeStrategy.java:195)
    at org.jvoicexml.android.callmanager.AndroidDocumentServer.getDocu
ment(AndroidDocumentServer.java:212)
    at org.jvoicexml.interpreter.VoiceXmlInterpreterContext.acquireVoi
ceXmlDocument(VoiceXmlInterpreterContext.java:715)
    at org.jvoicexml.interpreter.VoiceXmlInterpreterContext.loadDocume
nt(VoiceXmlInterpreterContext.java:671)
    at org.jvoicexml.interpreter.JVoiceXmlSession.call(JVoiceXmlSessio
n.java:198)
    at org.jvoicexml.android.callmanager.Interpreter.run(Interpreter.j
ava:91)
org.jvoicexml.event.error.NoresourceError: error.noresource: Sessio
n is already closed
    at org.jvoicexml.interpreter.JVoiceXmlSession.waitSessionEnd(JVoic
eXmlSession.java:250)
    at org.jvoicexml.android.callmanager.Interpreter.run(Interpreter.j
ava:97)
Se cazó el error con causa: null
Se cazó el error con Localizedmessage: error.noresource: Session is
 already closed
Se cazó el error con menssage: error.noresource: Session is already
 closed
saliendo del run y por tanto matando el thread
```

Figura 3-1: traza del error 403 devuelto por varios servicios de Hosting al intentar descargar documentos con el método GET del protocolo HTTP

Después de varios intentos fallidos se encontró que la web de organización Trello permitía descargar los documentos alojados en ella utilizando el protocolo HTTP en vez del HTTPS. Se utilizó este servicio gratuito de internet hasta que la compañía decidió dejar de permitir este tipo de descargas.

Después de probar otros servicios de hosting sin éxito se decidió montar un servidor web Apache en el ordenador del autor y configurar un dominio gratuito con nombre accesibilidadandroid.hopto.org asociado a ese servidor web. Para que se pudiera referenciar al documento VXML que constituye la aplicación demostradora Help Desk con una URI al uso y para evitar que los cambios de IP (debido a tener una IP dinámica) rompieran el enlace se utilizó la herramienta gratuita No-IP DUC del servicio de hosting utilizado, No-IP. Esta herramienta comprueba periódicamente la IP y actualiza consecuentemente su servidor DNS.

Con estas herramientas se construyó un servidor web casero donde poder alojar los documentos VXML a ejecutar.

3.3 Módulos de la portabilidad

Como se ha mencionado anteriormente el objetivo del proyecto es conseguir interpretar un documento VXML en un teléfono Android utilizando el intérprete JVoiceXML. Para ello se estudió tanto las bases de programación Android así como la estructura modular del proyecto JVoiceXML. De ese estudio surgió el diseño que muestra la figura siguiente y que define a alto nivel la estructura de la portabilidad:

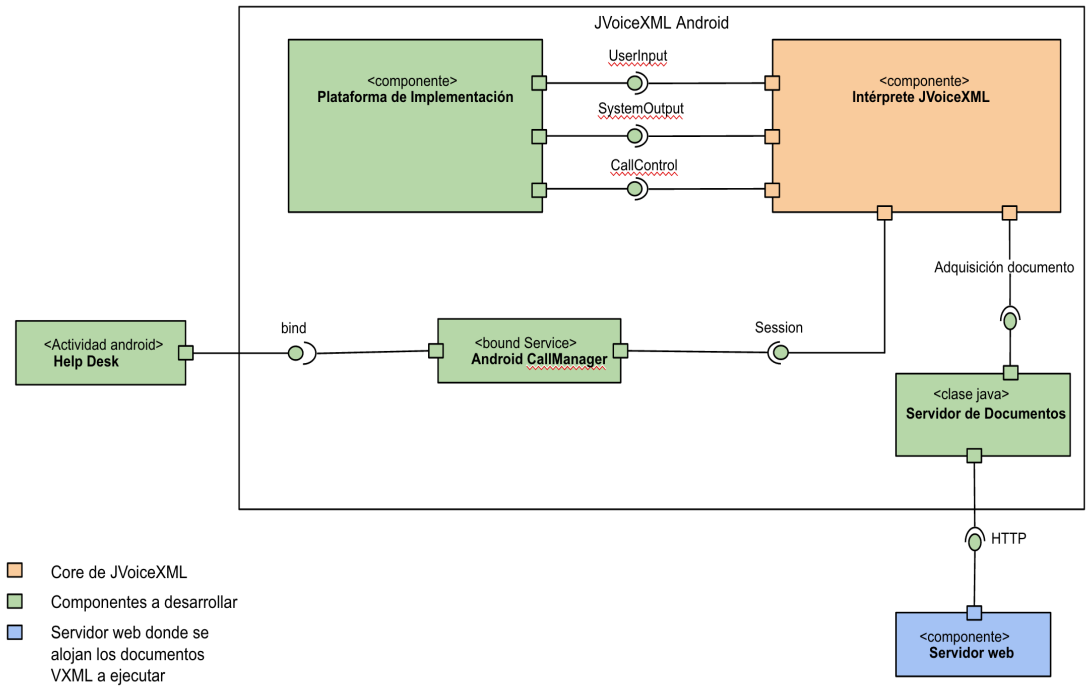

Figura 3-2: Visión de alto nivel de la portabilidad

Los nuevos componentes deben integrarse en la estructura global de JVoiceXML. Como ya se dijo en el Estado del Arte una de las ventajas del sistema JVoiceXML es su modularidad y el énfasis que pone en facilitar la extensión del sistema a otras plataformas. En la sección de Implementación veremos como la mayoría de los

componentes a desarrollar están planificados en JVoiceXML y cuentan con una o más clases Java de tipo interfaz (clases abstractas puras) ya desarrolladas y que se toman como base para los módulos a desarrollar en este proyecto. Gracias a todo esto el desarrollo de esta portabilidad debería permitir una fácil adaptación a los cambios futuros que se produzcan en el intérprete JVoiceXML.

A continuación se describen los diferentes módulos a desarrollar en el proyecto.

3.3.1 Android CallManager

El CallManager se utiliza para iniciar y parar el intérprete JVoiceXML. Ofrece una interfaz que puede utilizarse para iniciar llamadas (sesiones) con una única llamada a un método. Los argumentos que se pasan a ese método serán como mínimo una URI que apunte a un documento VoiceXML alojado en un servidor web utilizando el protocolo HTTP. Además el CallManager ofrece métodos para terminar las llamadas (sesiones) y otros métodos secundarios.

Este componente está constituido por el *bound service* CallManager. Tal y como explica la documentación de Google sobre los Servicios [16]:

> *A Service is an application component representing either an application's desire to perform a longer-running operation while not interacting with the user or to supply functionality for other applications to use*

La plataforma Android obliga a que las operaciones de larga duración (más de 5 segundos) se hagan fuera de la Actividad que está en primer plano (la que ve el usuario). Si esto no se cumple, el sistema genera una excepción y muestra un diálogo ANR (Android Not Responding) al usuario, preguntándole si quiere esperar o cerrar la aplicación. Para evitar este problema se deben realizar esas operaciones de larga duración en alguno de los siguientes componentes Android:

- Service
- Asynctask
- Thread (hilo Java clásico)

Se decide utilizar un Servicio Android para construir la portabilidad Android como una especie de librería que pueda ser utilizada por otras aplicaciones Android para ejecutar documentos VXML, así como para evitar que la aplicación deje de responder.

Android ofrece diferentes tipos de Servicio según la funcionalidad que se quiera implementar. En este proyecto se escogió el tipo BoundService porque facilita la comunicación entre la Actividad y el Servicio y por la gestión del ciclo de vida de estos servicios que hace Android. En particular el Servicio permanecerá activo mientras haya alguna Actividad unida a él y el sistema "matará" automáticamente el Servicio en cuanto no haya ningún componente Android unido a él. Este ciclo de vida concuerda perfectamente con el funcionamiento esperado para una aplicación que haga uso de esta portabilidad para ejecutar un documento VXML.

3.3.2 Plataforma de Implementación

La Plataforma de Implementación constituye, junto con el CallManager el componente más importante de la portabilidad Android. Este componente se encarga de gestionar los recursos externos al intérprete, en este caso los motores ASR y TTS, y generar eventos en respuesta a acciones del usuario (p.ej recepción de input, desconexión) o a eventos del sistema (p.ej timeouts).

En la siguiente figura podemos ver las clases más importantes de la Plataforma de Implementación y qué recursos gestiona cada una:

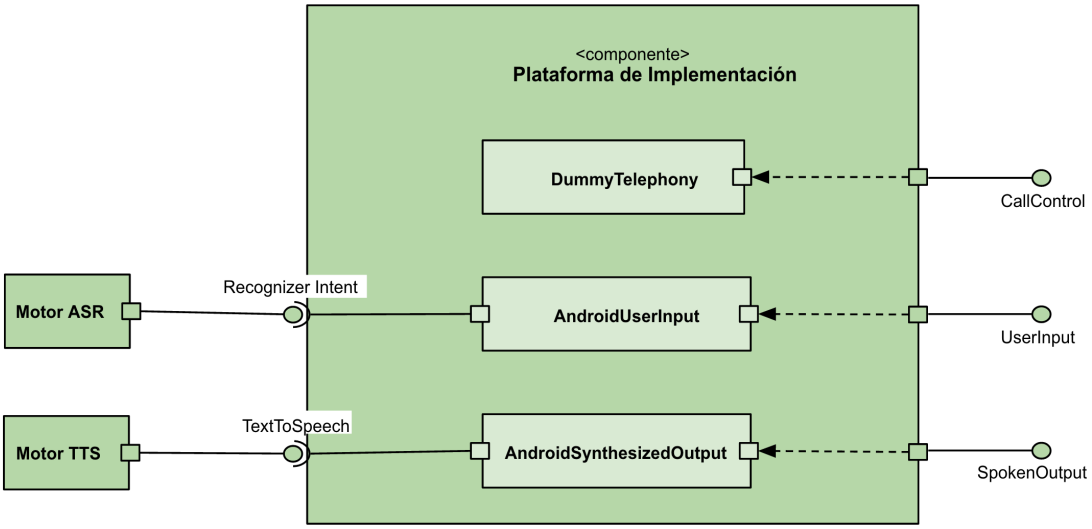

Figura 3-3: Componentes de la Plataforma de Implementación

En este proyecto el soporte para telefonía no es requisito, por lo que se utiliza la clase DummyTelephony que está básicamente vacía de contenido pero que permite el correcto funcionamiento de todo el sistema.

3.3.3 Document Server

Este componente se encarga de procesar las peticiones que le llegan desde el intérprete JvoiceXML para descargarse documentos VXML de un servidor web a través del protocolo HTTP.

3.3.4 Aplicación demostradora Help Desk

Este componente constituye la aplicación Android que demuestra que se han cumplido los requisitos del proyecto. Está compuesta por una actividad que se unirá (mediante el método bind()) al servicio AndroidCallManager para ejecutar documentos VXML. Para

ello la actividad pasará como argumento de la función bind() el URI del documento VXML a ejecutar por el Intérprete JVoiceXML.

Por la importancia de este componente la siguiente sección se dedica a describirlo con mayor detalle.

3.4 Aplicación Help Desk

A continuación se describen los requisitos, objetivos, funcionalidad e interfaz de la aplicación Help Desk a desarrollar.

3.4.1 Requisitos

La aplicación Help Desk deberá cumplir los siguientes requisitos:

- Representar un módulo de una aplicación potencialmente comercial que utilice la interacción por voz entre usuario y teléfono
- Que el reconocimiento y síntesis sea en español
- Que su uso sea intuitivo
