Ingeniería Técnica de Telecomunicación en Telemática

PROYECTO FIN DE CARRERA

Desarrollo de un sistema de diálogo para la elaboración de una lista de compra y la consulta de recetas en dispositivos móviles Android

Autor: Marina Malo Malo
Tutor: Dr. David Griol Barres

Leganés, octubre de 2015

Título: Desarrollo de un sistema de diálogo para la elaboración de una lista de compra y la consulta de recetas en dispositivos móviles Android.
Autor: Marina Malo Malo.
Director: Dr. David Griol Barres

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Agradezco a los que siguen. Hasta el final sólo llegan los mejores.

Resumen

El objetivo principal del presente Proyecto Fin de Carrera es ilustrar mediante el desarrollo de una aplicación móvil el funcionamiento de un sistema de diálogo, poniendo de manifiesto las ventajas que este medio de interacción ofrece a los usuarios.

Los sistemas de diálogo constituyen una interfaz que recibe como entrada frases del lenguaje natural expresadas de forma oral y generan como salida frases del lenguaje natural expresadas, así mismo, de forma oral.

Siguiendo la estructura del sistema de diálogo, se ha desarrollado una aplicación móvil para el sistema operativo Android. La aplicación consta de dos módulos fundamentales:
Un primer módulo en el que los usuarios pueden consultar mediante la voz una gran variedad de alimentos, organizados por categorías, en el que se muestra información relevante sobre cada uno de ellos. El usuario puede almacenar alimentos que sean de su interés en una lista de la compra
Un segundo módulo en el que los usuarios mediante la voz, pueden seleccionar entre una gran variedad de recetas y consultar la información de cada una de ellas mediante la síntesis de voz

Para llevar a cabo esta funcionalidad la aplicación hace uso del servicio de reconocimiento de voz de Google (Google Voice Recognizer), la librería que incorporan los dispositivos móviles Android para la creación de bases de datos (SQLite), el sistema de reproducción de texto en forma sonora nativo en Android (Text to Speech) y el lenguaje de programación establecido en dicho sistema operativo (Java).

En este proyecto se incluye un estudio detallado de los sistemas de dialogo, haciendo hincapié en las ventajas de su uso en aplicaciones, facilitando la interacción natural mediante el habla de una persona y un sistema informático y favoreciendo la integración de personas con discapacidades motoras o visuales

Palabras clave: Sistemas de Diálogo, Android, Interacción multimodal, App, Reconocimiento Automático del Habla, Síntesis de Voz.

Abstract

The main objective of the present thesis is illustrated by developing a mobile application, the operation for a system dialogue, showing the advantages that offers.

The dialogue systems, outperforms an interface that receives natural oral expresions as an input, and generate natural oral expresions as an output.

Following the dialogue systems structure, it's been developed an mobile application for Android Operator System.
The application is composed by two fundamental modules:
In the first one the, users can ask for feed, organized in categories, in which fundamental information is shown for each one of them. The user can save feed from its own interest in a shopping list.
In the second one, users can select by voice between a lot of receipts and check the information of all of them by the voice.

To take into account this functionality, this application use Android Voice Recognizer, the library incorporated in Android Mobiles Systems SQLite, the Text Reproduction System (Text To Speech) and Java.

In this thesis, it is included a detailed study of dialogue systems, showing the advantages in its own use on applications, making easy the natural interaction between a person and a computer, improving the integration by the voice between dicapacitated people and computers.

Keywords: Dialogue systems, Android, Multimodal interaction, App, Automatic Speech Recognition, Speech Synthesis.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En la primera parte de este capítulo se realiza un análisis de la sociedad actual y su grado de familiarización con los dispositivos y aplicaciones móviles, junto con un pequeño resumen de los sistemas de diálogo y las aplicaciones desarrolladas en la actualidad.

A continuación, se detallan los motivos que promovieron el desarrollo de una aplicación basada en un sistema de diálogo para el presente Proyecto Fin de Carrera.

Para concluir el capítulo se describen los objetivos que se pretenden conseguir con el desarrollo de la aplicación junto con una descripción de las fases del desarrollo y los medios empleados.

Durante los últimos años los teléfonos móviles han experimentado una gran evolución, desde los primeros terminales pesados y grandes, pensados solamente para hablar por teléfono, hasta los actuales dispositivos inteligentes o *smartphones*.

1.1 Introducción

Ya no solamente usamos el móvil para llamadas o envío de mensajes, sino que ahora, se utilizan servicios de datos, correo electrónico, Internet, redes sociales, comercio electrónico, entre otros.

Así lo recoge la segunda parte del estudio "*El futuro de las comunicaciones móviles"* realizado por Oracle [1].De dicho estudio, en el que se entrevistaron a 3.000 consumidores de dispositivos móviles, se extrajeron las siguientes conclusiones:

El 67% de los usuarios de teléfonos móviles utilizan un smartphone y un 47% ha aumentado su uso de datos en los últimos años. El 43% de los encuestados ya ha reemplazado su cámara, el 34% su reproductor MP3 y el 24% su dispositivo GPS por su teléfono móvil. En cuanto a la demanda de aplicaciones, el 55% de los encuestados ha descargado una aplicación gratuita y el 25% ha pagado por el uso de una aplicación en su dispositivo móvil.

Este auge de la tecnología ha generado en el mundo moderno la necesidad cada vez mayor de acceder a los datos desde cualquier sitio y a la máxima velocidad posible.

Desde el inicio de los tiempos el modo de comunicación más eficiente, rápido y natural que posee el ser humano es el habla. Equipar los dispositivos móviles con la capacidad de comprensión y reproducción de mensajes del habla cotidiana, es uno de los objetivos fundamentales para lograr su plena integración en la sociedad actual.

Una de las principales opciones que se están desarrollando para dotar a los dispositivos electrónicos de la capacidad de comunicación son las aplicaciones basadas en sistemas de diálogo.

Los sistemas de diálogo o sistemas conversacionales (SLS, Spoke Language Systems) son una tecnología concebida para facilitar la interacción natural mediante el habla, entre una persona y un ordenador.

Constituyen interfaces o intermediarios entre los sistemas informáticos y los usuarios, donde el medio de comunicación es la voz y el objetivo que se persigue es que el sistema cumpla una determinada tarea.

Para cumplir con los objetivos de los usuarios estas aplicaciones requieren varias interacciones entre la persona y la máquina (varios turnos de diálogo), de manera que gradualmente se vaya definiendo la acción a realizar.

Así mismo, el sistema debe poseer capacidad para:

- Identificar el estado en el que se encuentra el diálogo.
- Tomar decisiones en base a dicho estado.
- Solicitar información adicional en caso de que fuese necesario.
- Solicitar repeticiones de los turnos de diálogo en caso de duda.
- Sintetizar sentencias del lenguaje natural.

Diseñar una aplicación que pueda mantener un diálogo con el usuario de manera natural, es un reto difícil de conseguir a día de hoy, debido a que los sistemas de diálogo que podemos usar están sujetos a las limitaciones de los reconocedores del habla actuales.

Dichos reconocedores son capaces de comprender enunciados emitidos por los locutores en dominios específicos de información, dónde la cantidad de enunciados posibles a reconocer está restringida.

A pesar de las dificultades técnicas que el reconocimiento de voz presenta, actualmente son muchos los que señalan el gran potencial de esta nueva interfaz de acceso a dispositivos electrónicos.

Según palabras de Zig Serafín, Vicepresidente Corporativo de Microsoft Corporation "*La voz es el nuevo 'touch'. Es la evolución natural de teclados*

y pantallas táctiles. Hoy en día el habla se está convirtiendo en una parte esperada de nuestra experiencia diaria a través de una variedad de dispositivos" [2].

Una mayor velocidad en el ingreso de datos, una mayor capacidad de interacción con los dispositivos electrónicos para las personas con discapacidades físicas o motoras o una mayor liberación de los usuarios a la hora de interactuar con sus dispositivos son, entre otras, las razones que han llevado a Microsoft a venir desarrollando desde hace diez años la tecnología de voz a la par que desarrollan su tecnología táctil.

La tecnología desarrollada por Microsoft para el reconocimiento de voz ya ha sido integrada dentro de Bing para móviles, en el sistema operativo Windows Mobile 6.5 y en Windows 7.

Vlad Sejnoha, director de la empresa Nuance Communications señala en una entrevista concedida para el periódico "El Diario" que: "*Estamos en un punto de transición dónde la voz y la comprensión del lenguaje natural han cobrado de pronto mucha importancia*". Además añade "*El habla es ideal para la informática móvil en parte porque los usuarios tienen un solo comando hablado, que puede realizar tareas que normalmente requieren una gran cantidad de gestos con los dedos*" [3].

La empresa Nuance ha integrado con destreza el reconocimiento del habla dentro de los mercados emergentes de las interfaces de voz con su software Dragon.

Dicho software permite interactuar con el PC, a través de la voz, ejecutando tareas como: "*Abrir Microsoft Word*", o "*Minimizar todas las ventanas*". Incluso a través de instrucciones de voz, puede editar y dar formato a un texto, buscar en internet o enviar correos electrónicos.En 2012 presentaron Dragon Naturally Speaking 2, su nuevo software de reconocimiento de voz compatible con PC, teléfonos móviles e incluso coches.

El sistema de entretenimiento Sync, presente en los automóviles de la marca Ford, hace uso de la tecnología de Nuance, permitiendo a los conductores solicitar canciones, información meteorológica y direcciones de destino.

Por su parte Google, empresa multinacional estadounidense especializada en productos y servicios relacionados con Internet, software, dispositivos electrónicos y otras tecnologías, tampoco se ha quedado atrás en lo referente al desarrollo de tecnologías de reconocimiento de voz.

Recientemente la compañía ha sacado la nueva versión de su navegador, Google 25, que entre las principales novedades que incorpora destaca el reconocimiento de voz, permitiendo a los usuarios interactuar con él y con las aplicaciones a través de la voz.

En lo que a dispositivos móviles se refiere, el sistema operativo de Google, Android, ya cuenta con varias aplicaciones que permiten a los usuarios realizar diversas acciones mediante la voz:

- **VITA**
  Es uno de los principales asistentes de voz con los que cuenta Android, aunque actualmente se encuentra en fase beta de desarrollo. Con este asistente podremos activar gran variedad de funciones en el teléfono, mandar mensajes de texto e emails, buscar números de teléfonos, información meteorológica o búsquedas en la web. La aplicación es gratuita y requiere de conexión de datos para su funcionamiento [4].
- **SHERPA**
  Es otro de los asistentes de voz disponibles en Android, creado por el vasco Xabier Uribe-Etxebarría. Permite realizar tareas cómo actualizar, llamar a contactos o consultar información en la web. Actualmente se encuentra en la versión beta que puede adquirirse de forma gratuita y necesita conexión de datos para su funcionamiento.
- **SIRI**
  Sin duda, la empresa pionera en el campo de los asistentes de voz fue Apple con la presentación de su asistente de voz Siri. Actualmente este asistente ha salido de la fase beta de su desarrollo para iOS9 y iphone4 y necesita de conexión Wi-Fi o 3G para su funcionamiento [5].
  Este asistente ofrece interacción conversacional con otras aplicaciones como recordatorios, servicio de mensajería, la bolsa, consulta del estado del tiempo, contactos, navegador web, etc.
  Soporta varios idiomas como inglés, francés, alemán, italiano, chino, etc. Entre sus muchas ventajas y características cabe destacar que ofrece compatibilidad con el iPad y con algunos modelos de coches mediante Bluetooth.

