Universidad Nacional Autónoma de Nicaragua, UNAN – León
Facultad de Ciencias y Tecnología
Programa de Maestría en Computación

Sistema informático piloto de comandos de voz para el envío y recepción de mensajes de texto (SMSs) desde un teléfono móvil-celular

Tesis para optar al grado de
Magister Scientiae en Computación
con Énfasis en Redes y Comunicaciones

Autor: Julio César González Moreno

Tutor: Eduardo Santiago Molina Poveda, M.sc.

León, Nicaragua
Julio de 2012

Sistema informático piloto de comandos de voz para el envío y recepción de mensajes de texto (SMSs) desde un teléfono móvil-celular

Resumen

La evolución que han sufrido las tecnologías y las comunicaciones han contribuido a la mejora en los lazos relacionales que guardan las personas, obligándolos a permanecer comunicados la mayor parte del tiempo, incluso en situaciones donde las condiciones no son las propicias.

En este trabajo se presenta el desarrollo de un sistema informático piloto, implementado bajo la plataforma Android, que por medio de comandos de voz intenta facilitar las condiciones de comunicación mediante SMSs; permitiendo que las personas continúen en comunicación constante. Este sistema implementa acciones como son: *lectura*, *recepción*, *borrado* y *escritura* de SMSs, todo ello utilizando como mecanismo de interacción la voz. Esto permite que tanto las manos como la vista permanezcan libres y sean útiles en el desarrollo de otras actividades cotidianas como por ejemplo: manejar, hacer ejercicio, cocinar, etc.

La funcionalidad que sustenta todo el trabajo está basada en la definición de una gramática que permite representar un lenguaje finito para que las personas interactúen con su teléfono móvil-celular mediante comandos dictados a través de la voz. Para la incorporación de dicha funcionalidad se decidió utilizar una plataforma software que funciona sobre dispositivos móviles-celulares y que a su vez accede a servicios propios del teléfono móvil-celular como lo son el envío y la recepción de SMSs, así como también permite incorporar tecnologías proporcionadas por terceros para, posteriormente, unificar diversas funcionalidades y soluciones que permitiesen dar respuesta a la problemática planteada. Es por ello que se seleccionó a la plataforma Android como el entorno de trabajo para el sistema informático piloto desarrollado.

El desarrollo del sistema informático ha sido nombrado como “piloto” ya que no es posible identificar de forma directa a los usuarios potenciales del sistema. Muchas de las ideas modernas que han sido concebidas para finalidades y para usuarios con ciertas características específicas resultan ser útiles y explotadas para otros fines. Los usuarios potenciales del sistema informático no se pueden identificar de forma directa ya que el sistema puede ser empleado por cualquier persona que posea conocimientos básicos del funcionamiento de tecnologías móviles-celulares que emplean Android y que su sistema vocal se encuentre en buenas condiciones para poder emitir, por medio de su voz, los comandos que se encargan de gobernar el sistema.

Con la realización del presente trabajo se ha podido determinar que una mayor cantidad de personas (clientes en potencia) pueden hacer uso de la tecnología de SMSs ofrecida por empresas de servicios telefónicos móviles-celulares en Nicaragua. Esto permite que se amplié, aún más, el segmento de mercado para los servicios de mensajería de texto móvil. Además, aquellos usuarios que presentan algún tipo de discapacidad visual pueden emplear su voz para comunicarse, a través de SMSs con otras personas dentro de una creciente sociedad que gira en torno a un mundo globalizado e interconectado mediante la tecnología. El sistema también apoya el proceso de comunicación de personas que presentan alguna discapacidad o lesión temporal o permanente (lesiones en brazos o piernas) y de personas de la tercera edad. Por otro lado, se cuenta con un moderno y novedoso mecanismo que permite que se realicen labores cotidianas (hacer deporte, conducir un automóvil, cuidar bebes, etc.) al mismo tiempo que se emplea la voz para comunicarse, a través de SMSs, con otras personas.

El trabajo es presentado a lo largo de cuatro capítulos. Inicialmente en el primer capítulo se presentan todos los fundamentos que justifican el trabajo realizado; en el segundo capítulo se abordan aspectos

conceptuales que se relacionan directamente con el sistema propuesto; en el tercer capítulo se explican detalladamente los aspectos de diseño y programación que fueron tomados en cuenta; así como también se describen los detalles de implementación y finalmente, en el cuarto y último capítulo se presentan las conclusiones y recomendaciones del autor. También se ha agregado una sección de anexos en donde se pueden encontrar las instrucciones de instalación y segmentos de código de mayor relevancia.

Dedicatoria

A mi bella esposa Valeria Medina, por ser la fuerza que me empujó siempre a seguir adelante con este trabajo y por ser el camino que guiaba mis pasos. Gracias por brindarme siempre tu apoyo incondicional.

Agradecimientos

A Dios y a la virgen María, por iluminarme y guiarme a lo largo de todo este camino.

A mis padres y hermanos, por haberme dado siempre su apoyo incondicional y por confiar en mí en todo momento; gracias por enseñarme que la educación siempre premia por encima de todas las cosas.

A mis amigos, por llevarme en sus pensamientos deseándome siempre lo mejor. En especial a mi buen amigo y compañero de labores Ing. Denis Espinoza por brindarme su apoyo y conocimiento incondicional cuando más lo necesite.

A todas las personas, organismos y proyectos que han hecho posible que mis compañeros de Maestría y mi persona hayamos podido disfrutar de un aprendizaje de calidad consiguiendo de esta manera ampliar y desarrollar aún más nuestros conocimientos. Sin su apoyo nada de esto hubiese sido posible.

A mi tutor Msc. Santiago Molina, por cultivar su conocimiento, educación y cultura sobre mi persona. Gracias por tener paciencia y apoyarme en los momentos críticos de este trabajo, así como también por sus acertadas sugerencias que siempre fueron una ayuda incalculable en la realización de este trabajo.

Índice general

Índice de ilustraciones

Índice de tablas

Sistema informático piloto de comandos de voz para el envío y recepción de mensajes de texto (SMSs) desde un teléfono móvil-celular

Capítulo I: Introducción

1 Definición del problema

Según los resultados mostrados en el VIII censo de población y IV de vivienda realizado en Nicaragua por el Instituto Nacional para el Desarrollo (INIDE) en el periodo comprendido entre el 28 de mayo al 11 de junio del 2005 (Paguaga, 2006), de un total de 5,128,074 personas censadas; 39,562 cuentan con servicio de Internet, 198,918 cuentan con un ordenador personal, 616,636 cuentan con teléfono convencional y un total de 1,186,623 hacen uso del teléfono móvil-celular para el desarrollo de sus comunicaciones. Los resultados obtenidos a través del censo muestran claramente que los elementos tecnológicos utilizados con mayor frecuencia son: el teléfono móvil-celular, el teléfono convencional, el ordenador personal e Internet (en ese orden respectivamente). Cabe destacar que a través del censo se pudo determinar que el teléfono móvil-celular es el elemento tecnológico que presentó mayor presencia en las personas censadas.

Lo anterior nos lleva a realizar una prospección sobre futuros escenarios donde el uso del teléfono móvil-celular resulta indispensable, más no así imprescindible, para el desarrollo de algunas actividades cotidianas de la vida.

Primer escenario: personas con algún tipo de discapacidad visual.

La gran mayoría de personas que presentan algún tipo de discapacidad visual (ceguera total o parcial) se privan de hacer uso de ciertas tecnologías incluyéndose dentro de estas el teléfono móvil-celular. Las personas que presentan dichas discapacidades se ven limitadas en cuanto a poder escribir un SMS desde su teléfono móvil-celular, ya que su capacidad visual es deficiente, nula o casi nula. Esto mismo sucede al momento en que se recibe un SMS y se quiere emitir una respuesta sobre dicho mensaje.

Para enfocar mejor el escenario antes planteado, se puede hacer uso de los datos suministrados por el movimiento asociativo de los ciegos en América Latina (Varsavsky, 2001), el cual cita que sobre una extensión de tierra de 20.000.000 de kilómetros cuadrados, que se extiende desde México (en la parte sur de América del norte), hasta Argentina y Chile (en el extremo más austral de América del Sur) se encuentra una población total que supera los 500.000.000 de habitantes de los cuales se calcula que entre 4 y 5 millones de personas (de forma estimativa, ¡no hay datos estadísticos, lamentablemente!) son ciegas o presentan algún tipo de problema de baja visión. Además es necesario agregar que este tipo de dolencia es algo creciente en América Latina y el mundo, sobre todo a partir de cierta edad (generalmente de los 40 años en adelante) y principalmente por las carencias que en la dieta se tienen respecto a nutrientes apropiados para conservar la visión o controlar los efectos de deterioro por la edad.

Según lo antes mencionado es posible dar forma a una innovación que le permita a las personas que presentan alguna discapacidad visual hacer uso de la tecnología móvil-celular para comunicarse a través de SMSs, esto con el objetivo de que dichas personas se integren familiar y laboralmente a una sociedad que actualmente crece y se transforma de la mano de la tecnología.

Este mismo escenario también se puede retomar para brindar apoyo a personas que presentan algún tipo de discapacidad o lesión que puede ser del tipo temporal o permanente, así como también personas de la tercera edad a las cuales les resulta difícil utilizar un esquema de comunicación manual basado en envío y recepción de SMSs desde un teléfono móvil-celular.

En el caso de Nicaragua este escenario tiene también mucha validez. La Encuesta Nicaragüense para Personas con Discapacidad (ENDIS2003) (Vásquez) indica que de un total de 4.456.199

personas encuestadas cuya edad era de seis años a más, un total de 462.172 personas (10.37% de la población encuestada), presentan algún tipo de discapacidad. Entre el 22% y el 30% de las personas con discapacidad presentan problemas de auto-cuidado y el 12% de las discapacidades tienen su origen en accidentes (de tránsito, laborales, etc.).

Segundo escenario: envío de SMSs mientras se conduce un automóvil.

Según un estudio realizado por el Laboratorio de Investigación del Transporte del Reino Unido (pablotossi, 2008) se determinó que enviar SMSs desde el teléfono móvil-celular mientras se conduce el automóvil es más peligroso que manejar bajo los efectos del alcohol o las drogas. Según el estudio, enviar SMSs mientras se conduce disminuye en un 35% el tiempo promedio de reacción del conductor comparado con aquellos conductores que han ingerido el límite permitido de alcohol o marihuana a los cuales su capacidad de reacción sólo les desciende un 21% y 12% respectivamente.

La gran mayoría de las personas que conducen automóviles y que viajan junto con su teléfono móvil-celular han recibido algún SMS mientras conducen. La recepción del SMS puede implicar la tentativa lectura y emisión de una respuesta a dicho mensaje mientras se conduce. Pero, ¿cuántas personas se atreven a contestar el SMS recibido?. Según el estudio, el 50% de los conductores de entre 18 y 24 años de edad aceptaron que envían SMSs mientras conducen su automóvil.

Los dos escenarios antes planteados guardan una estrecha relación. En ambos casos resulta necesario utilizar la tecnología móvil-celular para la comunicación a través de SMSs. El problema aparece porque en el primer escenario el sentido de la vista es nula o casi nula y en el segundo escenario el sentido de la vista se encuentra ocupado ya que la concentración visual es crítica. En ambos casos se dificulta, o imposibilita, el envío de SMSs para la comunicación.

El problema planteado en los dos escenarios anteriores puede tener una solución aproximada si se incorporasen características de interacción vocal, referidas al intercambio de SMSs, dentro de los dispositivos móviles-celulares. De acuerdo a lo observado en dicho problema se plantean las siguientes incógnitas:

- ¿Cómo facilitar el envío y la recepción de SMSs a través de un teléfono móvil-celular mediante comandos de voz?

- ¿Cuáles son las necesidades de incorporar a los dispositivos de telefonía móvil-celular, funciones de interacción por medio de la voz?

- ¿Cómo integrar funcionalidades de interacción por medio de la voz, dentro de los dispositivos móviles empleados para la comunicación celular?

- ¿Cuál es la tecnología y cuáles son los dispositivos móviles-celulares que ofrecen mayores facilidades de integración e implementación para sistemas que funcionen mediante comandos de voz?

2 Antecedentes

Actualmente existen pocas soluciones que permiten que las personas interactúen con ciertas tecnologías y dispositivos a través de comandos de voz. Dichas soluciones deben ofrecer a los usuarios la posibilidad de poderse comunicar con el dispositivo electrónico a través del dictado de órdenes o comandos indicados mediante la voz. Una vez que la persona dicta la orden o el comando deseado, el dispositivo electrónico capta la onda de la voz, para luego digitalizarla y transformarla en una representación lingüística simbólica (frases o palabras con un sentido lógico); posteriormente dicha representación es transformada en texto (cadenas de caracteres) con el objetivo de que el sistema verifique (sentencias condicionales) internamente cuál es la acción solicitada por el usuario.

La mayoría de las soluciones encontradas hasta la fecha de finalización del presente trabajo estaban enfocadas a la lectura de texto a través de un dispositivo electrónico como puede ser un ordenador personal o un teléfono móvil-celular. Estas soluciones toman el texto desde una página web o desde una base de datos y se lo suministran al usuario en un formato de voz artificial, o voz sintética, dejando a un lado la interacción entre el usuario y el dispositivo a través de la emisión de órdenes o comandos dictados mediante la voz. Esta falta de interacción entre los usuarios y los distintos dispositivos tecnológicos es lo que se ha considerado como la oportunidad de desarrollo tecnológico para la realización del presente trabajo.

Los sitios web y aplicaciones que se citan a continuación, forman parte de los antecedentes de mayor impacto o contribución a la innovación y desarrollo tecnológico, que sirvieron de base al presente trabajo. De cada uno de ellos se detalla su funcionalidad así como también cualquier aporte que estos pudiesen ofrecer a la propuesta aquí planteada.

	Descripción	*Ventajas*	*Aportes*
Web Talks (Web talks)	Aplicación desarrollada para la plataforma Android. Permite escuchar las páginas web favoritas de sus usuarios, como si de una radio se tratase.	Funcionamiento sencillo. Admite un número reducido de comandos de voz.	Desarrollar un sistema que funcione bajo la plataforma Android. Utilizar comandos de voz para la realización de acciones.
RSS To Speech 1.2 (RSS to Speech)	Gadget para la aplicación Google Desktop. Descarga las alimentaciones de noticias RSS desde cualquier sitio web, posteriormente las lee al usuario.	Utiliza tecnología Text-to-Speech (TTS).	Uso del motor Text-to-Speech (TTS) para hacer que el sistema lea, interactué y se comunique con el usuario.
Text To Speech (Text To Speech)	Portal de idiomas que ofrece un servicio de lectura de texto.	Gesticula el texto introducido. Lee el texto introducido en diferentes acentos.	Utilización de diferentes voces para la lectura de textos.
Text to speech by oddcast (oddcast)	Muestra personajes que presentan capacidades para hablar/pronunciar textos de forma dinámica, en tiempo real y con sincronización de labios.	Permite aplicar efectos de audio. Hace uso de señales emotivas y expresiones faciales.	Mejorar el diseño de la aplicación para que parezca lo más realista posible.

Tabla 1: Antecedentes de mayor impacto

3 Justificación

El uso de la telefonía móvil-celular ha permitido que las comunicaciones personales puedan desarrollarse de una manera más rápida sin importar la ubicación geográfica de las personas. El teléfono móvil-celular se ha convertido en un elemento imprescindible para la vida de las personas y a medida que la tecnología ha evolucionado los teléfonos móviles-celulares han incorporando diversas funciones cada vez más complejas. Actualmente, ya no son sólo un teléfono sino un dispositivo con cámara de fotos y de vídeo, Internet, juegos, agenda, reproductor mp3, televisión, acceso a redes inalámbricas, editores de textos, etc.

Con tantos usos y aplicaciones que se asocian a un teléfono móvil-celular surge la siguiente incógnita: en la mayoría de los casos, ¿para qué se utiliza diariamente el teléfono móvil-celular?. Según un estudio (¿Para qué se usan los celulares?, 2006) realizado por Psicofxp.com (una de las mayores comunidades online en español) para el cual se aplicó una encuesta (Legion, 2008) efectuada durante 15 días entre sus visitantes, el envío de SMSs es el mayor uso que los encuestados le dan a su teléfono móvil-celular, mientras que el segundo lugar lo ocupa su originaria aplicación, hablar a través de él.

El contexto en el que se realizó el estudio anterior puede ser extrapolado hacia todos los usuarios a nivel mundial que hacen uso de la telefonía móvil-celular para hacer evidente la necesidad de comunicación a través del envío y recepción de SMSs. Si a esta necesidad evidente de comunicación le incorporamos su uso por parte de personas que presentan alguna discapacidad visual o alguna discapacidad temporal o permanente o personas de la tercera edad o personas que se encuentren realizando actividades donde la concentración visual es critica (por ejemplo: manejar un automóvil, realizar ejercicios físicos, etc.) se puede determinar que es necesario el desarrollo de una solución innovadora en donde las personas con las características antes mencionadas puedan seguir comunicándose por medio del envío y recepción de SMSs desde su teléfono móvil-celular.

Dicha solución innovadora se basa en la elaboración de un producto software que permita el envío y la recepción de SMSs desde un teléfono móvil-celular a través de comandos de voz. El sistema final debe ofrecer al usuario una interfaz dirigida completamente mediante comandos dictados a través de la voz. Desde dicha interfaz se podrán realizar todas las actividades relacionadas al envío y recepción de SMSs como son: redactar un nuevo mensaje, responder un mensaje recibido, borrar un mensaje, borrar todos los mensajes, entre otras.

El efecto producido por el desarrollo de la solución software innovadora recae en el ofrecimiento de facilidades de interacción oral entre las personas que la utilicen, que pueden o no presentar alguna discapacidad visual o alguna discapacidad temporal o permanente, y los dispositivos móviles empleados para la comunicación celular a través del envío y recepción de SMSs. El desarrollo de dicha solución podrá ser utilizada para incrementar la posibilidad de incorporar la idea de utilizar comandos de voz para la interacción con cualquier otro dispositivo electrónico así como también potenciar las posibilidades de comunicación en personas que tienen algún tipo de discapacidad visual o algún tipo de discapacidad temporal o permanente. En escenarios donde la concentración visual sea medianamente critica se ofrecerán facilidades para mantener la comunicación con otros individuos a través de la reducción de la distracción visual, lo que a su vez ayudará a la reducción de accidentes vehiculares que provocan daños a la infraestructura (vial, eléctrica, etc.) y que a su vez pueden conllevar a fallecimientos, con toda la secuela de efectos morales, jurídicos y económicos que implica la accidentalidad vehicular.

El impacto del sistema, el cual será utilizado ampliamente en distintos ambientes sociales y por

personas con diversos tipos de necesidades, repercute en la mejora de la calidad de vida de sus usuarios (con o sin discapacidades visuales, con o sin discapacidades temporales o permanentes, etc.) ya que permite que estos se integren, aún más, a la sociedad a través de un mecanismo de comunicación basado en SMSs y dirigido por comandos de voz. El impacto del sistema también es visible cuando éste ofrece apoyo en el desarrollo de actividades donde la concentración visual sea medianamente crítica (actividades físicas deportivas, conducción vehicular, cuidados de recién nacidos, etc.). Es importante aclarar que aunque se incluyan este tipo de soluciones en la interacción usuario – teléfono móvil-celular para el envío y recepción de SMSs, el factor humano (propensión a errores) no elimina del todo la posibilidad de sufrir algún tipo de accidente.

4 Objetivos

A continuación se detallan tanto el objetivo general, como los objetivos específicos planteados en el desarrollo del presente trabajo.

4.1 Objetivo general

Desarrollar un sistema informático piloto que mediante comandos de voz permita el envío y recepción de SMSs a través de un teléfono móvil-celular.

4.2 Objetivos específicos

1. Determinar y analizar las necesidades de incorporar a los dispositivos de telefonía móvil-celular funciones de interacción por medio de la voz.

2. Evaluar las tecnologías móviles-celulares y determinar aquella que ofrece mayores facilidades de integración e implementación para sistemas que funcionen mediante comandos de voz.

3. Identificar las funciones necesarias para incorporar mecanismos de interacción por medio de la voz, en dispositivos móviles empleados para la comunicación celular.

4. Incorporar a la sociedad una idea innovadora, basada en el desarrollo del sistema informático aquí propuesto, que ofrezca una solución a los problemas que sustentan la elaboración del presente trabajo.

Capítulo II: Marco conceptual

1 Presentación

En el presente capítulo se desarrollan todos los aspectos teóricos que conforman la base y el fundamento para la elaboración del trabajo. Estos aspectos, aunque estudiados a profundidad por el autor, se presentan de forma general pero muy clara, con la intención de establecer un contexto sobre el cual se presenta y se aborda el presente trabajo.

2 Síntesis y reconocimiento de voz mediante ordenador

La síntesis de voz mediante ordenador es un proceso de sintetización del habla que genera una voz sintética o artificial (no pregrabada).

La síntesis de voz es la producción artificial de habla humana. Un sistema usado con este propósito recibe el nombre de sintetizador de voz y puede implementarse a través de soluciones basadas en software o en hardware. La síntesis de voz se llama a menudo en inglés ***T**ext **T**o **S**peech*, *TTS*, en referencia a su capacidad de convertir texto en voz artificial.

La calidad de una voz sintética vendrá dada por los siguientes aspectos:

- **Factor de inteligibilidad**: determina con qué facilidad/dificultad es entendida la voz artificial
- **Factor de naturalidad**: determina en qué medida se asemeja la voz artificial a la voz real de un ser humano

Un sistema sintetizador de voz mediante ordenador, que convierte texto en voz artificial, esencialmente se compone de dos partes: un *front-end* y un *back-end*. A grandes rasgos, el *front-end* toma como elemento de entrada el texto y produce una representación lingüística fonética del mismo, luego el *back-end* retoma como entrada la representación lingüística fonética y produce una forma de onda sintetizada.

El *front-end* desempeña dos tareas primordiales. Primero, toma el texto y convierte las partes problemáticas como números y abreviaturas en palabras muy equivalentes. Este proceso a menudo recibe el nombre de normalización de texto o preprocesado. A continuación, se asigna una transcripción fonética a cada palabra luego se divide y se marca el texto en varias unidades prosódicas, como frases y oraciones. El proceso de asignar transcripciones fonéticas a las palabras recibe el nombre de conversión de texto a fonema (TTP en inglés) o grafema a fonema (GTP en inglés). La combinación de transcripciones fonéticas e información prosódica constituyen la representación lingüística fonética (Wikipedia, 2010).

La otra parte, el *back-end*, toma la representación lingüística simbólica y la convierte en sonido. El *back-end* se llama a menudo sintetizador de voz.

2.1 Engine TTS (*Text To Speech*)

Antes de describir lo que es un Engine TTS (Text To Speech) es necesario indicar: ¿qué es la API Speech de Java?; se trata de una interfaz estándar que sólo se puede utilizar en combinación con un producto de lenguaje externo que soporte esta misma interfaz.

El Engine TTS es un sistema diseñado para hacerle frente a la entrada de voz o bien a la salida de voz. Tanto reconocedores de voz como sintetizadores de voz son motores de habla. Los motores de voz de la API Speech de Java se encuentran dentro de un paquete llamado *javax.speech*.

Cada Engine TTS tiene un descriptor de objeto que indica el modo en que deben ser invocados cada uno de sus métodos. Una aplicación también puede tener un descriptor de modo que indique los requerimientos deseados del Engine TTS. Algunas propiedades comunes para todos los motores son *EngineName*, *Modename*, *Locale* y Running.

Cuando un motor de voz externo es solicitado, la aplicación puede pasar información sobre el tipo de motor de voz externo que necesita. A continuación, un motor de voz que cumpla todos los requisitos es seleccionado y un objeto del tipo Java speech engine es creado.

Un Engine TTS puede dar consejos sobre la pronunciación de las palabras mediante la adición de nuevas palabras a un vocabulario. La forma escrita, oral, pronunciación y categoría gramatical de la palabra también pueden ser definidas.

El Engine TTS empleado por el sistema aquí planteado fue Svox Pico TTS, el cual es una solución telefónica dedicada para complementar el creciente éxito de la tecnología de texto a voz en el mercado de móviles-celulares y permitir que aumente la cantidad de personas que disfrutan de una tecnología de manos libres basada en mensajes sonoros emitidos mediante voz sintética.

El proceso de reconocimiento de voz mediante ordenador dota a las máquinas de la capacidad de recibir mensajes orales tomando como entrada la señal acústica de la voz recogida a través de un micrófono; el proceso de reconocimiento de voz mediante ordenador tiene como objetivo final decodificar el mensaje contenido en la onda acústica de la voz para realizar las acciones pertinentes. Para lograr este objetivo es necesario conjugar una gran cantidad de conocimientos acerca del sistema auditivo humano, sobre la estructura del lenguaje humano, la representación del significado de los mensajes y sobre todo el auto-aprendizaje de la experiencia diaria.

Actualmente se está lejos de contar con un sistema completo que pueda comprender cualquier mensaje oral en cualquier contexto, tal y como lo puede hacer un ser humano. Sin embargo, la tecnología actual sí permite desarrollar sistemas de reconocimiento de voz que trabajen, con errores aceptables, en entornos semánticos restringidos.