- Que el documento VXML que constituye la aplicación haga uso de al menos 10 etiquetas del lenguaje VXML
- Que se registre algún dato del usuario que luego se utilice en un diálogo sintetizado
- El documento VXML que constituye la aplicación estará alojado en Internet y se descargará en tiempo de ejecución

3.4.2 Objetivos

La aplicación Help Desk constituye un módulo que se podría añadir a cualquier tipo de aplicación y que pretende proporcionar al usuario la capacidad de solucionar los problemas de uso a los que se enfrenta al utilizar esa aplicación así como descubrir

nuevas funcionalidades de la misma o pedir que le contacte un asesor. En particular este Help Desk sería el módulo de ayuda de una aplicación de música en streaming tipo Spotify.

El objetivo principal de la aplicación Help Desk es demostrar el correcto funcionamiento de la portabilidad Android del sistema JVoiceXML. El segundo objetivo es crear un ejemplo real de mejora de la accesibilidad para personas con déficit de visión en el ecosistema Android.

3.4.3 Funcionalidad

La aplicación pretende dar solución a tres casos de uso, a saber:

- Un usuario tiene algún problema o no consigue realizar alguna acción deseada y busca ayuda
- El usuario no encuentra la solución en el listado y pide que le llame un asesor de soporte técnico
    - Se le pide el teléfono y la hora y día más conveniente para llamarle
- Un usuario quiere descubrir funcionalidades que desconoce de la aplicación

La siguiente figura describe el flujo de la interacción entre el usuario y la aplicación:

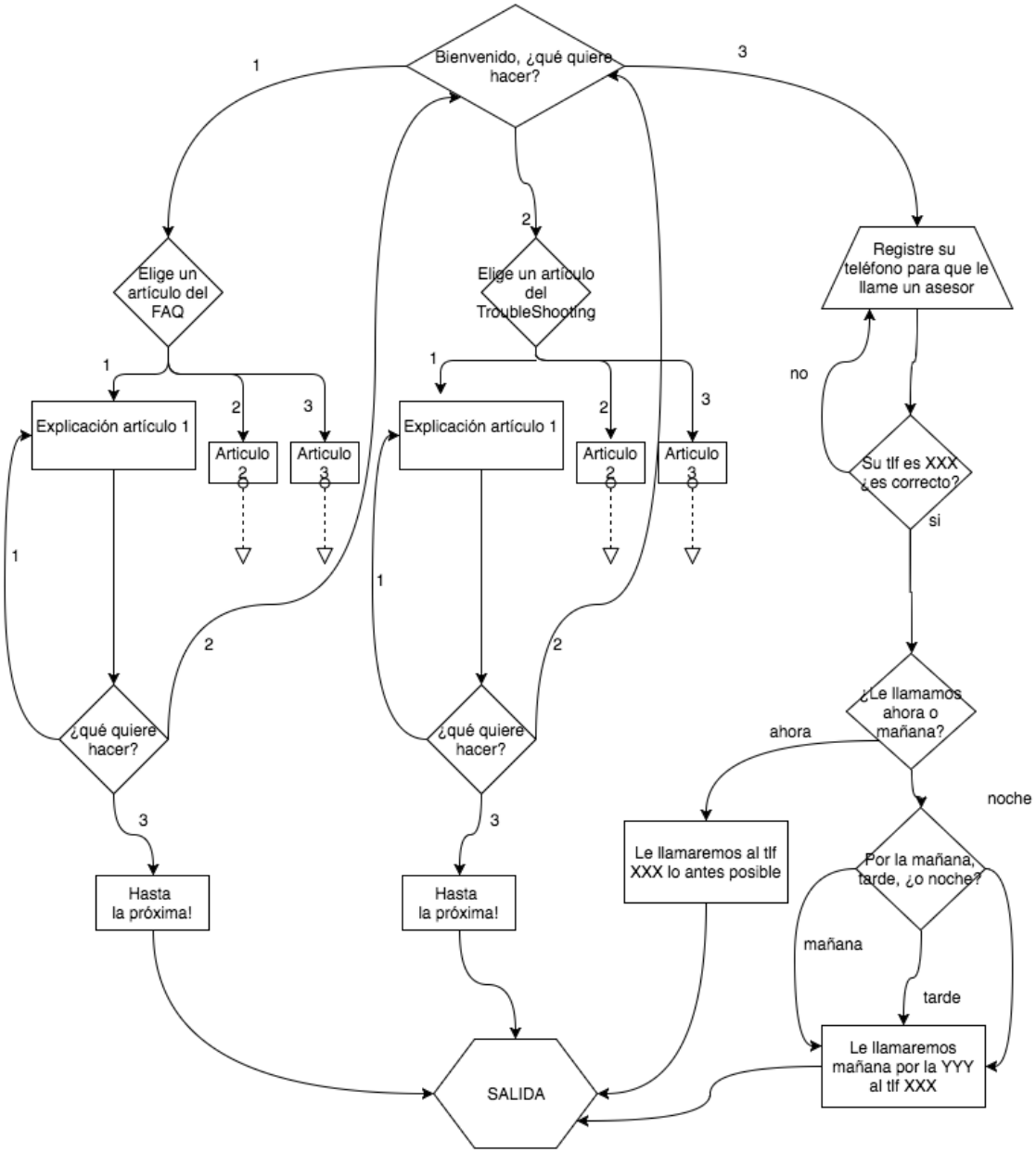

Figura 3-4: Diagrama de flujo de la aplicación Help Desk

3.4.4 Interfaz

Debido a que la aplicación está pensada como demostración de la portabilidad y su potencial usuario es una persona con déficit de visión la interfaz es absolutamente minimalista con fondo del logotipo de la Escuela Politécnica Superior de la UAM y sólo dos botones, Iniciar y Parar:

Figura 3-5: Captura de pantalla de la aplicación Help Desk

4 Implementación

En la sección 4.1 se describen los materiales físicos y de software utilizados en el proyecto. En las secciones 4.2 y 4.3 se introducen los componentes de Android y VoiceXML relevantes para este trabajo que se citarán en las secciones desde la 4.4 a la 4.7 donde se detallan las tareas de implementación más importantes y se explican las decisiones tomadas respecto a la estructura del proyecto y las problemáticas surgidas durante su desarrollo.

4.1 Material utilizado

Para la realización del proyecto han sido necesarios una serie de materiales físicos:

- Ordenador personal Mac
- Smartphone Nexus 5 con versión Android > 4.0
- Cable USB macho a microUSB macho para instalar y depurar el proyecto

Y varios programas de software:

- Eclipse IDE con el plugin ADT
- XAMPP, distribución Apache gratuita para montar un servidor web casero donde alojar los documentos VXML a ejecutar
- No-IP DUC, programa que actualiza las DNS del dominio ligado al servidor web para mantener ese vínculo cuando la IP dinámica cambia

4.2 Componentes Básicos de Android

En esta sección se describen los componentes Android más relevantes que han sido utilizados en el desarrollo de este proyecto: Activity, Service, Layout, BroadcastReceiver, AndroidManifest y Intent.

4.2.1 Activity

Las Activity (Actividad en adelante) son los bloques fundamentales de cualquier aplicación Android. Según la documentación oficial de Android [17] representan una acción única y focalizada que el usuario puede realizar. Casi todas las actividades interactúan con el usuario, por lo que la clase Activity se encarga de crear una ventana en la que colocar la interfaz de usuario (UI) con el método setContentView(View). Aunque las Actividades normalmente son ventanas a pantalla completa también se pueden utilizar de otras maneras, como ventanas flotantes o incluidas dentro de otras Actividades. Una aplicación Android estará constituida por una o más Actividades que se invocan entre ellas conforme el usuario va interactuando con la UI.

Las Actividades gestionan automáticamente su ciclo de vida (figura 4-1) según acciones del usuario como ir a otra aplicación, abrir la barra de notificaciones o apagar el teléfono.

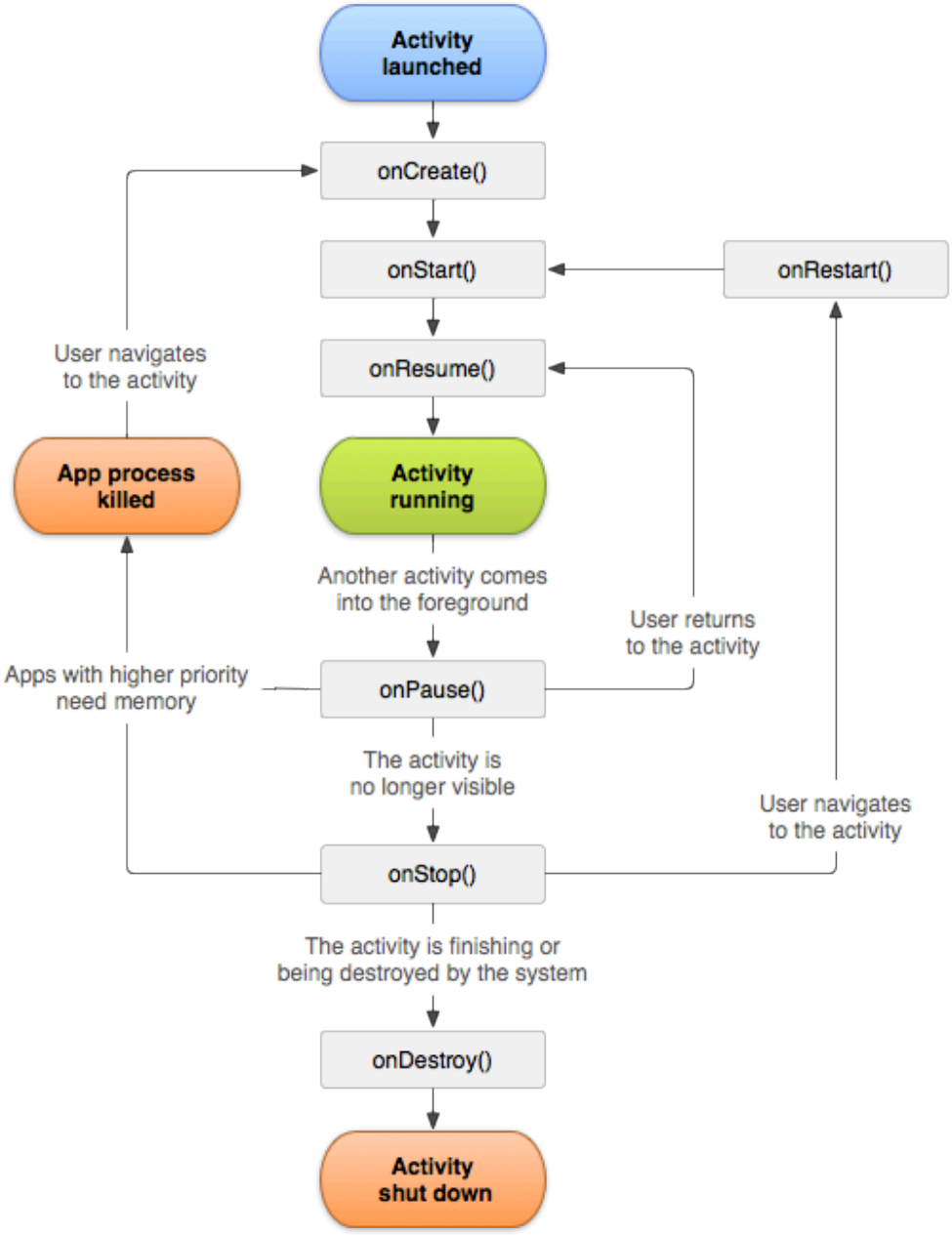

Figura 4-1: Ciclo de vida de una Activity, según documentación oficial de Google [16]

Aunque la gestión del ciclo de vida se realiza de forma automática, todos los estados por los que pasa la Activity tienen un método asociado (onPause(), onStop(), etc.) en los que se pueden programar tareas como modificar la interfaz al iniciar la aplicación o liberar recursos al cerrarla.

Como veremos en la sección 4.2.3 todas las Actividades están ligadas a un archivo

XML que representa la interfaz asociada a la misma, aunque Android permite también modificar la interfaz de forma dinámica en tiempo de ejecución.

4.2.2 Service

Otro de los componentes principales de Android son los Service (Servicios en adelante). Según la documentación oficial de Android [16] un Servicio es un componente que representa una de dos cosas: o bien una operación de larga duración que no requiera de la interacción con el usuario, o bien una funcionalidad creada para que la utilicen otras aplicaciones. Los Servicios, dependiendo del tipo que sean, se pueden iniciar con los métodos Context.startService() y Context.bindService().

Hay que tener en cuenta que los Servicios se ejecutan en el hilo principal, por lo que si el Servicio va a ejecutar operaciones que sean intensivas en uso de CPU debería crear otro hilo de ejecución donde ejecutar esas operaciones.

Los Servicios tienen un ciclo de vida semejante al de las Actividades con estados iguales como onCreate() y onDestroy() y otros propios de los Servicios como onStartCommand(). Android intentará mantener el proceso donde se ejecuta el Servicio siempre que éste se haya iniciado o tenga clientes ligados a él (para los Servicios de tipo Bound Service). Cuando el sistema está bajo de memoria o necesita matar procesos existentes, lo hace según la prioridad que tenga cada uno de ellos. Esta prioridad será mayor si el Servicio está ejecutando código de sus alguno de los métodos que reflejan o si hay clientes ligados a él (para Bound Services). A mayor prioridad menor posibilidad de que el sistema mate ese proceso.

4.2.3 Layout

Como adelantábamos todas las Actividades tienen asociado un archivo XML que define de forma estática la disposición de los elementos que componen la interfaz de usuario. Gracias al plugin ADT para Eclipse (figura 4-2) es posible modificar ese archivo

haciendo “drag&drop” de elementos gráficos disponibles en la paleta de objetos y ver en directo si el efecto visual es el esperado. Esta herramienta permite visualizar los layouts en diferentes dispositivos con tamaños de pantalla y resoluciones diferentes para desarrollar aplicaciones que se vean correctamente en los dispositivos objetivo. Las modificaciones que se hagan con la herramienta gráfica se trasladan automáticamente al archivo XML (figura 4-3) que será compilado junto con el resto de componentes de la aplicación en un archivo.apk instalable en dispositivos Android

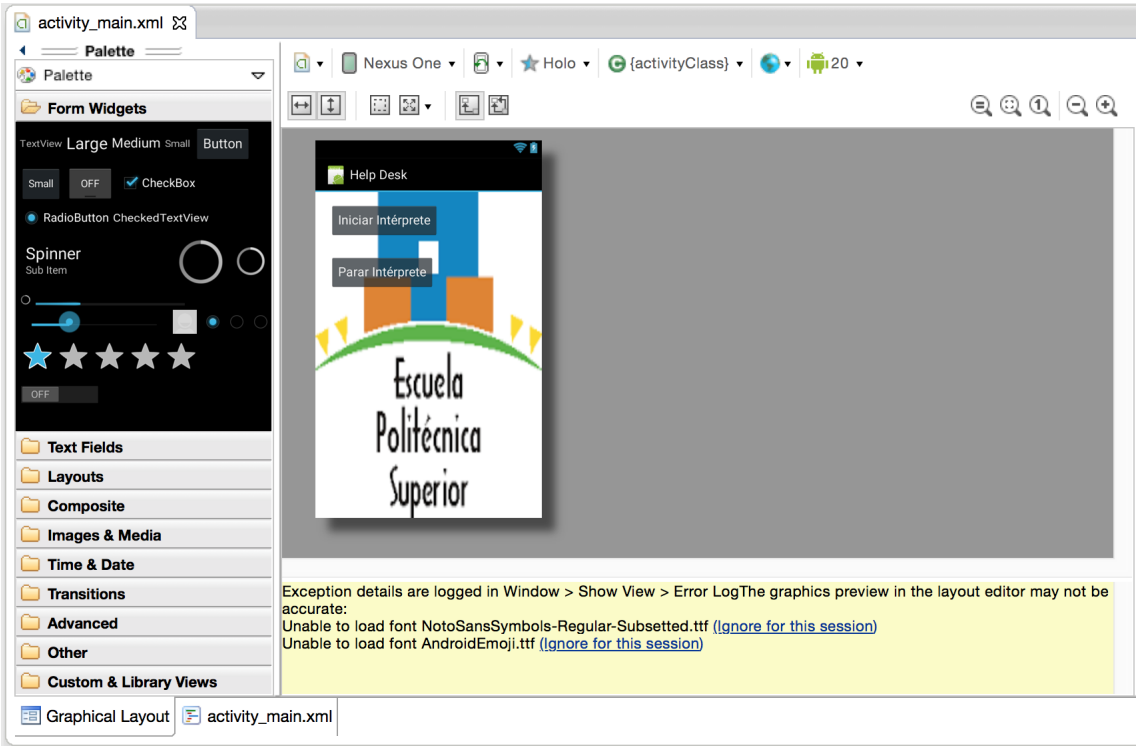

Figura 4-2: Herramienta gráfica para modificar Layouts

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/logo"
    tools:context="${relativePackage}.${activityClass}" >

    <Button
        android:id="@+id/start_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="16dp"
        android:text="Iniciar Intérprete" />

    <Button
        android:id="@+id/stop_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/start_button"
        android:layout_below="@+id/start_button"
        android:layout_marginTop="25dp"
        android:text="Parar Intérprete" />

</RelativeLayout>
```