1.2 Motivación del Proyecto Fin de Carrera

La posibilidad de que el usuario puede elegir entre más de una forma de acceso a la aplicación (voz, teclado, etc) se denomina multimodalidad.

Durante los últimos años las aplicaciones basadas en sistemas de diálogo con interfaces multimodales han sido implementadas mediante una combinación de lenguaje XHTML [6], para el diseño de la parte gráfica de la aplicación y lenguaje Voice XML [7], propuesto por el W3C, para la parte de voz.

El lenguaje VoiceXML contiene etiquetas que proveen al navegador web las funciones de síntesis de voz, reconocimiento automático del habla, gestión del diálogo y reproducción de audio.

Siguiendo con esta línea en el presente proyecto se realiza un estudio detallado de los sistemas de diálogo mostrando su funcionamiento, su arquitectura y los diferentes tipos de sistemas de diálogo.

Como propuesta de aplicación que ejemplifique el modo de implementar un sistema de estas características sobre un dispositivo móvil, se decidió crear

una aplicación en el sistema operativo Android con dos módulos de funcionamiento:

- **MÓDULO 1: LISTA DE LA COMPRA**

  Emula el funcionamiento de una lista de la compra. Los usuarios a través de varios turnos de diálogo pueden ir seleccionando, de entre una amplia base de datos, los alimentos que desean. Dichos alimentos se encuentran organizados en categorías y proporcionan la siguiente información:

  → Nombre del alimento.
  → Marca del alimento.
  → Breve descripción del alimento.
  → Precio del alimento.
  → Cantidad de unidades que el usuario ha añadido.
  → Imagen del alimento.

  Las acciones que el usuario puede realizar dentro de este módulo son:

  → Añadir nuevos alimentos a la lista de la compra.
  → Eliminar alimentos almacenados con anterioridad en la lista de la compra.

  Con cada nueva modificación, la aplicación calcula automáticamente, en función de la cantidad de alimento especificada por el usuario para cada alimento, el precio total de todos los productos almacenados en un determinado momento en la aplicación.

- **MÓDULO 2: RECETARIO**

  El segundo módulo, proporciona a los usuarios información sobre distintos tipos de recetas almacenadas en diversas categorías, que los usuarios pueden ir consultando en varios turnos de diálogo. Cada receta proporciona la siguiente información:

  → Nombre de la receta.
  → Imagen de la receta.
  → Nombre y cantidad de los alimentos a utilizar.
  → Instrucciones para la realización de la receta.

  En este módulo ofrece a los usuarios la posibilidad de reproducir una a una, repitiendo en caso de que fuese necesario, las instrucciones a seguir para la elaboración de la receta.

Mediante la voz, proporcionamos al usuario una manera más fácil de acceso a la información y una mayor accesibilidad a la misma, sobre todo en casos en los que se pueda ver disminuida por factores de discapacidad o falta de visión, o por situaciones en las que debido a la acción que se está desempeñando (por ejemplo, cuando estamos cocinando y tenemos las manos la mayor parte del tiempo ocupadas) resulta complicado consultar información en nuestro dispositivo móvil.

Para el estudio, gestión y posterior implementación de ambos módulos, la aplicación define un algoritmo y un modelo de sistema de diálogo, que

pasaremos a describir y analizar en detalle durante los siguientes capítulos del presente documento.

1.3 Objetivos

El objetivo fundamental de este Proyecto Fin de Carrera es el estudio de los Sistemas de Diálogo, ejemplificando su arquitectura, funcionamiento e implementación en dispositivos móviles mediante la creación de una aplicación en el sistema operativo Android, que ponga de manifiesto la ventaja que puede suponer para los usuarios el reconocimiento y la síntesis de voz.

Utilizando los API's para el reconocimiento y la síntesis de voz proporcionados por Android, hemos desarrollado una aplicación con la que los usuarios van interactuando a través de las interfaces táctiles y de voz, hasta obtener la información deseada. Esta información, a su vez, puede consultarse de manera visual o a través de la síntesis de voz.

De este objetivo principal se derivan los siguientes objetivos parciales:

- Conseguir una comunicación con la aplicación que sea lo más natural posible, aportando una mayor comodidad al usuario y reduciendo el tiempo y las acciones requeridas para obtener el acceso a la información.
- Disminuir las barreras arquitectónicas entre las aplicaciones móviles y las personas con algún tipo de discapacidad visual o motora, facilitando su acceso y su integración.
- Ampliar los ámbitos de uso de la aplicación a aquellos entornos en los que resulte complicado su utilización debido al uso exclusivo de las interfaces táctiles o manuales.

1.4 Medios Empleados

Las herramientas empleadas en el desarrollo de la aplicación han sido:

- **HARDWARE**
  - → Ordenador portátil Asus A53S.
  - → Dispositivo móvil Samsung Galaxy Ace.
- **SOFTWARE**
  - → Plataforma de desarrollo Elipse Índigo.
  - → Sistema Operativo Android.

1.5 Estructura de la Memoria

Para facilitar la lectura de la memoria, se incluye a continuación un breve resumen de cada capítulo:

- **Capítulo 1: Introducción**

  En la primera parte de este capítulo se realiza un breve análisis de los sistemas de diálogo. A continuación, se detallan los motivos que promovieron el desarrollo de una aplicación móvil, basada en un sistema de diálogo para el presente Proyecto Fin de Carrera.

  Seguidamente se describen los objetivos que se pretenden conseguir con el desarrollo de la aplicación y finalmente se describen las fases de desarrollo y los medios empleados.

- **Capítulo 2: Estado del Arte**

  Este capítulo comienza con un estudio detallado de los sistemas de diálogo en el que se incluye su evolución histórica, los criterios de clasificación de los mismos y las principales aplicaciones móviles que podemos encontrar en la actualidad, que basan su funcionamiento en estrategias de diálogo.

  El capítulo termina con un estudio de los actuales dispositivos móviles y sus sistemas operativos.

- **Capítulo 3: El Sistema Operativo Android**

  En este capítulo se realiza un estudio del Sistema Operativo Android, incluyendo su arquitectura, los principales componentes que lo forman, los medios para crear bases de datos y una pequeña guía de instalación para su uso.

- **Capítulo 4: Descripción General del Trabajo Realizado.**

  Este capítulo comienza describiendo en líneas generales el trabajo realizado para el desarrollo de la aplicación.

  A continuación, detalla las principales clases desarrolladas para su diseño, las bases de datos definidas y los *frames* generados para la implementación de la estrategia de diálogo.

- **Capítulo 5: Módulo Lista de la Compra**

  A lo largo de este capítulo se describen los diferentes submódulos desarrollados para cumplir con la funcionalidad que se pretende que ofrezca el Módulo Lista de la Compra.

  Para cada submódulo, se detalla su funcionalidad, su diagrama de flujo y su arquitectura.

- **Capítulo 6: Módulo Recetario.**

  A lo largo de este capítulo se describen los diferentes submódulos desarrollados para cumplir con la funcionalidad que se pretende que ofrezca el Módulo Recetario.

  Para cada submódulo se detalla su funcionalidad, su diagrama de flujo y su arquitectura.

- **Capítulo 7: Evaluación, Planificación y Presupuesto**

  En este capítulo, primeramente se recogen los resultados obtenidos tras realizar una encuesta de evaluación a diez usuarios que probaron la aplicación en sus dispositivos móviles.

A continuación se detalla la planificación temporal de las distintas fases del proyecto y finalmente se recoge su presupuesto total en euros.

- **Capítulo 8: Conclusiones y Trabajo Futuro.**

  En este capítulo se detallan las conclusiones obtenidas con el desarrollo de la aplicación en base a los objetivos marcados en el Capítulo 1 y se estudian las diferentes líneas de trabajo futuro para mejorar el funcionamiento y diseño de la misma.

CAPÍTULO 2

Estado Del Arte

En el presente capitulo se realiza en primer lugar un estudio detallado de los sistemas de diálogo analizando los distintos módulos que componen su arquitectura, las distintas clasificaciones de los mismos, su evolución histórica y las aplicaciones disponibles actualmente en el mercado.

A continuación, se realiza un estudio sobre el panorama actual de los dispositivos móviles y los diferentes tipos de sistemas operativos disponibles, realizando un análisis más detallado del sistema operativo Android.

Finalmente, el capítulo concluye presentando de forma detallada la aplicación desarrollada para el presente Proyecto Fin de Carrera.

2.1 Los Sistemas de Diálogo

"Un diálogo es una conversación entre dos o más personas llamadas interlocutores que alternativamente (haciendo uso de varios turnos) manifiestan sus deseos, intenciones y creencias, mientras hacen parte de un proceso de negociación" [8].

Un sistema de diálogo puede entenderse como un sistema automático capaz de emular a un ser humano en un diálogo con otra persona, con el objetivo de que el sistema cumpla con una cierta tarea (normalmente suministrar una cierta información o llevar a cabo una determinada acción).

Se constituyen como una interfaz entre un sistema informático y un usuario en el que el modo de interacción es la voz, permitiendo realizar tareas de forma más eficiente.

Un sistema de diálogo ideal reconocería el habla espontánea, comprendiendo enunciados sin restricciones de contenido. Proporcionaría

respuestas con sentido, gramaticalmente bien formadas, y pragmáticamente adecuadas, respondería con voz completamente natural y sería multimodal.

Como hemos comentado anteriormente, desarrollar un sistema que cumpla con estos requisitos a día de hoy sigue siendo todo un reto debido a que la capacidad de comprensión de los actuales reconocedores del habla está limitada a lo que se denominan dominios restringidos, es decir, ámbitos conversacionales muy concretos en los que se persiguen la realización de tareas muy específicas.

2.2 Funcionamiento y Arquitectura

Para que un sistema de diálogo pueda determinar la tarea final a realizar, son necesarios varios turnos de diálogo en los que, tras cada interacción del usuario, el sistema lleva a cabo las siguientes tareas:

- Reconocer la secuencia de palabras emitida por el usuario.
- Análisis lingüístico (morfológico, sintáctico, semántico, pragmático) de la secuencia de palabras reconocida.
- Creación de una representación interna.
- Tratamiento de dicha representación interna y decisión de la tarea o tareas a realizar en función de ella (acceso a bases de datos, solicitud de repetición, etc.).
- Construcción de una secuencia de respuesta informando al usuario de la tarea realizada.
- Síntesis de texto a voz de la secuencia de respuesta.

Dada toda esta multitud de operaciones a realizar, los sistemas de dialogo se diseñan cómo una estructura modular, de forma que cada módulo cumple con una de las funciones, trabajando de manera coordinada con el resto de módulos.

La Figura 1 muestra los diferentes módulos que componen la arquitectura de un sistema de diálogo.