2.2 Speech Recognition

El reconocimiento de voz generalmente es utilizado como una interfaz entre el ser humano y una computadora mediante algún software de aplicación.

Un sistema de reconocimiento de voz debe cumplir con tres tareas esenciales:

- **Pre-procesamiento**: se encarga de convertir la entrada de la voz a un formato que el sistema reconocedor de voz sea capaz de procesar.

- **Reconocimiento**: trata de identificar lo que se dijo. En esta tarea se lleva a cabo una traducción de la señal de voz a un texto o cadenas de caracteres específicos.

- **Comunicación**: formado el texto correctamente, este es enviado al sistema (Software/Hardware) que lo requiera.

Utilizando una representación gráfica; los componentes de un sistema de reconocimiento de voz antes citados estarían ubicados según se muestra en la siguiente ilustración:

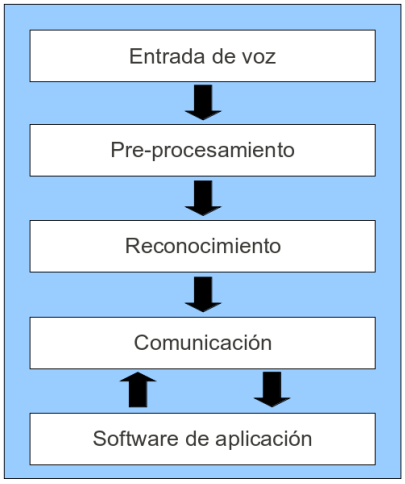

Ilustración 1: Componentes de un sistema de reconocimiento de voz

En ciertos casos, existe una comunicación bilateral en aplicaciones, en las que la interfaz de voz está íntimamente relacionada al resto de la aplicación. Estas pueden guiar al reconocedor de voz especificando las palabras o estructuras que el sistema puede utilizar. Otros sistemas sólo poseen una comunicación unilateral.

Las tareas de pre-procesamiento, reconocimiento y comunicación deberían ser invisibles/transparentes al usuario final de la interfaz. Lo único que el usuario debería notar es el desempeño del sistema de una manera indirecta, a través de características como son: certeza en el reconocimiento de su voz y la velocidad a la que debe dictar las frases. Estas características son generalmente utilizadas por el usuario para evaluar una interfaz de reconocimiento de voz.

Los sistemas de reconocimiento de voz se enfocan en las palabras y los sonidos que distinguen una palabra de la otra en un idioma determinado. Esto es lo que se denomina como los fonemas. Por ejemplo: tapa, capa, mapa y napa, son palabras diferentes debido a que su sonido inicial es distinto.

Existen distintos escenarios utilizados para analizar y describir el habla. Los enfoques más comúnmente usados son:

- **Articulación**: se encarga de analizar cómo el ser humano produce los sonidos del habla

- **Acústica**: se encarga de analizar la señal generada por la voz como una secuencia de sonidos

- **Percepción auditiva**: se encarga de analizar cómo el ser humano procesa el habla

Los tres enfoques proveen ideas y herramientas para obtener resultados eficientes en el reconocimiento de la voz.

La codificación de una señal de voz se refiere a los distintos métodos para codificar digitalmente el habla para utilizarla en diversos ambientes: desde juguetes parlantes, CD's, hasta transmisiones vía telefónica.

Para utilizar la voz como dato en aplicaciones tan diversas como el correo de voz, anotaciones en un texto o un directorio parlante, es necesario almacenar la voz de manera que una computadora pueda recuperarla. La presentación digital de la voz nos provee también las bases para el reconocimiento y la síntesis de voz.

El método "convencional" o secuencial de almacenamiento de datos, la cinta magnética, requiere que se le adelante y regrese hasta encontrar la posición buscada. Es propensa a daño mecánico, no se pueden editar (cut/paste) y no duran mucho tiempo en uso. Otra alternativa puede ser utilizar métodos de codificación digital, esto permite manipular el sonido en memoria. Por lo tanto la meta es capturar fielmente la señal con el menor número de bits.

Lo anterior provoca que se tengan que tomar en cuenta diversos factores como son:

- Memoria y ancho de banda necesario para mayor flexibilidad de uso
- Costo de transmisión
- Diversos rangos de calidad
- Codificadores de Voz (algoritmo de codificación)

3 Gramática libre de contexto

Las gramáticas libres de contexto permiten describir la mayoría de los lenguajes de programación, de hecho, la sintaxis de la mayoría de lenguajes de programación está definida mediante gramáticas libres de contexto. Por otro lado, estas gramáticas son suficientemente simples como para permitir el diseño de eficientes algoritmos de análisis sintáctico que, para una cadena de caracteres dada, determinen como puede ser generada desde la gramática. Dichas gramáticas son las que generan los lenguajes libres o independientes del contexto.

Una gramática independiente del contexto es una gramática formal en la que cada regla de producción es de la forma:

Exp → x

Donde *Exp* es un símbolo no terminal y *x* es una cadena de terminales y/o no terminales. El término independiente del contexto se refiere al hecho de que el no terminal *Exp* puede siempre ser sustituido por *x* sin tener en cuenta el contexto en el que ocurra. Un lenguaje formal es independiente de contexto si hay una gramática libre de contexto que lo genera, este tipo de gramática fue creada por *Backus-Naur*[1] y se utiliza para describir la mayoría de los lenguajes de programación.

1 Una de las dos técnicas principales de notación para las gramáticas libres de contexto utilizada para describir la sintaxis de los lenguajes de programación

Una gramática independiente del contexto está compuesta por cuatro elementos (Cortés, 2008):

1. Símbolos terminales: son los elementos que no generan nada
2. Símbolos no terminales: son los elementos del lado izquierdo de una producción, antes de la flecha "→"
3. Producciones: son sentencias que se escriben en la gramática
4. Símbolo inicial: es el primer elemento de la gramática

Ejemplo: teniendo un lenguaje que genera expresiones del tipo: 9 + 5 - 2

Para determinar si una gramática independiente del contexto está bien escrita se utilizan los arboles de análisis sintáctico, así:

Producciones:

lista -> lista + dígito

lista -> lista – dígito

lista -> dígito

dígito -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

3.1 Árbol de análisis sintáctico

Un árbol de análisis sintáctico permite mostrar gráficamente cómo se puede derivar cualquier cadena de un lenguaje a partir del símbolo distinguido de una gramática que genera ese lenguaje.

Un árbol es un conjunto de puntos, llamados nodos, unidos por líneas, llamadas arcos. Un arco conecta dos nodos distintos. Para crear un árbol formado por un conjunto de nodos y arcos se deben satisfacer ciertas propiedades:

- Existe un único nodo distinguido, llamado raíz (se dibuja en la parte superior) que no tiene arcos incidentes.
- Todo nodo *c* excepto el nodo raíz está conectado con un arco a otro nodo *k*, llamado el padre de *c* (*c* es el hijo de *k*). El padre de un nodo, se dibuja por encima del nodo.
- Todos los nodos están conectados al nodo raíz mediante un único camino.
- Los nodos que no tienen hijos se denominan hojas, el resto de los nodos se denominan nodos interiores.

El árbol de análisis sintáctico tiene las siguientes propiedades:

- El nodo raíz está rotulado con el símbolo distinguido de la gramática;
- Cada hoja corresponde a un símbolo terminal o un símbolo no terminal;
- Cada nodo interior corresponde a un símbolo no terminal.

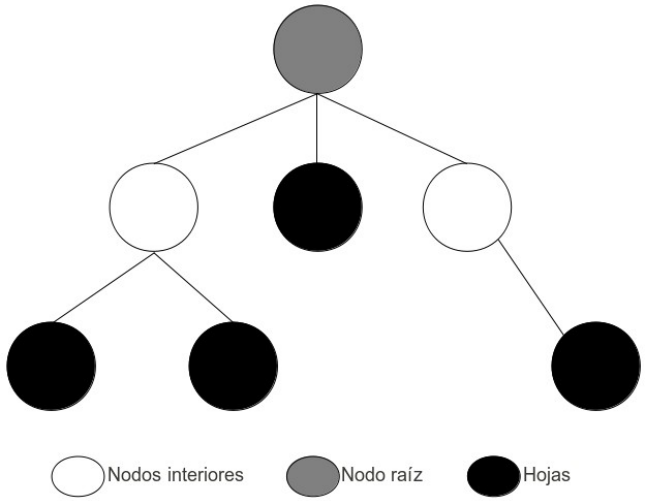

Ilustración 2: Árbol de análisis sintáctico

Para cada cadena del lenguaje generado por una gramática es posible construir (al menos) un árbol de derivación, en el cual cada hoja tiene como rótulo uno de los símbolos de la cadena.

Una gramática es correcta siempre y cuando el símbolo inicial esté al lado izquierdo de las producciones y sea la raíz del árbol.

Para el caso del ejemplo anterior el árbol de análisis sintáctico quedaría representado de la siguiente manera:

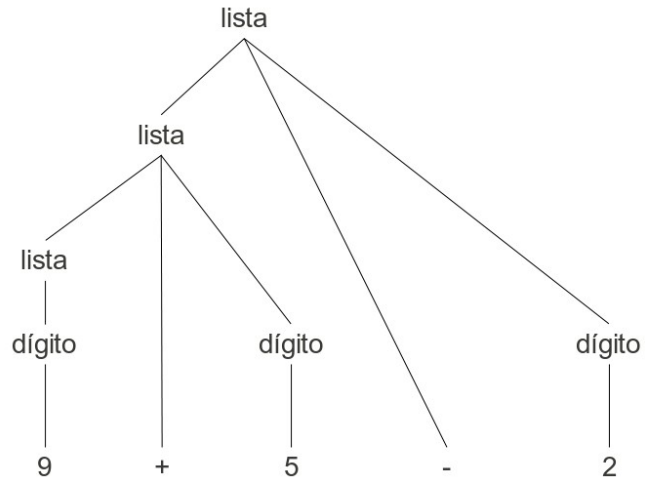

Ilustración 3: Ejemplo de árbol de análisis sintáctico

3.2 Vocabulario finito

Una gramática es una cuádrupla que se corresponde con la siguiente expresión:

$$G = (VT, VN, S, P)$$

en donde:

- VT: representa a un conjunto finito de símbolos terminales
- VN: representa a un conjunto finito de símbolos no terminales
- S: representa al símbolo inicial, el cual pertenece a VN
- P: representa a un conjunto de producciones o de reglas de derivación

Todas las cadenas del lenguaje definido por la gramática están formadas con símbolos del vocabulario terminal VT. El vocabulario terminal se define por una enumeración de los símbolos terminales.

El vocabulario no terminal VN es el conjunto de símbolos introducidos como elementos auxiliares para la definición de la gramática y que no figuran en las sentencias del lenguaje. El vocabulario no terminal se define por enumeración de los símbolos no terminales.

La intersección entre el vocabulario terminal y el vocabulario no terminal es el conjunto vacío:

$$\{VN\} \cap \{VT\} = \{\varnothing\}$$

La unión entre el vocabulario terminal y el vocabulario no terminal es el vocabulario:

$$\{VN\} \cup \{VT\} = \{V\}$$

En ocasiones es importante distinguir si un determinado vocabulario incluye o no la cadena vacía, indicándose respectivamente con súper-índice + o súper-índice *, tal como se muestra a continuación:

$$V^{+} = V - \{\lambda\}$$
$$V^{*} = V + \{\lambda\}$$

El símbolo inicial S es un símbolo no terminal a partir del cual se aplican las reglas de la gramática para obtener las distintas cadenas del lenguaje.

Las producciones P son las reglas que se aplican desde el símbolo inicial para obtener las cadenas del lenguaje. El conjunto de producciones P se definen por medio de la enumeración de las distintas producciones, en forma de reglas o por medio de un metalenguaje.

El siguiente ejemplo tratará de presentar lo antes citado:

Supóngase la siguiente gramática **G = (VN, VT, S, P),** donde:

VN = { <número> , <dígito> }
VT = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
S = <número>

Las reglas de producción P son:

<número> := <dígito> <número>
<número> := <dígito>
<dígito> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Generalmente, los elementos que pueden formar parte de un vocabulario terminal se representan por:

- Letras minúsculas de comienzo del abecedario: a, b, c, …, g
- Operadores tales como: +, -, *, /, etc.
- Caracteres especiales: #, @, (,), etc.
- Dígitos: 0, 1, ..., 9
- Palabras reservadas de un determinado lenguaje de programación con letras minúsculas y en negrita: **if**, **then**, **else**, etc.

4 Sistema de mensajería de texto SMS (Short Message Service)

El servicio de mensajes cortos o SMS (**S**hort **M**essage **S**ervice) es un servicio disponible en los teléfonos móviles-celulares que permite el envío de mensajes de texto cortos entre teléfonos móviles-celulares. SMS fue diseñado originariamente como parte del estándar de telefonía móvil digital GSM, pero en la actualidad está disponible en una amplia variedad de redes, incluyendo las redes 3G.

Un SMS es una cadena alfanumérica de hasta 140 caracteres o de 160 caracteres de 7 bits y cuyo encapsulado incluye una serie de parámetros. En principio, se emplean para enviar y recibir SMSs normal, pero existen extensiones del protocolo básico que permiten incluir otros tipos de contenido, dar formato a los mensajes o encadenar varios SMSs para permitir mensajes de mayor longitud.

En GSM existen varios tipos de SMSs: SMSs "puros", mensajes de configuración (que contienen los parámetros de conexión para otros servicios, como WAP o MMS), mensajes WAP Push, notificaciones de mensajes MMS, etc. Es importante destacar que el transporte de todos los tipos de SMS se realiza de la misma manera. En otros estándares de telefonía móvil (como CDMA2000 o UMTS) el proceso de transporte de los mensajes se realiza de otra forma, pero el funcionamiento es transparente al usuario.

5 Tecnologías de apoyo

En general, toda tecnología se desarrolla con el objetivo de facilitar o mejorar, directa o indirectamente, los procesos humanos. El proceso de comunicación entre personas mediante la utilización de comandos de voz para el envío y la recepción de SMSs es un ejemplo claro en donde el desarrollo de una tecnología innovadora puede ser utilizado para aportar enormes beneficios a la

sociedad. Así mismo, muchas tecnologías son desarrolladas con fines generales, quedando abierta la posibilidad de potenciarlas en usos, ambientes, necesidades o aplicaciones específicas, ya sea de forma aislada o combinada.

En este contexto, el presente trabajo no sólo se ha apoyado en previos estudios teóricos y prácticos, sino en tecnologías emergentes (tanto hardware como software) de gran potencial tales como las redes de comunicación móviles-celulares, las redes inalámbricas accesibles desde un teléfono móvil-celular, la computación móvil, la plataforma Android y su kit de desarrollo (SDK, ***S**oftware **D**evelopment **K**it*), el entorno de desarrollo integrado Eclipse y el lenguaje de programación Java. Algunas de estas tecnologías de apoyo son abordadas de forma breve a continuación.

5.1 Redes de comunicación móviles-celulares

La red GSM (Sistema global de comunicaciones móviles) constituye el estándar de comunicaciones móviles-celulares más utilizado en la actualidad. Se denomina estándar "de segunda generación" (2G) porque, a diferencia de la primera generación de teléfonos móviles-celulares, las comunicaciones se producen de un modo completamente digital.

En 1982, cuando fue estandarizado por primera vez, fue denominado "Groupe Special Mobile" y en 1991 se convirtió en un estándar internacional llamado "Sistema Global de Comunicaciones Móviles". En Europa, el estándar GSM usa las bandas de frecuencia correspondientes a los 900MHz y 1800 MHz. Sin embargo, en Estados Unidos se usa la banda de frecuencia correspondiente a los 1900 MHz. Razón por la cual, los teléfonos móviles-celulares que funcionan tanto en Europa como en Estados Unidos se denominan tribanda y aquellos que funcionan sólo en Europa se denominan bibanda.

El estándar GSM permite un rendimiento máximo de 9,6 kbps, que permite la realización de transmisiones de voz y de datos digitales de volumen bajo, por ejemplo, SMSs o mensajes multimedia (MMS, Servicio de mensajes multimedia).

Las redes de telefonía móvil-celular se basan en el concepto de celdas, es decir zonas circulares que se superponen para cubrir un área geográfica. Dichas redes se basan en el uso de un transmisor-receptor central en cada celda, denominado "estación base" (**E**stación **B**ase **T**ransceptora, BTS).

Cuanto menor sea el radio de una celda, mayor será el ancho de banda disponible. Por lo tanto, en zonas urbanas muy pobladas, hay celdas con un radio de unos cientos de metros mientras que en zonas rurales hay celdas enormes de hasta 30 kilómetros que proporcionan cobertura.

En una red GSM, el equipo de comunicaciones empleado por el usuario se llama estación móvil. Una estación móvil está constituida por una tarjeta SIM (módulo de identificación de abonado), que permite identificar de manera única al usuario y al equipo de comunicaciones.

Las terminales (dispositivos) se identifican por medio de un número único de identificación de quince dígitos denominado IMEI (identificador internacional de equipos móviles). Cada tarjeta SIM posee un número de identificación único (y secreto) denominado IMSI (identificador internacional de abonados móviles).

5.2 Redes inalámbricas

Tal como su propio nombre lo indica, las redes inalámbricas son aquéllas que carecen de cables. Gracias a las ondas de radio, se lograron redes de computadoras de este tipo, aunque su creación refirió varios años de búsqueda. Esta tecnología facilita en primer lugar el acceso a recursos en lugares donde se imposibilita la utilización de cables, como zonas rurales poco accesibles. Además, estas redes pueden ampliar una ya existente y facilitar el acceso a usuarios que se encuentren en un lugar remoto, sin la necesidad de conectar sus computadoras a un equipo de encaminamiento a través de cables. Estos usuarios pueden acceder a la red de su empresa o a la computadora de su casa en forma inalámbrica, sin configuraciones adicionales.

Algunas tecnologías que emplean un esquema de comunicación inalámbrica son las siguientes:

- **Wireless**: se denomina así a los dispositivos que no utilizan cables para realizar el envío y la recepción de datos

- **Wi-Fi**: abreviatura del término inglés **Wi**reless **Fi**delity. Es el término utilizado corrientemente para una red local sin cables (WLAN) de alta frecuencia

- **WLAN** (**W**ireless **L**ocal **A**rea **N**etwork): una WLAN es un tipo de red de área local (LAN) que utiliza ondas de radio de alta frecuencia en lugar de cables para comunicar y transmitir datos

- **Bluetooth**: es una tecnología y protocolo de conexión entre dispositivos inalámbricos. Incluye un chip específico para comunicarse en la banda de frecuencia comprendida entre 2402 y 2480 GHz con un alcance máximo de 10 metros y tasas de transmisión de datos de hasta 721 kbps

La principal ventaja que supone una red Wireless frente a una cableada, es la movilidad. En la actualidad, muchos usuarios y empleados de empresas requieren, para la realización de sus tareas, acceder en forma remota a sus archivos, trabajos y recursos; la red Wireless permite hacerlo sin realizar ninguna tarea compleja de conexión o configuración y evita que cada usuario viaje hasta su empresa o su casa para poder acceder a los recursos de su red de datos.

En síntesis, las redes inalámbricas proporcionan las siguientes características:

- Simplicidad en la instalación

- Facilidad en el proceso de ampliación

- Menor complejidad en su administración

El hecho de que no posean cables, permite adaptarlas a casi cualquier estructura y prescindir de la instalación de pisos técnicos y la instalación de cables molestos que crucen oficinas, habitaciones, etc.

A través de esta tecnología, puede disponerse de una conexión a Internet casi en cualquier lugar donde se cuente con tal servicio y, de esta forma, también a todas las ventajas que nos ofrece la red de redes respecto de lo que es comunicación e información.

5.3 Plataforma Android

Antes de indicar qué es la plataforma Android, resulta necesario comprender el concepto de una plataforma software, para posteriormente enlazar este concepto al de Android.

Una plataforma software es un elemento crucial en el desarrollo del mismo. Mediante el marco de trabajo que proporcionan dichas plataformas es posible crear un nuevo software para que este sea ejecutado sobre ellas (ver ilustración 4). Las plataformas de desarrollo típicas incluyen una arquitectura de ordenadores, un sistema operativo, un lenguaje de programación en conjunto con sus correspondientes librerías o interfaces gráficas (UI, ***U**ser **I**nterface*).

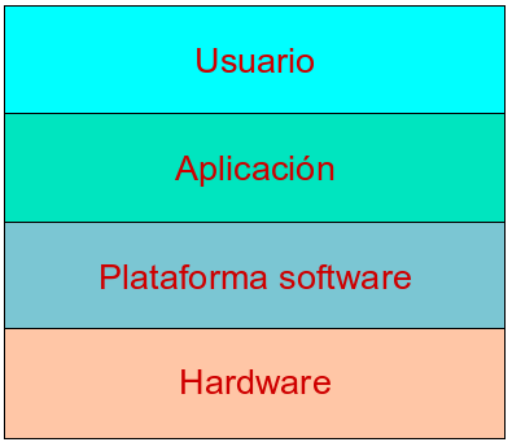

Ilustración 4: Estructura de capas

Las plataformas son comúnmente mencionadas con las APIs, ***A**pplication **P**rogram **I**nterfaces* las cuales no son más que una serie de bibliotecas o librerías que ofrecen un conjunto de funciones y métodos, como una capa de abstracción, para que puedan ser utilizadas por otro software. Un conjunto completo de APIs constituyen una plataforma software. Estas plataformas son normalmente dependientes de los sistemas operativos aunque existen algunas plataformas que no lo son.

Java es un ejemplo de plataforma no dependiente del sistema operativo. Debido a que Java es tanto el lenguaje como la plataforma de desarrollo, la cual incluye la ***J**ava **V**irtual **M**achine* (JVM) cuya función es interpretar el conjunto de bytes (byte code) resultante del proceso de compilación del código fuente del programa escrito en lenguaje Java.

Android es una plataforma software y un sistema operativo para dispositivos móviles (Comparison of Android devices, 2011) basada en un kernel Linux, desarrollado por Google y más tarde por la Open Handset Alliance. Esta plataforma permite a los desarrolladores escribir código en lenguaje Java para que este se ejecute en dispositivos móviles mediante las librerías Java desarrolladas por Google. También se pueden escribir aplicaciones en otros lenguajes, como por ejemplo C, para posteriormente ser compiladas en código nativo ARM y ejecutarlas, aunque este proceso de desarrollo no está soportado oficialmente por Google. La mayor parte de la plataforma de Android está disponible bajo la licencia de software libre de Apache versión 2.0 y otras licencias de código abierto.

5.3.1 Características de Android

A continuación se citan algunas de las características más importantes proporcionadas por la plataforma y que hacen de Android una plataforma muy versátil.

- Ofrece una amplia variedad de diseños (VGA, librerías de gráficos 2D y 3D, etc.)
- Permite el almacenamiento de información en bases de datos SQLite
- Proporcionan diversos mecanismos de conectividad (GSM/EDGE, CDMA, EV-DO, UMTS, Bluetooth y WiFi)
- Soporte para sistemas de mensajería del tipo SMS y MMS
- Incorpora un navegador web para acceder a contenidos disponibles sobre Internet
- Permite trabajar con una máquina virtual de Java
- Las aplicaciones escritas en Java pueden ser compiladas y ejecutadas en la maquina virtual de Dalvik, la cual es una máquina virtual diseñada por Google para su uso en dispositivos móviles
- Ofrece soporte para diversos formatos multimedia (audio, video e imágenes) (MPEG-4, H.264, MP3, AAC, OGG, AMR, JPEG, PNG, GIF)
- Posibilidad de soportar hardware adicional como pueden ser: cámaras de video, pantallas táctiles, GPS, acelerómetros, etc.
- Entorno de desarrollo completo (Android SDK) incluyendo emulador (**A**ndroid **V**irtual **D**evice, AVD), herramientas de depuración, profiling de memoria y rendimiento y plugin para el IDE Eclipse

Una de las características que motivaron a que el presente trabajo se elaborara sobre la plataforma Android es que ofrece una serie de posibilidades que permiten trabajar con una gran variedad de elementos propios del dispositivo como son: redes inalámbricas, acceso a GSM, acceso a GPS, acceso a interfaces de lista de contactos, interacción con mensajería instantánea como es SMS y MMS, comunicación entre aplicaciones existentes, etc. Otro aspecto motivador es que según un estudio de mercado (World, 2011) afirma que las ventas de smartphones aumentó un 83% en el año 2011. Según el mismo estudio; Android continua en crecimiento y ocupa ya el 35% del mercado; en los tres primeros meses del año 2011 se vendieron 101 millones de smartphones en todo el mundo, de los cuales 35.7 millones fueron Android esto arroja un indicio de que dicha plataforma tiene un futuro prometedor.