Figura 4-3: contenido del archivo activity_main.xml que define el layout de MainActivity

Como se comentó anteriormente, los elementos de la UI también se pueden crear y modificar de forma dinámica. Los desarrolladores disponen así de dos métodos diferentes para crear la UI, lo que les permite aumentar la modularidad de la aplicación (utilizando layouts que pueden reutilizar) sin perder flexibilidad (se pueden cargar o modificar layouts por código).

Generalmente algunos elementos de la UI como fondos, botones u otros estarán constituidos por imágenes. Éstas se guardan en las 3 carpetas de nombre *drawable,* cada una pensada para albergar las mismas imágenes pero con resoluciones diferentes. Así al iniciarse una aplicación se cargarán automáticamente las imágenes que mejor se vean según el *screen density* (densidad de pantalla) del dispositivo.

4.2.4 Broadcast receivers

BroadcastReceiver es un componente de Android que permite registrar una aplicación para recibir eventos del sistema o de una aplicación (la misma u otra). Cuando un

evento del tipo que se ha registrado ocurre, el sistema Android notifica a todos los BroadcastReceivers registrados invocando el método onReceive().

Antes de la versión de Android 3.0, no se podía realizar ninguna operación asíncrona en el método onReceive(), ya que una vez que éste finaliza su ejecución el sistema puede reciclar este componente. Desde la versión 3.0 en adelante se puede llamar al método goAsync() que devuelve un objeto del tipo PendingResult. De esta forma el sistema no reciclará (en circunstancias normales) el BroadcastReceiver hasta que se llame al método PendingResult.finish(). Con esta opción se puede ejecutar procesamiento asíncrono en un BroadcastReceiver.

Este componente puede ser registrado de forma estática en el archivo AndroidManifest.xml o de forma dinámica con el método Context.registerReceiver().

Para que una aplicación mande un Broadcast debe utilizar el método sendBroadcast(Intent).

4.2.5 AndroidManifest

El archivo AndroidManifest.xml (figura 4-4) es una parte muy importante de cualquier aplicación Android. Se genera automáticamente cuando creamos un proyecto en Eclipse o Android Studio y en él se declaran todas las especificaciones de nuestra aplicación. Entre estas especificaciones está el nombre de la aplicación, la versión mínima de Android que debe tener un dispositivo para ejecutar esa aplicación, las declaraciones de todas las Actividades, Intents o bibliotecas utilizadas así como los permisos que necesita la aplicación para funcionar y que el usuario deberá aceptar cuando la instale en su dispositivo.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jvoicexml.android.callmanager"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14"
        android:targetSdkVersion="21" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.GET_TASKS" />

    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name" >

        <!-- <uses-library android:name="org.apache.log4j" /> -->


        <activity
            android:name=".MainActivity"
            android:label="@string/title_activity_main" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>