Figura 1: Arquitectura Modular de un Sistema de Diálogo Hablado

Cada módulo se encarga de realizar las siguientes funciones dentro de la aplicación:

- **MÓDULO DE RECONOCIMIENTO AUTOMÁTICO DEL HABLA :**
  Procesa la secuencia de sonidos emitida por el usuario y proporciona la secuencia o secuencias de palabras reconocidas más probable en formato texto.
- **MÓDULO DE COMPRENSIÓN DEL HABLA:**
  Recibe como entrada la secuencia o secuencias de palabras reconocidas y genera como salida una representación semántica de su significado, basándose en un modelo semántico establecido.
- **GESTOR DE DIÁLOGO:**
  Es el módulo fundamental del sistema ya que su objetivo es determinar que acción debe realizarse en cada momento, de manera que la interacción con el usuario sea lo más cómoda e "inteligente" posible. Recibe como entrada la representación semántica obtenida por el módulo de comprensión del habla y en base a ello, siguiendo la estrategia del diálogo, determina para cada estado en el que se encuentra el diálogo, la siguiente acción a realizar.
- **MÓDULO REGISTRO DE DIÁLOGO:**
  Almacena las estructuras semánticas o "frames" obtenidas por el módulo de comprensión del habla a lo largo de las interacciones del usuario. En base a esta información el gestor de diálogo puede determinar en qué punto de la conversación se encuentra la aplicación y actuar en consecuencia, así como referenciar cierta información relevante expresada por el usuario en anteriores turnos de diálogo.
- **MÓDULO DE ACCESO A LA BASE DE DATOS:**
  Proporciona los mecanismos necesarios para el acceso a la información almacenada a la base de datos.
- **MÓDULO DE GENERACIÓN DE RESPUESTAS:**
  Construye enunciados gramaticalmente correctos y en un lenguaje lo más cercano posible al lenguaje natural, transmitiendo el mensaje generado por el gestor de diálogo.

- **MÓDULO SINTETIZADOR DE VOZ:**
  Recibe la sentencia construida por el módulo generador de respuestas y genera la correspondiente señal de audio que será la respuesta que llegará al usuario

2.3 Tipos de Sistemas de Diálogo

Dependiendo de cómo se construya el sistema de diálogo y la forma en la que se administre, los sistemas de diálogo se clasifican en:

- **SISTEMAS DE DIÁLOGO GUIADOS:**
  Se caracterizan porque sólo el sistema tiene la capacidad para dirigir el diálogo mediante interacciones cerradas basadas en un algoritmo de pregunta-respuesta. La iniciativa del usuario queda por tanto bastante restringida.
- **SISTEMAS DE DIÁLOGO COOPERATIVOS:**
  Ambos interlocutores (usuario y sistema) tienen la capacidad de guiar el diálogo por lo que hay un reparto equilibrado del turno de palabra. Los usuarios pueden interrumpir el flujo del diálogo y el sistema incorpora mecanismos de detección de incoherencias gramaticales.
- **SISTEMAS DE DIÁLOGO ADAPTATIVOS:**
  El sistema es capaz de aprender nuevas estrategias comunicativas en función del comportamiento del usuario.
- **SISTEMAS DE DIÁLOGO GUIADOS POR EL USUARIO:**
  A lo largo de la interacción es el usuario el que lleva la iniciativa del sistema pudiendo utilizar un lenguaje natural que no esté limitado a un dominio específico por el sistema.

A parte de esta clasificación general, atendiendo a otros aspectos no menos relevantes como el tipo de discurso o el tipo de comunicación, encontramos otras clasificaciones para los sistemas de diálogo, entre ellas:

- **ATENDIENDO AL TIPO DE COMUNICACIÓN:**
  - → **Unimodal**: el único medio de comunicación es el habla.
  - → **Multimodal**: además de la voz, el sistema cuenta con varios canales de comunicación como el teclado, la pantalla táctil, el ratón, etc.
- **ATENDIENDO A LA DEPENDENCIA DEL HABLANTE:**
  - → **Dependientes del hablante:** son aquellos sistemas que están diseñados para funcionar con un único hablante. Estos sistemas son más baratos y sencillos en su implementación pero como contraprestación son poco flexibles.
  - → **Independientes del hablante:** A diferencia de los anteriores, estos sistemas funcionan para cualquier hablante por lo que son más flexibles, pero también son más caros y complicados de desarrollar.
- **ATENDIENDO AL TIPO DE DISCURSO:**

- → **Sistemas de reconocimiento de palabras aisladas:** en este tipo de sistemas sólo es posible reconocer un determinado número de palabras, almacenadas como una secuencia acústica. Los modelos acústicos de las palabras a reconocer se comparan con las secuencias acústicas almacenadas seleccionando aquel que más similitudes presenta.
- → **Sistemas de reconocimiento de palabras conectadas:** el sistema recibe como entrada secuencias conectadas pertenecientes a un mismo dominio lingüístico y el reconocimiento se lleva a cabo analizando de manera individual la coincidencia entre cada una de las palabras de la secuencia de entrada y las palabras clave almacenadas en la aplicación.
- → **Sistemas de reconocimiento continuo:** este tipo de sistemas es capaz de reconocer habla continua sin necesidad de realizar pausas entre las palabras. Desarrollar un sistema de estas características, es una tarea laboriosa y complicada debido a factores como: las múltiples entonaciones de las palabras, la coarticulación, la segmentación de unidades, etc.

2.4 Historia de los Sistemas de Diálogo

Como señala Zoraida Callejas Carrión en su tesis doctoral sobre los sistemas de diálogo oral, adaptativos y portables [9], las primeras investigaciones sobre los sistemas de diálogo surgen a principio de los años ochenta, como resultado de las mejoras en los campos de reconocimiento del habla, procesamiento del lenguaje natural y síntesis del habla.

Surgen así los dos primeros grandes proyectos: el programa de sistemas de lenguaje hablado DARPA (Defense Advanced Research Projects Agency) [10], en Estados Unidos y el programa Espirit SUNDIAL (Speech Understanding and Dialogue) en Europa.

El objetivo del programa DARPA fue el estudio del desarrollo de tecnologías de reconocimiento del habla y comprensión del lenguaje, aplicadas dentro de un dominio de consulta sobre información de vuelos (Air Travel Information System, ATIS). El proyecto ATIS sirvió como referencia en múltiples investigaciones desarrolladas posteriormente como es el caso de los proyectos desarrollados en AT&T, concretamente el proyecto AMICA que aplicaba diferentes métodos estocásticos dando lugar a un sistema de diálogo de iniciativa mixta.

Por su parte el proyecto SUNDIAL, financiado por la Comunidad Europea, tuvo como objetivo construir sistemas de diálogo en tiempo real, capaces de mantener una conversación con el usuario siguiendo una estrategia cooperativa, dentro de un dominio de consulta de horarios de trenes y aviones.

De los estudios llevados a cabo en SUNDIAL, surgen posteriormente un gran número de proyectos con financiación europea centrados en el modelado del diálogo, como ARISE (Automatic Railway Information System for Europe) sistema que proporciona información acerca del servicio de trenes en Europa.

A finales de esta época los sistemas tuvieron que modificarse para adaptarse a los nuevos dispositivos móviles, cada vez más populares, y que requerían el poder trabajar con bajas relaciones señal a ruido y menores anchos de banda. Se trabajó también en la definición de lenguajes estándar para el desarrollo de sistemas de diálogo. A finales de 1993 el W3C Voice Browser Working Group, presentó los primeros estudios sobre los requisitos para navegadores web que supusieron la base de los lenguajes de etiquetas como Voice XML.

A partir de 2003, las investigaciones comenzaron a perseguir líneas cada vez más complejas, orientadas a la realización de sistemas de diálogo con un razonamiento más avanzado, capacidad de resolución de problemas, multimodalidad y multilingüismo.

Los avances en la investigación permiten desarrollar interfaces multimodales, dejando a un lado las interfaces tradicionales de entrada como el ratón y el teclado. Destaca entre otros el proyecto SMARTKOM (Sistema de diálogo multimodal alemán con iniciativa mixta para diversos dominios de aplicación, entre ellos la reserva de entradas de cine) [11].

Otro de los objetivos fue la adaptabilidad, es decir, dotar al sistema de la capacidad para responder a diferentes necesidades ante diferentes interacciones del usuario con el sistema. En los sistemas multimodales los usuarios disponen de diversas formas de comunicación lo que hace que el sistema deba adaptarse a las diferentes interfaces de entrada.

2.5 Principales Aplicaciones Disponibles en la Actualidad

Todos estos esfuerzos y avances en la investigación han dado pie a que hoy en día existan multitud de aplicaciones de Sistemas de diálogo con diferentes niveles de complejidad, dentro de una gran variedad de ámbitos.

Uno de los ámbitos más extendido son las aplicaciones de consulta de información. Algunos ejemplos de estas aplicaciones son:

- **Voyager**: sistema de información turística del área de Boston [13].
- **Júpiter**: sistema de predicción meteorológica de EE.UU [14].
- **Dihana**: sistema de información de viajes de tren en España [15].

Existen sistemas que realizan operaciones más complejas cómo pueden ser reservas de billetes o interacción con servicios del automóvil:

- **Mercury:** Interfaz conversacional que proporciona información sobre vuelos y permite hacer reservas [16].

Dentro del ámbito del comercio electrónico, existen sistemas que pueden llevar a cabo transacciones, como ventas de entradas o servicios relacionados con la banca electrónica:

- **Cinentradas:** Sistema de diálogo para la venta telefónica de entradas de cine [17].
- **Línea BBVA:** servicio ofrecido por el Banco Bilbao Vizcaya Argentaria, que permite a los usuarios realizar operaciones bancarias cómo transferencias, domiciliaciones o compra de seguros [18].

Los Sistemas de diálogo también se emplean en ámbitos educativos:

- **ItSpoke:** Sistema de diálogo oral de tutorización que conversa con estudiantes corrigiendo errores lingüísticos [19].
- **Vocaliza:** Sistema de Diálogo para terapias de voz en español que ayuda a los terapeutas en el entrenamiento con usuarios que presentan discapacidades lingüísticas [20].

La interacción oral también está teniendo presencia en ámbitos como la robótica o los entornos inteligentes:

- **Odisea:** Sistema de Diálogo oral que permite la interacción entre usuarios y entornos inteligentes [21].

2.6 Dispositivos Móviles y Principales Sistemas Operativos

2.6.1 Dispositivos Móviles

Los dispositivos móviles son aparatos de pequeño tamaño con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función pero que pueden llevar a cabo otras funciones más generales.

La principal característica de estos dispositivos es la movilidad, es decir, la capacidad para poder portarse y ser usados durante su transporte.

Cada dispositivo móvil posee un número de funcionalidades determinado y por esta razón T38 y Du Pont Global Mobility Inovation Team propusieron en el año 2005 la siguiente clasificación de dispositivos móviles:

- **Dispositivo móvil de datos limitados (Limited Data Mobile Device):** Dispositivos móviles de pantalla pequeña con servicios de datos limitados al envío y recepción de mensajes de texto y conexiones de tipo WAP.
- **Dispositivos móviles de datos básicos (Basic Data Mobile Device):** Dispositivos móviles con una pantalla de tamaño medio y una navegación basada en iconos. Ofrecen servicios como el acceso a emails, SMS, navegación web, entre otros, Dentro de esta categoría están los llamados teléfonos inteligentes o smartphones.

- **Dispositivos móviles de datos mejorados (Enhanced Data Mobile Device):** dispositivos móviles con un tamaño de pantalla medio-grande que ofrecen la misma funcionalidad que los dispositivos móviles de datos básicos, añadiendo mas funcionalidades como aplicaciones de Microsoft Office Mobile (Word, Excel, Power Point) o aplicaciones web de uso cotidiano desde ordenadores, adaptadas en versiones móviles.