5.3.2 Arquitectura de Android

Los componentes principales que forman parte de la plataforma Android se muestran en la ilustración 5 y se describen a continuación:

- **Aplicaciones:** esta capa representa las aplicaciones que pueden ser ejecutadas sobre la plataforma y que están escritas en lenguaje de programación Java. Es en esta capa en donde se ubicará el sistema plasmado en el presente trabajo

- **Framework de aplicaciones:** esta capa permite que los desarrolladores de aplicaciones tengan acceso a las API's del framework. La arquitectura de la plataforma está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework)

- **Bibliotecas:** Android incluye un conjunto de bibliotecas C/C++ utilizadas por varios componentes de la plataforma. Estas características se exponen a los desarrolladores a través del framework; algunas son: System C library (implementación biblioteca C standard), bibliotecas de medios, bibliotecas de gráficos, 3D, SQLite, entre otras

- **Android Runtime:** Android incluye un conjunto de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para hacer uso de una mínima parte de la memoria. La Máquina Virtual está basada en registros y ejecuta clases compiladas por el compilador de Java

- **Kernel Linux:** la plataforma emplea la versión 2.6 de Linux modificada para ejecutar los servicios base del sistema como son: seguridad, gestión de memoria, gestión de procesos, stack de red y modelo de controladores. El kernel también actúa como una capa de abstracción entre el hardware del dispositivo, a través de los drivers o manejadores de los mismos, y el software que se ejecuta sobre él. Esta capa hace que las capas superiores funcionen independientemente del hardware que posea el dispositivo

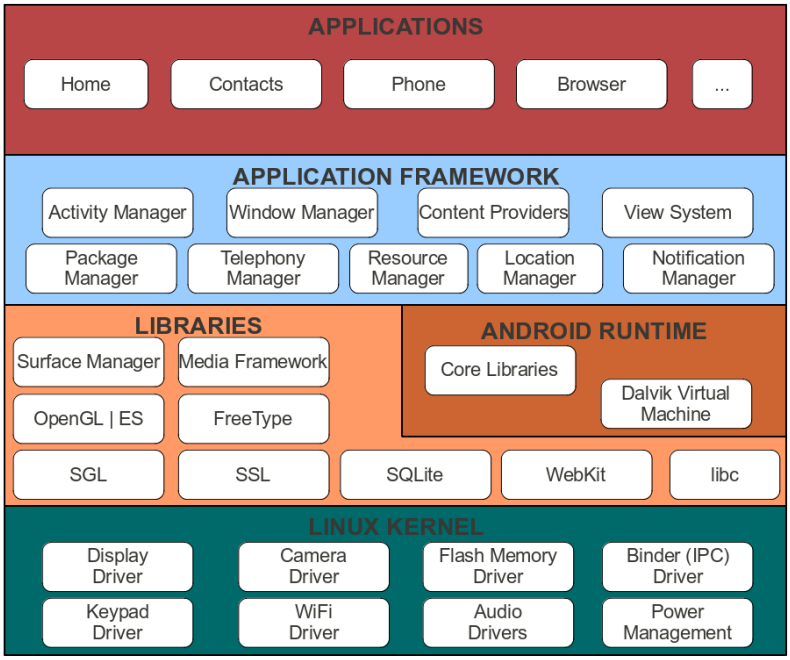

Ilustración 5: Arquitectura de la plataforma Android

5.3.3 Kit de desarrollo Android SDK

El kit de desarrollo de software (**S**oftware **D**evelopment **K**it, SDK) incluye un conjunto de herramientas de desarrollo, tales como un depurador o debugger, un conjunto de librerías, un emulador (basado en QEMU), documentación, códigos de ejemplo y tutoriales. Los sistemas operativos sobre los que se puede instalar el kit de desarrollo son: Microsoft Windows, GNU/Linux y MacOS. El entorno de desarrollo (**I**ntegrated **D**evelopment **E**nvironment, IDE) oficialmente soportado es Eclipse conjuntamente con el plugin ADT proporcionado por Google.

La versión del Android SDK empleada para la codificación y pruebas del sistema plasmado en el presente trabajo fue descargada desde la siguiente dirección web: http://developer.android.com/sdk/index.html

5.4 Entorno de desarrollo integrado Eclipse

Eclipse constituye un entorno de desarrollo integrado de código abierto multiplataforma que permite desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de las clases de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. Eclipse también dispone de un editor de texto con resaltado de sintaxis. Proporciona un esquema de compilación en tiempo real. Incorpora pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes (wizards) para la creación de proyectos, clases, tests, etc.

Una de las ventajas de hacer uso de Eclipse, y que resultaron vitales para la realización del presente trabajo, es que el entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés plug-in) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas y precargadas de forma predeterminada, las necesite el usuario o no. A través de esto fue posible incorporar el plug-in ADT para acoplar toda la funcionalidad que permite desarrollar aplicaciones sobre la plataforma Android.

La versión de Eclipse empleada para la codificación y pruebas del sistema plasmado en el presente trabajo fue la versión Helios Service Release 1 descargado desde la siguiente dirección web: http://www.eclipse.org/downloads/

5.5 Lenguaje de programación Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990's. Las aplicaciones Java están típicamente compiladas en un bytecode (archivos con extensión *.class*). En tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para su ejecución.

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina elementos de bajo nivel como punteros.

Sun Microsystems proporciona una implementación GNU General Public License de un compilador Java y una máquina virtual Java, conforme a las especificaciones del Java Community Process, aunque la biblioteca de clases que se requiere para ejecutar los programas Java no es software libre.

Android en realidad no ejecuta código Java; primero compila el código Java y luego ejecuta una segunda compilación del mismo código. Android ejecuta archivos con extensión *.apk* (**A**ndroid **P**ac**K**age) a través de una máquina virtual especial denominada *Dalvik Virtual Machine*.

La *Dalvik Virtual Machine* no ejecuta código Java ya que no es capaz de interpretar archivos con extensión *.class.* La *Dalvik Virtual Machine* únicamente interpreta archivos con extensión *.dex*; básicamente los archivos con extensión *.dex* son una versión comprimida de los *.class*; a partir de estos últimos el SDK de la plataforma Android hace una segunda compilación y genera los *.dex*, los cuales son archivos de menor tamaño cuya ejecución hace un uso eficiente de los componentes hardware acoplados al dispositivo.

Capítulo III: Diseño y programación del sistema

En el presente capítulo se aborda el trabajo de diseño y de programación empleados para la realización del sistema. Se presenta una visión completa y estructurada de las partes que lo conforman. El capítulo inicia con una descripción general y funcional del sistema, necesarias para comprender algunas decisiones que fueron tomadas durante la fase de diseño. El capítulo termina con una amplia descripción de la arquitectura que conforma todo el sistema y los aspectos de diseño alrededor de dicha arquitectura.

1 Descripción general del sistema

El sistema desarrollado provee una herramienta para la comunicación vía SMSs a través de comandos dictados mediante la voz en donde cada usuario hace uso de un teléfono móvil-celular con conexión a Internet a través de una red inalámbrica. La lógica de los comandos que rigen el sistema está basada en un vocabulario de términos finito que indica la sintaxis que se debe seguir para cada uno de los comandos admitidos. El sistema incorpora capacidades de habla mediante la generación de voz artificial; de está manera le comunica al usuario las acciones que suceden mientras se hace uso del sistema; además integra capacidades de conversión de voz (capturada mediante el micrófono) a texto a través del reconocimiento de voz.

Para hacer uso del sistema es necesario dar clic en el botón indicado (“De clic para dictar el comando”); a continuación se activará el reconocimiento de voz a la espera de que se emita alguno de los siguientes comandos:

- **nuevo**: permite crear un nuevo mensaje para ser enviado a un destinatario

  Sintaxis: nuevo <*cuerpo del mensaje*> <*número del destinatario* | *buscar*> <*si* | *no*>
  Al indicar la palabra *buscar* se mostrará una lista con todos los números telefónicos de los contactos almacenados

- **borrar recibidos**: permite eliminar todos los mensajes que han sido recibidos y que ya fueron leídos

  Sintaxis: borrar recibidos <*si* | *no*>

- **borrar enviados**: permite eliminar todos los mensajes que han sido enviados

  Sintaxis: borrar enviados <*si* | *no*>

- **borrar todo**: permite eliminar todos los mensajes tanto enviados como recibidos

  Sintaxis: borrar todo <*si* | *no*>

- **borrar**: permite eliminar el último mensaje recibido que ya fue leído

  Sintaxis: borrar <*si* | *no*>

- **leer**: permite leer algún mensaje que ha sido recibido

  Sintaxis: leer <*número del mensaje a leer*>

Una vez emitido el comando; el sistema automáticamente irá guiando al usuario, a través de una voz artificial, hasta que se complete la sintaxis según el comando indicado.

Si se recibe un nuevo mensaje entrante mientras el sistema se encuentra en ejecución; el sistema informará de este suceso al usuario a través de un mensaje generado mediante una voz artificial; también se mostrará dicho suceso en la barra de notificaciones del teléfono móvil-celular. A continuación el sistema interactuará con el usuario siguiendo la sintaxis mostrada a continuación:

> Sintaxis: {¿Desea leer el mensaje?} <si {lectura del mensaje} {¿Desea emitir una respuesta sobre el mensaje recibido?} <si <cuerpo del mensaje> {¿Seguro de enviar el mensaje?} <si | no> | no> | no>

El sistema proporciona una serie de opciones que son configurables y que modifican el comportamiento del mismo. El acceso a la configuración de dichas opciones se puede llevar a cabo al pulsar el botón MENU del teléfono móvil-celular. Dichas opciones son:

Ilustración 6: Opciones configurables del sistema

- **Volumen multimedia**: permite cambiar la intensidad del volumen multimedia entre los valores bajo, medio o alto

- **Velocidad de voz**: permite seleccionar la velocidad a la que se quiere que la voz artificial del sistema hable, es posible establecer los valores rápida, normal o lenta

- **Idioma**: permite seleccionar el idioma que se quiere que sea empleado por el sistema para su funcionamiento, es posible establecer los valores español e inglés

- **Acerca de**: muestra una breve descripción y el nombre del autor del sistema

El sistema resulta sencillo e intuitivo ya que está pensado para personas que tienen algún tipo de discapacidad visual o personas cuya visión se encuentra ocupada en otras actividades. Una vez que el sistema se encuentra en ejecución éste presenta al usuario dos botones de pulsación uno de ellos permite hacer uso de gran parte de la funcionalidad del sistema y el otro permite salir del mismo.

Ilustración 7: Interfaz principal del sistema

2 Arquitectura del sistema

El diseño y consolidación de una arquitectura adecuada a los requerimientos del sistema fue, sin duda, el trabajo de más profundidad en el desarrollo de éste. Éste trabajo obligó a realizar una mayor revisión bibliográfica, especialmente en la búsqueda de alternativas para las decisiones de diseño. Se propusieron y exploraron alternativas hasta seleccionar aquellas que se ajustaran mejor a los requerimientos planteados. Incluso, algunas decisiones de diseño fueron parcial o totalmente implementadas y se tuvieron que descartar o re-formular para optimizar el rendimiento del sistema. En este apartado se presenta de forma general este trabajo de diseño y la arquitectura definitiva empleada por el sistema.

2.1 Consideraciones para el diseño del sistema

A continuación se describen los aspectos que se consideraron para llevar a cabo el diseño del sistema:

- El sistema debe ser capaz de operar con todos los tipos de mensajes SMSs (entrantes,

recibidos, leídos, no leídos, salientes, etc.).

- El sistema debe proporcionar una serie de comandos cortos, fáciles de memorizar por un usuario y de sencilla pronunciación.

- El sistema debe ser ejecutado tanto en idioma inglés como en español sin que esto influya en su correcto funcionamiento.

- El sistema debe procesar únicamente comandos dictados mediante la voz y que sean comandos debidamente admitidos por el mismo.

- El sistema debe ser capaz de conservar los valores que sean configurados por el usuario aún cuando el teléfono móvil-celular sea apagado o el sistema sea finalizado y ejecutado nuevamente.

- El diseño del sistema debe estar pensado específicamente para tecnología móvil-celular, por lo tanto, debe considerar todas las limitaciones alrededor de esta tecnología y potenciar sus propiedades.

- El sistema está dirigido a entornos en donde se tiene acceso y control a tecnología de redes inalámbricas conectadas a Internet.

- El diseño lógico y visual del sistema debe estar pensado para ofrecer facilidades de uso a personas que presentan algún tipo de discapacidad visual así como también a personas que presentan alguna discapacidad o lesión temporal o permanente (lesiones en brazos o piernas) y a personas de la tercera edad.

2.2 Descripción general de la arquitectura del sistema

El sistema está basado en una arquitectura orientada a objetos y por lo tanto está compuesto de una serie de clases que lo proveen de toda su funcionalidad. Todo el sistema ha sido desarrollado a través de lenguaje Java apoyándose para ello del **E**ntorno de **D**esarrollo **I**ntegrado (EDI) Eclipse Helios; las interfaces gráficas de usuario fueron desarrolladas haciendo uso del metalenguaje XML (e**X**tensible **M**ark-up **L**anguage). En general, el sistema funciona en gran parte de forma autónoma a través de las clases que lo componen, dichas clases proporcionan funcionalidades como acceso a los SMS, acceso a la barra de notificaciones, funcionalidad de texto a voz (síntesis de voz), acceso a la lista de contactos y funcionalidad de voz a texto (reconocimiento de voz). Esta última hace uso de Internet para conseguir su finalidad.

Lo que sucede cuando se hace uso del reconocimiento de voz es que el sistema emplea un paquete Java nombrado como: *android.speech.RecognizerIntent*, el cual captura la onda de la voz emitida mediante un flujo de audio y la envía a un servidor de Google para que la procese y obtenga a partir de dicho flujo una representación en cadena de caracteres (texto plano); luego esta representación es devuelta por el servidor de Google hacia el sistema en forma de texto el cual ya puede ser utilizado en la lógica de programación que proporciona la interacción con los comandos.

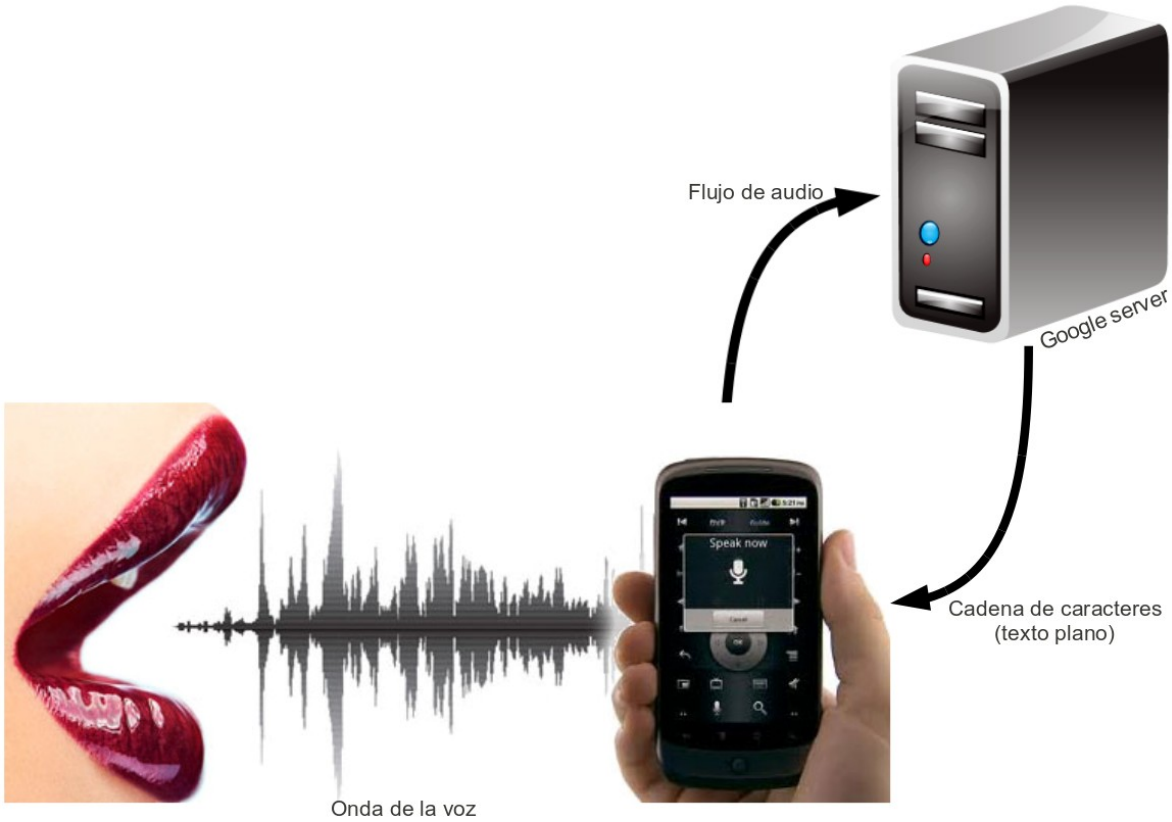

Ilustración 8: Reconocimiento de voz de Google

Así mismo, para que todo esto sea posible, debe existir un canal de comunicación (físico y lógico) entre el teléfono móvil-celular y el servidor de procesamiento de flujo de audio de voz de Google. El sistema actual se apoya sobre la tecnología de redes inalámbricas para proveer un canal físico de comunicación y ofrece un protocolo de nivel de aplicación, montado sobre la arquitectura TCP/IP, como mecanismo de comunicación lógica.

Hasta ahora se han mencionado, de forma muy general y breve, los componentes y elementos más importantes que conforman el sistema, así como aquellas tecnologías sobre las cuales se apoya. Es una primera visión general del sistema, necesaria para una comprensión más fácil de los detalles de diseño e implementación que se presentan en los apartados siguientes.

3 Programación del sistema

Todo el sistema fue programado siguiendo el paradigma de programación orientada a objetos mediante el lenguaje Java. El resultado final de esta etapa del desarrollo del sistema fueron decenas de clases, centenas de atributos, propiedades, métodos y en general, miles de líneas de código que cumplieran con los requerimientos y el diseño del sistema. El proceso de programación fue cuidadoso en todo momento en la búsqueda de maximizar el rendimiento del sistema.

3.1 Gramática libre de contexto

Antes de describir el proceso de programación empleado para el desarrollo del sistema es necesario analizar la gramática libre de contexto utilizada para la implementación de los comandos que son admitidos por el sistema.

La gramática libre de contexto propuesta en el presente trabajo se diseño para dar respuesta a la forma de operar (sintaxis) de cada uno de los comandos admitidos por el sistema.

La propuesta de gramática libre de contexto es la siguiente:

<inicio> := (<comandos> <mensaje> <num> | <contactos> <confirmación>) | (<comandos> <confirmación>) | (<comandos> <num>)
<comandos> := nuevo | borrar recibidos | borrar enviados | borrar todo | borrar | leer
<mensaje> := <mensaje> (<palMay> | <palMin> <espacio>)
<confirmación> := <afirmativo> | <negativo>
<palMay> := A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
<palMin> := a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
<espacio> := " "
<num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<contactos> := buscar
<afirmativo> := si
<negativo> := no

El código Java asociado a la implementación de la gramática libre de contexto mostrada anteriormente ha sido escrito en los métodos *onActivityResult* y *validateDataCapture* de la clase *messageSpeak*.

3.2 Diagrama de clases

Todas las clases que forman parte del sistema se encuentran dentro del paquete Java *org.messageSpeak* y su estructura se detalla en el siguiente diagrama de clases:

Como se puede observar en la siguiente ilustración el sistema está constituido por un total de doce clases; cada una de estas clases aporta la funcionalidad necesaria para dar respuesta a diversos aspectos ofrecidos por el sistema.

org.messageSpeak

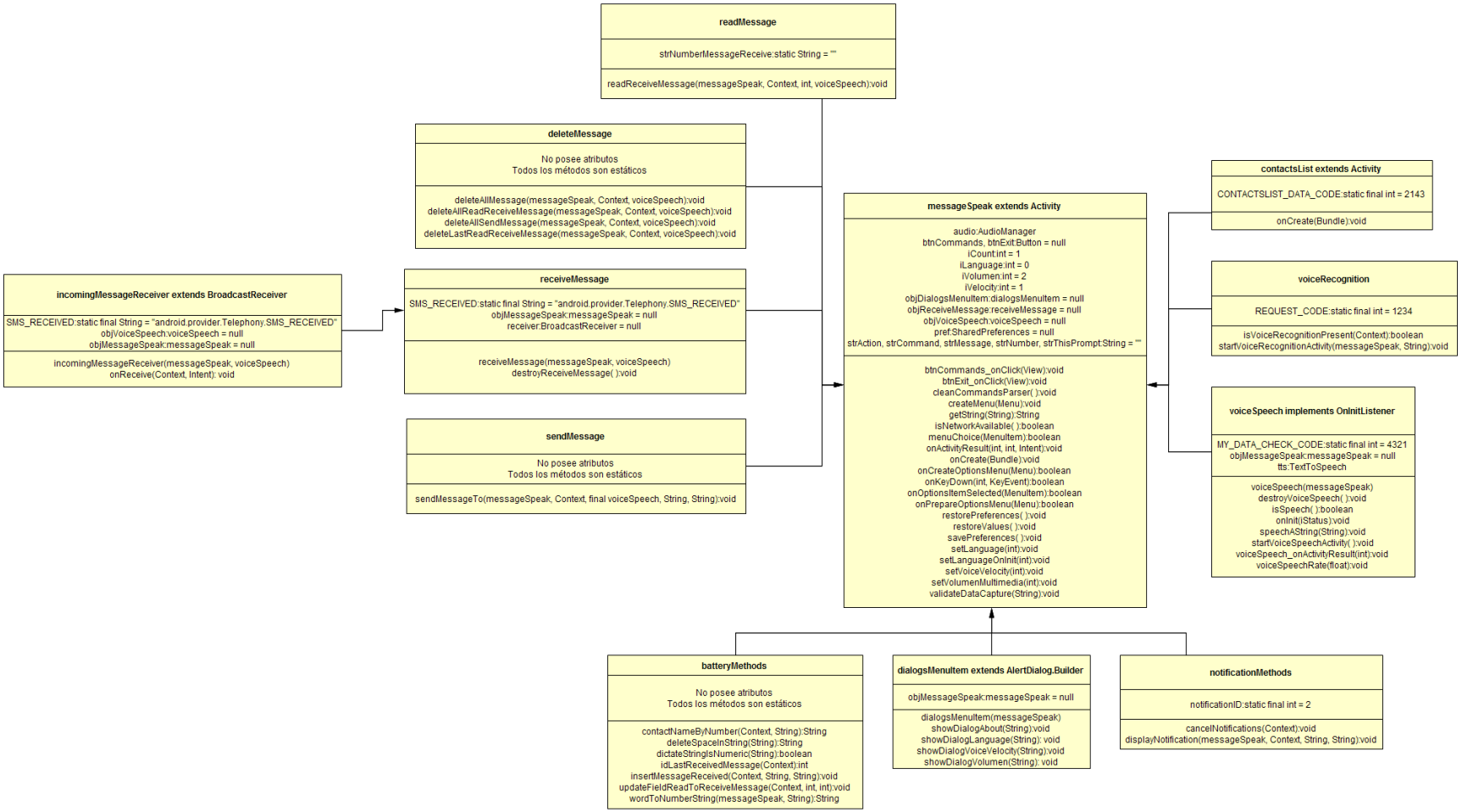

Ilustración 9: Diagrama de clases

Cada una de estas clases es posible agruparlas, según el aspecto al que dan respuesta, de la siguiente manera:

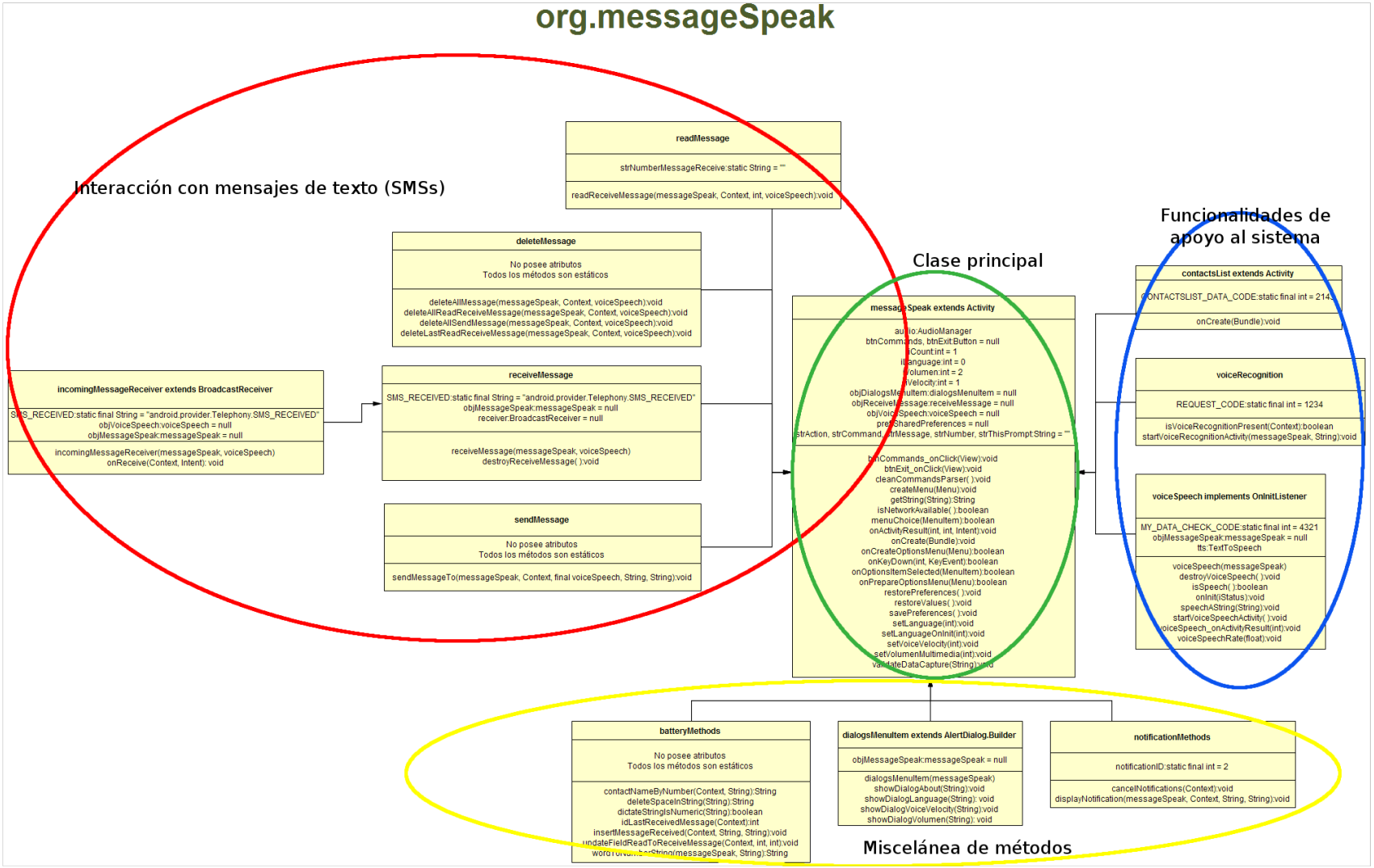

Ilustración 10: Diagrama de clases agrupado

3.2.1 Interacción con mensajes de texto (SMSs)

Dentro de este grupo encontramos a todas aquellas clases que trabajan de forma directa con los SMSs, ya sea para leer, borrar o actualizar. Estas clases sirven de apoyo a comandos propios del sistema como son: borrar y leer, además se emplean cuando el sistema recibe y procesa un nuevo mensaje entrante.