</manifest>

```

Figura 4-4: contenido del archivo del proyecto AndroidManifest.xml

4.2.6 Intent

Según la documentación de Android [18] un Intent es una descripción abstracta de una operación a realizar. Se puede utilizar como argumento de los métodos startActivity() para iniciar una Actividad, sendBroadcast() para mandarlo a cualquier BroadcastReceiver registrado y startService() o bindService() para comunicarse con un Servicio que esté ejecutándose en el background.

Un Intent permite ligar código de diferentes aplicaciones. Su uso más significativo es para iniciar Actividades, donde jugaría el papel de pegamento entre ellas. Técnicamente un Intent es una estructura de datos que contiene una descripción abstracta de la acción

a realizar, además de posiblemente uno o varios objetos dónde almacenar información extra.

4.3 Bloques básicos de VoiceXML

A continuación se hace una breve descripción de los elementos y los conceptos más importantes del estándar tal y como están descritos en la recomendación VoiceXML 2.0 [1].

4.3.1 Bloques

Aunque la mayoría de ellos ya se han descrito en anteriores secciones de este documento conviene recordar los bloques principales que componen un sistema VoiceXML.

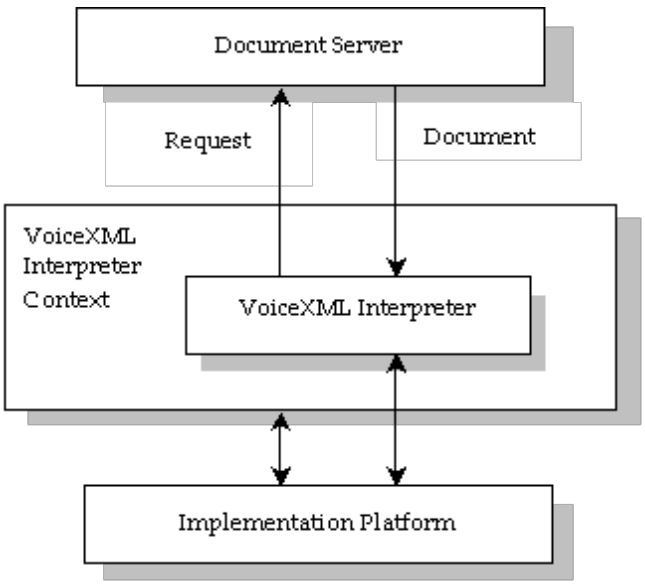

Figura 4-5: Arquitectura de un sistema VoiceXML, extraída de la recomendación VoiceXML 2.0 [1]

Como podemos ver en la figura 4-5, un sistema VoiceXML está compuesto por un Servidor de Documentos, un Intérprete VoiceXML con su Contexto asociado (por simplicidad, en el resto del documento nos referimos a la suma de ambos bloques como "Intérprete") y una Plataforma de Implementación.

Un servidor de documentos (p.ej. un servidor web) procesa peticiones de una aplicación cliente, el Intérprete VoiceXML, a través del Contexto del Intérprete. Ese servidor responde con un documento VXML que será procesado por el Intérprete VoiceXML.

La Plataforma de Implementación es controlada por el Contexto y por el Intérprete. Por ejemplo en una aplicación IVR el Contexto puede ser responsable de detectar una llamada entrante, conseguir el documento VXML inicial y responder la llamada, mientras que el Intérprete gestiona el diálogo después de responder. La Plataforma de Implementación genera eventos en respuesta a acciones del usuario y a eventos del sistema. Algunos de esos eventos activarán una respuesta del Intérprete mientras que otros serán utilizados por el Contexto.

4.3.2 Conceptos

Antes de describir los conceptos más relevantes del estándar VXML conviene explicar de forma sucinta qué es un documento VXML y cómo se ejecuta el código en él contenido. Un documento VXML describe una conversación que puede verse como una máquina de estados finita. El usuario siempre está en un estado de la conversación, o diálogo, en un momento dado. Cada diálogo determina el siguiente diálogo al que ir o el siguiente documento a ejecutar. La ejecución se termina cuando un diálogo no especifica un sucesor, o si tiene un elemento que explícitamente termina la conversación.

4.3.2.1 Diálogos y subdiálogos

Hay dos tipos de diálogos: formularios y menús. Los formularios definen una interacción que recoge del usuario valores para una serie de variables. Mediante las gramáticas (ver sección 4.3.2.4) se pueden los valores permitidos para esas variables. Los menús ofrecen al usuario una serie de opciones. Según la opción escogida la ejecución pasará a un diálogo diferente.

Un subdiálogo es como una llamada a una función, ofrece un mecanismo para invocar una nueva interacción y al finalizar ésta se vuelve al punto de ejecución anterior a la llamada. Los subdiálogos pueden utilizarse por ejemplo para crear una serie de componentes que puedan ser reutilizados por varios documentos de una misma aplicación o por distintas aplicaciones.

4.3.2.2 Sesiones

Una sesión empieza cuando el usuario empieza a interactuar con el Contexto del Intérprete VoiceXML, se mantiene activa mientras los documentos se cargan y procesan y se termina cuando lo solicita el usuario, un documento o el propio Contexto debido por ejemplo a un error.

4.3.2.3 Aplicaciones

Una aplicación es un conjunto de uno o más documentos que comparten el mismo documento raíz. Siempre que un usuario interactúa con un documento en una aplicación, el documento raíz también se carga. Éste permanecerá cargado mientras el usuario se mueve entre los diversos documentos de la aplicación, y se desactiva cuando la interacción con el usuario pasa a un documento que no pertenece a esa aplicación. Mientras el documento raíz está cargado, sus variables son accesibles desde el resto de documentos de la aplicación.

4.3.2.4 Gramáticas

Cada diálogo tiene una o más gramáticas de voz y/o de DTMF (estas últimas no soportadas por el presente trabajo). Hay dos tipos de aplicaciones según quién dirige la interacción, las dirigidas por el sistema y las de iniciativa mixta. En las dirigidas por el sistema las gramáticas de cada diálogo están activas sólo mientras el usuario está en ese diálogo. En las de iniciativa mixta sin embargo, el usuario y el sistema se alternan para decidir qué hacer. En este último tipo de aplicaciones algunos de los diálogos se marcan en el documento para permitir que sus gramáticas permanezcan activas incluso cuando el usuario esté en otro diálogo. En estas aplicaciones si el usuario dice algo que coincida con las gramáticas activas de otro diálogo, la ejecución se trasladará a ese diálogo, tratando las palabras del usuario como si se hubieran dicho allí. Este comportamiento añade flexibilidad y potencia a las aplicaciones VXML.

4.3.3 Etiquetas VXML

El lenguaje de programación VXML es un "lenguaje de etiquetado" que sigue las normas de XML. A continuación se describen algunas de sus etiquetas más relevantes utilizadas en la aplicación demostradora Help Desk y en algunos casos sus atributos más importantes:

<vxml>
Es el elemento de mayor nivel de todo documento. Caracteriza al documento con una serie de atributos y permite diferenciarlo de otros documentos XML

<form>
Los formularios son el principal componente de un documento. Está compuesto por todos o algunos de los siguientes elementos:

- Una serie de *form items,* nodos que se visitan de forma individual. Se subdividen en *input items* que pueden rellenarse por el usuario y *control items*
- Declaraciones de variables no ligadas al formulario
- Controladores de eventos que ejecutan alguna acción cuando ocurre ese evento

- Bloques de lógica que se ejecutan cuando se el input del usuario coincide con alguna gramática activa

<field>

Cada uno de los *input items* que conforman un formulario. Tienen asociado una *item variable* a rellenar por el usuario. Los atributos más relevantes de este elemento son:

- Name: el nombre de la variable que guardará el resultado de la interacción. El nombre debe ser único en el formulario
- Expr: el valor inicial de la *item variable*. Por defecto su valor es indefinido, pero si se inicializa a cualquier valor el *input item* no será visitado por el algoritmo de ejecución a no ser que se borre ese valor o se obligue explícitamente a visitarlo por ejemplo con la etiqueta <go to>
- Cond: una expresión lógica utilizando alguna variable del documento que debe ser cierta para que se visite el *input item.* Es un atributo opcional

<prompt>

Este elemento hace que se ponga en cola automáticamente para su reproducción el texto o audio pregrabado que se incluya en él. Esto implica que la interpretación del documento continuará sin parar en este elemento hasta que la ejecución llegue a un elemento que requiera de input del usuario. Sus atributos más importantes son:

- Bargein: controla si el usuario puede interrumpir la reproducción del prompt
- Cond: condición lógica que debe ser cierta para que se reproduzca el prompt
- Timeout: un límite de tiempo que se aplicará a la siguiente petición de input del usuario
- Count: un contador que permite reproducir diferentes prompts si el usuario hace algo repetidamente, como no decir nada o decir algo que no coincida con ninguna gramática

<grammar>

Permite definir una gramática de voz que:

- Especifica un set de palabras o frases que el usuario puede decir para realizar una acción o introducir información
- Devuelve una interpretación semántica correspondiente a cada frase detectada que coincida con la gramática. Esta interpretación puede tomar la forma de un valor simple (p.ej. una cadena de texto), un set parejas de atributos y valores (p.ej. "Día", "mes" y "año" y sus correspondientes valores), o un objeto anidado (para peticiones complejas)

<filled>

El código contenido entre la apertura y cierre de esta etiqueta se ejecuta cuando el usuario pronuncia alguna palabra o frase que coincida con alguna de las gramáticas activas. Permite efectuar operaciones, ir a otro diálogo, salir del documento o cualquiera de las otras acciones válidas especificadas por el estándar.

<nomatch>

El bloque de código definido por esta etiqueta se ejecuta cuando el input del usuario no coincide con ninguna gramática activa en ese *input item.* Permite ejecutar las mismas operaciones que en el caso de la etiqueta <filled>.

<noinput>

Esta etiqueta se activa cuando se supera el tiempo límite para el input del usuario, el timeout. Como en las dos etiquetas anteriores permite ejecutar diferentes acciones.

<goto>

Esta etiqueta permite cambiar el orden el orden establecido de ejecución del documento obligando a que se ejecute el diálogo especificado en la misma.

<var>

Esta etiqueta permite declarar una variable y asignarle un valor inicial.

4.4 Integración de JVoiceXML en Android

El núcleo de este proyecto consiste en conseguir que un proyecto de código libre escrito en Java que está compuesto por cientos de clases y preparado para funcionar en PC pueda funcionar también en los dispositivos con sistema operativo Android. Para facilitar esta tarea se tomó como base una implementación de JVoiceXML ya existente y funcional, la TextImplementation (sus clases están disponibles en la página del proyecto en GitHub [19]). Como su propio nombre indica esta implementación es una extensión de JVoiceXML en el que los diálogos, los inputs del usuario y los outputs del sistema son texto y no voz, pero sirve como punto de arranque del proyecto. Todas las clases de la Plataforma de Implementación (descritas en la sección 4.7) tienen su base en sus contrapartes de la TextImplementation aunque todas ellas fueron modificadas para cumplir los requisitos de este proyecto, en algunos casos implementando hasta 4 interfaces más y en muchos casos creando métodos propios.

En las siguientes dos secciones se explican las decisiones tomadas en torno a dos de las tareas más importantes de este proyecto, encontrar e implementar una estructura de clases y componentes Android que permitiera el correcto funcionamiento del Intérprete JVoiceXML en Android y conseguir que el texto reconocido por el ASR se contrastará con las gramáticas y permitiera al Intérprete tomar decisiones según hubiera o no coincidencia.

4.4.1 Cambio en la estructura del proyecto

Para intentar que el proyecto funcionara como un servicio a utilizar por futuras aplicaciones Android que quisieran ejecutar documentos VXML para mejorar la accesibilidad o por otras razones, se diseñó inicialmente la estructura Android tal y como se presenta en la figura 3-1. En esa estructura el código del Intérprete corre sobre el hilo del Bound Service llamado AndroidCallmanager al que se ligaría una aplicación Android mediante el comando bind().

Una vez se resolvieron las excepciones y problemas resultantes de incompatibilidades del código JVoiceXML con la plataforma Android (ver sección 5) y se implementaron las clases de la PI y los componentes necesarios para implementar la estructura diseñada se vió que ésta presentaba un problema: el ASR tal y como está programado es una Actividad que se inicia con el método startActivityForResult(Intent) y éste sólo puede ser llamado por otra Actividad, y como se puede ver en la figura 3-1 el código de este proyecto estaba repartido entre varias clases, pero ninguna de ellas una Actividad.

Para solucionar esto se probaron distintas estrategias:

- Hacer que la clase AndroidSpokenInput (encargada de la gestión del input por voz del usuario) extendiera la clase Activity e invocara el método startActivityForResult()
- Intentar hacer cast del Servicio a Activity para iniciar el ASR desde allí
- Iniciar una Actividad desde el Servicio que a su vez iniciara el ASR (ya que el resultado del reconocimiento se envía a la función onActivityResult() de la Actividad que lo inicia)
- Extender la clase SpeechRecognizer para permitir iniciar el reconocimiento desde el Servicio

Por diferentes motivos ninguna de estas estrategias fue exitosa por lo que se decidió modificar la estructura completa del proyecto que fue la que se implementó finalmente y que está descrita en la siguiente figura.

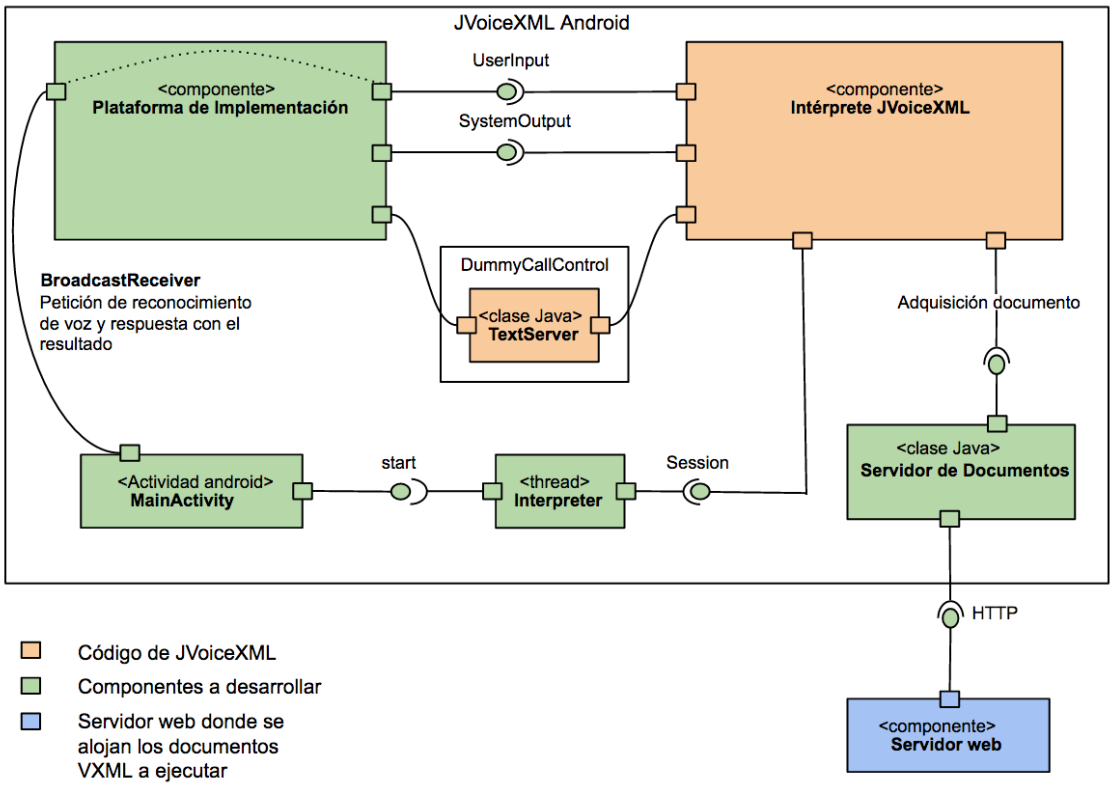

Figura 4-6: estructura definitiva de la portabilidad

Los cambios más importantes son:

- Se cambia el Servicio por un hilo Java donde corre tanto el Intérprete JVoiceXML como la PI
- Se crea una Actividad (MainActivity) que genera el hilo Interpreter y le pasa una URI que referencia a un documento VXML para iniciar la ejecución del mismo
- Se implementa un BroadcastReceiver en MainActivity para recibir la petición desde la PI de creación del motor TTS y devolver la referencia al mismo
- Se implementa un BroadcastReceiver en MainActivity para recibir la petición desde la PI para abrir la Actividad del ASR y devolver el resultado del reconocimiento mediante un Intent
- Se crea un objeto de tipo TextServer en la clase AndroidPlatformImplementationFactory al que se conecta el Intérprete y que permite emular una conexión telefónica entre cliente (clase Interpreter) y el servidor (TextServer).

La siguiente sección describe la forma en la que se implementaron tanto estos Broadcast Receivers como el TextServer.

4.4.2 Comunicación Android-Intérprete

Como se vio en la sección 4.2, los Broadcast Receivers permiten mandar un mensaje que puedan recibir tanto otros componentes de una misma aplicación como otras aplicaciones. Esta capacidad de que otras aplicaciones escuchen los mensajes así como que envíen Broadcast que interactúen con nuestra aplicación genera un agujero de seguridad. Por ello en este proyecto se decide utilizar un tipo de envío más seguro utilizando la clase LocalBroadcastManager, que según la documentación oficial [20] tiene las siguientes ventajas:

- La información que se envía no saldrá de la aplicación, por lo que no hay riesgo de que se filtre información privada
- Otras aplicaciones no pueden enviar Broadcasts de este tipo a tu aplicación, evitando el riesgo de tener agujeros de seguridad potencialmente explotables
- Es más eficiente que enviar un Broadcast global a través de todo el sistema

Para enviar un Broadcast de esta manera se utiliza el siguiente método:

LocalBroadcastManager.getInstance(this.context).sendBroadcast(intent);

Para utilizar este método desde la PI se necesita el *contexto* de Android. Éste es muy fácilmente asequible desde un Actividad o un Servicio con la función getApplicationContext() o incluso utilizar la palabra clave *this* para según qué operaciones ya que estas clases heredan de la clase abstracta Context. En este proyecto en cambio se necesitaba utilizar el contexto desde clases Java tradicionales y para ello se les pasa una referencia al contexto en su inicialización.
Para lograr esto se tuvo que propagar el contexto desde MainActivity hasta las clases que lo necesitaban. Esto requirió varias fases:

- Se añaden dos métodos getContext() y setContext() a la clases AndroidConfiguration, AndroidSpokenInput y AndroidSynthesizedOutput

- Se añade una variable de tipo Context en los constructores de la clase Interpreter y y AndroidPlatformImplementationFactory
- Cuando MainActivity crea el hilo Interpreter le pasa el contexto utilizando la palabra clave *this*
- En el método run() de la clase Interpreter se crea un objeto de tipo AndroidConfiguration y se le pasa el contexto con su método setContext()
- Cuando la clase Interpreter inicia el hilo JVoiceXmlMain éste utiliza la clase AndroidConfiguration para crear una una instancia de AndroidImplementationPlatformFactory creada ya con el contexto
- JVoiceXmlMain ejecuta el método init() de AndroidImplementationPlatformFactory donde se crean los objetos de tipo AndroidSpokenInput y AndroidSynthesizedOutput. Acto seguido se utilizan sus métodos setContext() para que guarden copia local de la referencia al contexto.

Para poder comunicar la PI con la clase MainActivity ya sólo faltaba registrar sendos BroadcastReceivers en MainActivity, lo cual se hace con el código de la figura 4-7.

```
recognitionReceiver = createRecognitionReceiver();
LocalBroadcastManager.getInstance(this).registerReceiver(
        recognitionReceiver,
        new IntentFilter("android.spoken.input.startrecognition"));

Log.v("MainActivity.onCreate()", "created recognition BroadcastReciever");

TTSReceiver = createTTSReceiver();
LocalBroadcastManager.getInstance(this).registerReceiver(
        TTSReceiver,
        new IntentFilter("android.spoken.output.initializeTTS"));

Log.v("MainActivity.onCreate()", "created texttoSpeech BroadcastReciever");
```

Figura 4-7: registro de los BroadcastReceivers en MainActivity

En la figura anterior se observa cómo se crean los BroadcastReceivers en dos funciones para mayor modularidad. En la función createRecognitionReceiver se crea el BroadcastReceiver y se define el código que se ejecuta cuando se detecta un Broadcast creado con un Intent con la acción “android.spoken.input.startrecognition”. Este código crea un Intent y lo configura con los parámetros adecuados (ver sección 4.6) para iniciar el reconocimiento de voz. Después en el método onActivityResult() se comprueba que el ASR no haya tenido problemas y se envía los resultados a otra función,

recognitionFinished(), que será la que guarde esos resultados en el Intent original que forma parte del Broadcast enviado por la clase AndroidSpokenInput.

La función creatTTSReceiver crea el BroadcastReceiver y define el código que se ejecuta cuando se detecta un Broadcast con el Intent "android.spoken.output.initializeTTS". En este caso sólo se utiliza una vez este Broadcast ya que el código a ejecutar en este caso es la creación e inicialización del motor TTS (ver sección 4.5) y su envío al AndroidSynthesizedOutput con el método AndroidSynthesizedOutput.setSpeechEngine().

Cuando se implementaron estos cambios la aplicación ya era capaz de sintetizar correctamente los mensajes al usuario contenidos en los documentos VXML de prueba, pero fallaba a la hora de comprobar si el texto reconocido por el ASR coincidía con las gramáticas de esos documentos. La expresión concreta del problema era un error generado en el Intérprete JVoiceXML cuya traza parcial puede verse en la siguiente figura:

```
org.jvoicexml.event.error.NoresourceError: error.noresource: Pool o ⏎
f type 'android' is unknown!
    at org.jvoicexml.implementation.pool.KeyedResourcePool.borrowObjec ⏎
t(KeyedResourcePool.java:113)
    at org.jvoicexml.android.callmanager.AndroidImplementationPlatform ⏎
Factory.getExternalResourceFromPool(AndroidImplementationPlatformFa ⏎
ctory.java:401)
    at org.jvoicexml.android.callmanager.AndroidImplementationPlatform ⏎
Factory.getCallControl(AndroidImplementationPlatformFactory.java:19 ⏎
7)
    at org.jvoicexml.implementation.jvxml.JVoiceXmlPromptAccumulator.r ⏎
enderPrompts(JVoiceXmlPromptAccumulator.java:124)
    at org.jvoicexml.android.callmanager.AndroidImplementationPlatform ⏎
Factory.renderPrompts(AndroidImplementationPlatformFactory.java:294 ⏎
)
```

Figura 4-8: traza parcial del error noresource

Después de muchas pruebas infructuosas se decidió investigar cómo había resuelto esta problemática el alumno checo Martin Nekula en su propia portabilidad Android (ver sección 2.4.1). Esto fue posible gracias a que el código de Martin está disponible en la rama android de la página del repositorio de JVoiceXML en SourceForge [21]. De su código se copiaron las líneas de código mostradas en las siguientes figuras:

```
textServer = new TextServer(4242);
textServer.addTextListener(this);
textServer.start();

 synchronized (textServer) {
        try {
            textServer.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Figura 4-9: código de la clase AndroidImplementationPlatformFactory copiado de la portabilidad de Martin Nekula

```
client = new TextConnectionInformation(4242);
```

Figura 4-10: código de la clase Interpreter copiado de la portabilidad de Martin Nekula

Con estas líneas de código se pudo resolver el error anterior. Esto se explica porque el sistema JVoiceXML está diseñado para permitir el control de una llamada telefónica, en la que la persona que llama es el cliente y el Intérprete alojado en un ordenador es el servidor. Ambas clases TextServer y TextConnectionInformation fueron creadas por el administrador Dirk Schnelle-Walka y están disponibles en la página de JVoiceXML del repositorio GitHub [19].