2.6.2 Teléfonos Inteligentes o Smartphones

Un teléfono inteligente o "Smartphone" es un teléfono móvil construido sobre una plataforma informática móvil, con una mayor capacidad de almacenar datos y realizar actividades semejantes a una minicomputadora y conectividad, que un teléfono móvil convencional [12].

Estos teléfonos son la evolución de los teléfonos móviles tradicionales que cuentan con nuevas características y prestaciones, como la mejora del almacenamiento de datos, conexiones a Internet mediante una tarifa de datos contratada con un operador o mediante el uso de redes Wi-Fi, pantalla táctil, descarga de nuevas aplicaciones, etc.

De entre todos los tipos de dispositivos móviles, los teléfonos inteligentes son los más usados en la actualidad y los que más posibilidades de evolución presentan.

Según el último informe presentado por la ITU (Unión Internacional de las Telecomunicaciones) presentado el 7 de Octubre de 2013 en Ginebra (Suiza) en la Tierra hay casi tantos móviles como habitantes [23].

Según revela dicho informe, durante el último año la banda ancha en smartphones y tabletas se ha convertido en el segmento de mayor crecimiento dentro de las telecomunicaciones.

A final de año habrá 6.800 millones de usuarios móviles de los cuáles 2.100 millones son suscripciones a banda ancha con un crecimiento del 46%.

Durante el pasado año, 250 millones de personas contrataron servicios de datos en su smartphones, lo que hace que el 40% de la población mundial acceda a la red.

2.6.3 Principales Sistemas Operativos Móviles

Un sistema operativo móvil o SO móvil, es un sistema operativo que controla un dispositivo móvil del mismo modo que controlan los PCs sistemas operativos como Windows o Linux, pero con una mayor simplicidad y orientados a la conectividad inalámbrica, a los formatos multimedia para móviles y a las diferentes formas de introducir información en ellos [24].

A medida que el uso de teléfonos móviles se vuelve cada día más importante entre los usuarios, las compañías de teléfonos móviles han ido desarrollando una competencia bastante reñida en lo que al desarrollo de sistemas operativos se refiere.

La cuota de mercado de sistemas operativos móviles durante el primer trimestre de 2015 se muestra en la Figura 2:

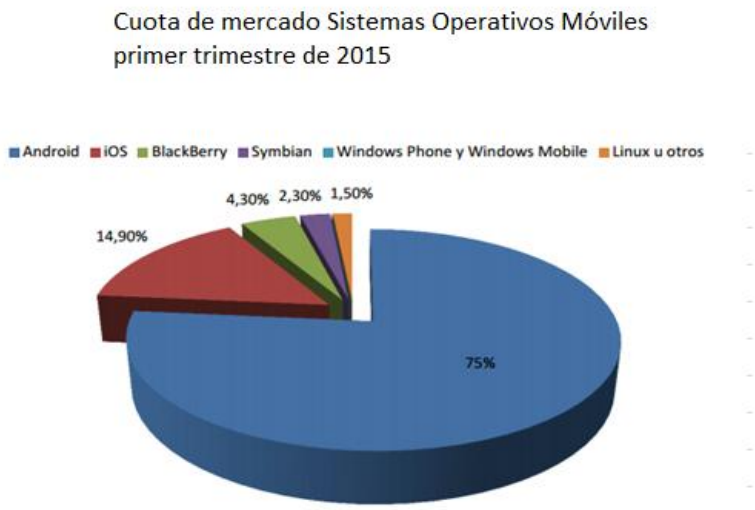

Figura 2: Cuota de Mercado de Sistemas Operativos Móviles durante el tercer trimestre de 2012 [22].

Android es el que presenta la mayor cuota de mercado desde enero de 2011 experimentando un gran crecimiento, que en solo dos años ha pasado a ser el SO móvil más usado por más de la mitad del mercado.

A continuación, se incluye un estudio más detallado de las principales características de los sistemas operativos móviles más importantes en la actualidad.

2.6.3.1 Android

Es un sistema operativo basado en Linux desarrollado por Android Inc. Firmac, comprada posteriormente por Google en 2005, diseñado principalmente para su uso en dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tabletas.

El anuncio del sistema operativo Android se realiza el 5 de noviembre de 2007, junto con la creación de la Open Handset Alliance; un consorcio de compañías de hardware, software y telecomunicaciones. El primer terminal que incluía este sistema operativo fue el HTC Dream vendido en Octubre de 2008.

Actualmente cuenta con una gran comunidad de desarrolladores de aplicaciones, ya que, al contrario que otros sistemas operativos como iOS o Windows Phone, es un sistema operativo de código abierto que permite acceder tanto al código fuente como a la lista de incidencias, lo que permite detectar problemas aún no resueltos.

De entre las principales características de este sistema operativo destacan:

- Almacenamiento: incorpora SQLite, una base de datos liviana que permite almacenamiento de datos estructurados.

- Conectividad: soporta las siguientes tecnologías de conectividad: GSM/EDGE, IDEN, CDMA, EV-DO, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC y WiMAX.
- Su máquina virtual Dalvik está optimizada para funcionar en múltiples plataformas.
- Gran cantidad de servicios disponibles: como GPS, mapas, lectores de códigos de barras, etc.
- Multimedia: mejora de la capacidad visual a lo largo de las sucesivas versiones añadiendo mejor calidad en gráficos y sonido y mayor soporte a formatos de vídeo y audio.
- Entorno de desarrollo completo que incluye: emulador de dispositivos, herramienta para la depuración y plugin para el IDE de Eclipse.
- Cuenta con la tienda de aplicaciones Android Market.

2.6.3.2 iOS

Sistema operativo móvil propiedad de la empresa Apple Inc, desarrollado inicialmente para su teléfono inteligente iPhone y trasladado posteriormente a dispositivos como el iPad o el iPod Touch.

iOS es una versión reducida y optimizada del sistema operativo Mac OS X. En su implementación, iOS cuenta con cuatro capas de niveles de abstracción: la capa del núcleo del sistema operativo, la capa de "Servicios Principales", la capa de "Medios", y la capa "Cocoa Touch".

Cabe destacar las múltiples restricciones a las que está sometido este sistema operativo por parte de Apple Inc: restricciones sobre el hardware en el que se ejecutará el sistema, restricciones sobre el software que puede ejecutar el sistema o restricciones sobre las aplicaciones que se pueden utilizar en segundo plano.

De entre las principales características de este sistema operativo destacan:

- Interfaz gráfica diseñada para pantallas táctiles con capacidad multitouch.
- Carpetas: a partir de la versión iOS 4 se introdujo un sistema de carpetas permitiendo mover unas aplicaciones por encima de otras.
- Su interfaz está constituida básicamente por sliders, interruptores y botones, proporcionando una respuesta fluida e inmediata.
- Soporte multitarea.
- Incluye múltiples aplicaciones para la gestión de emails, fotos, cámara, mensajes, notas, contactos, etc.
- No proporciona soporte a tecnologías como Adobe Flash o Java, en cambio iOS usa HTML5 como alternativa a Flash.
- Los lenguajes de programación soportados son C, C++ y Objective-C.
- Cuenta con la tienda de aplicaciones App Store.

2.6.3.3 Windows Phone

Windows Phone es el sistema operativo móvil desarrollado por Microsoft, enfocado principalmente al mercado de consumo. Cuenta con una interfaz de usuario que tiene como punto fuerte la simplicidad, con un aspecto cuidado, que integra varios servicios propios como SkyDrive, Skypo, Xbox Live.

De entre las principales características de este sistema operativo destacan:

- Interfaz de usuario dividida en mosaicos dinámicos que enlazan con aplicaciones, contactos, páginas web, archivo multimedia y demás funcionalidades.
- Tecnología multitouch.
- Arquitectura ARM.
- Sincronización con la nube mediante SkyDrive.
- Servicio de mensajería mediante Skype.
- Servicio remoto de localización GPS.
- Servicio remoto de bloqueo y eliminación de datos.
- Navegador web Internet Explorer Mobile 9.
- Organización en hub, que permite clasificar acciones y aplicaciones correspondientes a una actividad determinada.
- Tienda de aplicaciones Market place.

2.6.3.4 Blackberry OS

Blackberry OS es un sistema operativo móvil desarrollado por Research In Motion (RIM) para los dispositivos Blackberry.

Su desarrollo se remonta a 1999 con la aparición de las primeras PDA (Asistentes Personales Digitales), que ofrecían a los usuarios nuevas funciones como el acceso al correo electrónico, la navegación web y la sincronización con programas como Microsoft Exchange o Lotus Note.

Actualmente este sistema operativo está orientado al mundo profesional como gestor de correo electrónico y agenda, siendo la versión Blackberry 10 la última estable hasta el momento, solamente disponible para dispositivos Blackberry Z10.

De entre las principales características de este sistema operativo destacan:

- Modelo de desarrollo basado en código cerrado.
- Desarrollado en los lenguajes de programación Java y C++.
- Las plataformas soportadas son las líneas de smartphones de Blackberry.
- Soporte para 26 idiomas.
- Soporte multitarea.
- Con la versión 10 conexión a redes 4G LTE y soporte para NFC.

Además, mientras las aplicaciones se ejecutan, varias capas de seguridad garantizan el aislamiento de datos, entre ellas;

- Gestión del ciclo de vida automático: los ciclos de vida de las aplicaciones son controlados por el propio sistema operativo.
- Múltiple hardware: Desde los inicios de su desarrollo, el sistema operativo Android ha sido diseñado para adaptarse a cualquier tipo de teléfono, desde los primeros dispositivos que no contaban con pantalla táctil hasta los actuales smartphones.
- Se programa en Java aunque también puede programarse en C o C++ haciendo uso de una serie de herramientas llamadas NDK (Native Development Kit) que Google pone a disposición de los desarrolladores.

Capítulo 3

El Sistema Operativo Android

A lo largo de este capítulo se realiza un estudio del Sistema Operativo Android, incluyendo su arquitectura, los principales componentes que lo forman, los medios para crear bases de datos y una pequeña guía de instalación para su uso.

3.1 Características de la Plataforma Android

Como ya hemos comentado en el Capítulo 2, Android es el sistema operativo de código abierto creado por Google. Detallamos a continuación las principales características de esta plataforma:

- El núcleo del sistema operativo está basado en Linux 2.6, con ciertas modificaciones, para adaptarlo a los requerimientos de los dispositivos móviles.
- Aplicaciones hechas de componentes: cada aplicación está compuesta por distintas piezas que se integran cumpliendo una determinada funcionalidad, con la ventaja de que las piezas diseñadas para un programa pueden ser reutilizadas para otros o sustituidas por otros.
- Multimedia: a lo largo de las diferentes versiones se ha ido mejorando la calidad de los gráficos y sonidos, dando soporte a un mayor número de formatos de audio y vídeo.
- Seguridad: la plataforma cuenta con diversos mecanismos mediante los cuales, a la hora de desarrollar un programa, se establece a qué servicios

accederá y que elementos del teléfono utilizará. Además, mientras las aplicaciones se ejecutan, varias capas de seguridad garantizan el aislamiento de datos.

- Gestión del ciclo de vida automático: los ciclos de vida de las aplicaciones son controlados por el propio sistema operativo.
- Múltiple hardware: desde los inicios de su desarrollo, el sistema operativo Android, ha sido diseñado para adaptarse a cualquier tipo de teléfono, desde los primeros dispositivos que no contaban con pantalla táctil, hasta los actuales smartphones.
- Se programa en Java, aunque también puede programarse en C o C++, haciendo uso de una serie de herramientas llamadas NDK (Native Development Kit) que Google pone a disposición de los desarrolladores.