Las clases que forman parte de este grupo son las siguientes:

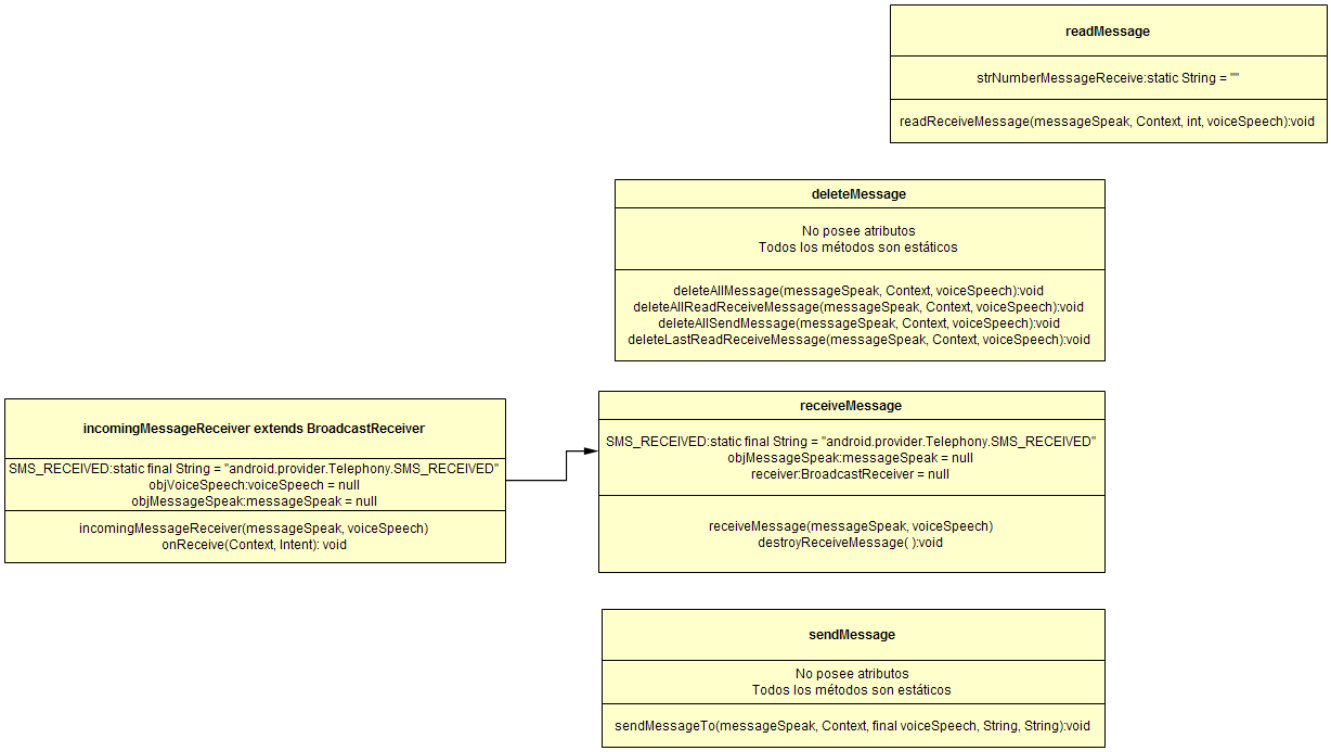

Ilustración 11: Interacción con mensajes de texto (SMSs)

La clase *readMessage* permite obtener de la bandeja de entrada del teléfono móvil-celular algún SMS almacenado dentro de ella (método *readReceiveMessage*).

La clase *deleteMessage* se encarga de la eliminación de SMSs, ya sean mensajes que han sido recibidos o mensajes que han sido enviados (métodos *deleteAllSendMessage* y *deleteAllMessage*), también posee métodos que permiten borrar el último mensaje recibido que ya fue leído por el usuario (método *deleteLastReadReceiveMessage*) así como también borrar todos los mensajes recibidos que ya han sido leídos (método *deleteAllReadReceiveMessage*).

La clase *receiveMessage* es la clase encargada de gestionar los nuevos mensajes entrantes, para ello se apoya de la clase *incomingMessageReceiver* la cual detecta y reacciona ante determinados eventos globales (método *onReceive*) generados por el sistema (batería baja, tarjeta SD insertada o SMS recibido como es nuestro caso).

La clase *sendMessage* se encarga de enviar SMSs hacia un determinado destinatario (método *sendMessageTo*).

3.2.2 Funcionalidades de apoyo al sistema

Este grupo se conforma de un total de tres clases. La funcionalidad aportada por este grupo radica en implementar el mecanismo de reconocimiento de voz y el mecanismo de texto a voz además de incorporar una clase que muestra una interfaz gráfica con una lista de todos los contactos almacenados en el teléfono móvil-celular.

Las clases que conforman este grupo son las siguientes:

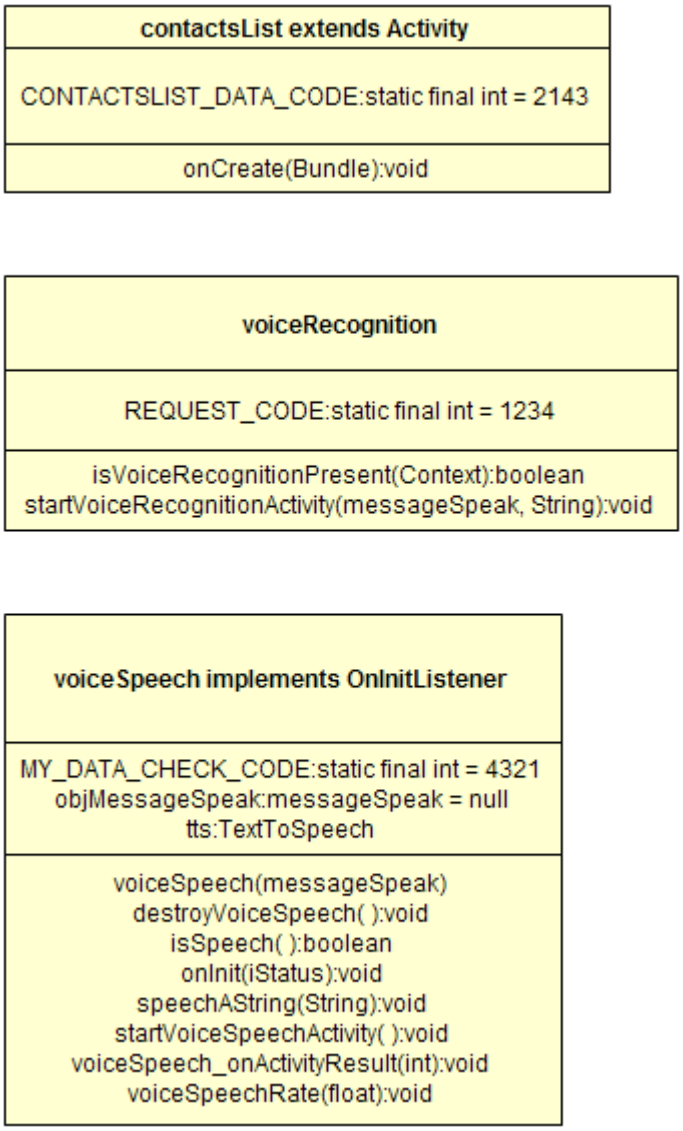

Ilustración 12: Funcionalidades de apoyo al sistema

La clase *contactsList* representa una interfaz gráfica que se encarga de mostrar una lista con los números telefónicos de todos los contactos almacenados en el teléfono móvil-celular. Esta lista se utiliza cuando el usuario indica el número del destinatario al que quiere enviar un nuevo mensaje utilizando el comando *buscar*, esto le permite al usuario seleccionar el número del destinatario al cual

será enviado el nuevo mensaje sin que tenga que saberlo de memoria.

La clase *voiceRecognition* es la clase encargada de tratar los comandos dictados por el usuario y convertirlos en cadenas de caracteres (método *startVoiceRecognitionActivity*) para que estas sean utilizadas en la lógica del código que forma parte del sistema. Esta clase utiliza el método *isVoiceRecognitionPresent* el cual determina si el mecanismo de reconocimiento de voz está presente en el teléfono móvil-celular[2], también dispone de otro método que se encarga de iniciar el reconocimiento de voz.

La clase *voiceSpeech* genera una voz artificial encargada de guiar al usuario en la utilización de los distintos comandos admitidos por el sistema, implementa un mecanismo de conversión de texto a voz, el texto se encuentra precargado en un archivo de recursos y luego es convertido por esta clase en mensajes sonoros. La clase emplea una serie de métodos que le permiten: iniciar el mecanismo de texto a voz (método *startVoiceSpeechActivity*), finalizar el mecanismo de texto a voz (método *destroyVoiceSpeech*), determinar si el mecanismo de texto a voz se encuentra actualmente emitiendo un mensaje sonoro (método *isSpeech*), emitir mensajes sonoros dado una cadena de caracteres (método *speechAString*), modificar la velocidad a la que se emiten los mensajes sonoros (método *voiceSpeechRate*), entre otros.

3.2.3 Miscelánea de métodos

Este grupo está constituido por un total de tres clases, dichas clases aportan funcionalidades para el manejo de cadenas de caracteres con números telefónicos, supresión de espacios en blanco entre cadenas de caracteres con números telefónicos, obtenciones de identificadores de mensajes almacenados en la bandeja de entrada, visualización de menús de configuración del sistema, visualización de notificaciones en la barra de notificaciones, entre otras. Por lo descrito anteriormente podemos concluir que este grupo alberga una serie de métodos que no calzan en ninguno de los grupos anteriores pero que son importantes para completar la funcionalidad total del sistema.

Las clases que se encuentran en este grupo son las siguientes:

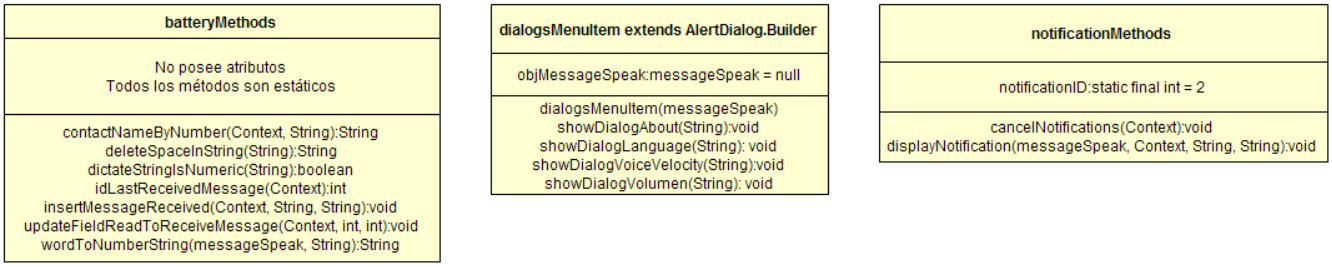

Ilustración 13: Miscelánea de métodos

La clase *batteryMethods* alberga métodos de distinta índole que apoyan al funcionamiento del sistema, dentro de estos métodos encontramos métodos que devuelven el nombre de un contacto a partir de su número de teléfono (método *contactNameByNumber*), método que permite eliminar los espacios en blanco que puedan existir en una cadena de texto que contiene algún número de teléfono (método *deleteSpaceInString*), método que indica si una cadena de caracteres dictada por el

2 La aplicación Google Voice Search (generalmente pre-instalada en dispositivos Android) responde a un RecognizerIntent realizando el streaming de audio hacia los servidores de Google. Es posible comprobar si Google Voice Search está instalado accediendo a Ajustes → Aplicaciones → Administrar aplicaciones

usuario contiene únicamente valores numéricos (método *dictateStringIsNumeric*), método que obtiene de la bandeja de entrada de SMSs el identificador del último mensaje recibido (método *idLastReceivedMessage*), método que permite ingresar un nuevo mensaje recibido a la bandeja de entrada (método *insertMessageReceived*), método que permite marcar un mensaje de la bandeja de entrada como leído (método *updateFieldReadToReceiveMessage*) y un método (*wordToNumberString*) que permite pasar una cadena de caracteres formada por números en forma de palabras (uno, dos, tres, etc.) a una cadena de caracteres formada por números en forma de dígitos (1, 2, 3, etc.).

La clase *dialogsMenuItem* se encarga de mostrar las opciones configurables del sistema (a través de los métodos *showDialog...*) las cuales pueden ser seleccionadas por el usuario a través del botón MENU del teléfono móvil-celular.

La clase *notificationMethods* permite mostrar una nueva notificación sobre la barra de notificaciones del teléfono móvil-celular (método *displayNotification*), esto último se emplea cuando se recibe un nuevo SMS. Esta misma clase también posee un método (*cancelNotifications*) que permite eliminar de la barra de notificaciones todas las notificaciones que han sido generadas por el sistema.

3.2.4 Clase principal

Constituido por una única clase que corresponde con la clase integradora de todas las clases analizadas en las secciones anteriores. Aquí se crean los objetos necesarios, se obtienen y almacenan las opciones configurables del sistema (métodos *restorePreferences*, *restoreValues* y *savePreferences*), se valida si el dispositivo se encuentra conectado a una red inalámbrica (método *isNetworkAvailable*), se analiza la sintaxis que debe ser respetada por cada uno de los comandos (métodos *onActivityResult* y *validateDataCapture*), se establece el idioma de la aplicación (métodos *setLanguageOnInit* y *setLanguage*), se establece la intensidad del volumen multimedia (método *setVolumenMultimedia*), entre otras cosas.

Los atributos y métodos que forman parte de la clase principal son los siguientes:

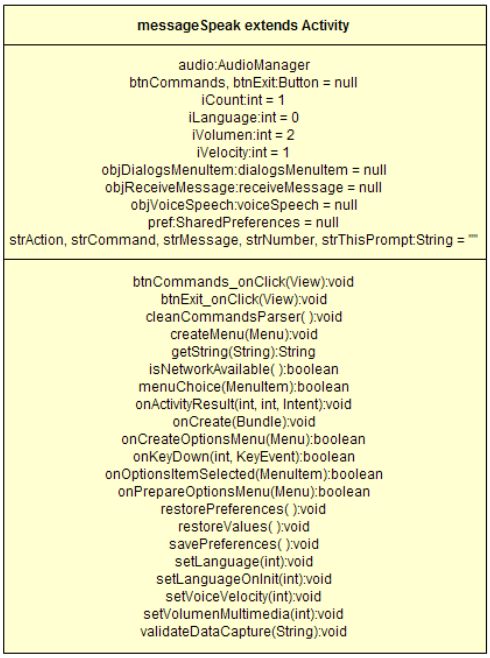

Ilustración 14: Clase principal

3.3 Diagrama de casos de uso

En esta sección se describen los principales casos de uso que identifican las operaciones que se pueden realizar en el sistema.

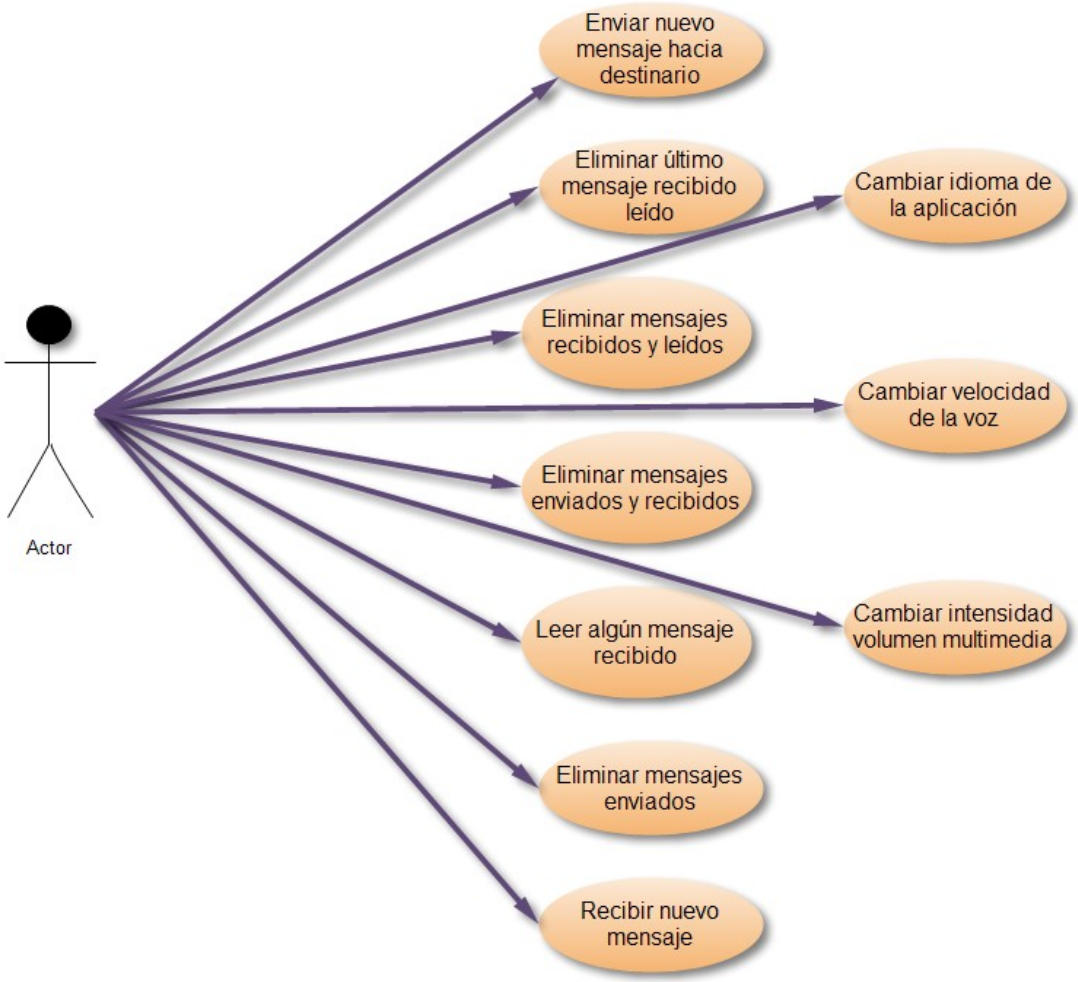

Ilustración 15: Diagrama de casos de uso

A continuación se detallan los casos de uso antes mostrados y se asocia a cada caso de uso con su respectivo comando admitido por el sistema, así como también con cada una de las clases que se emplean para proporcionan la funcionalidad deseada.

	Comando	*Clases*	*Descripción*
Enviar nuevo mensaje hacia destinatario	*nuevo*	messageSpeak voiceSpeech voiceRecognition batteryMethods sendMessage	Se toman en cuenta toda las acciones que se deben llevar a cabo cuando el usuario desea emitir un nuevo SMS
Eliminar último mensaje recibido leído	*borrar*	messageSpeak voiceSpeech voiceRecognition deleteMessage	Se toman en cuenta las acciones necesarias para eliminar de la bandeja de entrada el último mensaje recibido y que ya ha sido leído por el usuario
Eliminar mensajes recibidos y leídos	*borrar recibidos*	messageSpeak voiceSpeech voiceRecognition deleteMessage notificationMethods	Elimina todos los mensajes almacenados en la bandeja de entrada y que ya han sido leídos por el usuario
Eliminar mensajes enviados y recibidos	*borrar todo*	messageSpeak voiceSpeech voiceRecognition deleteMessage notificationMethods	Elimina todos los mensajes almacenados en las bandejas de entrada y de salida
Leer algún mensaje recibido	*leer*	messageSpeak voiceSpeech voiceRecognition readMessage batteryMethods notificationMethods	Lee algún mensaje almacenado en la bandeja de entrada. El usuario debe indicar cuál es el número del mensaje que quiere leer. Números bajos indican mensajes recientemente recibidos y números altos indican mensajes antiguos
Eliminar mensajes enviados	*borrar enviados*	messageSpeak voiceSpeech voiceRecognition deleteMessage	Elimina todos los mensajes almacenados en la bandeja de salida
Recibir nuevo mensaje		messageSpeak receiveMessage incomingMessageReceiver batteryMethods notificationMethods voiceSpeech voiceRecognition readMessage sendMessage	Indica al usuario cuando se ha recibido un nuevo mensaje y pregunta si se desea que el nuevo mensaje recibido sea leído; también consulta si el usuario desea emitir una respuesta sobre dicho mensaje
Cambiar idioma del sistema		messageSpeak dialogsMenuItem	Permite modificar el idioma del sistema entre español e inglés, para esto se hace uso de un menú de opciones
Cambiar velocidad de		messageSpeak	Permite modificar la velocidad de la

	Comando	Clases	Descripción
voz		dialogsMenuItem voiceSpeech	voz a la que se lee la información, para esto se hace uso de un menú de opciones
Cambiar intensidad volumen multimedia		messageSpeak dialogsMenuItem	Permite modificar la intensidad del volumen multimedia, para esto se hace uso de un menú de opciones

Tabla 2: Descripción de casos de uso

3.4 Diagramas de secuencia

Para modelar la interacción entre las clases y los métodos de dichas clases que forman parte del sistema se hará uso de los diagramas de secuencia. Dichos diagramas mostrarán la interacción que sucede entre un conjunto de objetos que forman parte del sistema para lograr una determinada funcionalidad. Los diagramas de secuencia se reflejan en el eje del tiempo y se modelan para cada uno de los casos de uso.

Los diagramas de secuencia se utilizarán para mostrar los detalles de implementación de cada uno de los escenarios, incluyéndose los objetos y las clases que se emplean para la implementación de dicho escenario, así como también se muestran los mensajes intercambiados entre los objetos.

A continuación se muestra para cada uno de los casos de uso su respectivo diagrama de secuencia junto con una breve explicación del mismo.