Aunque este proyecto no tiene como requisito el poder gestionar una llamada telefónica el Intérprete JVoiceXML necesita establecer esa conexión cliente-servidor para su correcto funcionamiento.

En la implementación que tanto Martin como el autor utilizaron como base de sus respectivos proyectos, la TextImplementation, el TextServer se utiliza para recibir los mensajes introducidos por texto por el usuario y mandarlos al Intérprete para su procesado. No es el caso en el proyecto que nos ocupa, ya que la comprobación de coincidencia entre los resultados del reconocimiento de voz y las gramáticas de la aplicación VXML se realiza en el método AndroidSpokenInput.notifyRecognitionResults(). En este proyecto la clase TextServer

se utiliza sólo para que el Intérprete pueda crear esa conexión cliente-servidor aunque no se envíe información a través de ella.

4.5 Motor TTS

El motor TextToSpeech permite sintetizar audio a partir de una cadena de texto. Para poder utilizarlo en una aplicación Android se deben cumplir una serie de requisitos:

- El dispositivo debe tener una versión de Android mayor o igual a la 1.6. Esto en la actualidad lo cumplen la inmensa mayoría de dispositivos
- Descarga en el dispositivo del lenguaje a utilizar en Settings -> Idioma e introducción de texto -> Síntesis de voz -> Síntesis de Google -> Instalar archivos de voz
- Tener acceso al contexto de Android en la clase que vaya a crear el motor e implementar el *listener* onInitLister para saber cuándo y si se ha inicializado el motor correctamente

Android permite modificar varios parámetros del motor, desde cambiar el lenguaje (funcionará si se tienen descargados los archivos correspondientes) hasta ralentizar o acelerar la velocidad con la que se sintetiza el texto, pasando por cambiar la voz que "lee" el texto o modificar su tono.

La configuración utilizada en este proyecto es la siguiente:

- Lenguaje: español
- Velocidad de lectura: 1.5 veces la velocidad estándar
- Voz: la estándar, no se modifica
- Tono: el estándar, no se modifica

4.6 Motor ASR

El Android Speech Recognizer permite reconocer la voz humana en multitud de lenguajes y traducirla a texto. Para poder utilizar esta funcionalidad de Android se deben cumplir los siguientes requisitos:

- El dispositivo debe tener una versión de Android igual o mayor que la 1.6.
- Se debe tener conexión a Internet o descargar los archivos del lenguaje a reconocer. Para descargarlos ir a Ajustes -> Idioma e introducción de texto -> Dictado por voz de Google -> Reconocimiento de voz sin conexión
    - No todas las distribuciones de Android permiten el reconocimiento offline
- Es necesario contar con una conexión a Internet en el momento del reconocimiento ya que éste se lleva a cabo en los servidores de Google
- En su forma de uso estándar se crea mediante un RecognizerIntent con el método startActivityForResult() que inicia la Activity del reconocedor (figura 4-11) Esto implica que deberá ser la actividad en primer plano la que inicie el ASR

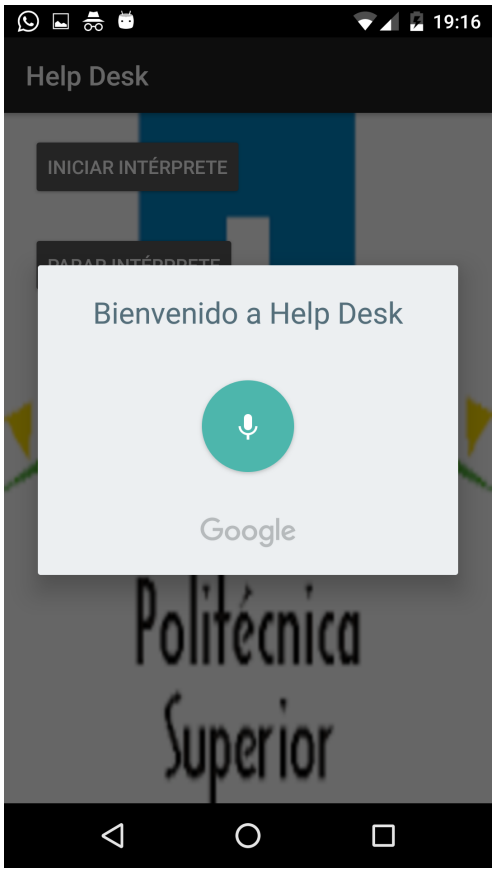

Figura 4-11: Imagen de la Actividad del ASR

Para aumentar la precisión del reconocimiento y para facilitar el uso de los resultados Android permite configurar una serie de parámetros del RecognizerIntent modifican el comportamiento del ASR. Entre ellos cabe destacar el poder cambiar de lenguaje, especificar que sólo se utilice el reconocimiento offline, modificar el texto que aparece en la Actividad, escoger entre diferentes modelos de reconocimiento o limitar el número de resultados que se devuelve el ASR.

En este proyecto se utilizaron los siguientes parámetros de configuración:

- Lenguaje: español
- Modelo de reconocimiento: free form
- Texto de la Actividad: “Bienvenido a Help Desk”
- Número de resultados máximo: 5
- Sólo offline: no

Un punto interesante del ASR es que puede devolver uno o varios resultados del reconocimiento. Hay veces que el sistema no está seguro de qué dijo el usuario y devuelve varias interpretaciones diferentes. Cuando esto ocurre el sistema puede devolver (Android no lo asegura) en el propio RecognizerIntent un array con un nivel de confianza para cada resultado. Este nivel es un número real que va desde el 1.0 (poca confianza) al 2.0 (mucha confianza).

4.7 Descripción de las clases

A continuación se describen brevemente las clases implementadas en el proyecto.

4.7.1 MainActivity

- Tipo de clase: Activity
- Layout asociado: activity_main.xml
- Hilo de ejecución: UI thread

- Descripción de la clase: Es la clase principal de la aplicación. Toda aplicación Android debe tener al menos una actividad que pueda ser utilizada como entrada a la aplicación, y se define en el AndroidManifest con el valor "android.intent.action.MAIN". En el manifiesto del proyecto (ver figura 4-4) esta actividad se define también como LAUNCHER. Esto implica que será la actividad lanzada cuando el usuario haga click sobre su icono en el menú de aplicaciones de su dispositivo. Esta actividad cuenta con dos botones definidos de forma estática (en el layout asociado) y cada uno cuenta con un *listener* definido por el método de la clase Button "setOnClickListener" que permite ejecutar código cuando detecta el evento onClick. El botón Iniciar Intérprete crea una clase Interpreter y le pasa en el constructor la URI del documento VXML a ejecutar. Después ejecuta el método Interpreter.start() para iniciar la ejecución de ese hilo, ya que como se explicó anteriormente las operaciones de larga duración no pueden ejecutarse en el UI thread. El botón Parar Intérprete ejecuta los métodos necesarios para liberar los recursos reservados y matar los hilos creados.

  Además de esto y como se vio en la sección 4.4.2, esta actividad es la encargada de iniciar los motores TTS y ASR, hacer su configuración inicial y pasar referencia del TTS a la Plataforma de Implementación para su control.

  Esta actividad también se encarga de pasar los resultados obtenidos del reconocimiento de voz iniciado con el método "startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE)". Mientras se reciben los resultados del ASR se debe pausar la ejecución del hilo de la PI para no incurrir en problemas de concurrencia. Una vez recibidos los resultados se guardan en un Intent creado por la PI y se reactiva el hilo pausado que ya puede utilizar de forma segura el resultado del reconocimiento de voz.

- Visualización del layout: La siguiente figura muestra el layout de Mainctivity:

Figura 4-12: Layout de la clase MainActivity

4.7.2 Interpreter

- Tipo de clase: hilo Java
- Hilo de ejecución: es en sí mismo un hilo
- Descripción de la clase: Esta clase extiende la clase Thread (hilo) e implementa la interfaz del core de JVoiceXML, JVoiceXmlMainListener.
  Como ya se ha mencionado se optó por utilizar un hilo para ejecutar en él el Intérprete JVoiceXML y no bloquear el UI thread.

  En esta clase se crea y se mantiene referencia a la clase principal del *core,* JVoiceXmlMain y se añade a su lista de *listeners* la instancia de la clase Interpreter gracias a que éste implementa la interfaz JVoiceXmlMainListener.

Esto permite pausar la ejecución del hilo Interpreter hasta que finalice la inicialización y configuración del Intérprete. Cuando esto ocurra éste llamará al método jvxmlStarted() del Intérprete para que continúe la ejecución (ver figura 4-13).

```
public synchronized void jvxmlStarted() {
    synchronized(this){
        this.notifyAll();
    }
}
```

Figura 4-13: Código del método Interpreter.jvxmlStarted()

El otro método heredado de la interfaz JVoiceXmlMainListener es jvxmlTerminated(), invocado cuando se cierra el Intérprete. Y por último se crea un método finish() para que desde MainActivity se pueda cerrar el Intérprete y liberar todos los recursos y luego poder matar a su vez el hilo Interpreter.

4.7.3 AndroidDocumentServer

- Tipo de clase: clase Java
- Hilo de ejecución: hilo Interpreter
- Descripción de la clase: Esta clase es la encargada de descargar los recursos que necesita el contexto del Intérprete VoiceXML para su correcta ejecución. Para descargarlos utiliza el URI de los mismos. Los recursos más importantes son los propios documentos VXML (método getDocument()), las gramáticas externas (método getGrammarDocument()) y los archivos de audio para su reproducción (método getAudioInputStream()).También se encarga de la gestión de errores trasladando al Intéprete los errores de timeout, de URIs mal formadas o de errores diversos del servidor donde esté alojado el recurso a descargar, entre otros.

El código de esta clase fue proporcionado en su gran mayoría por el administrador del proyecto JVoiceXML, Dirk Schnelle-Walka.

4.7.4 AndroidConfiguration

- Tipo de clase: clase Java
- Hilo de ejecución: hilo Interpreter
- Descripción de la clase: Esta clase permite al Intérprete JVoiceXML configurarse en tiempo de ejecución creando instancias de clases diferentes según la plataforma en la que se esté ejecutando. Cada extensión de JVoiceXML debe implementar esta clase (que hereda de la interfaz Configuration) y modificar los tipos de clases que devuelven sus diferentes métodos.

  Para poder lograr la creación de instancias de una clase u otra dependiendo de la extensión se utilizan los llamados tipos genéricos de Java (<T>). En el método run() de la clase JVoiceXmlMain se configura el Intérprete y se llama al método loadObject(*ClasePadre*.class) de la clase AndroidConfiguration (ver figura 4-14) pasándole la clase padre de la que heredará la clase en cuestión de la PI. A su vez el método loadObject devuelve un objeto de tipo genérico <T> T, y su argumento está definido como una clase de tipo genérico (Class<T> baseClass), tal y como se aprecia en la figura 4-15.

```
try {
    implementationPlatformFactory = configuration.loadObject(
              ImplementationPlatformFactory.class);
    implementationPlatformFactory.init(config);
} catch (Exception e) {
    LOGGER.error(e.getMessage(), e);
    Log.e("implementationFactory",e.getMessage());
    synchronized (shutdownSemaphore) {
        shutdownSemaphore.notifyAll();
    }
    return;
}
```

Figura 4-14: extracto del método JVoiceXmlMain.run()

```
public <T> T loadObject(final Class<T> baseClass) {
```

Figura 4-15: definición del método AndroidConfiguration.loadObject

En el caso de este proyecto se modificó la clase DummyConfiguration para devolver instancias de las clases de la PI implementada, en concreto las clases AndroidImplementationPlatformFactory y AndroidDocumentServer:

```
} else if (baseClass == DocumentServer.class) {
    return (T) new AndroidDocumentServer();
    else if (baseClass == ImplementationPlatformFactory.class) {
        try {
            return (T) new AndroidImplementationPlatformFactory(callManagerContext);
        } catch (Exception e) {
            Log.e("creando AndroidImplementationPlatformFactory",e.getMessage());
            return null;
        }
```

Figura 4-16: extracto del método AndroidConfiguration.loadObject()

Esta forma de generar instancias diferentes tipos según la clase AndroidConfiguration generó la excepción ClassNotFoundException debido a la estructura del proyecto y las dependencias entre los distintos paquetes. Para solucionarlo se decidió juntar todas las clases implementadas en un único paquete resolviendo así la excepción.