3.2 Arquitectura

Como puede verse en la Figura 3, el sistema operativo Android está estructurado en varias capas, de modo que cada una utiliza elementos de las capas inferiores, permitiendo a los desarrolladores acceder a las capas más bajas, sin necesidad de tener que programar funcionalidades de bajo nivel

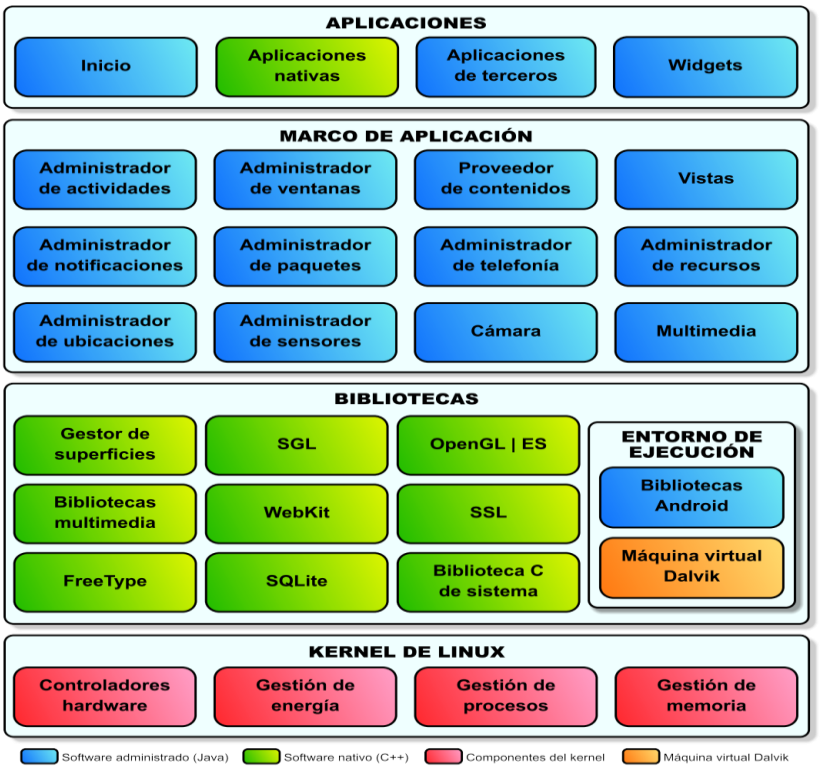

Figura 3: Arquitectura del Sistema Operativo Android

Detallamos a continuación cada una de las capas

- **Kernel de Linux:**
  Como hemos comentado anteriormente, el núcleo del sistema operativo Android está basado en la versión 2.6 del kernel de Linux, adaptado a las características de los dispositivos móviles. El núcleo actúa como una capa de abstracción entre el hardware y el resto de capas, encargándose de gestionar los diferentes recursos del teléfono (energía, memoria, etc.), así como los diferentes servicios básicos, como seguridad, gestión del proceso pila de red y modelo de controladores.

- **Librerías Android o Bibliotecas**
  La capa superior al kernel la componen las bibliotecas nativas de Android o librerías, escritas en C o C++ y compiladas para la arquitectura hardware específica de cada dispositivo móvil. Su función es la de proporcionar a las aplicaciones funcionalidades para realizar tareas de forma eficiente, que se repiten con frecuencia, evitando tener que codificarlas cada vez que se usen. Las diferentes librerías disponibles son:
  - Gestor de superficies: gestión del acceso a la pantalla.
  - Bibliotecas multimedia: reproducción de imágenes, audio y vídeo.
  - SQLite: pequeña base de datos relacional.
  - Open GL/ES: librerías 3D.
  - FreeType: fuentes de texto.
  - Webkit: navegador optimizado.
  - SGL: gráficos 2D.
  - SSL: cifrado de comunicaciones.
  - Biblioteca C de sistema: cabeceras y funciones según el lenguaje C.

- **Entorno de Ejecución**
  No está considerada una capa en sí misma, debido a que está formado por librerías que contienen funcionalidades habituales de Java y funcionalidades específicas de Android. El componente principal del entorno de ejecución de Android es la máquina virtual Dalvik. Las aplicaciones se codifican en Java y se compilan en un formato específico denominado .dex (Dalvik Executables), para que dicha máquina virtual pueda ejecutarlas. Este sistema permite que solamente sea necesario compilar las aplicaciones una vez para poder ser ejecutadas en cualquier tipo de dispositivo Android que disponga de la versión mínima del SO requerida por la aplicación
  - Marco de aplicaciones: esta capa contiene las clases y servicios que utilizan las aplicaciones para poder llevar a cabo las distintas funcionalidades:
  - Administrador de actividades: administra la pila de actividades de la aplicación y el ciclo de vida de la misma.
  - Administrador de ventanas: gestiona las ventanas de las aplicaciones y hace uso de la librería Surface Manager.
  - Administrador de contenidos: permite a las aplicaciones compartir sus datos con otras aplicaciones Android.

- Vistas: proporciona una gran variedad de elementos para poder construir las interfaces de usuario.
- Administrador de paquetes: biblioteca que proporciona información acerca de los paquetes instalados en el SO Android, gestionando también la instalación de nuevos paquetes.
- Administrador de telefonía: librería que incluye todas las API's relacionadas con las funciones propias del teléfono como llamadas, envío de mensajes, etc.
- Administrador de recursos: librería con la que podemos gestionar todos los elementos de la aplicación que no forman parte del código, como layouts, cadenas de texto, imágenes, sonidos, etc.
- Administrador de ubicaciones: permite determinar la localización geográfica del dispositivo a través del GPS o de las redes disponibles y trabajar con mapas.
- Administrador de notificaciones: permite a las aplicaciones comunicar a los usuarios los eventos que ocurren durante su ejecución, como llamadas entrantes, conexiones Wi-Fi disponibles, recepción de mensajes, etc.

- **Aplicaciones**
  Es la última capa de la arquitectura y en ella se incluyen todas las aplicaciones del dispositivo, tanto las incluidas por defecto en Android, como las que el usuario vaya añadiendo posteriormente, ya sean de su propio desarrollo o pertenecientes a otras empresas.

3.3 Conceptos Básicos

Los bloques son los elementos principales que Android pone a disposición de los usuarios para desarrollar aplicaciones. Los cuatro bloques principales son:

- **ACTIVITY**
  Es el bloque más usado y se corresponde con una ventana o un cuadro de diálogo en una aplicación de escritorio. Una Activity es una clase dentro de la cuál mostraremos las Views o vistas, que son los elementos básicos de las interfaces gráficas y actúan como nexo de unión entre la Activity y el usuario, ya que se encargan de recibir los eventos realizados por éste.
- **BROADCAST INTENT RECEIVERS**
  Un Broadcast Intent Receiver o Receptor de Mensajes, es un componente que tiene como función recibir y reaccionar frente a ciertos mensajes emitidos por el sistema.
- **SERVICE**
  Los Services o Servicios, son componentes diseñados para mantenerse en ejecución por sí solos, independientemente de las Activities, como es el caso de un reproductor de música.

3.3.1 Intents

Los Intents, son mensajes asíncronos que sirven para activar las Activitys, los Broadcast Intent Receivers y los Service. Estos mensajes son lanzados continuamente para notificar de diversos eventos, como son avisar que el dispositivo se está quedando sin batería, la inserción de una tarjeta SD, etc. Las aplicaciones, además de responder a los Intents, pueden crear los suyos propios para lanzar actividades o para avisar de que algo ha sucedido.

3.3.2 Interfaz Gráfica

En Android la interfaz de usuario se construye con los elementos View Group y View.

La clase View Group es la base de las subclases denominadas layouts, que ofrecen diferentes tipos de disposiciones gráficas y puede contener otros objetos Views.

Los objetos View son las unidades básicas de la interfaz de usuario y son la base de las subclases llamadas "Widgets", que son objetos totalmente implementados, como botones o campos de texto.

3.3.3 Ciclo de Vida de una Actividad

A lo largo de la ejecución de una aplicación, cada una de las Actividades que la componen tiene un ciclo de vida dependiente del estado en el que se encuentra la aplicación. La Figura 4 muestra los diferentes estados por los que pasa una aplicación a lo largo del ciclo de vida, en función del método que se invoque:

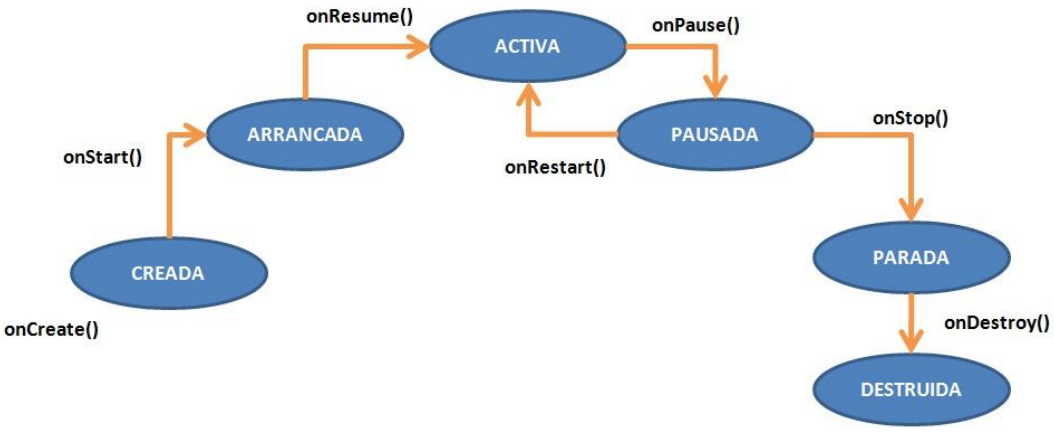

Figura 4: Ciclo de Vida de una Aplicación en Android

- **Creada:**
  La Actividad se crea e inicia el ciclo de vida.
- **Arrancada:**
  La Actividad entra en la cola de actividades que van a pasar a ser mostradas al usuario.
- **Activa:**
  El usuario visualiza la Actividad en la pantalla, pudiendo interactuar con ella. En este estado la Actividad está la primera en la pila de ejecución del sistema.
- **Pausada:**
  La Actividad ha pasado a segundo plano porque otra Actividad se ha colocado por encima de ella, pero sigue siendo visible en la pantalla.
- **Parada:**
  La Actividad ha pasado a segundo plano y está completamente tapada por la nueva Actividad.
- **Destruida:**
  La Actividad ya no está disponible, se han liberado sus recursos y en caso de ser llamada tiene que comenzar un nuevo ciclo de vida.

Del control del ciclo de vida se encarga el sistema operativo. Cada vez que se produce un cambio de estado, se van a producir una serie de eventos que son capturados por los siguientes métodos:

- **onCreate(Bundle):** se llama nada más crear la Actividad y en él se realizan todas las inicializaciones necesarias para poner en marcha la Actividad, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Recibe cómo parámetros un objeto Bundle, que guarda el estado en el que se encontraba anteriormente la Actividad.
- **onRestart():** se llama cuando una Actividad que se ha parado vuelve a estar activa.
- **onStart():** se llama cuando la Actividad va a ser mostrada al usuario.
- **onResume():** Se llama cuando la Actividad va a comenzar a interactuar con el usuario.
- **onPause():** se llama cuando una Actividad va a pasar a segundo plano, normalmente porque otra Actividad va a ser lanzada.
- **onStop():** se llama cuando la Actividad se vuelve invisible para el usuario, bien porque otra Actividad haya pasado a primer plano y la tapa completamente o bien porque haya sido destruida.
- **onDestroy():** Se llama antes de que la Actividad sea totalmente destruida, bien porque el usuario ejecuta sobre ella el método de finalización *finish()* o bien porque el sistema necesita más recursos y elimina la Actividad para conseguirlos.