3.4.1 Enviar nuevo mensaje hacia destinatario

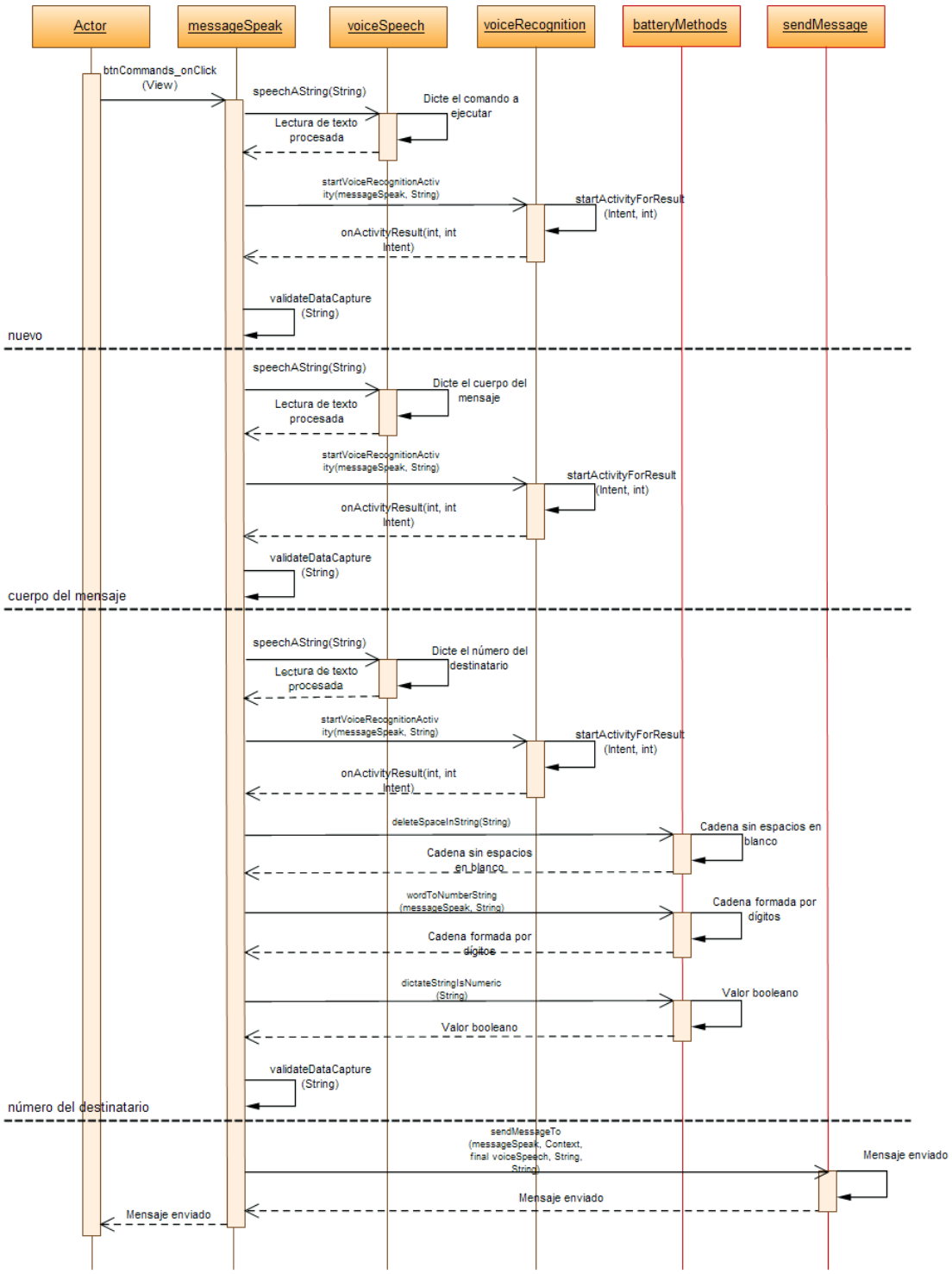

Ilustración 16: Diagrama de secuencia - Enviar nuevo mensaje hacia destinatario

La ilustración anterior muestra la interacción de las clases más importantes que forman parte del

sistema y que se utilizan para que un nuevo mensaje sea enviado hacia un destinatario. Para hacer uso de esta funcionalidad el usuario deberá emplear el comando *nuevo*, una vez que el usuario ha dictado el comando anterior, se le solicitará el cuerpo del nuevo mensaje, a continuación el sistema solicitará el número al cual deberá ser enviado el nuevo mensaje; aquí es necesario destacar que solamente se admiten valores numéricos o se admite el comando *buscar* el cual mostrará al usuario una lista seleccionable con todos los contactos almacenados junto a su respectivo número de teléfono, una vez que se tienen todos los datos antes mencionados el sistema procederá a enviar el nuevo mensaje a través del método *sendMessageTo* de la clase *sendMessage*.

3.4.2 Eliminar último mensaje recibido leído

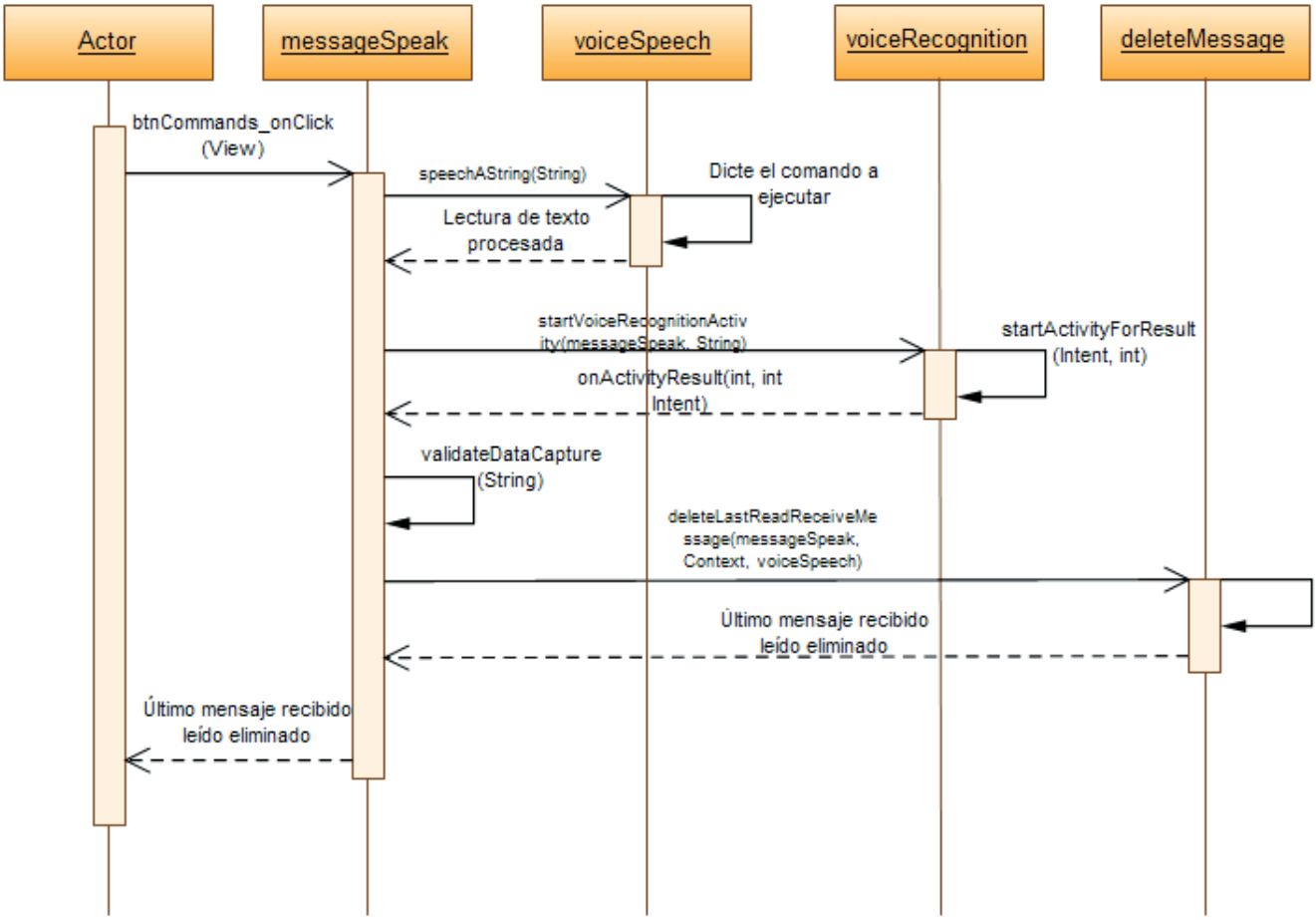

Ilustración 17: Diagrama de secuencia - Eliminar último mensaje recibido leído

Para poder eliminar el último mensaje recibido que ya ha sido leído por el usuario; el usuario deberá hacer uso del comando *borrar*, a continuación el sistema eliminará dicho mensaje de la bandeja de entrada a través del método *deleteLastReadReceiveMessage* de la clase *deleteMessage*.

3.4.3 Eliminar mensajes recibidos y leídos

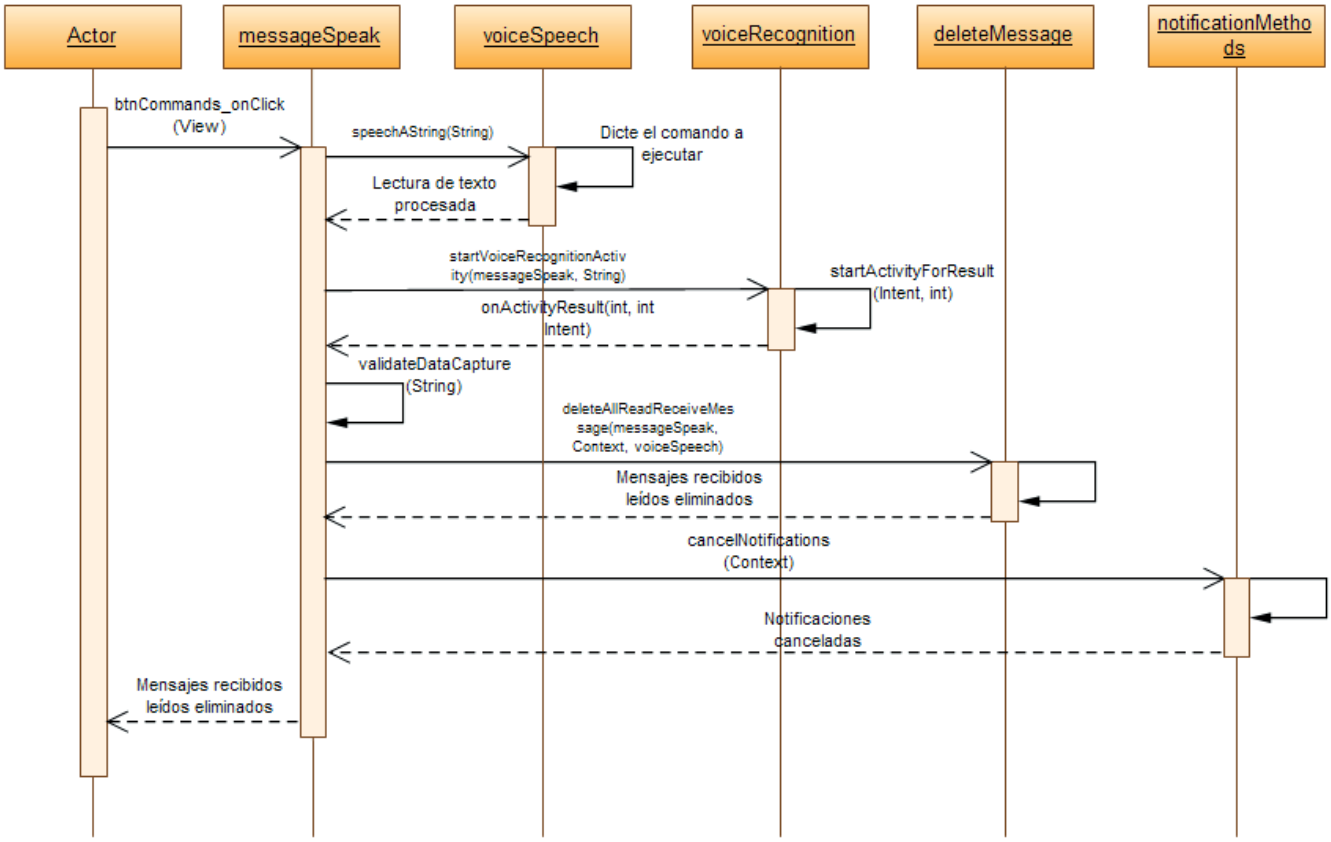

Ilustración 18: Diagrama de secuencia - Eliminar mensajes recibidos y leídos

El diagrama de secuencia anterior muestra que para poder eliminar los mensajes recibidos y leídos se deberá hacer uso del comando *borrar recibidos*, a continuación el sistema eliminará de la bandeja de entrada todos los mensajes almacenados a través del método *deleteAllReadReceiveMessage* de la clase *deleteMessage*, posteriormente se limpiará la barra de notificaciones del teléfono móvil-celular a través del método *cancelNotifications* de la clase *notificationMethods*.

3.4.4 Eliminar mensajes enviados y recibidos

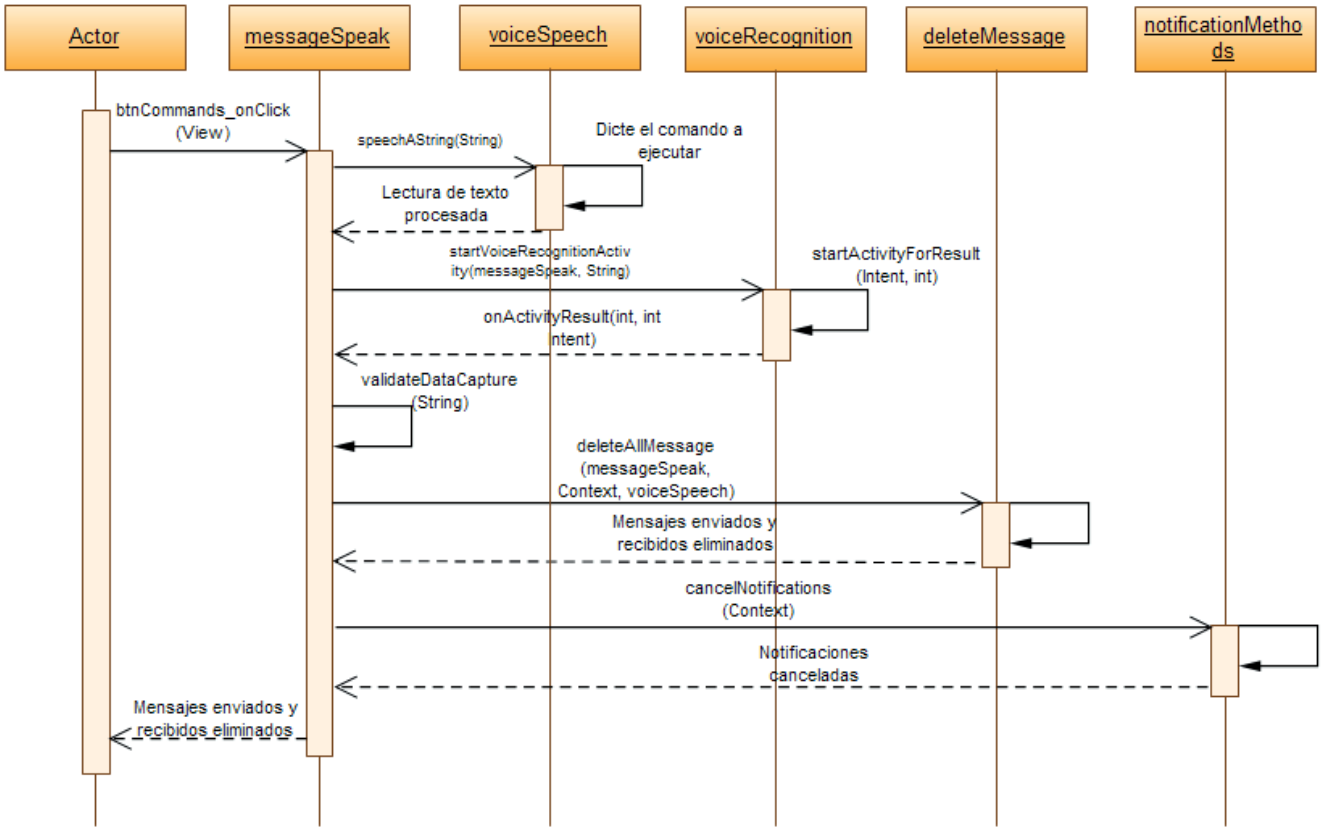

Ilustración 19: Diagrama de secuencia - Eliminar mensajes enviados y recibidos

Para poder eliminar los mensajes enviados y recibidos se deberá hacer uso del comando *borrar todo*, a continuación el sistema eliminará de la bandeja de entrada y de salida todos los mensajes almacenados a través del método *deleteAllMessage* de la clase *deleteMessage*, posteriormente se limpiará la barra de notificaciones del teléfono móvil-celular a través del método *cancelNotifications* de la clase *notificationMethods*.

3.4.5 Leer algún mensaje recibido

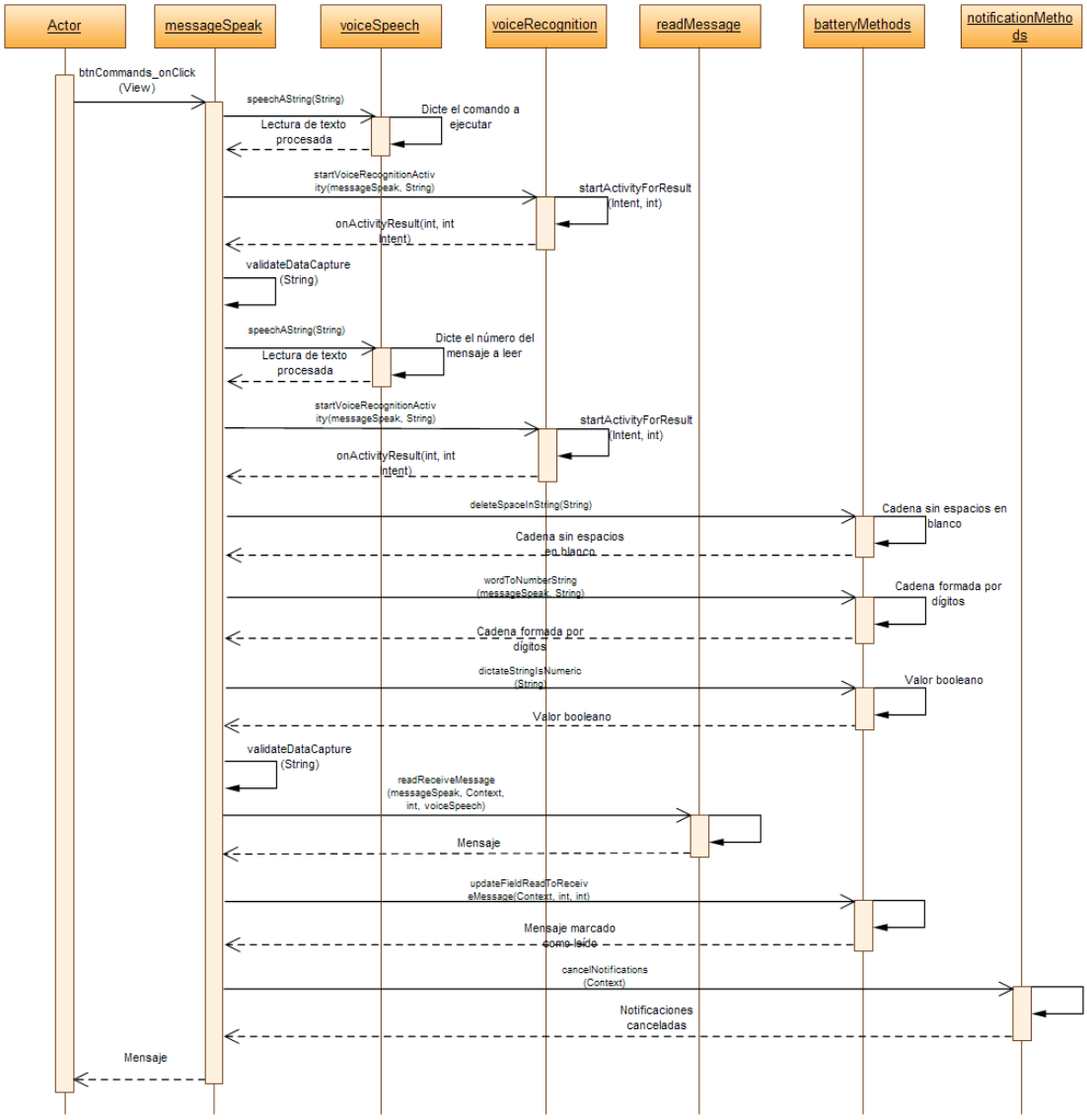

Ilustración 20: Diagrama de secuencia - Leer algún mensaje recibido

La ilustración anterior muestra la interacción entre algunas de las clases que forman parte del sistema para leer algún mensaje almacenado en la bandeja de entrada. Para hacer uso de esta funcionalidad el usuario debe emplear el comando *leer*, una vez que el usuario dicte dicho comando, el sistema pedirá al usuario que ingrese el número del mensaje que desea leer; números pequeños corresponderán con mensajes recibidos recientemente y números grandes corresponderán con mensajes antiguos; aquí es necesario destacar que solamente se admiten valores numéricos. Una vez que se ha indicado el número del mensaje que se quiere leer se procede a extraer el mensaje correspondiente de la bandeja de entrada, para ello se hace uso del método *readReceiveMessage* de la clase *readMessage*. Una vez leído el mensaje, este se marca como leído a través del método *updateFieldReadToReceiveMessage* de la clase *batteryMethods*.

3.4.6 Eliminar mensajes enviados

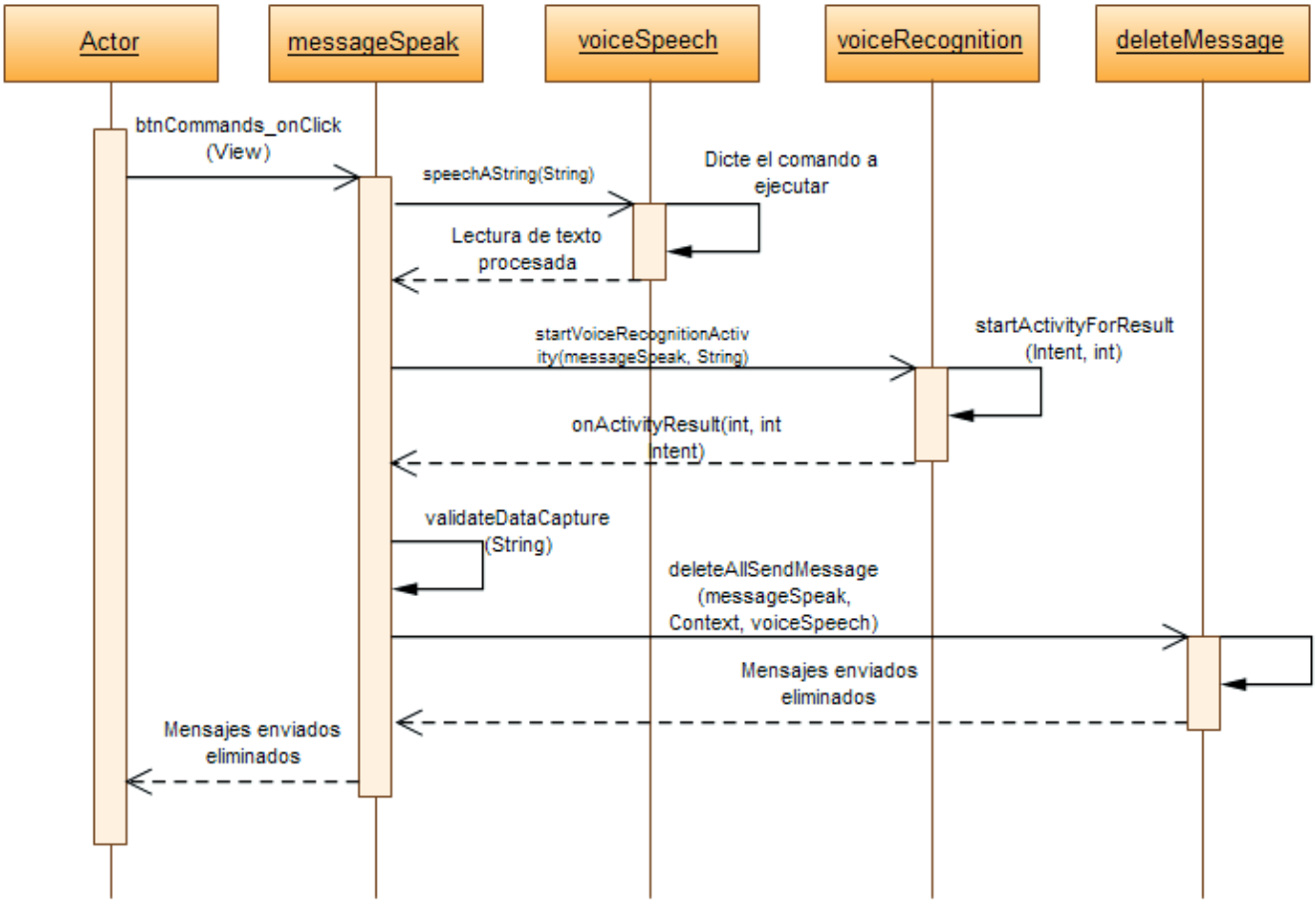

Ilustración 21: Diagrama de secuencia - Eliminar mensajes enviados

Para poder eliminar los mensajes enviados se deberá hacer uso del comando *borrar enviados*, a continuación el sistema eliminará de la bandeja de salida todos los mensajes almacenados a través del método *deleteAllSendMessage* de la clase *deleteMessage*.