4.7.5 Plataforma de Implementación

Como vimos en la sección 3.3.2 este módulo se encarga de gestionar los recursos externos al Intérprete como los motores ASR y TTS, y de generar eventos en respuesta a acciones del usuario o a eventos del sistema. Cada extensión del sistema JVoiceXML, cada portabilidad a una plataforma diferente a las ya soportadas requiere de la implementación de una PI compatible con los recursos y métodos de esa plataforma en particular. A continuación se describirán las diferentes clases que componen la PI implementada para dar soporte a JVoiceXML en Android.

4.7.5.1 AndroidImplementationPlatformFactory

- Tipo de clase: clase Java

- Hilo de ejecución: hilo Interpreter
- Descripción de la clase: Esta es la clase central de la PI y contiene referencias a las instancias del resto de clases de la PI que se describirán en las siguientes páginas.

  La clase JVoiceXmlMain crea una instancia de esta clase en su método run() y propaga la referencia a las distintas clases pertinentes del Intérprete y del VoiceXML Context, en particular a la clase FormInterpretationAlgorithm. Como ya hemos visto el FIA contiene el algoritmo de procesado de los documentos VXML y se comunica con el AndroidImplementationPlatformFactory para conseguir referencias a los distintos recursos de la plataforma, para sintetizar diálogos o para comunicar errores.

  El apellido "Factory" de ésta y otras clases que veremos a continuación implica que éstas deben ser capaces de generar múltiples instancias de la clase a la que representan para soportar la ejecución concurrente de varios documentos VXML al mismo tiempo. Debido a que esta concurrencia no era requisito del proyecto y queda fuera del ámbito del mismo no se garantiza que el desarrollo actual permita ejecutar varios documentos VXML a la vez.

  Esta clase hereda métodos de cinco clases Java de tipo interfaz, tres de ellas *listeners*. Algunos de estos métodos heredados están vacíos como los heredados de la clase TelephonyListener ya que no se requiere soporte para llamadas telefónicas y sólo se implementa esta interfaz para evitar excepciones en el *core* de JVoiceXML o tener que modificar el código del mismo.

  Un listener importante de los que hereda es el TextListener. Cuando se crea el TextServer en el método init() (figura 4-17) se registra el AndroidImplementationPlatformFactory como *listener*. Esto parar la ejecución del hilo Interpreter con el método wait() sincronizado sobre el objeto TextServer mientras se inicializa y configura el TextServer y después reiniciarlo con el método notifyAll().

```
textServer = new TextServer(4242);
textServer.addTextListener(this);
textServer.start();

Log.e("AndroidImplementationPlatformFactory.init()","esperando a que el TextServer se inicialice");
 synchronized (textServer) {
        try {
            textServer.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

 Log.e("AndroidImplementationPlatformFactory.init()"," TextServer inicializado, salimos del wait");
```

Figura 4-17: Inicialización del TextServer

4.7.5.2 AndroidSpokenInput

- Tipo de clase: clase Java
- Hilo de ejecución: hilo Interpreter
- Descripción de la clase: Esta clase se encarga de controlar y monitorear los inputs del usuario por medio de la voz así como de propagar los eventos relacionados al Intérprete.

  Como vimos en la sección 4.4.2, el motor ASR lo gestiona la clase MainActivity ya que en su forma estándar el ASR es una Actividad, y por tanto para invocarla se utiliza el método startActivityForResult() que sólo está disponible para las clases de tipo Actividad. Por ello para poder iniciar el reconocimiento de voz, la clase AndroidSpokenInput utiliza un Broadcast para enviar una petición de inicio de reconocimiento a la MainActivity con un objeto Intent donde guardar el resultado del reconocimiento. De la misma manera que hemos visto anteriormente se detiene la ejecución del hilo Interpreter hasta que llega un resultado del ASR. Una vez llega este resultado se comunica al Intérprete y se le pregunta si el resultado coincide con alguna gramática. Si coincide se genera un evento de tipo *acceptedEvent* que se comunica al Intérprete a través de la clase AndroidImplementationPlatformFactory. En caso contrario se genera un evento de tipo *rejectedEvent* y se comunica de igual forma.

Las gramáticas pueden estar activas en todo el documento VXML, en un diálogo concreto o en un nodo, entre otros. Esta clase se encarga también de activar y desactivar las gramáticas contra las que se comprueba si el resultado reconocido coincide o no. Por último esta clase se encarga de liberar los recursos utilizados cuando se cierra el Intérprete o en otros casos particulares.

4.7.5.3 AndroidSpokenInputFactory

- Tipo de clase: clase Java
- Hilo de ejecución: hilo Interpreter
- Descripción de la clase: Esta clase se encarga de crear múltiples instancias de la clase AndroidSpokenInput y mantener un contador de las mismas. Como se dijo anteriormente esta clase no se utiliza en el proyecto ya que no se necesita ejecutar varios documentos VXML de forma concurrente.

4.7.5.4 AndroidSynthesizedOutput

- Tipo de clase: clase Java
- Hilo de ejecución: hilo Interpreter
- Descripción de la clase: Esta clase se encarga de gestionar y controlar los recursos que tiene que ver con la síntesis de voz. Como ya se explicó esta clase permite mandar un Broadcast en su método open() para que el MainActivity cree el motor TTS, y para la ejecución del hilo Interpreter mientras el TTS se inicializa. Para poder reiniciar el hilo esta clase implementa la clase OnInitListener, y así cuando el TTS termina de inicializarse se hace una llamada al método onInit() donde se ejecuta el método notifyAll().

  Esta clase contiene un objeto Java de tipo Queue (cola) que almacena los textos a sintetizar por el TTS y después los va sintetizando uno por uno. También es esta clase la que permitiría soportar la funcionalidad de barge in (permitir al usuario interrumpir la síntesis de voz con algún comando o elección), que borraría todos los textos a sintetizar que no tuvieran definido en el documento

VXML que no se les puede interrumpir (atributo *bargein="false"*). Esta funcionalidad no está soportada por esta portabilidad ya que requiere extender y modificar el motor ASR y queda fuera de los objetivos, por lo que se propone como trabajo futuro en la sección 6.

Otra de las funciones importantes de esta clase es propagar eventos relacionados con la síntesis de voz al Intérprete JVoiceXML.
También utilizando los métodos de esta clase se puede comprobar si el TTS soporta la síntesis de voz en un lenguaje determinado y cambiar el lenguaje en el que se sintetiza la voz. Y como en casos anteriores esta clase libera la memoria y los recursos utilizados cuando se le notifica que se cierra el Intérprete.

4.7.5.5 AndroidSynthesizedOutputFactory

- Tipo de clase: clase Java
- Hilo de ejecución: hilo Interpreter
- Descripción de la clase: Esta clase se encarga de crear múltiples instancias de la clase AndroidSynthesizedOutput y mantener un contador de las mismas. Como ya se ha dicho anteriormente esta clase no se utiliza en el proyecto ya que no se necesita ejecutar varios documentos VXML de forma concurrente.

5 Pruebas realizadas

El intérprete JVoiceXML junto con sus módulos adyacentes está compuesto de 7 proyectos con decenas de clases cada uno, mientras que las clases creadas por mi desde cero o las que implementan y extienden clases existentes de JVoiceXML son sólo 9. Debido a ello el diseño sólo pudo ser parcialmente modular. Esto implica que haya sido necesario una mezcla de pruebas unitarias y de integración que en gran medida no pudieron ser planificadas de antemano.

Durante todo el desarrollo del proyecto se realizaron pruebas exhaustivas de diferentes configuraciones de componentes Android, de la comunicación Android-JVoiceXML, de los motores TTS y ASR, de la aplicación Help Desk, etc. Lo que sigue es una descripción de las pruebas más relevantes ordenadas según la fase de desarrollo del proyecto en las que se ejecutaron y haciendo énfasis en aquellas que condujeron a cambios en la implementación.

Todas las pruebas se realizaron depurando o ejecutando el código en un móvil Nexus 5 con Android > 4.0 (durante el tiempo de duración del proyecto la versión Android del móvil se fue actualizando).

5.1 Configuración e inicio del JVoiceXML voice browser

La clase principal del intérprete es JVoiceXmlMain que extiende la clase Java Thread. Una instancia de esta clase se crea y se inicia en su método run() dentro del método run() de la clase Interpreter.

En este caso se quería probar que el intérprete JVoiceXML se inicializara y configurara correctamente. Para ello se depuró el proyecto poniendo un breakpoint en la línea 67 de la clase Interpreter (ver figura 5-1). Este hilo de ejecución se pausa en la línea 62 por lo que si la ejecución llegaba a la línea 67 significaría que la clase JVoiceXmlMain no habría generado ninguna excepción y que habría llamado al método jvxmlStarted() de la clase Interpreter donde se reanuda la ejecución del hilo. En caso contrario se vería la

traza del error en Eclipse y se pasaría a afinar la búsqueda del origen del error añadiendo breakpoints en puntos anteriores de ejecución para poder depurar con más precisión la aplicación, como finalmente ocurrió.

Esta prueba se repitió iterativamente eliminando problemas con tres causas diferentes:

- una mala implementación de la clase AndroidConfiguration (la variable *config* es instancia suya) que se corrigió fácilmente
- incompatibilidades entre AndroidConfiguration y la clase JVoiceXmlMain. Se tuvo que modificar ésta última y pedir al gestor de JVoiceXML que copiara el cambio en el repositorio SourceForge
- ClassNotFoundException causada porque el Android Java Runtime Machine no soporta las clases de la librería *javax.sound.sampled*. De nuevo hubo que modificar dos clases del core de JVoiceXML cambiando el tipo de los argumentos de varios métodos de AudioInputStream a InputStream que sí es soportado por Android

```
55          jvxml = new JVoiceXmlMain(config);
56          jvxml.addListener(this);
57          jvxml.start();
58
59
60          synchronized (this) {
61              try {
62                  this.wait();
63              } catch (InterruptedException e) {
64                  e.printStackTrace();
65              }
66          }
67          Log.e("Interpreter","el Intérprete terminó su inicialización");
68
```

Figura 5-1: Extracto de la clase Interpreter.java

Después de estos cambios el intérprete se inició y configuró correctamente.

5.2 Pruebas TTS y ASR

Antes de incluir en el proyecto los motores TTS y ASR se probó en una Actividad vacía el funcionamiento de ambos motores y se probaron diferentes configuraciones de los mismos así como comprobar que las funciones *callback* (funciones que no se llaman directamente por código si no que son llamadas por el sistema cuando ocurre algún evento) ejecutadas después de la inicialización de los motores o cuando se reciben resultados del ASR funcionaran correctamente. De estas pruebas salió la configuración utilizada en el proyecto y descrita en las secciones 4.5 y 4.6.

Una vez probados por separado se integró el código generado en las diferentes clases de la Plataforma de Implementación. Al integrarlo surgió un problema que hizo necesaria una reorganización completa del proyecto; el ASR debía ser llamado desde el *UI Thread,* es decir, el hilo de la Actividad cuyo *layout* esté viendo el usuario en ese momento. El problema era que la llamada se hacía desde el hilo del Intérprete y generaba una excepción. Además en la configuración normal, el ASR devuelve el resultado en la función *callback* onActivityResult() despué8 s de iniciar el ASR con la función startActivityForResult(RecognizerIntent) que en principio sólo estaba definida para una Actividad y no para un Servicio.

Para solucionarlo se buscaron y probaron varias formas de hacer la llamada desde el BoundService sin éxito, por lo que se decidió modificar la estructura del proyecto eliminando el servicio y aplicando la solución descrita en la sección 4.4 que utiliza BroadcastReceivers tanto para inicializar los motores como para la comunicación de resultados y peticiones entre el MainActivity y la Implementation Platform.

5.3 Ejecutando documentos VXML

Durante todo el proceso de desarrollo del proyecto se utilizó un documento VXML muy sencillo (“Hello World”) para probar primero la funcionalidad básica. Una vez que se terminó el grueso del trabajo se sometió el proyecto a pruebas con varios documentos diferentes centrados cada uno en una funcionalidad diferente.

A continuación se describen algunas de esas pruebas, los resultados que ofrecieron así como las decisiones o conclusiones que se derivaron de ellas.

5.3.1 ECMAScript

Como podemos leer en Wikipedia [22]:

> *ECMAScript es una especificación de lenguaje de programación publicada por ECMA International. El desarrollo empezó en 1996 y estuvo basado en el popular lenguaje JavaScript propuesto como estándar por Netscape Communications Corporation. Actualmente está aceptado como el estándar ISO 16262.*

ECMAScript es el lenguaje de scripting que se utiliza en el estándar VXML. Permite hacer operaciones sobre las variables generadas al ejecutar un documento VXML que robustecen el sistema y posibilitan por ejemplo tomar decisiones durante la ejecución o formatear datos que se utilicen en diálogos o que se pasen a un script local o externo.