3.4 Base de Datos SQLite

La plataforma Android pone a disposición de los desarrolladores la herramienta SQLite, un manejador de bases de datos que permite trabajar con

poca memoria y con una velocidad bastante aceptable. Con SQLite podemos almacenar información estructurada en tablas a las cuáles podemos acceder mediante sentencias de tipo SQL.

SQLite utiliza el lenguaje SQL para las consultas de selección (SELECT), la manipulación de datos (INSERT, DELETE, etc) y para la definición de datos (CREATE TABLE, etc.).

Para acciones cómo el uso de claves externas o transacciones anidadas, hay modificaciones respecto al estándar SQL-92.

SQLite soporta los tipos TEXT, INTEGER y REAL, y no requiere de ninguna configuración inicial o administrador de bases de datos para su uso.

La creación de bases de datos con SQLite proporciona un acceso privado que únicamente engloba la aplicación en que fueron creadas. Para poder acceder a la base de datos de una aplicación desde otra, tenemos que hacerlo a través de los Content Providers.

Las bases de datos creadas por una aplicación se guardan dentro del directorio: *"DATA/data/APP_NAME/databases/FILENAME"*

- DATA: es la ruta que devuelve el método *Enviroment.getDataDirectory().*
- APP_NAME: es el nombre de nuestra aplicación.
- FILENAME: es el nombre que le damos a nuestra base de datos al crearla.

3.5 Estructura de un Proyecto en Android con Eclipse

Cuando ejecutamos un nuevo proyecto Android en Eclipse, se genera una estructura de carpetas como las mostradas en la Figura 5.

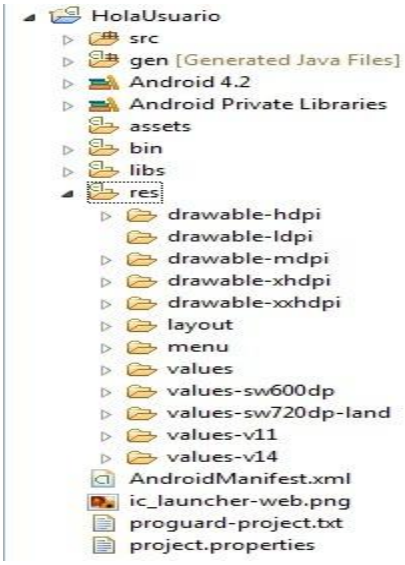

Figura 5: Estructura de carpetas de un Proyecto en Android

De entre todas las carpetas las más importantes son:

- **src:** contiene todo el código fuente de la aplicación. Dentro de esta carpeta es dónde debemos incluir las nuevas clases .java que generemos.
- **gen:** contiene los archivos que genera Eclipse automáticamente al compilar el proyecto. Estos archivos no deben tocarse ya que es el propio Eclipse el que se encarga de modificarlos.
- **Android versión:** contiene la librería de desarrollo de Android.
- **assets:** contiene los recursos usados por la aplicación.
- **res:** contiene todos los ficheros de recursos necesarios para la aplicación. Este directorio contiene las siguientes carpetas:
  - **drawable**: contiene las imágenes clasificadas en función de la resolución.
  - **layout:** contiene los archivos .xml pertenecientes a los diferentes archivos de las diferentes pantallas de la interfaz gráfica.
  - **values:** contiene archivos .xml de otros recursos de la aplicación como colores (colores.xml) o cadenas de texto (string.xml).
- **AndroidManifest:** archivo que contiene información esencial sobre el sistema Android, sin la cual no es posible ejecutar ninguna línea de código. Dentro de este archivo se especifica que actividades o servicios ejecuta nuestra aplicación, que acciones es capaz de realizar y que permisos necesita usar en el caso de que se quiera acceder a recursos compartidos, como la lista de contactos o el uso del GPS.

3.6 Descarga de los componentes

Para comenzar a programar aplicaciones en Android son necesarios los siguientes elementos:

Lo primero es tener un ordenador con los siguientes sistemas operativos:

- Windows XP en 32 bits, Windows Vista de 32 bits o 64 bits o Windows 7 de 32 bits o de 64 bits.
- Linux de 32 bits.
- Mac OS con versión 10.5.8 o superior.

Como el lenguaje de programación más habitual es Java, es necesario tener descargado el JDK (Java Development Kit) de Oracle con versión 1.5 o 1.6. Podemos descargarlo gratuitamente desde la página de Oracle [25].

Para facilitar la programación, usaremos el entorno de desarrollo de Eclipse que permite programar en varios lenguajes de manera rápida y sencilla, gracias a sus complementos.

Eclipse permite su descarga gratuita directamente de su propia web [26].El entorno Eclipse que debe instalarse es alguno que tenga integrado el plugin JDT, para que podamos programar en Java.

Necesitaremos también descargar el SDK de Android, para lo que iremos a su sitio web y descargaremos una versión compatible con nuestro sistema operativo [27].

3.7 Configuración Inicial del SDK

Una vez que tengamos descargados los componentes mencionados en el apartado 3.6 (IDE Eclipse y SDK de Android), comenzaremos con la instalación de los mismos.

Primeramente, descomprimiremos el archivo .rar de Eclipse (si no lo hemos hecho aún) en el directorio que prefiramos y ejecutaremos la aplicación contenida en esa carpeta.

La primera vez que lo ejecutemos nos aparecerá un cuadro de diálogo, como el de la Figura 6, donde tenemos que indicar la carpeta en la que Eclipse almacenará todos los proyectos que creemos.

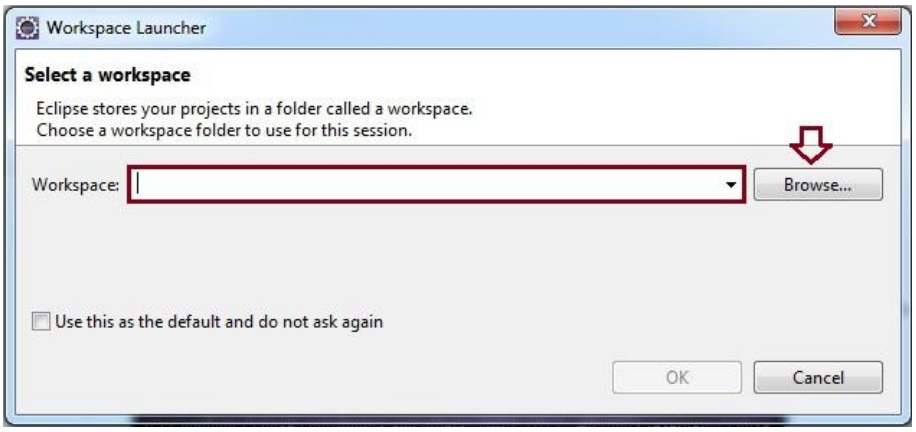

Figura 6: Pantalla de Eclipse para indicar el WorkSpace

A continuación, si la versión que hemos descargado del SDK de Android no viene descomprimida, deberemos descomprimirla en el directorio que elijamos.

El siguiente paso consiste en instalar la herramienta ADT (Android Development Tool), un plug-in específico de Android para la plataforma Eclipse, que facilita la creación de proyectos y la implementación, depuración y ejecución de los mismos.

Para instalar el plug-in ADT seguiremos los siguientes pasos:

- Iniciamos Eclipse.
- De la barra de herramientas superior seleccionamos la opción de *Help-> Install New Software.*

Se nos abrirá una ventana como la de la Figura 7, llamada Available Software donde deberemos pulsar sobre el botón *Add* situado en la parte superior derecha.

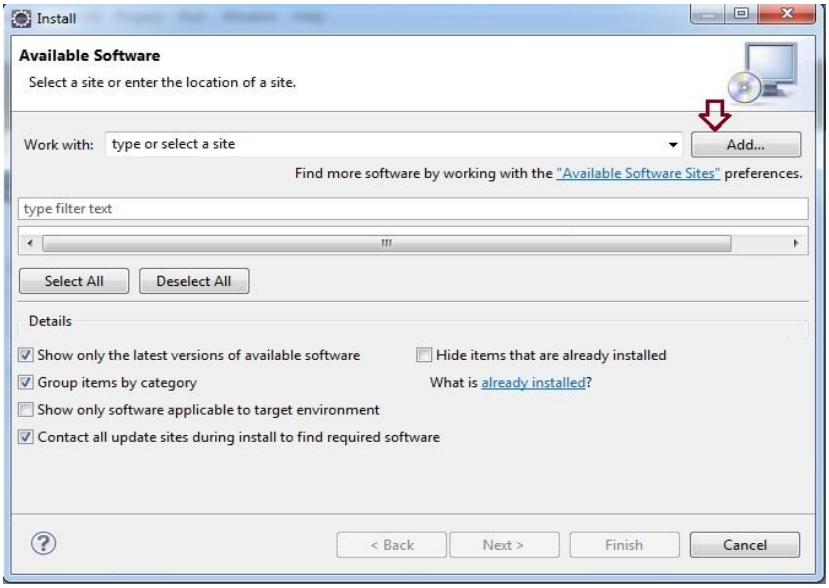

Figura 7: Ventana Available Software

Al pulsar el botón *Add*, se nos abrirá una pestaña como la de la Figura 8, dónde en el campo llamado *Name* pondremos el nombre que queramos darle al nuevo software que vayamos a instalar, en nuestro caso ADT Plugin y en el campo *Location* escribiremos la siguiente URL *https://dl-ssl.google.com/android/eclipse/*.

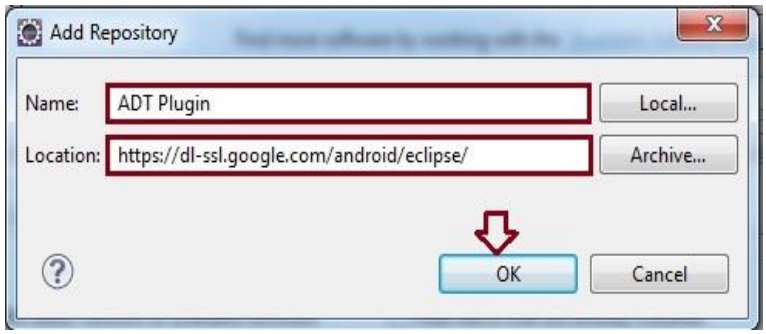

Figura 8: Pestaña para añadir el plugin ADT

Pulsamos sobre el botón de *OK* y volveremos a la ventana de Available Software, mostrada en la Figura 9, donde en el cuadro del centro nos aparecen las opciones: Developer Tools y NDK Plugins. Seleccionamos la opción *Developer Tools* y le damos a *Next*.