3.4.7 Recibir nuevo mensaje

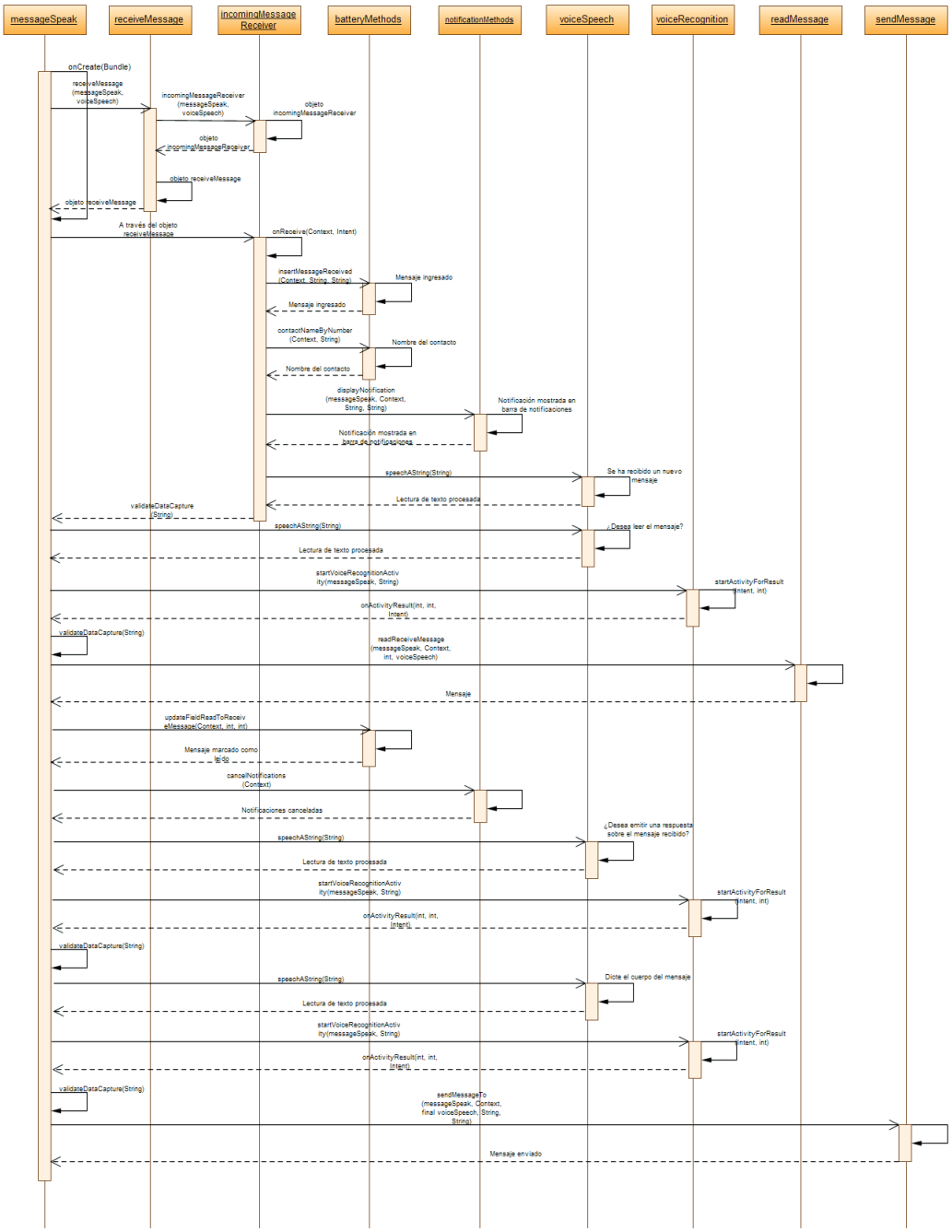

Ilustración 22: Diagrama de secuencia - Recibir nuevo mensaje

El diagrama de secuencia antes mostrado indica que para poder recibir un nuevo mensaje; el usuario no deberá hacer uso de comandos ya que los mensajes serán recibidos de forma asíncrona, sin previo aviso y en cualquier intervalo de tiempo. Para poder proporcionar esta funcionalidad lo primero que se hace es crear un objeto receptor de mensajes de difusión, en este caso se trata de un objeto de la clase *receiveMessage*, este objeto será el encargado de atender los mensajes recibidos de forma asíncrona y que se produzcan cuando se reciba un nuevo SMS.

Una vez recibido un nuevo mensaje éste se introduce dentro de la bandeja de entrada a través del método *insertMessageReceived* de la clase *batteryMethods*, luego se obtiene (cuando sea posible[3]) el nombre del emisor del mensaje, para ello es necesario acceder a los contactos y obtener el nombre a partir de su número de teléfono para ello se utiliza el método *contactNameByNumber* de la clase *batteryMethods*.

Para informar que se ha recibido un nuevo mensaje el sistema utiliza dos mecanismos, primero informa de ello al usuario haciendo uso de la barra de notificaciones del teléfono móvil-celular, luego hace uso del texto a voz y le indica de forma sonora que se ha recibido un nuevo mensaje. A continuación el sistema le consulta al usuario si desea leer el nuevo mensaje recibido, en caso de que su respuesta sea afirmativa se procederá a leer dicho mensaje a través del método *readReceiveMessage* de la clase *readMessage*, en este punto el mensaje se marca como un mensaje leído y a continuación se limpiará la barra de notificaciones del teléfono móvil-celular a través del método *cancelNotifications* de la clase *notificationMethods*. Posteriormente, el sistema le consulta al usuario si desea emitir una respuesta sobre el mensaje recibido, en caso de que su respuesta sea afirmativa el sistema le pedirá al usuario el cuerpo del mensaje de respuesta y luego procederá a enviar el mensaje de respuesta a través del método *sendMessageTo* de la clase *sendMessage*.

3.4.8 Cambiar idioma del sistema

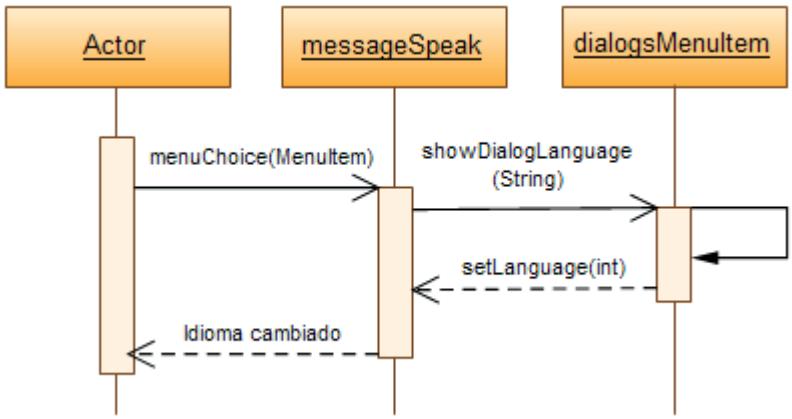

Ilustración 23: Diagrama de secuencia - Cambiar idioma del sistema

Para cambiar el idioma es necesario que el usuario seleccione la opción correspondiente del menú incorporado en el sistema. Los idiomas admitidos son: español (idioma por defecto) e inglés; ambos mostrados al usuario a través del método *showDialogLanguage* de la clase *dialogsMenuItem*. Posteriormente, para establecer el idioma seleccionado por el usuario se hace uso del método *setLanguage* de la clase *messageSpeak*.

3 Si el número de teléfono del emisor del mensaje no posee una entrada dentro de la agenda de contactos no será posible obtener el nombre del emisor de dicho mensaje

3.4.9 Cambiar velocidad de voz

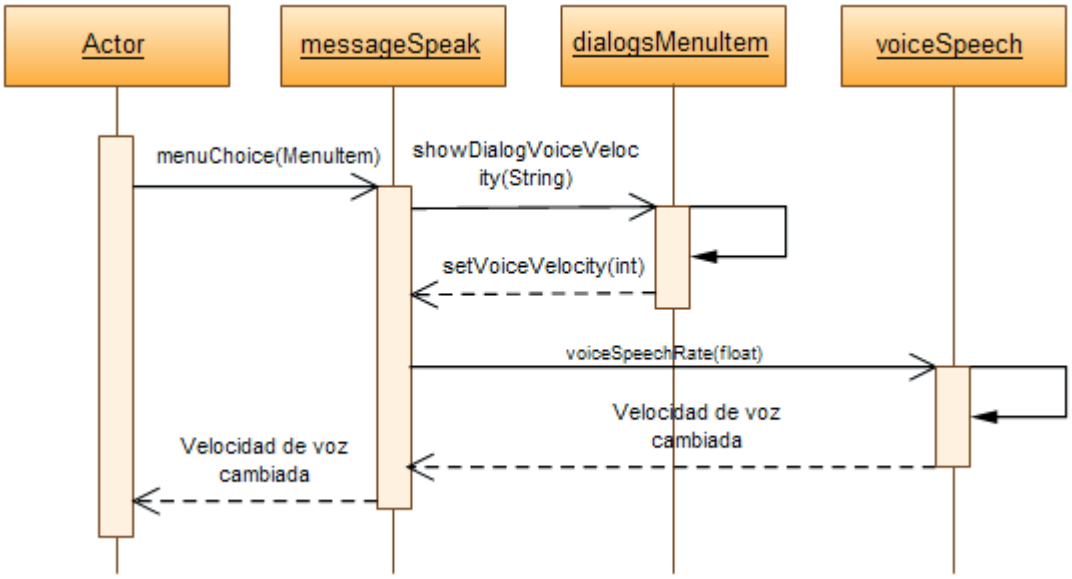

Ilustración 24: Diagrama de secuencia - Cambiar velocidad de voz

La ilustración anterior muestra que para cambiar la velocidad de la voz que emite los mensajes sonoros es necesario que el usuario seleccione la opción correspondiente del menú incorporado en el sistema. Las velocidades admitidas son: rápida, normal (valor por defecto) o lenta; todas mostradas al usuario a través del método *showDialogVoiceVelocity* de la clase *dialogsMenuItem*. Posteriormente, para establecer la velocidad de la voz seleccionada por el usuario se hace uso del método *setVoiceVelocity* de la clase *messageSpeak*. Por último se modifican los valores que actualmente posea el objeto de la clase *voiceSpeech* a través del método *voiceSpeechRate*.

3.4.10 Cambiar intensidad volumen multimedia

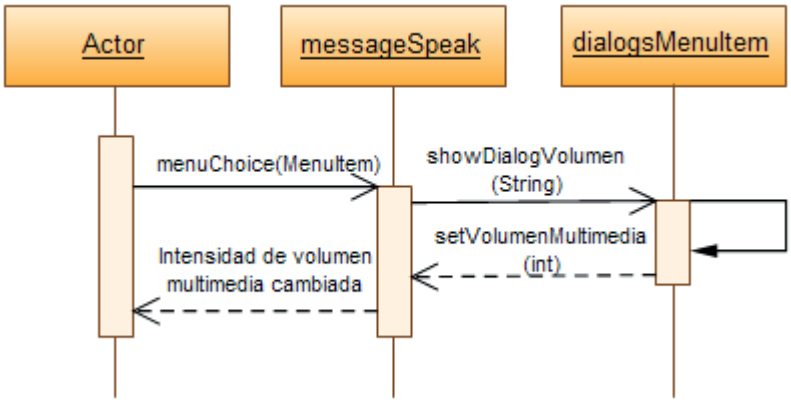

Ilustración 25: Diagrama de secuencia - Cambiar intensidad volumen multimedia

Para cambiar la intensidad del volumen multimedia es necesario que el usuario seleccione la opción correspondiente del menú incorporado en el sistema. Las velocidades admitidas son: bajo, medio o alto (valor por defecto); todas mostradas al usuario a través del método *showDialogVolumen* de la clase *dialogsMenuItem*. Posteriormente, para establecer la nueva intensidad del volumen multimedia seleccionada por el usuario se hace uso del método *setVolumenMultimedia* de la clase *messageSpeak*.

3.5 Consideraciones importantes

Un aspecto importante a citar es que el usuario puede indicar sus acciones (comandos) a través de un máximo de cinco intentos dictados mediante su voz, si después de cinco intentos el sistema no ha sido capaz de obtener la información solicitada de forma correcta (según la sintaxis de cada comando) se abortará la acción que se esté desarrollando en ese momento. Esto es aplicable para todos aquellos casos de uso que utilicen algún comando de voz admitido por el sistema.

Al iniciar el sistema se verifica si el mecanismo de reconocimiento de voz está presente en el teléfono (se emplea el método *isVoiceRecognitionPresent* de la clase *voiceRecognition*), en caso de no estarlo se notificará dicho suceso por medio del mensaje: “Detección de voz no presente en el teléfono” y se inhabilitará la posibilidad de que el usuario pueda emplear algún comando de voz para interactuar con el sistema; en caso de que el mecanismo de reconocimiento de voz se encuentre presente en el teléfono se procederá a verificar si éste se encuentra conectado a alguna red inalámbrica (se emplea el método *isNetworkAvailable* de la clase *messageSpeak*), en caso de no estarlo se notificará dicho suceso a través del mensaje: “Conexión de red no disponible” y se inhabilitará la posibilidad de que el usuario pueda emplear algún comando de voz para interactuar con el sistema; en caso de que el teléfono si se encuentre conectado a alguna red inalámbrica se habilitará la posibilidad de que el usuario pueda utilizar el sistema mediante algún comando de voz. Es importante aclarar que cuando se verifica si el teléfono está conectado a alguna red inalámbrica no se verifica si la red inalámbrica a la que se encuentra conectado cuenta con acceso a Internet.

El sistema tiene la capacidad de guardar las preferencias que han sido configuradas por el usuario. Dentro de estas preferencias se encuentran todas aquellas opciones que se muestran al pulsar el botón MENU del teléfono exceptuando la opción *Acerca de...* dichas opciones son: seleccionar voz, idioma, volumen multimedia y velocidad de voz. Una vez que el usuario modifica alguna de las opciones antes citadas su nuevo valor será almacenado (a través del método *savePreferences* de la clase *messageSpeak*) para que perdure aún cuando el sistema sea cerrado o el teléfono sea apagado. Una vez que el usuario inicia el sistema se restablecen los valores (a través del método *restorePreferences* de la clase *messageSpeak*) almacenados para cada una de las opciones antes citadas.

Resulta necesario dejar muy claro que el sistema puede presentar un funcionamiento no deseado o incorrecto bajo ciertas circunstancias como pueden ser:

- Palabras complejas dictadas con mucha rapidez.
- El Engine TTS empleado por el sistema no es capaz de realizar entonaciones interrogativas o exclamativas, por lo que suenan igual que las enunciativas.
- En el sistema de reconocimiento de voz cada frase, palabra u oración dictada requiere de

acceso a Internet, por lo que puede sufrir una leve latencia que dependerá de la velocidad de la conexión a Internet que el usuario posea.

- El volumen, el eco y el ruido de fondo juegan un papel importante en el reconocimiento de las frases, palabras u oraciones. En algunas situaciones este aspecto puede ser un factor no influyente, ya que el propio micrófono del dispositivo aplica mecanismos que tratan de paliar dichos problemas.

Capítulo IV: Conclusiones y recomendaciones

En este último capítulo se presentan las conclusiones y recomendaciones derivadas del presente trabajo. Estas conclusiones y recomendaciones se enumeran a continuación.

1 Conclusiones

Una vez diseñado, codificado y puesto a prueba el sistema plasmado en el presente trabajo se pudieron obtener las siguientes conclusiones:

1. La arquitectura y programación del sistema permitieron que éste mostrara un funcionamiento autónomo en respuesta a las necesidades originales como son: envío, recepción, lectura y eliminación de SMSs todo ello mediante comandos de voz

2. El diseño sencillo e intuitivo permite que personas con capacidades especiales (personas de tercera edad, lesionados, con discapacidades visuales, etc.) pudieran hacer un uso eficiente y frecuente del sistema

3. La incorporación de los mecanismos de texto a voz y de reconocimiento de voz de Google permitieron el desarrollo de un sistema que se vale de dichos mecanismos para la interacción de SMSs haciendo uso de la voz

4. La combinación de la tecnología móvil-celular con plataformas de trabajo flexibles y programables permiten potenciar la creación de sistemas que aprovechan los recursos gestionados por un dispositivo móvil-celular

5. Los aportes del trabajo de diseño y de programación del sistema no están limitados únicamente a entornos relacionados con el envío y la recepción de SMSs desde teléfonos móviles-celulares mediante comandos de voz. Diversos aspectos plasmados en el presente trabajo (la gramática libre de contexto, el mecanismo de reconocimiento de voz, etc.) pueden ser de gran utilidad incluso para trabajos investigativos desarrollados en otros entornos

Con todos los puntos antes mencionados es posible afirmar que se ha diseñado y desarrollado un sistema sencillo y práctico que resuelve la problemática o la necesidad planteada en la Definición del problema del presente trabajo.

2 Recomendaciones

Una vez finalizado el presente trabajo a continuación se proponen una serie de recomendaciones para la continuación y mejora del sistema aquí propuesto:

- El sistema puede ser fácilmente traducido a otros idiomas, agregándolos como opción en el menú de idiomas y creando nuevos ficheros de recursos para cada uno de los idiomas objetivos

- Utilizar un Engine TTS, diferente al aquí plasmado, que permita establecer el tipo de voz (hombre o mujer) emisora de los mensajes sonoros. El planteado únicamente tiene la capacidad de emitir mensajes sonoros a través de una voz artificial de mujer

- Publicar en el Android Market el sistema planteado para que su uso sea ampliamente

difundido por Internet

Referencias bibliográficas

(s.f.). Obtenido de http://text-to-speech.imtranslator.net/

4 feet software. (14 de Abril de 2009). Obtenido de http://www.4feets.com/2009/04/speech-recognition-in-android-sdk-15/

Ableson, W. F., Sen, R., & King, C. (2011). *Android in action* (Second edition ed.). Manning.

Araña, R., & Caballero, Y. (2007). Creación de aplicaciones móviles en Android (Curso de orientación profesional). España.

Burnette, E. (2010). *Hello, Android Introducing Google's Mobile Development Platform* (3erd edition ed.). Pragmatic Bookshelf.

Chami, F. (23 de Septiembre de 2010). *Java Code Geeks*. Obtenido de http://www.javacodegeeks.com/2010/09/android-text-to-speech-application.html

Cooper, T. (31 de Agosto de 2011). *stackoverflow*. Obtenido de http://stackoverflow.com/questions/7257615/how-to-access-r-string-xxx-resources-from-a-method-by-passing-string-xxx-as-pa

Cortés, C. A. (29 de Mayo de 2008). *Gramaticas Independientes del Contexto, ejemplos y ejercicios*. Obtenido de http://cnx.org/content/m16320/latest/

Elsey, J. (12 de Febrero de 2011). *James Elsey - Digital learnings and other such nonsense*. Obtenido de http://www.jameselsey.co.uk/blogs/techblog/android-how-to-implement-voice-recognition-a-nice-easy-tutorial/

Google Desktop. (s.f.). Obtenido de http://desktop.google.com/plugins/i/rsstospeech.html?hl=es

Lee, W.-M. (2011). *Beginning Android Application Development*. Wiley Publishing.

Legion. (28 de Marzo de 2008). *psicofxp*. Obtenido de http://www.psicofxp.com/forums/discusiones-generales.13/678750-para-que-utilizas-tu-telefono-movil.html

Murphy, M. L. (2009). *Beginning Android*. Apress.

oddcast. (s.f.). Obtenido de http://www.oddcast.com/demos/tts/tts_example.php?client

oriolpons. (18 de Octubre de 2010). *desctrl*. Obtenido de http://www.desctrl.com/blog/2010/10/18/android-fullscreen-application/

oriolpons. (25 de Octubre de 2010). *desctrl Webs & Mobile Applications*. Obtenido de http://www.desctrl.com/blog/2010/10/25/android-check-network-available/

pablotossi. (20 de Septiembre de 2008). *Celularis*. Obtenido de http://www.celularis.com/noticias/cuidado-enviar-sms-mientras-se-conduce-es-mas-peligroso-que-fumar-marihuana-o-beber-alcohol/

Paguaga, N. D. (Noviembre de 2006). *INIDE - Instituto Nacional de Información de Desarrollo*. Obtenido de http://www.inide.gob.ni/censos2005/censo2005.htm

Project, A. O. (s.f.). *Android Developers*. Obtenido de http://developer.android.com/resources/articles/tts.html

psicofxp. (7 de Abril de 2006). Obtenido de http://www.psicofxp.com/blog/backstage/50.%C2%BFpara-que-se-usan-los-telefonos-celulares.html

Rodríguez, I. P. (13 de Marzo de 2009). *Portal d'accès obert al coneixement de la UPC*. Obtenido de http://upcommons.upc.edu/pfc/handle/2099.1/6752

Varsavsky, J. A. (1 de julio de 2001). *Gesta-MP*. Obtenido de http://www.gesta.org/gesta01/artigo05.htm

Vásquez, A. (s.f.). *Organización Panamericana de la salud*. Obtenido de www.paho.org/spanish/DD/PUB/Discapacidad-SPA.pdf

Web talks. (s.f.). Obtenido de http://www.webtalks.es/

Wikipedia. (Julio de 2010). Obtenido de http://en.wikipedia.org/wiki/Speech_synthesis

Wikipedia. (Noviembre de 2011). Obtenido de http://en.wikipedia.org/wiki/Comparison_of_Android_devices

World, P. (5 de Mayo de 2011). *PC World Mexico*. Obtenido de http://www.pcworld.com.mx/Articulos/12794.htm

Anexos

1 Instalación de la aplicación

En este pequeño apartado se explica cómo instalar el sistema plasmado en el presente trabajo dentro de un teléfono móvil-celular con Android.

Para emplear el sistema se necesita del archivo ejecutable; dicho ejecutable se corresponde con un archivo con extensión *.apk*; el nombre del archivo es: *Message speak.apk*

Una vez que se cuenta con el archivo antes citado; el archivo debe ser copiado a la raíz de la memoria Micro SD del teléfono. Esto se puede realizar conectando vía USB el teléfono móvil-celular al ordenador y activando el almacenamiento USB desde el propio teléfono o utilizando un adaptador de tarjetas.

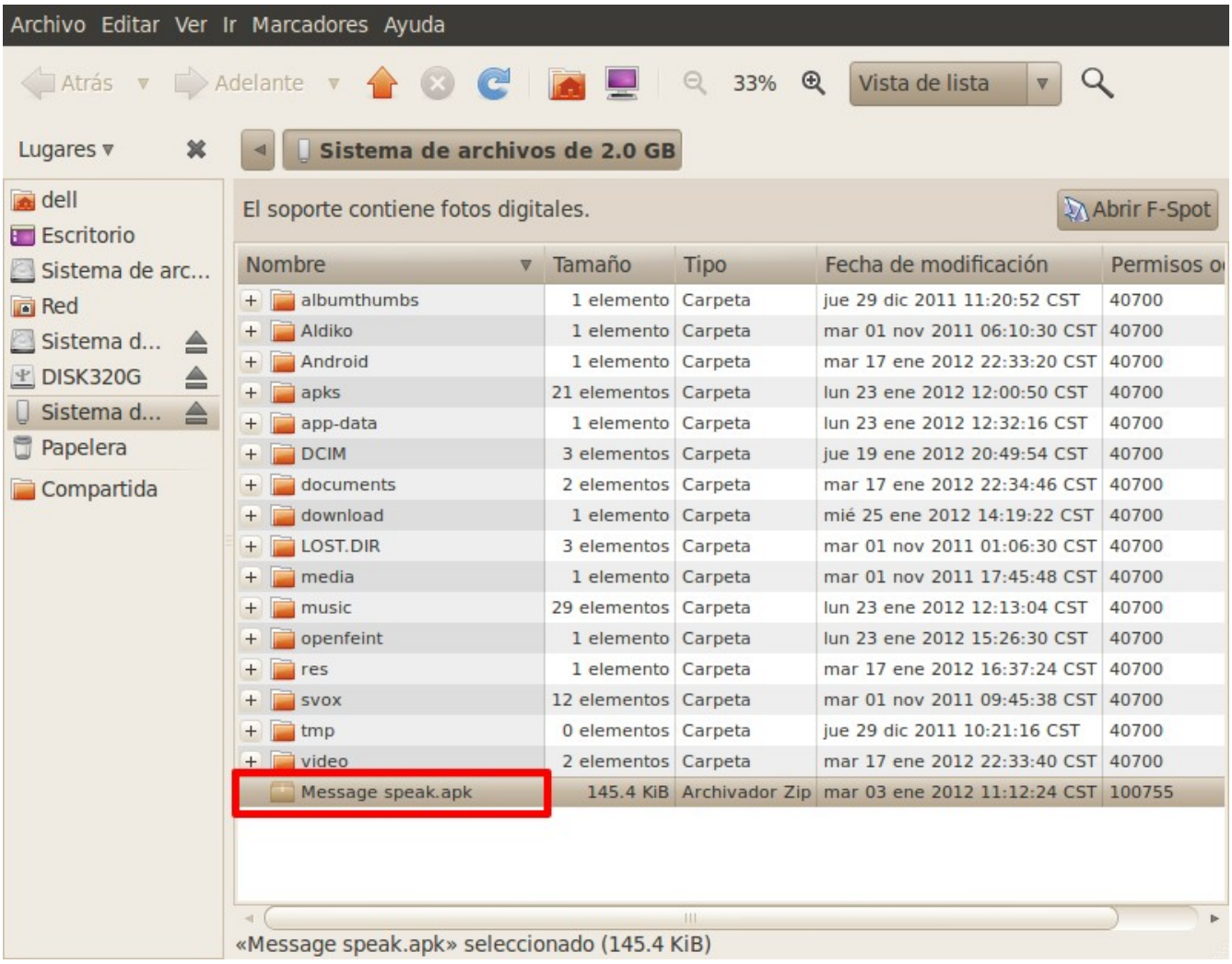

Ilustración 26: Message speak.apk en la raíz de la memoria Micro SD

Una vez que la memoria Micro SD contenga en su raíz el archivo binario *Message speak.apk* se

deberá acceder a la memoria desde el teléfono. Para ello se puede hacer uso de la aplicación *ES File Explorer* que no es más que un explorador de archivos que por defecto viene instalado en muchos teléfonos con Andorid.