La plataforma Android compila las aplicaciones en código de bytes Java y luego lo convierte a un formato propio .dex compatible con su máquina virtual llamada Dalvik. Por esta razón al intentar evaluar la variable “hi” (<var name="hi" expr="'Hello World!'"/>) del documento “Hello World” se generaba una excepción.
Para solventar este problema se fuerza a que el código que permite utilizar el lenguaje ECMAScript no se compile si no que se interprete utilizando la siguiente línea de código:

```
ctx.setOptimizationLevel( -1 );
```

Esto permitió que funcionara ECMAScript en el proyecto.

5.3.2 Sólo prompts

Se probó hasta tres documentos distintos que sólo contenían sólo *prompts* (frases a ser sintetizadas por el TTS). Se comprobó que las diferentes etiquetas VXML involucradas en la síntesis de voz funcionaban correctamente.

5.3.3 Navegación entre diálogos y documentos

Por navegación se entiende que la ejecución de los diferentes nodos de un documento VXML se haga en el orden correcto así como que sea posible ejecutar un diálogo de un documento externo al inicial o *raíz* y que la ejecución continúe después en el nodo correcto.

Ejecutar cualquier documento VXML implica siempre una cierta navegación entre diálogos más o menos compleja, pero para asegurar que la funcionalidad importante referente a la navegación no tuviera fallos se utilizó un documento especialmente complejo preparado para probar este ámbito del lenguaje VXML. Esta prueba permitió comprobar que la navegación se hacía de manera correcta.

5.3.4 Reconocimiento de voz

Se utilizó un documento sencillo con una única pregunta de sí o no al usuario para comprobar que el reconocimiento funcionaba correctamente y que el resultado se enviaba desde la MainActivity al intérprete JVoiceXML.

El reconocimiento de voz y la propagación hasta la Implementation Platform se consiguió de manera sencilla, pero la comunicación desde la Implementation Platform hasta el intérprete del resultado de comprobar si lo reconocido coincidía con las gramáticas definidas fue bastante más complicado. Después de muchas pruebas infructuosas se implementó la solución descrita en la sección 4.4.2.

5.4 Pruebas finales: ejecutar el documento demostrador Help Desk

Para demostrar el correcto funcionamiento del proyecto y que se cumplían los requisitos de diseño se crea el documento Help Desk. Durante la creación del mismo se fueron probando sus distintos módulos y corrigiendo los errores surgidos durante el mismo.

El error más común fue la excepción SAXParsingException que en la mayoría de los casos tenía como causa una mala confección del documento VXML. Pero hubo un caso que fue tortuoso de solucionar debido a la aparente arbitrariedad del error. Si se realizaba algún cambio en el documento y se subía a TinyUpload (servicio gratuito de hosting de archivos utilizado después de que Trello dejase de permitir la descarga de archivos mediante el método GET de HTTP) la aplicación funcionaba correctamente, pero la segunda ejecución de la aplicación generaba la excepción SAXParsingException. Después de muchas pruebas se vio que el problema radicaba en el propio servicio de hosting y se tomó la decisión de crear un servidor web en el ordenador de trabajo para alojar el documento Help Desk. Esto solucionó el problema.

Además de esta excepción es importante resaltar que se intentó utilizar las etiquetas VXML *null* y *void* en el documento para robustecer la interacción pero no tenían ningún efecto en la ejecución del documento. Después de varias pruebas se comprobó que el problema consistía en que la versión del código JVoiceXML utilizado no soportaba estas etiquetas. Esto supone otra razón más por la que se propone como trabajo futuro el actualizar a la última versión de JVoiceXML las clases que utiliza el proyecto.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este proyecto se ha diseñado, implementado y probado una portabilidad a Android del sistema para PC JVoiceXML así como la aplicación demostradora Help Desk. Partiendo de este trabajo un desarrollador Android puede fácilmente utilizar la potencia y sencillez del estándar VoiceXML para gestionar la interacción por voz humano-máquina para mejorar la accesibilidad de sus aplicaciones.

Este proyecto aúna dos aspectos que lo hacen muy interesante. Por un lado el trabajo apoya a un grupo de desarrollo de código abierto liderado por Dirk Schnelle-Walka cuya versión de VoiceXML es una de las más utilizadas dentro del espectro de código libre de VoiceXML y que continúa activamente mejorando. Y por otro lado ayuda a que las personas con déficit de visión pronunciado puedan extraer de los smartphones un valor semejante a los demás.

Durante la ejecución del proyecto se ha aprendido un lenguaje de programación orientado a objetos como Java, así como el lenguaje de programación VoiceXML. También se han aprendido las particularidades del desarrollo en Android con la utilización de métodos y clases propias de Android como el Speech Recognizer Engine o el Text to Speech. Además de estos lenguajes se ha aprendido a depurar sistemas complejos con varios hilos de ejecución concurrentes.

6.2 Trabajo futuro

Aunque se han cumplido todos los requisitos obligatorios planteados en la sección 3.1, no se cumplieron dos de los opcionales. Éstos son el permitir la ejecución de documentos VXML en español e inglés (ahora sólo se puede en español) y el no modificar el código de JVoiceXML. A continuación se enumeran, para un posible trabajo futuro, tanto esos objetivos no cumplidos como otras posibles mejoras que

quedan fuera del alcance de este trabajo:

- Implementar el reconocimiento continuo de voz para permitir el barge-in (poder elegir una opción por voz antes de que termine la pregunta)
- Eliminar las modificaciones al código JVoiceXML cuando se pueda o coordinarse con el grupo de JVoiceXML para que hagan las modificaciones necesarias en el código del repositorio
- Actualizar el código JVoiceXML a la última versión y resolver los errores que surjan
- Modificar la estructura de la portabilidad para que funcione como una librería en vez de como una Activity
- Permitir el multilenguaje (ahora está programado para que funcione en español)
- Implementar la capacidad de fijar un timeout en las aplicaciones VoiceXML que se procesen con esta plataforma de implementación
- Permitir ejecutar documentos en local además de documentos online

Referencias

[1] "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C Recommendation 16 March 2004 https://www.w3.org/TR/voicexml20, accedido el 05/06/16

[2] "Informe Mobile en España y en el Mundo 2015", informe anual de la empresa Ditrendia http://www.ditrendia.es/wp-content/uploads/2015/07/Ditrendia-Informe-Mobile-en-Espa%C3%B1a-y-en-el-Mundo-2015.pdf, accedido el 29/05/16

[3] "Acceso y uso de las TIC por las personas con discapacidad", informe la Fundación Vodafone España publicado el 11 de noviembre de 2013 http://www.aspaym.es/pdf/publicaciones/Acceso%20y%20uso%20de%20las%20TIC%20por%20las%20personas%20con%20discapacidad.pdf, accedido el 22/06/16

[4] https://en.wikipedia.org/wiki/Interactive_voice_response, accedido el 01/06/16

[5] https://en.wikipedia.org/wiki/VoiceXML, accedido el 03/06/16

[6] https://www.w3.org/Voice/, accedido el 02/06/16

[7] http://kevin-junghans.blogspot.com.es/2011/11/voicexml-and-aspnet-developer.html, accedido el 09/06/16

[8] http://voicemodel.codeplex.com/, accedido el 25/05/16

[9] http://www.asterisk.org/get-started, accedido el 25/05/16

[10] https://github.com/voiceglue/voiceglue/wiki, accedido el 24/05/16

[11] "JVoiceXML and CMUSphinx", blog post publicado el 11 de octubre de 2010 http://cmusphinx.sourceforge.net/2010/10/jvoicexml-and-cmusphinx/, accedido el 26/05/16

[12] M. Bruneti, "Porting JVoiceXML su Android", PFC Universita Politecnica delle Marche 2012 http://airtlab.dii.univpm.it/it/system/files/thesis/brunetti-matteo-thesis.pdf accedido el 27/05/16

[13] E. Federici, "Human-Computer vocal interaction: un portiere robotico", PFC Universita Politecnica delle Marche 2013 http://airtlab.dii.univpm.it/it/system/files/thesis/federici-eros-thesis.pdf, accedido el 27/05/16

[14] Vídeo de demostración de la portabilidad Android de Eros Federici, https://www.youtube.com/watch?v=Fa06SJT28wg, accedido el 25/05/16

[15] M. Nekula, "Textový klient pro JVoiceXML na platforme Google Android", PFC Masarykova Univerzita 2014, https://is.muni.cz/th/282398/fi_b/text_prace.pdf, accedido el 27/05/16

[16] Documentación de Google sobre la clase abstracta Service de Android https://developer.android.com/reference/android/app/Service.html, accedido el 12/06/16

[17] Documentación de Google sobre la clase abstracta Activity de Android https://developer.android.com/reference/android/app/Activity.html, accedido el 27/06/16

[18] Documentación de Google sobre la clase Intent https://developer.android.com/reference/android/content/Intent.html, accedido el 27/06/16

[19] Página del proyecto JVoiceXML en el repositorio GitHub https://github.com/JVoiceXML/JVoiceXML, accedido el 28/06/16

[20] Documentación de Google sobre la clase LocalBroadcastManager https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html, accedido el 28/06/16

[21] Página web de la rama android del proyecto JVoiceXML alojado en SourceForge https://sourceforge.net/p/jvoicexml/code/4088/tree/branches/android/, accedido el 03/07/2016

[22] https://es.wikipedia.org/wiki/ECMAScript, accedido el 05/06/16

Glosario

W3C	World Wide Web Consortium
PFC	Proyecto final de carrera
TTS	Text To Speech, se refiere al motor de síntesis de voz de Google
ASR	Android Speech Recognizer, motor de reconocimiento de voz de Android
JVoiceXML	Java Voice eXtensible Markup Language
URI	Uniform Resource Identifier, una cadena de texto utilizada para identificar un recurso dentro de una red
VXML	Es el estándar VoiceXML que es una recomendación W3C
DTMF	Dual-tone multi-frequency signaling
DSP	Digital signal processing
CTI	Computer telephony integration
CCXML	Call Control eXtensible Markup Language
MVVM	Model-view-VoiceModel
RDC	Reusable Dialog Components
JVoiceXML	Java VoiceXML, distribución código libre de un intérprete VoiceXML
IDE	Integrated Development Environment, entorno de desarrollo integrado
UI	User Interface, Interfaz de usuario
PI	Plataforma de Implementación

Anexos

A CÓDIGO DE LA APLICACIÓN VXML HELP DESK

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml" version="2.1" xml:base="http://is.muni.cz/www/282398/input.vxml">
<form>
<field name="menu_principal">
    <grammar root="seleccion_menu_principal" mode='voice' version="1.0" xml:lang="en-US" >
        <rule id="seleccion_menu_principal" scope="public">
            <one-of>
              <item>1</item>
              <item>2</item>
              <item>3</item>
            </one-of>
          </rule>
    </grammar>

    <prompt>Bienvenido! Qué quieres hacer?  uno, ir a la lista de preguntas frecuentes.
    Dos, ir al solucionador de problemas. Tres, quiero que me llame un asesor  </prompt>

    <noinput>
        Por favor di algo
        <reprompt/>
    </noinput>
    <nomatch>
        No te e entendido
        <reprompt/>
    </nomatch>
    <filled>
        <if cond="menu_principal == '1'">
            <goto nextitem="preguntas_frecuentes"/>
        <elseif cond="menu_principal == '2'"/>
            <goto nextitem="troubleshooting"/>
        <else/>
            <goto nextitem="num_telefono"/>
        </if>
    </filled>
</field>

<field name="preguntas_frecuentes" expr="true">
    <grammar root="seleccion_pregunta_frecuente" mode='voice' version="1.0" xml:lang="en-US" >
        <rule id="seleccion_pregunta_frecuente" scope="public">
            <one-of>
              <item>1</item>
              <item>2</item>
              <item>3</item>
            </one-of>
          </rule>
```

```
46      </grammar>
47      <prompt>Elige un artículo: 1, Cómo compartir una canción. 2, Cómo escuchar mi música offline.
48              3, Puedo equalizar mi música?  </prompt>
49      <noinput>
50          Por favor di algo
51          <reprompt/>
52      </noinput>
53      <nomatch>
54          No te e entendido
55          <reprompt/>
56      </nomatch>
57      <filled>
58          <if cond="preguntas_frecuentes == '1'">
59              <goto nextitem="pf_compartir_cancion"/>
60          <elseif cond="preguntas_frecuentes == '2'"/>
61              <goto nextitem="pf_escuchar_offline"/>
62          <else/>
63              <goto nextitem="pf_equalizar"/>
64          </if>
65      </filled>
66  </field>
67
68  <field name="pf_compartir_cancion" expr="true">
69      <grammar root="seleccion_accion" mode='voice' version="1.0" xml:lang="en-US" >
70          <rule id="seleccion_accion" scope="public">
71              <one-of>
72                <item>1</item>
73                <item>2</item>
74                <item>3</item>
75              </one-of>
76            </rule>
77      </grammar>
78
79      <prompt>Compartir canción es muy fácil. Cuando estés reproduciendo una canción di
80          Compartir Facebook o Compartir Tuiter según prefieras.
81          Di 1 para repetir la explicacion. 2 para volver al menú principal. o 3 para salir</prompt>
82
83      <noinput>
84          Por favor di algo
85          <reprompt/>
86      </noinput>
87      <nomatch>
88          No te e entendido
89          <reprompt/>
90      </nomatch>
```