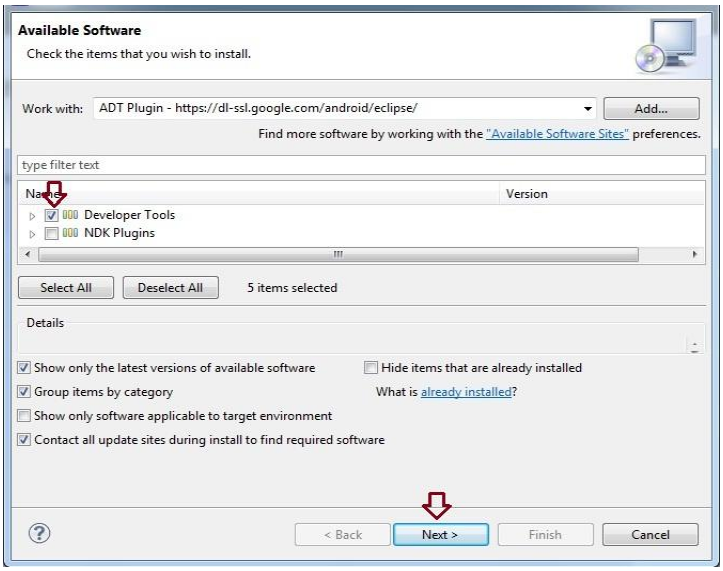

Figura 9: Ventana Available Software con el Plugin ADT añadido

A continuación, nos aparecerá otra ventana con la lista de herramientas que serán instaladas. Volvemos a pulsar sobre el botón *Next*.

Una vez pulsado *Next*, nos aparece una nueva ventana cómo la de la Figura 10, en la que debemos aceptar las condiciones de instalación. Para ello marcamos la opción "*I accept the terms of the license agreements"* y pulsamos sobre el botón *Finish* para que comience la instalación del software seleccionado.

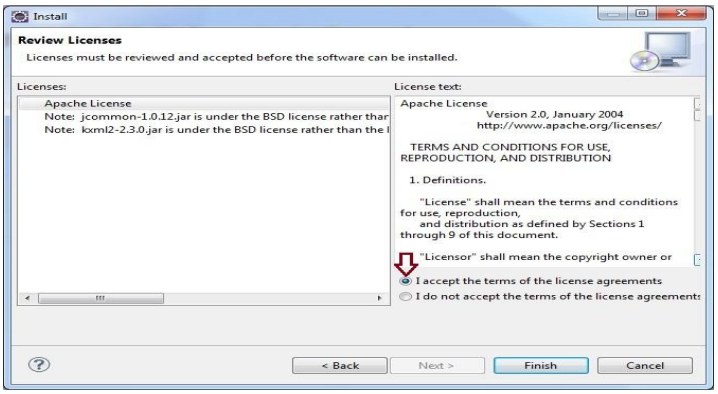

Figura 10: Ventana de aceptación de Términos y Condiciones

El siguiente paso será modificar las preferencias del plugin ADT en Eclipse, para indicarle el lugar en el que se encuentra el SDK de Android. Para ello tenemos que realizar los siguientes pasos sobre la ventana mostrada en la Figura 11:

- Seleccionamos la opción *Window-> Preferences* y se nos abrirá el panel de preferencias.
- Pulsamos la opción *Android* en el panel de la izquierda y sobre el panel de la derecha hacemos clic sobre el botón *Browse* para seleccionar la carpeta en la que queramos descomprimir el SDK.

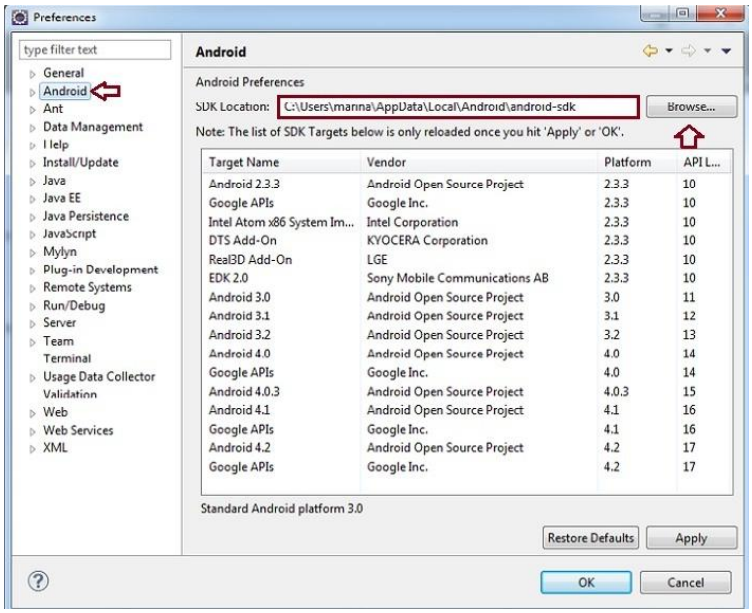

Figura 11: Panel de configuración de las preferencias del plugin ADT en Eclipse.

Pulsamos sobre el botón *Apply* y a continuación sobre el botón *OK*.

Por último para terminar con el proceso de instalación, debemos descargar los componentes esenciales del SDK en nuestro entorno de desarrollo. El SDK de Android utiliza una estructura modular, dónde las partes principales del mismo están divididas en un conjunto de componentes que podemos instalar por separado.

Para desarrollar una aplicación en Android necesitamos, como mínimo, descargar una plataforma Android con sus herramientas asociadas. Para ello tendremos que seguir los siguientes pasos:

- En Eclipse seleccionamos la opción *Windows->Android SDK Manager*.
- Nos aparecerá una ventana como la de la Figura 12, dónde debemos seleccionar la versión de la plataforma Android sobre la que queramos desarrollar nuestra aplicación. Si deseamos instalar algún otro componente solo tenemos que seleccionarlo del panel central. Una vez finalizada la selección, pulsaremos sobre el botón *Install* para finalizar.

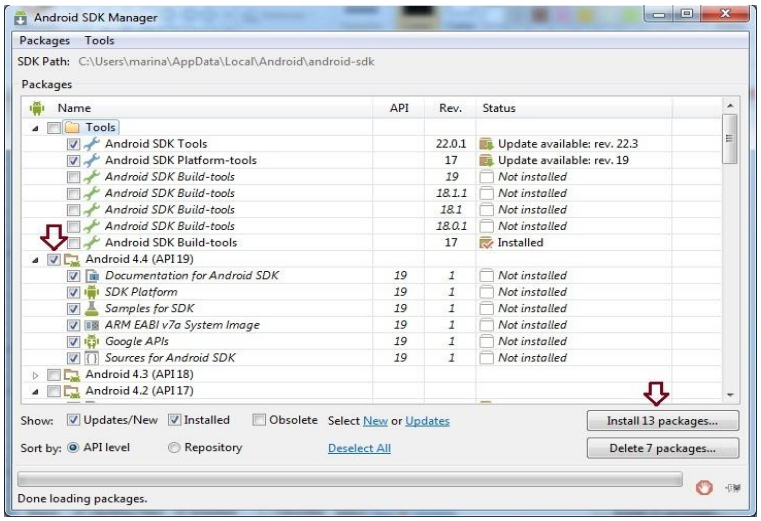

Figura 12: Ventana de configuración del Sistema Operativo Android.

3.8 Configuración Inicial del Android Virtual Device

El Android Virutal Device o AVD, es un conjunto de herramientas fundamentales para los desarrolladores de aplicaciones Android, que permite emular en un ordenador un entorno móvil al que apuntará nuestra aplicación Android.

Antes de comenzar a programar aplicaciones en Android, es importante tener disponible un AVD en Eclipse.

Para crear un nuevo AVD deberemos seguir los siguientes pasos:

- En Eclipse, seleccionamos la opción *Window-> Android Virutal Device Manager*.
- Nos aparecerá una ventana como la de la Figura 13. Seleccionamos el botón *New.*

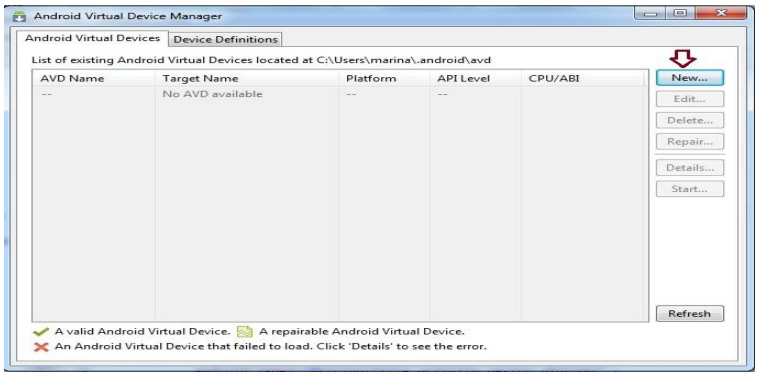

Figura 13: Ventana para añadir un nuevo Android Virtual Device

Nos aparecerá una nueva ventana, como la de la Figura 14, dónde deberemos rellenar los siguientes campos:

- *AVD Name*: es el nombre con el que aparecerá nuestro dispositivo virtual dentro del menú de dispositivos disponibles.
- *Target*: es la versión de Android que se ejecutará en el dispositivo virtual. Esta versión deberá ser como mínimo la que se use para crear las aplicaciones.
- *Device*: permite seleccionar las características correspondientes a un tipo concreto de terminal móvil.

De forma opcional, podremos configurar el resto de opciones que aparecen en la pantalla cómo el tamaño de la SD Card o las opciones de memoria.

Una vez finalizada la selección pulsamos sobre el botón *OK*

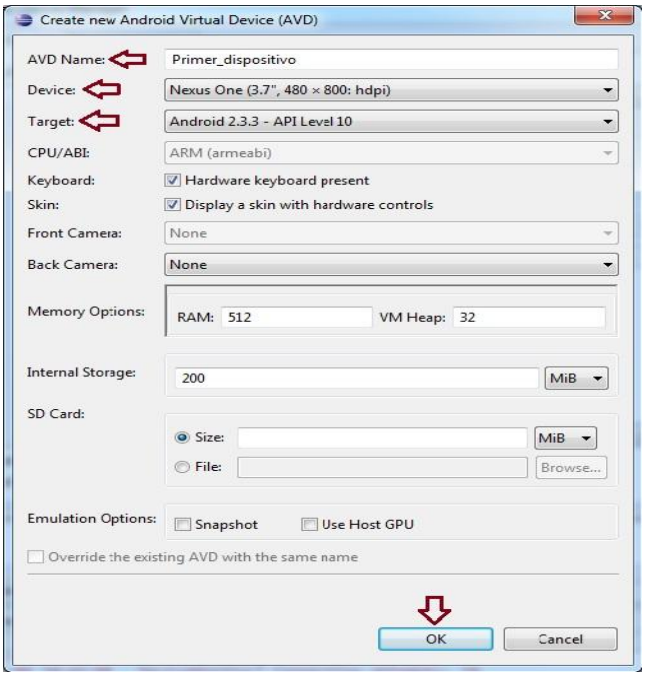

Figura 14: Pestaña para aceptar la configuración establecida para el AVD

Tras pulsar el botón *OK* vuelve a aparecernos la ventana "Android Virutal Device Manager" mostrada en la Figura 15. En el panel central, dentro de la lista de dispositivos AVD disponibles, debe aparecer el dispositivo creado, con el nombre que le hayamos asignado

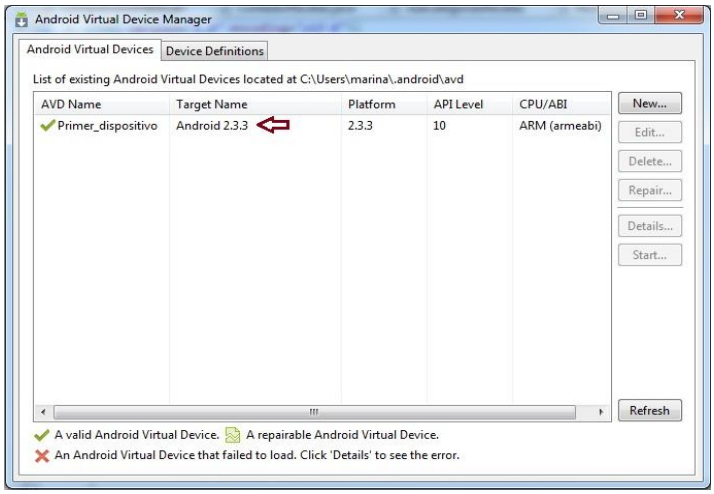

Figura 15: Ventana Android Virtual Device con la lista de dispositivos AVD disponibles

Para arrancar la ejecución de nuestro dispositivo virtual, seleccionamos sobre él dentro del panel central y pulsamos el botón *Start* como se muestra en la Figura 16.