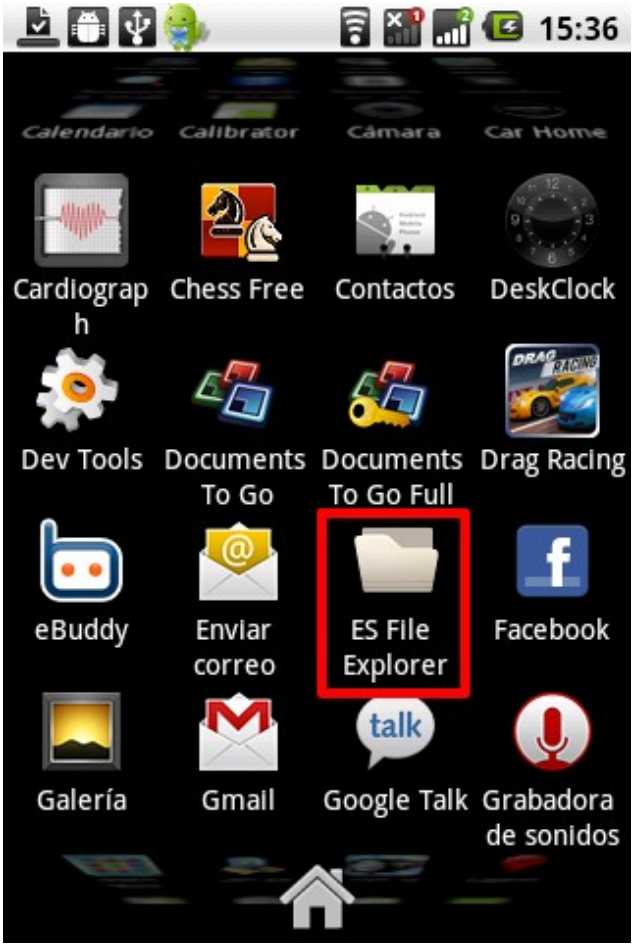

Ilustración 27: Explorador de archivos ES File Explorer

Ejecutado el explorador de archivos *ES File Explorer*, se deberá buscar, dentro del directorio *sdcard*, el archivo binario *Message speak.apk* y seleccionarlo para su instalación.

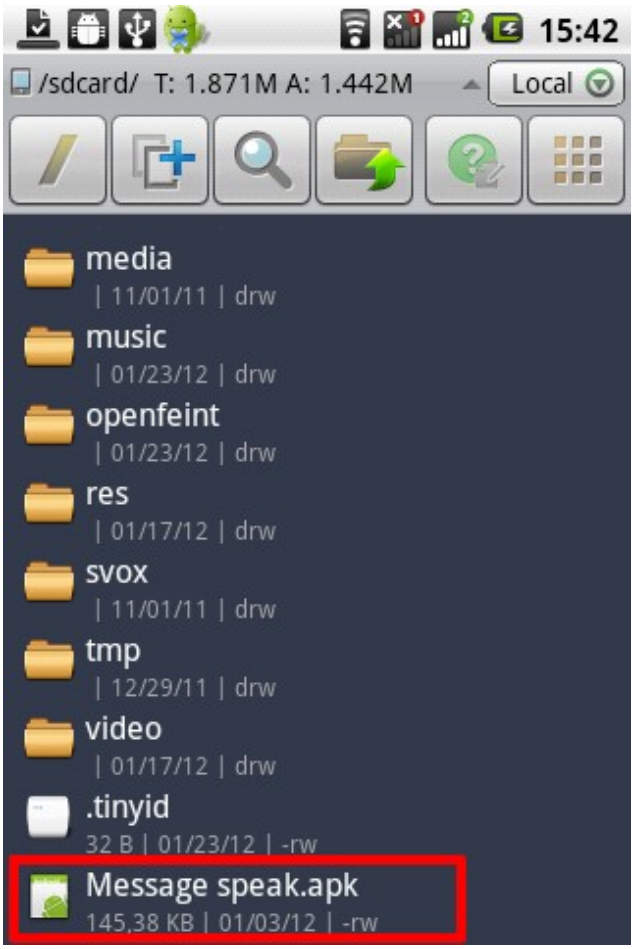

Ilustración 28: Acceso a Message speak.apk desde ES File Explorer

Una vez que se ha seleccionado el archivo binario *Message speak.apk* el sistema Android preguntará si se quiere instalar dicha aplicación; también mostrará una lista con todos los permisos que se le estarán concediendo a dicha aplicación si se acepta que sea instalada sobre el teléfono.

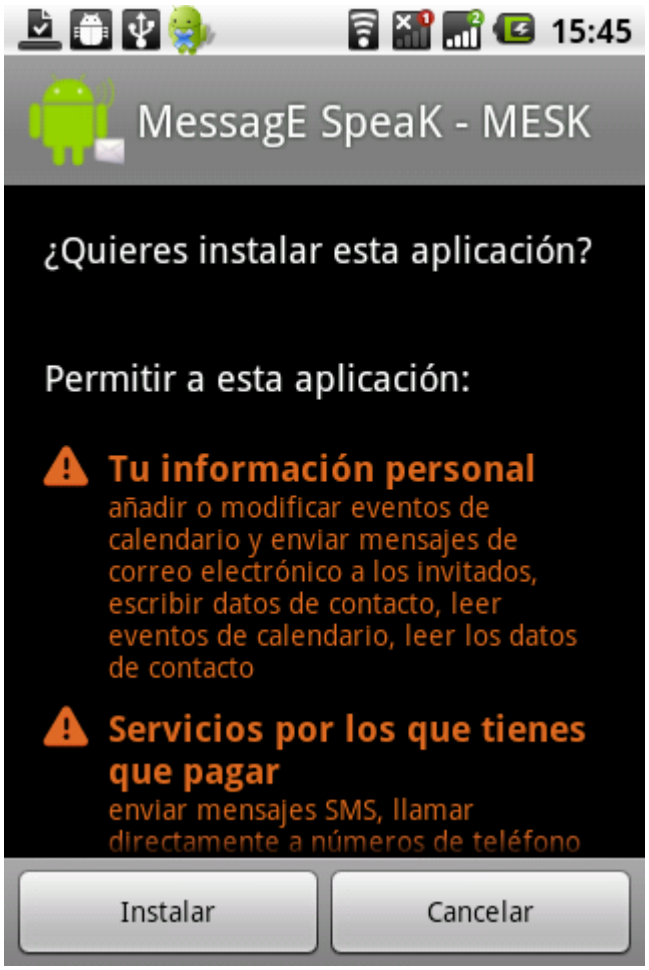

Ilustración 29: Instalando Message speak.apk

A continuación se deberá dar clic en el botón *Instalar* y después de unos segundos se tendrá la aplicación instalada con éxito.

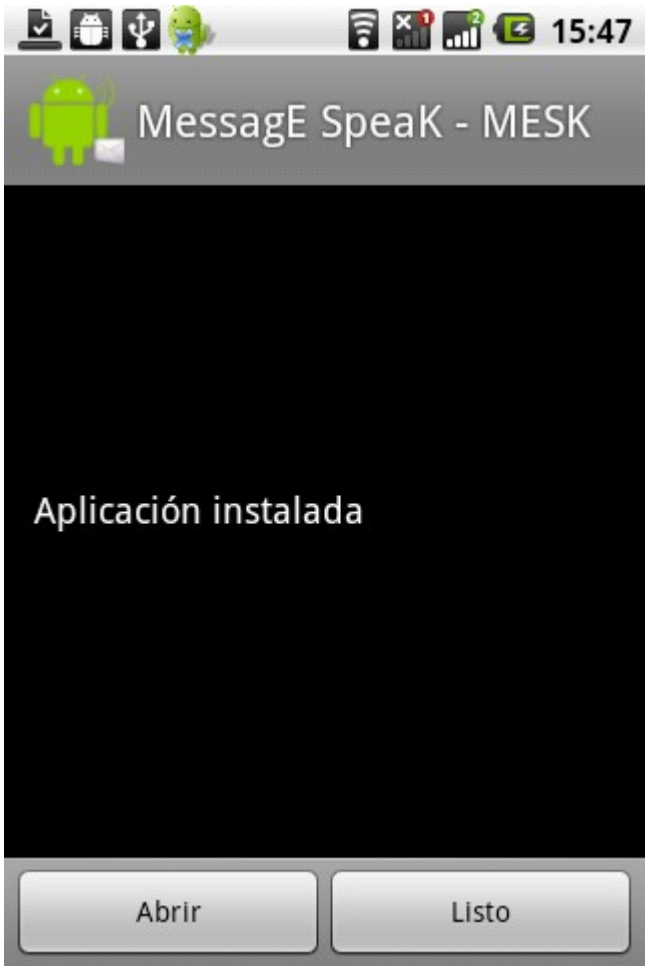

Ilustración 30: Message speak.apk instalado éxitosamente

Luego se deberá dar clic en el botón *Abrir* y la aplicación será ejecutada, a continuación estará lista para que el usuario pueda emplear los comandos que la gobiernan.

Ilustración 31: Message speak en ejecución

2 Segmentos de código o algoritmos de mayor relevancia

En este apartado se presentan algunos segmentos de código o algoritmos que se consideran de mayor relevancia para el sistema. Los segmentos de código presentados son muy simples de comprender, aún sin conocer del lenguaje Java, por lo que no se considera necesario presentarlos en un lenguaje neutral.

2.1 speechAString(String)

Método de la clase *voiceSpeech* utilizado para proporcionar la funcionalidad de texto a voz. El parámetro String corresponde con el texto a ser transformado en voz.

```
protected void speechAString(String strMessage) {
	if(strMessage.trim().length() == 0) {
		Log.d("Class: voiceSpeech - Method: speechAString", "The length of string: strMessage is zero");
		return;
	}
	objMessageSpeak.setLanguageOnInit(objMessageSpeak.iLanguage);
	tts.setLanguage(new Locale(objMessageSpeak.getString("VSSASstrLanguageArg1"),
```

```
objMessageSpeak.getString("VSSASstrLanguageArg2"))); //Configuramos el idioma de la voz que leerá los strings
        tts.speak(strMessage, TextToSpeech.QUEUE_FLUSH, null);  //Con el parámetro TextToSpeech.QUEUE_FLUSH detenemos todo lo que estuviera reproduciendose para que se pase a leer inmediatamente nuestro texto
}
```

2.2 sendMessageTo(messageSpeak, Context, final voiceSpeech, String, String)

Método de la clase *sendMessage* utilizado para enviar SMSs a un determinado destinatario. El primer parámetro String (de izquierda a derecha) corresponde con el número de teléfono del destinatario y el siguiente parámetro String corresponde con el cuerpo del mensaje.

```
protected static void sendMessageTo(messageSpeak objMessageSpeak, Context objContext, final voiceSpeech objVoiceSpeech, String strPhoneNumber, String strMessage) {
        if(objMessageSpeak == null) {
                Log.d("class: sendMessage - Method: sendMessageTo", "The object: objMessageSpeak is null");
                return;
        }

        final String SENT_SMS_FLAG = "SMS_SENT", DELIVER_SMS_FLAG = "SMS_DELIVERED", strSuccessSend = objMessageSpeak.getString("SMSMTstrSuccessSend"), strErrorSend = objMessageSpeak.getString("SMSMTstrErrorSend"), strSucessDelivered = objMessageSpeak.getString("SMSMTstrSucessDelivered"), strErrorDelivered = objMessageSpeak.getString("SMSMTstrErrorDelivered");

        if(objVoiceSpeech == null) {
                Log.d("class: sendMessage - Method: sendMessageTo", "The object: objVoiceSpeech is null");
                return;
        }

        if(objContext == null) {
                Log.d("class: sendMessage - Method: sendMessageTo", "The object: objContext is null");
                objVoiceSpeech.speechAString(strErrorSend);
                while(objVoiceSpeech.isSpeech());
                return;
        }

        if(strPhoneNumber.trim().length() == 0) {
                Log.d("class: sendMessage - Method: sendMessageTo", "The length of string: strPhoneNumber is zero");
                objVoiceSpeech.speechAString(strErrorSend);
                while(objVoiceSpeech.isSpeech());
                return;
        }

        if(strMessage.trim().length() == 0) {
                Log.d("class: sendMessage - Method: sendMessageTo", "The length of string: strMessage is zero");
                objVoiceSpeech.speechAString(strErrorSend);
                while(objVoiceSpeech.isSpeech());
                return;
        }

        Intent SentIn = new Intent(SENT_SMS_FLAG);
        PendingIntent SentPI = PendingIntent.getBroadcast(objContext, 0, SentIn, 0);

        Intent DeliverIn = new Intent(DELIVER_SMS_FLAG);
        PendingIntent DeliveredPI = PendingIntent.getBroadcast(objContext, 0, DeliverIn, 0);

        //Cuando el mensaje de texto fue enviado
        objContext.registerReceiver(new BroadcastReceiver() {
                @Override
                public void onReceive(Context context, Intent intent) {
```

```
                        switch(this.getResultCode()) {
                        case Activity.RESULT_OK:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Success: SMS
sent");
                                objVoiceSpeech.speechAString(strSuccessSend);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Error: generic
failure");
                                objVoiceSpeech.speechAString(strErrorSend);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        case SmsManager.RESULT_ERROR_NO_SERVICE:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Error: no
service");
                                objVoiceSpeech.speechAString(strErrorSend);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        case SmsManager.RESULT_ERROR_NULL_PDU:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Error: null PDU");
                                objVoiceSpeech.speechAString(strErrorSend);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        case SmsManager.RESULT_ERROR_RADIO_OFF:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Error: radio off");
                                objVoiceSpeech.speechAString(strErrorSend);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        default:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Error: sent sms
message failed");
                                objVoiceSpeech.speechAString(strErrorSend);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        }
                }
        }, new IntentFilter(SENT_SMS_FLAG));

        //Cuando el mensaje de texto fue entregado
        objContext.registerReceiver(new BroadcastReceiver() {
                @Override
                public void onReceive(Context context, Intent intent) {
                        switch(this.getResultCode()) {
                        case Activity.RESULT_OK:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Success: SMS
delivered");
                                objVoiceSpeech.speechAString(strSucessDelivered);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        case Activity.RESULT_CANCELED:
                                Log.d("class: sendMessage - Method: sendMessageTo -> onReceive", "Error: SMS not
delivered");
                                objVoiceSpeech.speechAString(strErrorDelivered);
                                while(objVoiceSpeech.isSpeech());
                                break;
                        }
                }
        }, new IntentFilter(DELIVER_SMS_FLAG));

        //Formamos grupos de mensajes, si es que estos exceden los 160 caracteres
        ArrayList<String> MultiSMS = SmsManager.getDefault().divideMessage(strMessage);
```

```
		ArrayList<PendingIntent> SentIns = new ArrayList<PendingIntent>();
		ArrayList<PendingIntent> DeliverIns = new ArrayList<PendingIntent>();

		for(int i = 0; i < MultiSMS.size(); i++) {
			SentIns.add(SentPI);
			DeliverIns.add(DeliveredPI);
		}

		try {
			//Enviamos el mensaje de texto
			SmsManager.getDefault().sendMultipartTextMessage(strPhoneNumber, null, MultiSMS, SentIns,
DeliverIns);
		}
		catch(Exception e) {
		Log.d("class: sendMessage - Method: sendMessageTo", e.getMessage());
			objVoiceSpeech.speechAString(strErrorSend);
			while(objVoiceSpeech.isSpeech());
		}
}
```

2.3 deleteLastReadReceiveMessage(messageSpeak, Context, voiceSpeech)

Método de la clase *deleteMessage* utilizado para eliminar, de la bandeja de entrada, el último mensaje recibido y marcado como leído.

```
protected static void deleteLastReadReceiveMessage(messageSpeak objMessageSpeak, Context objContext, voiceSpeech
objVoiceSpeech) {
		if(objMessageSpeak == null) {
			Log.d("class: deleteMessage - Method: deleteLastReadReceiveMessage", "The object: objMessageSpeak
is null");
			return;
		}

		String strErrorDelete = objMessageSpeak.getString("DMDLRRMstrErrorDelete"), strNoMessageToDelete =
objMessageSpeak.getString("DMDLRRMstrNoMessageToDelete"), SMS_READ_COLUMN = "read", WHERE_CONDITION
= SMS_READ_COLUMN + " = 1", SORT_ORDER = "date DESC";
		Uri SMS_INBOX_CONTENT_URI = Uri.parse("content://sms/inbox");
		Cursor cursor = null;

		if(objVoiceSpeech == null) {
			Log.d("class: deleteMessage - Method: deleteLastReadReceiveMessage", "The object: objVoiceSpeech is
null");
			return;
		}

		if(objContext == null) {
			Log.d("class: deleteMessage - Method: deleteLastReadReceiveMessage", "The object: objContext is null");
			objVoiceSpeech.speechAString(strErrorDelete);
			while(objVoiceSpeech.isSpeech());
			return;
		}

		try {
			cursor = objContext.getContentResolver().query(SMS_INBOX_CONTENT_URI, new String[] { "_id",
"thread_id", "address", "person", "date", "body" }, WHERE_CONDITION, null, SORT_ORDER);

			if(cursor == null) {
				Log.d("class: deleteMessage - Method: deleteLastReadReceiveMessage", "The cursor with Uri = "
+ SMS_INBOX_CONTENT_URI + " is null");
```

```
                        objVoiceSpeech.speechAString(strErrorDelete);
                        while(objVoiceSpeech.isSpeech());
                        return;
                }

                if(cursor.getCount() == 0)  //La consulta no regreso registros
                {
                        objVoiceSpeech.speechAString(strNoMessageToDelete);
                        while(objVoiceSpeech.isSpeech());
                        return;
                }

                if(cursor.moveToNext()) {
                        objContext.getContentResolver().delete(Uri.parse("content://sms/conversations/" +
cursor.getInt(cursor.getColumnIndexOrThrow("thread_id"))), "_id = ?", new String[]
{String.valueOf(cursor.getInt(cursor.getColumnIndexOrThrow("_id")))});

        objVoiceSpeech.speechAString(objMessageSpeak.getString("DMDLRRMstrOneMessageDelete"));
                        Log.d("class: deleteMessage - Method: deleteLastReadReceiveMessage", "1 message was
deleted");
                        Log.d("class: deleteMessage - Method: deleteLastReadReceiveMessage", "thread_id = " +
cursor.getInt(cursor.getColumnIndexOrThrow("thread_id")) + ", id = " + cursor.getInt(cursor.getColumnIndexOrThrow("_id")) +
", body = " + cursor.getString(cursor.getColumnIndexOrThrow("body")) + ", address = " +
cursor.getString(cursor.getColumnIndexOrThrow("address")) + ", date = " +
cursor.getString(cursor.getColumnIndexOrThrow("date")));
                        while(objVoiceSpeech.isSpeech());
                }
        }
        catch (Exception e) {
                Log.d("class: deleteMessage - Method: deleteLastReadReceiveMessage", e.getMessage());
                objVoiceSpeech.speechAString(strErrorDelete);
                while(objVoiceSpeech.isSpeech());
        }
        finally {
                if(cursor != null) {
                        cursor.close();
                        cursor = null;
                }
        }
}
```

2.4 deleteAllMessage(messageSpeak, Context, voiceSpeech)

Método de la clase *deleteMessage* empleado para eliminar de las bandejas de entrada y de salida todos los mensajes almacenados.

```
protected static void deleteAllMessage(messageSpeak objMessageSpeak, Context objContext, voiceSpeech
objVoiceSpeech) {
        if(objMessageSpeak == null) {
                Log.d("class: deleteMessage - Method: deleteAllMessage", "The object: objMessageSpeak is null");
                return;
        }

        String strErrorDelete = objMessageSpeak.getString("DMDARRMstrErrorDelete"), strNoMessageToDelete =
objMessageSpeak.getString("DMDAMstrNoMessageToDelete");
        Uri SMS_CONTENT_URI = Uri.parse("content://sms");
        Cursor cursor = null;

        if(objVoiceSpeech == null) {
```

```
                Log.d("class: deleteMessage - Method: deleteAllMessage", "The object: objVoiceSpeech is null");
                return;
        }

        if(objContext == null) {
                Log.d("class: deleteMessage - Method: deleteAllMessage", "The object: objContext is null");
                objVoiceSpeech.speechAString(strErrorDelete);
                while(objVoiceSpeech.isSpeech());
                return;
        }

        try {
                cursor = objContext.getContentResolver().query(SMS_CONTENT_URI, null, null, null, null);

                if(cursor == null) {
                        Log.d("class: deleteMessage - Method: deleteAllMessage", "The cursor with Uri = " +
SMS_CONTENT_URI + " is null");
                        objVoiceSpeech.speechAString(strErrorDelete);
                        while(objVoiceSpeech.isSpeech());
                        return;
                }

                if(cursor.getCount() == 0)  //La consulta no regreso registros
                {
                        objVoiceSpeech.speechAString(strNoMessageToDelete);
                        while(objVoiceSpeech.isSpeech());
                        return;
                }

                if(cursor.getCount() == 1)
                        objVoiceSpeech.speechAString(cursor.getCount()                                          +
objMessageSpeak.getString("DMDARRMstrMessageDelete"));
                else
                        objVoiceSpeech.speechAString(cursor.getCount()                                          +
objMessageSpeak.getString("DMDARRMstrMessagesDeletes"));

                while(objVoiceSpeech.isSpeech());
                Log.d("class: deleteMessage - Method: deleteAllMessage", String.valueOf(cursor.getCount()) + " message
was deleted");

                while(cursor.moveToNext()) {
                        Log.d("class: deleteMessage - Method: deleteAllMessage", "thread_id = " +
cursor.getInt(cursor.getColumnIndexOrThrow("thread_id")) + ", id = " + cursor.getInt(cursor.getColumnIndexOrThrow("_id")) +
", body = " + cursor.getString(cursor.getColumnIndexOrThrow("body")) + ", address = " +
cursor.getString(cursor.getColumnIndexOrThrow("address")) + ", date = " +
cursor.getString(cursor.getColumnIndexOrThrow("date")));
                        objContext.getContentResolver().delete(Uri.parse("content://sms/conversations/"                 +
cursor.getLong(1)), null, null);
                }
        }
        catch (Exception e) {
                Log.d("class: deleteMessage - Method: deleteAllMessage", e.getMessage());
                objVoiceSpeech.speechAString(strErrorDelete);
                while(objVoiceSpeech.isSpeech());
        }
        finally {
                if(cursor != null) {
                        cursor.close();
                        cursor = null;
                }
        }
}
```

2.5 deleteAllReadReceiveMessage(messageSpeak, Context, voiceSpeech)

Método de la clase *deleteMessage* empleado para eliminar de la bandeja de entrada todos los mensajes recibidos y que están marcados como leídos.

```
protected static void deleteAllReadReceiveMessage(messageSpeak objMessageSpeak, Context objContext, voiceSpeech objVoiceSpeech) {
	if(objMessageSpeak == null) {
		Log.d("class: deleteMessage - Method: deleteAllReadReceiveMessage", "The object: objMessageSpeak is null");
		return;
	}

	String strErrorDelete = objMessageSpeak.getString("DMDARRMstrErrorDelete"), strNoMessageToDelete = objMessageSpeak.getString("DMDARRMstrNoMessageToDelete"), SMS_READ_COLUMN = "read", WHERE_CONDITION = SMS_READ_COLUMN + " = 1", SORT_ORDER = "date DESC";
	Uri SMS_INBOX_CONTENT_URI = Uri.parse("content://sms/inbox");
	Cursor cursor = null;

	if(objVoiceSpeech == null) {
		Log.d("class: deleteMessage - Method: deleteAllReadReceiveMessage", "The object: objVoiceSpeech is null");
		return;
	}

	if(objContext == null) {
		Log.d("class: deleteMessage - Method: deleteAllReadReceiveMessage", "The object: objContext is null");
		objVoiceSpeech.speechAString(strErrorDelete);
		while(objVoiceSpeech.isSpeech());
		return;
	}

	try {
		cursor = objContext.getContentResolver().query(SMS_INBOX_CONTENT_URI, new String[] { "_id", "thread_id", "address", "person", "date", "body" }, WHERE_CONDITION, null, SORT_ORDER);

		if(cursor == null) {
		Log.d("class: deleteMessage - Method: deleteAllReadReceiveMessage", "The cursor with Uri: " + SMS_INBOX_CONTENT_URI + " is null");
		objVoiceSpeech.speechAString(strErrorDelete);
		while(objVoiceSpeech.isSpeech());
		return;
	}

		if(cursor.getCount() == 0)  //La consulta no regreso registros
		{
			objVoiceSpeech.speechAString(strNoMessageToDelete);
			while(objVoiceSpeech.isSpeech());
			return;
		}

		if(cursor.getCount() == 1)
			objVoiceSpeech.speechAString(cursor.getCount() + objMessageSpeak.getString("DMDARRMstrMessageDelete"));
		else
			objVoiceSpeech.speechAString(cursor.getCount() + objMessageSpeak.getString("DMDARRMstrMessagesDeletes"));
```

```
                while(objVoiceSpeech.isSpeech());
                Log.d("class: deleteMessage - Method: deleteAllReadReceiveMessage", String.valueOf(cursor.getCount())
+ " message was deleted");

                while(cursor.moveToNext()) {
                        Log.d("class: deleteMessage - Method: deleteAllReadReceiveMessage", "thread_id = " +
cursor.getInt(cursor.getColumnIndexOrThrow("thread_id")) + ", id = " + cursor.getInt(cursor.getColumnIndexOrThrow("_id")) +
", body = " + cursor.getString(cursor.getColumnIndexOrThrow("body")) + ", address = " +
cursor.getString(cursor.getColumnIndexOrThrow("address")) + ", date = " +
cursor.getString(cursor.getColumnIndexOrThrow("date")));
                        objContext.getContentResolver().delete(Uri.parse("content://sms/conversations/" +
cursor.getInt(cursor.getColumnIndexOrThrow("thread_id"))), "_id = ?", new String[]
{String.valueOf(cursor.getInt(cursor.getColumnIndexOrThrow("_id")))});
                }
        }
        catch (Exception e) {
                Log.d("class: deleteMessage - Method: deleteAllReadReceiveMessage", e.getMessage());
                objVoiceSpeech.speechAString(strErrorDelete);
                while(objVoiceSpeech.isSpeech());
        }
        finally {
                if(cursor != null) {
                        cursor.close();
                        cursor = null;
                }
        }
}
```