```
    <filled>
        <if cond="pf_compartir_cancion == '1'">
            <reprompt/>
        <elseif cond="pf_compartir_cancion == '2'"/>
            <goto nextitem="menu_principal"/>
        <else/>
            ¡Hasta la próxima!
            <exit/>
        </if>
    </filled>
</field>

<field name="pf_escuchar_offline" expr="true">
    <grammar root="seleccion_accion" mode='voice' version="1.0" xml:lang="en-US" >
        <rule id="seleccion_accion" scope="public">
            <one-of>
              <item>1</item>
              <item>2</item>
              <item>3</item>
            </one-of>
          </rule>
    </grammar>
    <prompt>Para escuchar música offline deberás descargarla primero. Para ello, desde el
        menú principal di: descargar. A continuación te preguntaremos qué lista
        o cancion quieres descargar. Una vez descargado podrás escucharlo offline igual
        que lo haces normalmente.  Di 1 para repetir la explicacion. 2 para volver al
        menú principal. o 3 para salir</prompt>

    <noinput>
        Por favor di algo
        <reprompt/>
    </noinput>
    <nomatch>
        No te e entendido
        <reprompt/>
    </nomatch>
    <filled>
        <if cond="pf_escuchar_offline == '1'">
            <reprompt/>
        <elseif cond="pf_escuchar_offline == '2'"/>
            <goto nextitem="menu_principal"/>
        <else/>
            ¡Hasta la próxima!
            <exit/>
        </if>
```

```
136     </filled>
137 </field>
138 
139 <field name="pf_equalizar" expr="true">
140     <grammar root="seleccion_accion" mode='voice' version="1.0" xml:lang="en-US" >
141         <rule id="seleccion_accion" scope="public">
142             <one-of>
143               <item>1</item>
144               <item>2</item>
145               <item>3</item>
146             </one-of>
147           </rule>
148     </grammar>
149     <prompt>Lo sentimos pero la función equalizador aún no está disponible. Avisaremos
150     cuando quede lista.  Di 1 para repetir la explicacion. 2 para volver al menú principal.
151     o 3 para salir</prompt>
152 
153     <noinput>
154         Por favor di algo
155         <reprompt/>
156     </noinput>
157     <nomatch>
158         No te e entendido
159         <reprompt/>
160     </nomatch>
161     <filled>
162         <if cond="pf_equalizar == '1'">
163             <reprompt/>
164         <elseif cond="pf_equalizar == '2'"/>
165             <goto nextitem="menu_principal"/>
166         <else/>
167             ¡Hasta la próxima!
168             <exit/>
169         </if>
170     </filled>
171 </field>
172 
173 <field name="troubleshooting" expr="true">
174     <grammar root="seleccion_problema" mode='voice' version="1.0" xml:lang="en-US" >
175         <rule id="seleccion_problema" scope="public">
176             <one-of>
177               <item>1</item>
178               <item>2</item>
179               <item>3</item>
180             </one-of>
```

```
          </rule>
    </grammar>

    <prompt>Elige un artículo: 1, No se escucha nada. 2, Se escucha mal. 3, Mi música
        descargada a desaparecido  </prompt>

    <noinput>
        Por favor di algo
        <reprompt/>
    </noinput>
    <nomatch>
        No te e entendido
        <reprompt/>
    </nomatch>
    <filled>
        <if cond="troubleshooting == '1'">
            <goto nextitem="ts_no_se_escucha"/>
        <elseif cond="troubleshooting == '2'"/>
            <goto nextitem="ts_se_escucha_mal"/>
        <else/>
            <goto nextitem="ts_musica_desaparecida"/>
        </if>
    </filled>
</field>

<field name="ts_no_se_escucha" expr="true">
    <grammar root="seleccion_accion" mode='voice' version="1.0" xml:lang="en-US" >
        <rule id="seleccion_accion" scope="public">
            <one-of>
              <item>1</item>
              <item>2</item>
              <item>3</item>
            </one-of>
          </rule>
    </grammar>

    <prompt>Por favor comprueba que el volumen no esté al mínimo. Si está usando auriculares,
        ¿están bien conectados? Antes de empezar a reproducir debería escuchar un bip.
        Si no es así por favor póngase en contacto con nosotros.
        Di 1 para repetir la explicacion. 2 para volver al menú principal. o 3 para salir</prompt>

    <noinput>
        Por favor di algo
        <reprompt/>
    </noinput>
```

```
    <nomatch>
        No te e entendido
        <reprompt/>
    </nomatch>
    <filled>
        <if cond="ts_no_se_escucha == '1'">
            <reprompt/>
        <elseif cond="ts_no_se_escucha == '2'"/>
            <goto nextitem="menu_principal"/>
        <else/>
            ¡Hasta la próxima!
            <exit/>
        </if>
    </filled>
</field>

<field name="ts_se_escucha_mal" expr="true">
    <grammar root="seleccion_accion" mode='voice' version="1.0" xml:lang="en-US" >
        <rule id="seleccion_accion" scope="public">
            <one-of>
              <item>1</item>
              <item>2</item>
              <item>3</item>
            </one-of>
          </rule>
    </grammar>
    <prompt> Por favor compruebe que los auriculares estén bien conectados.
    A veces si está el volumen al máximo puede distorsionarse el sonido.
    Si sigues con el problema por favor contáctanos Di 1 para repetir la explicacion.
    2 para volver al menú principal. o 3 para salir</prompt>

    <noinput>
        Por favor di algo
        <reprompt/>
    </noinput>
    <nomatch>
        No te e entendido
        <reprompt/>
    </nomatch>
    <filled>
        <if cond="ts_se_escucha_mal == '1'">
            <reprompt/>
        <elseif cond="ts_se_escucha_mal == '2'"/>
            <goto nextitem="menu_principal"/>
        <else/>
```

```
271             ¡Hasta la próxima!
272             <exit/>
273         </if>
274     </filled>
275 </field>
276
277 <field name="ts_musica_desaparecida" expr="true">
278     <grammar root="seleccion_accion" mode='voice' version="1.0" xml:lang="en-US" >
279         <rule id="seleccion_accion" scope="public">
280             <one-of>
281               <item>1</item>
282               <item>2</item>
283               <item>3</item>
284             </one-of>
285           </rule>
286     </grammar>
287     <prompt>Como mucho puedes tener 3 gigabaits de música descargada. Cuando
288         sobrepasas el límite se borran las canciones más antiguas.
289         Di 1 para repetir la explicacion. 2 para volver al menú principal. o
290         3 para salir</prompt>
291
292     <noinput>
293         Por favor di algo
294         <reprompt/>
295     </noinput>
296     <nomatch>
297         No te e entendido
298         <reprompt/>
299     </nomatch>
300     <filled>
301         <if cond="ts_musica_desaparecida == '1'">
302             <reprompt/>
303         <elseif cond="ts_musica_desaparecida == '2'"/>
304             <goto nextitem="menu_principal"/>
305         <else/>
306             ¡Hasta la próxima!
307             <exit/>
308         </if>
309     </filled>
310 </field>
311
312
313  <field name="num_telefono" expr="true">
314     <grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar" xml:lang="es-ESP"
315     tag-format="semantics/1.0" mode="voice" root="num_telefono_grammar">
```

```
316         <rule id="num_telefono_grammar" scope="public">
317          <item repeat="1-9">
318                 <ruleref uri="#digito">
319                     <tag>out.num_telefono = out.num_telefono + rules.digito;</tag>
320                 </ruleref>
321             </item>
322         </rule>
323
324          <rule id="digito">
325              <one-of>
326                 <item>0</item>
327                 <item>1</item>
328                 <item>2</item>
329                 <item>3</item>
330                 <item>4</item>
331                 <item>5</item>
332                 <item>6</item>
333                 <item>7</item>
334                 <item>8</item>
335                 <item>9</item>
336             </one-of>
337         </rule>
338      </grammar>
339 <prompt>Ahora registraremos su teléfono y horario en que quiere que le contactemos</prompt>
340 <prompt>Cual es tu teléfono? </prompt>
341 <noinput>
342     Por favor di algo
343 <reprompt/>
344     </noinput>
345  <nomatch>
346    No te e entendido
347    <reprompt/>
348  </nomatch>
349   <filled>
350      <if cond="num_telefono.length != 17">
351        El número de teléfono debe tener 9 dígitos
352            <clear namelist="num_telefono"/>
353        <else/>
354             <goto nextitem="confirmacion"/>
355      </if>
356   </filled>
357  </field>
358
359  <field name="confirmacion" expr="true">
360      <grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar" xml:lang="es-ESP"
```

```
361      tag-format="semantics/1.0"  mode="voice" root="siono">
362         <rule id="siono">
363            <one-of>
364                <item>
365                   <one-of>
366                      <item>si</item>
367                      <item>sí</item>
368                   </one-of>
369                   <tag>out.confirmacion="si";</tag>
370                 </item>
371                 <item>
372                   <one-of>
373                      <item>no</item>
374                      <item>No</item>
375                   </one-of>
376                   <tag>out.confirmacion="no";</tag>
377                 </item>
378            </one-of>
379         </rule>
380       </grammar>
381       <prompt>Su número es el <value expr="num_telefono"/>, ¿es correcto?</prompt>
382       <filled>
383          <if cond="confirmacion == 'sí'">
384            Perfecto
385            <goto nextitem="dia_llamada"/>
386            <else/>
387            <goto nextitem="num_telefono"/>
388          </if>
389       </filled>
390   </field>
391
392   <field name="dia_llamada" expr="true">
393       <grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar" xml:lang="es-ESP"
394       tag-format="semantics/1.0" mode="voice" root="dia">
395         <rule id="dia" scope="public">
396            <one-of>
397                <item>ahora</item>
398                <item>mañana</item>
399            </one-of>
400         </rule>
401       </grammar>
402       <prompt>¿Quiere que le llamemos ahora o mañana?</prompt>
403        <noinput>
404         Por favor di algo
405         <reprompt/>
```

```
406             </noinput>
407         <nomatch>
408     No te e entendido
409     <reprompt/>
410     </nomatch>
411       <filled>
412          <if cond="dia_llamada == 'ahora'">
413            Le llamaremos al número de teléfono <value expr="num_telefono"/> a la máxima
414            brevedad, gracias.
415            <exit/>
416            <else/>
417            <goto nextitem="hora_llamada"/>
418          </if>
419       </filled>
420   </field>
421 <field name="hora_llamada" expr="true">
422   <grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar" xml:lang="es-ESP"
423    tag-format="semantics/1.0" mode="voice" root="hora">
424     <rule id="hora" scope="public">
425         <one-of>
426            <item>
427              <one-of>
428                 <item>mañana</item>
429                 <item>por la mañana</item>
430                 <item>por mañana</item>
431                 <item>la mañana</item>
432                 <item>en la mañana</item>
433                 <item>durante la mañana</item>
434              </one-of>
435              <tag>out="mañana";</tag>
436            </item>
437            <item>
438              <one-of>
439                 <item>tarde</item>
440                 <item>por la tarde</item>
441                 <item>por tarde</item>
442                 <item>la tarde</item>
443                 <item>en la tarde</item>
444                 <item>durante la tarde</item>
445              </one-of>
446              <tag>out="tarde";</tag>
447            </item>
448            <item>
449              <one-of>
450                 <item>noche</item>
```

```
                <item>por la noche</item>
                <item>por noche</item>
                <item>la noche</item>
                <item>en la noche</item>
                <item>durante la noche</item>
              </one-of>
              <tag>out="noche";</tag>
            </item>
          </one-of>
      </rule>
  </grammar>
   <prompt>Prefiere que le llamemos por la mañana, por la tarde o por la noche?
   </prompt>
  <help>
     Diga por la mañana,por la tarde o por la noche
  </help>
   <noinput>
    Por favor di algo
    <reprompt/>
     </noinput>
  <nomatch>
    No te e entendido
    <reprompt/>
  </nomatch>
  <filled>
        Le llamaremos mañana <value expr="hora_llamada"/> al número de teléfono
        <value expr="num_telefono"/>, gracias.
        <submit next="registrar_llamada.asp"
        namelist="num_telefono dia_llamada hora_llamada"/>
        <exit/>
  </filled>
 </field>
</form>
</vxml>
```

B PRESUPUESTO

1) Ejecución Material

- Compra de ordenador personal (Software incluido)..... 1.000 €
- Smartphone ..... 300 €
- Material de oficina..... 75 €
- Total de ejecución material..... 1.375 €

2) Gastos generales

- 16 % sobre Ejecución Material..... 220 €

3) Beneficio Industrial

- 6 % sobre Ejecución Material..... 82,5 €

4) Honorarios Proyecto

- 700 horas a 15 € / hora..... 10.500 €

5) Material fungible

- Gastos de impresión..... 60 €
- Encuadernación..... 100 €

6) Subtotal del presupuesto

- Subtotal Presupuesto..... 12337,5 €

7) I.V.A. aplicable

- 21% Subtotal Presupuesto..... 2590,9 €

8) Total presupuesto

- Total Presupuesto..... 14928,4 €

Madrid, julio 2016
El Ingeniero Jefe de Proyecto

Fdo.: J. Marcos del Ser Bartolomé
Ingeniero de Telecomunicación

C PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de una MEJORA DE LA ACCESIBILIDAD EN DISPOSITIVOS ANDROID CON EL SISTEMA JVOICEXML. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la

misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.