2.6 readReceiveMessage(messageSpeak, Context, int, voiceSpeech)

Método de la clase *readMessage* empleado para obtener de la bandeja de entrada un determinado mensaje recibido. El mensaje a obtener se pasa en el tercer parámetro int (de izquierda a derecha).

```
protected static void readReceiveMessage(messageSpeak objMessageSpeak, Context objContext, int iMessageNumber,
voiceSpeech objVoiceSpeech) {
        String strErrorReadMessage = objMessageSpeak.getString("RMRRMstrErrorReadMessage"), strNoMessage =
objMessageSpeak.getString("RMRRMstrNoMessage"), SORT_ORDER = "date DESC";
        Uri SMS_INBOX_CONTENT_URI = Uri.parse("content://sms/inbox");
        Cursor cursor = null;
        int iRetorno = -1;

        if(objVoiceSpeech == null) {
                Log.d("class: readMessage - Method: readReceiveMessage", "The object: objVoiceSpeech is null");
                return;
        }

        if(objContext == null) {
                Log.d("class: readMessage - Method: readReceiveMessage", "The object: objContext is null");
                objVoiceSpeech.speechAString(strErrorReadMessage);
                while(objVoiceSpeech.isSpeech());
                return;
        }

        try {
                cursor = objContext.getContentResolver().query(SMS_INBOX_CONTENT_URI, new String[] { "_id",
"thread_id", "address", "person", "date", "body" }, null, null, SORT_ORDER);

                if(cursor == null) {
                        Log.d("class: readMessage - Method: readReceiveMessage", "The cursor with Uri = " +
```

```
SMS_INBOX_CONTENT_URI + " is null");
                                objVoiceSpeech.speechAString(strErrorReadMessage);
                                while(objVoiceSpeech.isSpeech());
                                return;
                        }

                        if(cursor.getCount() == 0) {
                                objVoiceSpeech.speechAString(strNoMessage);
                                while(objVoiceSpeech.isSpeech());
                                return;
                        }

                        for(int i = 1; cursor.moveToNext(); i++) {
                                if(i == iMessageNumber) {

                objVoiceSpeech.speechAString(objMessageSpeak.getString("NMDNstrReceiveMessage")                        +
batteryMethods.contactNameByNumber(objContext,        cursor.getString(cursor.getColumnIndexOrThrow("address")))       +
objMessageSpeak.getString("RMRRMstrWithBody") + cursor.getString(cursor.getColumnIndexOrThrow("body")));
                                        strNumberMessageReceive                                                        =
cursor.getString(cursor.getColumnIndexOrThrow("address"));  //Guardamos el número del emisor del mensaje
                                        while(objVoiceSpeech.isSpeech());
                                        iRetorno = 1;
                                        break;
                                }
                        }

                        if(iRetorno != 1) {

                objVoiceSpeech.speechAString(objMessageSpeak.getString("RMRRMstrReceiveMessageNumber")                 +
String.valueOf(iMessageNumber) + objMessageSpeak.getString("RMRRMstrNotFound"));
                                while(objVoiceSpeech.isSpeech());
                        }
                }
                catch (Exception e) {
                        Log.d("class: readMessage - Method: readReceiveMessage", e.getMessage());
                        objVoiceSpeech.speechAString(strErrorReadMessage);
                        while(objVoiceSpeech.isSpeech());
                }
                finally {
                        if(cursor != null) {
                                cursor.close();
                                cursor = null;
                        }
                }
}
```

2.7 idLastReceivedMessage(Context)

Método de la clase *batteryMethods* empleado cuando se recibe un nuevo mensaje y el usuario confirma que quiere leer ese nuevo mensaje. Retorna el identificador del último mensaje recibido (almacenado en la bandeja de entrada) y que aún no ha sido leído.

```
protected static int idLastReceivedMessage(Context objContext) {
        String SMS_READ_COLUMN = "read", WHERE_CONDITION = SMS_READ_COLUMN + " = 0", SORT_ORDER =
"date DESC";
        Uri SMS_INBOX_CONTENT_URI = Uri.parse("content://sms/inbox");
        Cursor cursor = null;
        int iRetorno = -1;
```

```
	if(objContext == null) {
		Log.d("class: batteryMethods - Method: idLastReceiveMessage", "The object: objContext is null");
		return iRetorno;
	}

	try {
		cursor = objContext.getContentResolver().query(SMS_INBOX_CONTENT_URI, new String[] { "_id",
"thread_id", "address", "person", "date", "body" }, WHERE_CONDITION, null, SORT_ORDER);

		if(cursor == null) {
			Log.d("class: batteryMethods - Method: idLastReceiveMessage", "The cursor with Uri: " +
SMS_INBOX_CONTENT_URI + " is null");
			return iRetorno;
		}

		if(cursor.getCount() == 0)  //La consulta no regreso registros
		{
			Log.d("class: batteryMethods - Method: idLastReceiveMessage", "The cursor no contains
registers");
			return iRetorno;
		}

		if(cursor.moveToNext())
			iRetorno = cursor.getInt(cursor.getColumnIndexOrThrow("_id"));
	}
	catch (Exception e) {
		Log.d("class: batteryMethods - Method: idLastReceiveMessage", e.getMessage());
		return iRetorno;
	}
	finally {
		if(cursor != null) {
			cursor.close();
			cursor = null;
		}
	}

	Log.d("class: batteryMethods - Method: idLastReceiveMessage", "The _id of the last received message was: " +
String.valueOf(iRetorno));
	return iRetorno;
}
```

2.8 updateFieldReadToReceiveMessage(Context, int, int)

Método de la clase *batteryMethods* utilizado cuando se recibe un nuevo SMS y el usuario confirma que quiere leer dicho mensaje. Permite marcar como leído el nuevo mensaje recibido.

```
protected static void updateFieldReadToReceiveMessage(Context objContext, int iValue, int iIdMessage) {
	if(objContext == null) {
		Log.d("class: batteryMethods - Method: updateFieldReadToReceiveMessage", "The object: objContext is
null");
		return;
	}

	if(iValue == -1) {
		Log.d("class: batteryMethods - Method: updateFieldReadToReceiveMessage", "The value of iValue is -1");
		return;
	}

	if(iIdMessage == -1) {
```

```
                Log.d("class: batteryMethods - Method: updateFieldReadToReceiveMessage", "The value of iIdMessage is
-1");
                return;
        }

        ContentValues values = new ContentValues();
        values.put("read", iValue);

        Log.d("class: batteryMethods - Method: updateFieldReadToReceiveMessage", "The number of rows afected was: "
+ objContext.getContentResolver().update(Uri.parse("content://sms/inbox"), values, "_id=" + iIdMessage, null));
}
```

2.9 deleteAllSendMessage(messageSpeak, Context, voiceSpeech)

Método de la clase *deleteMessage* utilizado por el comando: "borrar enviados". El método elimina, de la bandeja de salida, todos los mensajes enviados desde el teléfono.

```
protected static void deleteAllSendMessage(messageSpeak objMessageSpeak, Context objContext, voiceSpeech
objVoiceSpeech) {
        if(objMessageSpeak == null) {
                Log.d("class: deleteMessage - Method: deleteAllSendMessage", "The object: objMessageSpeak is null");
                return;
        }

        String strErrorDelete = objMessageSpeak.getString("DMDARRMstrErrorDelete"), strNoMessageToDelete =
objMessageSpeak.getString("DMDASMstrNoMessageToDelete");
        Uri SMS_SENT_CONTENT_URI = Uri.parse("content://sms/sent");
        Cursor cursor = null;

        if(objVoiceSpeech == null) {
                Log.d("class: deleteMessage - Method: deleteAllSendMessage", "The object: objVoiceSpeech is null");
                return;
        }

        if(objContext == null) {
                Log.d("class: deleteMessage - Method: deleteAllSendMessage", "The object: objContext is null");
                objVoiceSpeech.speechAString(strErrorDelete);
                while(objVoiceSpeech.isSpeech());
                return;
        }

        try {
                cursor = objContext.getContentResolver().query(SMS_SENT_CONTENT_URI, null, null, null, null);

                if(cursor == null) {
                        Log.d("class: deleteMessage - Method: deleteAllSendMessage", "The cursor with Uri = " +
SMS_SENT_CONTENT_URI + " is null");
                        objVoiceSpeech.speechAString(strErrorDelete);
                        while(objVoiceSpeech.isSpeech());
                        return;
                }

                if(cursor.getCount() == 0)  //La consulta no regreso registros
                {
                        objVoiceSpeech.speechAString(strNoMessageToDelete);
                        while(objVoiceSpeech.isSpeech());
                        return;
                }
```

```
                if(cursor.getCount() == 1)
                        objVoiceSpeech.speechAString(cursor.getCount() +
objMessageSpeak.getString("DMDARRMstrMessageDelete"));
                else
                        objVoiceSpeech.speechAString(cursor.getCount() +
objMessageSpeak.getString("DMDARRMstrMessagesDeletes"));

                while(objVoiceSpeech.isSpeech());
                Log.d("class: deleteMessage - Method: deleteAllSendMessage", String.valueOf(cursor.getCount()) + "
message was deleted");

                while(cursor.moveToNext()) {
                        Log.d("class: deleteMessage - Method: deleteAllSendMessage", "thread_id = " +
cursor.getInt(cursor.getColumnIndexOrThrow("thread_id")) + ", id = " + cursor.getInt(cursor.getColumnIndexOrThrow("_id")) +
", body = " + cursor.getString(cursor.getColumnIndexOrThrow("body")) + ", address = " +
cursor.getString(cursor.getColumnIndexOrThrow("address")) + ", date = " +
cursor.getString(cursor.getColumnIndexOrThrow("date")));
                        objContext.getContentResolver().delete(Uri.parse("content://sms/conversations/" +
cursor.getLong(1)), null, null);
                }
        }
        catch (Exception e) {
                Log.d("class: deleteMessage - Method: deleteAllSendMessage", e.getMessage());
                objVoiceSpeech.speechAString(strErrorDelete);
                while(objVoiceSpeech.isSpeech());
        }
        finally {
                if(cursor != null) {
                        cursor.close();
                        cursor = null;
                }
        }
}
```

2.10 insertMessageReceived(Context, String, String)

Método de la clase *batteryMethods* utilizado cuando se recibe un nuevo SMS. El método inserta el nuevo mensaje recibido dentro de la bandeja de entrada y lo marca como no leído.

```
protected static void insertMessageReceived(Context objContext, String strAddress, String strBody) {
        if(objContext == null) {
                Log.d("class: batteryMethods - Method: insertMessageReceived", "The object: objContext is null");
                return;
        }

        if(strAddress.trim().length() == 0) {
                Log.d("class: batteryMethods - Method: insertMessageReceived", "The length of string: strAddress is
zero");
                return;
        }

        if(strBody.trim().length() == 0) {
                Log.d("class: batteryMethods - Method: insertMessageReceived", "The length of string: strBody is zero");
                return;
        }

        ContentValues values = new ContentValues();
        values.put("address", strAddress);
        values.put("read", 0);  //Lo marcamos como no leído
```

```
        values.put("type", 1);  //Indicamos que es de tipo sms
        values.put("body", strBody);

        //Insertamos el nuevo mensaje con los valores antes especificados
        objContext.getContentResolver().insert(Uri.parse("content://sms"), values);
}
```

2.11 contactNameByNumber(Context, String)

Método de la clase *batteryMethods* empleado cuando se recibe un nuevo mensaje. El método accede a la agenda telefónica, busca y retorna el nombre del contacto asociado con el número de teléfono pasado en el segundo parámetro (de izquierda a derecha) de tipo String.

```
protected static String contactNameByNumber(Context objContext, String strNumber) {
        String strRetorno = strNumber;
        Cursor cursor = null;
        String []mPhoneNumberProjection = { PhoneLookup._ID, PhoneLookup.NUMBER,
PhoneLookup.DISPLAY_NAME };
        Uri lookupUri = Uri.withAppendedPath(PhoneLookup.CONTENT_FILTER_URI, Uri.encode(strNumber));

        if(objContext == null) {
                Log.d("class: batteryMethods - Method: contactNameByNumber", "The object: objContext is null");
                return strRetorno;
        }

        if(strNumber.trim().length() == 0) {
                Log.d("class: batteryMethods - Method: contactNameByNumber", "The length of string: strNumber is
zero");
                return strRetorno;
        }

        try {
                cursor = objContext.getContentResolver().query(lookupUri, mPhoneNumberProjection, null, null, null);

                if(cursor.moveToFirst())
                        strRetorno = cursor.getString(2);
        }
        catch (Exception e) {
                Log.d("class: batteryMethods - Method: contactNameByNumber", e.getMessage());
                return strRetorno;
        }
        finally {
                if(cursor != null) {
                        cursor.close();
                        cursor = null;
                }
        }

        return strRetorno;
}
```

2.12 displayNotification(messageSpeak, Context, String, String)

Método de la clase *notificationMethods* empleado cuando se recibe un nuevo SMS. El método muestra una notificación en la barra de notificaciones del teléfono indicando que se ha recibido un nuevo mensaje. El tercer parámetro (de izquierda a derecha) de tipo String corresponde con el

número o el nombre del emisor del mensaje y el siguiente parámetro de tipo String corresponde con el cuerpo del mensaje.

```
protected static void displayNotification(messageSpeak objMessageSpeak, Context objContext, String strPersonOrNumber,
String strBody) {
	if(objMessageSpeak == null) {
		Log.d("class: notificationMethods - Method: displayNotification", "The object: objMessageSpeak is null");
		return;
	}

	if(objContext == null) {
		Log.d("class: notificationMethods - Method: displayNotification", "The object: objContext is null");
		return;
	}

	if(strPersonOrNumber.trim().length() == 0) {
		Log.d("class: notificationMethods - Method: displayNotification", "The length of string: strPersonOrNumber
is zero");
		return;
	}

	if(strBody.trim().length() == 0) {
		Log.d("class: notificationMethods - Method: displayNotification", "The length of string: strBody is zero");
		return;
	}

	NotificationManager manager = (NotificationManager)
objContext.getSystemService(Context.NOTIFICATION_SERVICE);
	Notification notification = new Notification(R.drawable.icon,
objMessageSpeak.getString("NMDNstrReceiveMessage") + strPersonOrNumber, System.currentTimeMillis());

	PendingIntent contentIntent = PendingIntent.getActivity(objContext, 1, new Intent(objContext,
messageSpeak.class), 0);
	notification.setLatestEventInfo(objContext, strPersonOrNumber, strBody, contentIntent);
	notification.vibrate = new long [] { 100, 250, 100, 500 };  //Hacemos vibrar el teléfono
	manager.notify(notificationID, notification);
}
```

2.13 cancelNotifications(Context)

Método de la clase *notificationMethods* empleado cuando el usuario hace uso de las siguientes acciones: utiliza el comando *borrar todo*, se sale del sistema o confirma que quiere leer un nuevo mensaje recibido. El método elimina, de la barra de notificaciones, las notificaciones colocadas por el sistema.

```
protected static void cancelNotifications(Context objContext) {
		if(objContext == null) {
			Log.d("class: notificationMethods - Method: cancelNotifications", "The object: objContext is null");
			return;
		}

	NotificationManager manager = (NotificationManager)
objContext.getSystemService(Context.NOTIFICATION_SERVICE);
	manager.cancel(notificationID);
}
```

2.14 showDialogVolumen(String)

Método de la clase *dialogsMenuItem* empleado para mostrar las opciones (bajo, medio y alto) asociadas al menú: *volumen multimedia* del sistema. El parámetro String contienen el título que aparecerá en el encabezado de dichas opciones.

```
protected void showDialogVolumen(String strTitulo) {
	setTitle(strTitulo);
	setCancelable(false);
	setItems(R.array.volumen,
	new DialogInterface.OnClickListener() {
		public void onClick(DialogInterface dialog, int which) {
			objMessageSpeak.setVolumenMultimedia(which);
		}
	});
	show();
}
```

2.15 setVolumenMultimedia(int)

Método de la clase *messageSpeak* empleado para ajustar los valores para el volumen multimedia. El parámetro int que recibe puede valer 0 para bajo, 1 para medio y 2 para alto.

```
protected void setVolumenMultimedia(int iWhich) {
	switch(iWhich) {
	case 0:  //Bajar el volumen multimedia
		for(int i = 0; i < 15; i++)
			audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,    AudioManager.ADJUST_LOWER,
AudioManager.FLAG_SHOW_UI);
		for(int i = 0; i < 5; i++)
			audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,     AudioManager.ADJUST_RAISE,
AudioManager.FLAG_SHOW_UI);
		break;
	case 1:  //Colocar a la mitad el volumen multimedia
		for(int i = 0; i < 15; i++)
			audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,    AudioManager.ADJUST_LOWER,
AudioManager.FLAG_SHOW_UI);
		for(int i = 0; i < 8; i++)
			audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,     AudioManager.ADJUST_RAISE,
AudioManager.FLAG_SHOW_UI);
		break;
	case 2:  //Subir el volumen multimedia
		for(int i = 0; i < 15; i++)
			audio.adjustStreamVolume(AudioManager.STREAM_MUSIC,     AudioManager.ADJUST_RAISE,
AudioManager.FLAG_SHOW_UI);
		break;
	}

	iVolumen = iWhich;
}
```

2.16 showDialogVoiceVelocity(String)

Método de la clase *dialogsMenuItem* empleado para mostrar las opciones (rápida, normal y lenta) asociadas al menú: *seleccionar voz del sistema*. El parámetro String contienen el título que aparecerá

en el encabezado de dichas opciones.

```
protected void showDialogVoiceVelocity(String strTitulo) {
	setTitle(strTitulo);
	setCancelable(false);
	setItems(R.array.voiceVelocity,
	new DialogInterface.OnClickListener() {
		public void onClick(DialogInterface dialog, int which) {
			objMessageSpeak.setVoiceVelocity(which);
		}
	});
	show();
}
```

2.17 setVoiceVelocity(int)

Método de la clase *messageSpeak* empleado para ajustar los valores para la voz artificial generada por el mecanismo de texto a voz. El parámetro int que recibe puede valer 0 para rápida, 1 para normal y 2 para lenta.

```
protected void setVoiceVelocity(int iWhich) {
	float fSlow = (float)0.4, fNormal = (float)1.0, fFast = (float)1.4;

	switch(iWhich) {
	case 0:  //Fast speech
		objVoiceSpeech.voiceSpeechRate(fFast);
		break;
	case 1:  //Normal speech
		objVoiceSpeech.voiceSpeechRate(fNormal);
		break;
	case 2:  //Slow speech
		objVoiceSpeech.voiceSpeechRate(fSlow);
		break;
	}

	iVelocity = iWhich;
}
```

2.18 showDialogLanguage(String)

Método de la clase *dialogsMenuItem* empleado para mostrar las opciones (rápida, normal y lenta) asociadas al menú: *seleccionar voz del sistema*. El parámetro String contienen el título que aparecerá en el encabezado de dichas opciones.

```
protected void showDialogLanguage(String strTitulo) {
	setTitle(strTitulo);
	setCancelable(false);
	setItems(R.array.language,
	new DialogInterface.OnClickListener() {
		public void onClick(DialogInterface dialog, int which) {
			objMessageSpeak.setLanguage(which);
		}
	});
	show();
}
```

2.19 setLanguage(int)

Método de la clase *messageSpeak* empleado para ajustar los valores para el idioma. El parámetro int que recibe puede valer 0 para español y 1 para inglés. El método *setLanguage* necesita del método *setLanguageOnInit* para poder funcionar, es por ello que su código se muestra más abajo.

```
protected void setLanguage(int iWhich) {
	this.setLanguageOnInit(iWhich);

	if(!voiceRecognition.isVoiceRecognitionPresent(this))
		btnCommands.setText(this.getString(R.string.txtVoiceNoPresent));
	else
	{
		if(!isNetworkAvailable())
			btnCommands.setText(this.getString(R.string.txtNoConnection));
		else
			btnCommands.setText(this.getString(R.string.btnCommands));
	}

	btnExit.setText(this.getString(R.string.btnExit));

	iLanguage = iWhich;
}
```

```
protected void setLanguageOnInit(int iWhich) {
	Locale locale = null;
	Configuration config = null;

	switch(iWhich) {
	case 0:  //Spanish
		locale = new Locale("spa", "ESP");
	Locale.setDefault(locale);
	config = new Configuration();
	config.locale = locale;
	getBaseContext().getResources().updateConfiguration(config,
getBaseContext().getResources().getDisplayMetrics());
	break;
	case 1:  //English
		locale = new Locale("eng", "USA");
	Locale.setDefault(locale);
	config = new Configuration();
	config.locale = locale;
	getBaseContext().getResources().updateConfiguration(config,
getBaseContext().getResources().getDisplayMetrics());
		break;
	}

	iLanguage = iWhich;
}
```

2.20 isVoiceRecognitionPresent(Context)

Método de la clase *voiceRecognition* utilizado para determinar si el mecanismo de reconocimiento de voz (de voz a texto) se encuentra presente en el teléfono. Retorna un valor booleano true cuando el reconocimiento de voz esta presente o false cuando no lo está.

```
protected static boolean isVoiceRecognitionPresent(Context objContext) {
	boolean bRetorno = false;

	if(objContext == null) {
		Log.d("class: voiceRecognition - Method: isVoiceRecognitionPresent", "The object: objContext is null");
		return bRetorno;
	}

	PackageManager pm = objContext.getPackageManager();

	if (pm.queryIntentActivities(new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0).size() > 0)
		bRetorno = true;

	return bRetorno;
}
```

2.21 isNetworkAvailable()

Método de la clase *messageSpeak* utilizado para determinar si el teléfono se encuentra conectado a alguna red inalámbrica. Retorna un valor booleano true cuando el teléfono se encuentra conectado a una red inalámbrica o false cuando no lo esta. El método no verifica si la red inalámbrica a la que se encuentra conectado el teléfono tiene acceso a Internet.

```
private boolean isNetworkAvailable() {
	ConnectivityManager objConectivityManager =
(ConnectivityManager)this.getApplicationContext().getSystemService(Context.CONNECTIVITY_SERVICE);

	if(objConectivityManager == null) {
		Log.d("Activity: messageSpeak - Method: isNetworkAvailable", "The connection is not available");
	}
	else {
	NetworkInfo[] arrayNetworkInfo = objConectivityManager.getAllNetworkInfo();

	if(arrayNetworkInfo != null) {
			for(int i = 0; i < arrayNetworkInfo.length; i++) {
				if(arrayNetworkInfo[i].getState() == NetworkInfo.State.CONNECTED)
					return true;
			}
		}
	}

	return false;
}
```

2.22 savePreferences()

Método de la clase *messageSpeak* utilizado para guardar de forma permanente las preferencias del sistema. Dentro de las preferencias que son guardas se encuentran: el idioma, la velocidad de la voz y la intensidad del volumen multimedia; todas ellas configurables por el usuario.

```
private void savePreferences() {
	int mode = Activity.MODE_PRIVATE;
	pref = this.getSharedPreferences("myPreferences", mode);
```

```
        SharedPreferences.Editor editor = pref.edit();

        editor.putInt("iVolumen", iVolumen);
        editor.putInt("iVelocity", iVelocity);
        editor.putInt("iLanguage", iLanguage);

        editor.commit();
}
```

2.23 restorePreferences()

Método de la clase *messageSpeak* utilizado para recuperar las preferencias del sistema. Dentro de las preferencias que son recuperadas se encuentran: el idioma, la velocidad de la voz y la intensidad del volumen multimedia.

```
private void restorePreferences() {
        int mode = Activity.MODE_PRIVATE;
        pref = this.getSharedPreferences("myPreferences", mode);

        if(pref == null)  //No existen preferencias guardadas
                return;

        iVolumen = pref.getInt("iVolumen", 2);
        iVelocity = pref.getInt("iVelocity", 1);
        iLanguage = pref.getInt("iLanguage", 0);
}
```