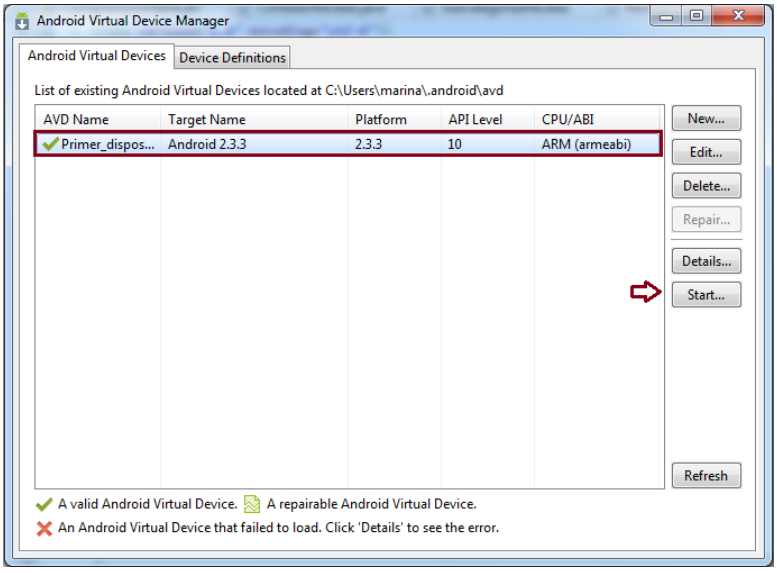

Figura 16: Ventana para arrancar el Android Virtual Device configurado.

Finalmente veremos correr en la pantalla del ordenador nuestro dispositivo virtual como ilustra la Figura 17.

Figura 17: Ejemplo de Android Virtual Device en ejecución

Capítulo 4

Descripción General del Trabajo Realizado

A lo largo de este capítulo se describen las principales características de la aplicación desarrollada para el presente Proyecto Fin de Carrera, comentando las principales clases definidas en la aplicación, las bases de datos implementadas para el almacenamiento de datos y los principales Frames definidos para la implementación de la estrategia de diálogo.

4.1 Características Generales

Las características principales de la aplicación desarrollada para el presente Proyecto Fin de Carrera son:

- La aplicación está desarrollada para el sistema operativo Android, siendo la versión 8 del SDK de Android la mínima requerida para poder ejecutarlo.
- El sistema de diálogo desarrollado se corresponde con un modelo guiado, en el que el sistema es el encargado de dirigir al usuario a través de las distintas fases de la aplicación.
- Las fases por las que pasa el diálogo a lo largo de la ejecución de la aplicación constituyen un sistema de estado finito.
- El modelo de lenguaje es un modelo restringido, con un vocabulario limitado perteneciente a un dominio preestablecido.
- El reconocedor busca entre la secuencia auditiva emitida por el usuario, palabras clave establecidas en cada punto del diálogo.

- El reconocimiento se lleva a cabo con independencia del hablante.
- La síntesis de voz se realiza a partir de sentencias determinadas, asociadas a cada uno de los posibles estados del diálogo.
- El único idioma disponible es el español.

4.2 Principales Clases Definidas para la Aplicación

Para la implementación de la aplicación se han definido las siguientes clases, cada una con una función específica, que se comunican entre ellas para funcionar en conjunto como un sistema de diálogo:

- **Registro De Diálogo:**
  Esta clase emula el comportamiento de una estructura pila con una capacidad de almacenamiento por defecto de 100 unidades. Se encarga de almacenar los distintos frames que se intercambian el resto de clases durante el diálogo. Los métodos principales de esta clase son:
  - **Método push():** permite almacenar un frame en el registro.

```
public void push(Object o) {
if (top == capacity - 1){
System.out.println("La pila está llena");
top = top+1;
data[top] = o;}}
```

  - **Método pop():** permite sacar un frame del registro.

```
public Object pop() {
Object o;if (top == -1) {return null;}
o = data[top];
data[top] = null;
top = top-1;
return o;}
```

  - **Método top():** devuelve el frame que está encima del registro sin sacarlo de él.

```
public Object top(){if (top == -1) {
return null;}
return data[top];}
```

- **Reconocimiento De Voz**
  Actividad encargada de reconocer la señal vocal pronunciada por el usuario y proporcionar la secuencia de palabras más probable.
  Para realizar el reconocimiento de voz en Android realizamos los siguientes pasos:
  - .Verificamos que existe una actividad para reconocer la voz a mediante las siguientes líneas de código:

```
PackageManager pm = getPackageManager();
activities=pm.queryIntentActivities(new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH), 0);
```

➢ Invocamos la actividad de reconocimiento de voz: para ello definimos el siguiente método, dónde primeramente creamos un Intent que iniciará la actividad de reconocimiento de voz. A continuación, configuramos los parámetros del reconocimiento de voz, indicando que el tipo de lenguaje será dictado y finalmente lanzamos el Intent configurado.

```
public void startVoiceRecognitionActivity(int CONSTANTE) {
Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "");
startActivityForResult(intent, CONSTANTE);}
```

➢ Procesar los resultados: para ello implementamos el siguiente código, dentro del método *onActivityResult*, con el que obtenemos la secuencia de palabras reconocidas.

```
protected void onActivityResult(int reqCode, int resultCode, Intent data) {
if (reqCode == CONSTANTE && resultCode == RESULT_OK) {
ArrayList<String> matches =
data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
```

- ### Comprensión del Habla

  Esta actividad, a partir de la secuencia o secuencias reconocidas, extrae su significado y lo encapsula en unos objetos denominados Frames. Para la comprensión de la secuencia, esta clase debe saber en qué punto del diálogo se encuentra la aplicación, para lo cual consulta la actividad Registro de Dialogo y analiza los objetos almacenados hasta el momento. En función de los objetos, buscará en la secuencia reconocida una serie de palabras claves que darán lugar a unos frames u a otros.
  Una vez elaborados los frames, los envía a través de objetos Intents a la actividad Gestor de Diálogo.
  Otra de las tareas que realiza esta actividad es establecer los diferentes elementos gráficos en función del estado de la conversación.

- ### Gestor de Diálogo

  Recibe los frames enviados por la actividad Comprensión del Habla y en función de ellos establece las siguientes acciones a realizar. Sus principales funciones son:

  ➢ Crear el Registro de Diálogo y las bases de datos.

  ➢ Actualizar el Registro de Diálogo con la información proporcionada por la actividad Comprensión del Habla.

  ➢ Lanzar el resto de actividades de la aplicación, proporcionándoles la información necesaria para cumplir con la estrategia de diálogo.

  ➢ Detectar posibles errores en la conversación y pedir retransmisiones

  ➢ Indicar al módulo de consulta de la base de datos la información que debe proporcionar en cada punto del diálogo.

- **Base de Datos Helper**
  Esta clase constituye una de las dos bases de datos de las que dispone la aplicación. Dentro de ella se engloba el módulo de consulta a la base de datos que recibe las peticiones de consulta por parte del Gestor de Diálogo, las procesa y le devuelve la información.
  La estructura de esta clase se detalla en la sección 4.3.1

- **Escenarios**
  Los escenarios son un conjunto de clases que emulan el comportamiento del módulo de generación de respuestas. Los objetos escenarios contienen frases gramaticalmente correctas, lo más cercanas posibles al lenguaje natural que transmiten el mensaje generado por el Gestor de Diálogo en cada estado de la conversación.

- **Texto A Voz**
  Esta actividad recibe los diferentes objetos Escenarios y genera la correspondiente señal de audio que será la respuesta que llegará al usuario. Para que esta actividad pueda realizar la síntesis de voz, hemos seguido los siguientes pasos:
  ➢ Implementar la interfaz *OnInitListener* de la clase *TextToSpeech*, para poder invocar los servicios del método de síntesis de voz de Android.

```
public class TextoAVoz extends Activity implements
TextToSpeech.OnInitListener
```

➢ Comprobar si hay un motor TTS válido, instalado en el dispositivo:

```
Intent comprobacionTTS = new Intent();
comprobacionTTS.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DA
TA);
startActivityForResult(comprobacionTTS,CODIGO_COMPROBACION_TTS);
```

➢ Cuando la comprobación finaliza, la devolución de la llamada devuelve el control al método *onActivityResult*. Dentro de este método analizamos el resultado de la comprobación: si el código de resultado es *CHECK_VOICE_DATA_PASS* significa que hay un motor TTS disponible y por tanto podemos crear una instancia de *TextToSpeech*. En caso contrario, lanzamos al usuario un mensaje informativo para que instale un motor *TextToSpeech* válido en el dispositivo.

```
public void onActivityResult(int resquestCode, int resultCode, Intent data){
if(resquestCode ==CODIGO_COMPROBACION_TTS){
if(resultCode==TextToSpeech.Engine.CHECK_VOICE_DATA_PASS){
tts = new TextToSpeech(this,this);
}else{
Toast.makeText(this,"no esta todo instalado",Toast.LENGTH_SHORT).show();
Intent instalacion = new Intent();
instalacion.setAction(TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
startActivity(instalacion);}}}
```

4.3 Bases de Datos

Las bases de datos son herramientas de gran potencial en la creación de aplicaciones informáticas. Hasta hace poco resultaba costoso y complejo incorporar bases de datos a nuestras aplicaciones móviles. No obstante, Android incorpora la librería SQLite, que permite usar bases de datos de forma sencilla, consumiendo pocos recursos del sistema. Esta librería está basada en el lenguaje de programación SQL, es de código abierto y proporciona las capacidades de una base de datos relacional.

Como hemos comentado anteriormente, las bases de datos que creemos en nuestra aplicación serán almacenadas en la carpeta */data/data/<package-name>/databases.*

Para esta aplicación se han definido dos bases de datos:

- **BaseDatosHelper:** base de datos que almacena toda la información que el usuario puede consultar en el sistema.
- **BaseDatosUsuarios:** base de datos que contiene la información que el usuario decide almacenar de forma permanente.

Para la creación de ambas bases de datos, primeramente hemos creado una clase privada interna que lleva a cabo todas las operaciones necesarias para crear la base de datos, tales como crear las tablas, rellenar las tablas con la información correspondiente y actualizar los datos en caso de modificación. Esta clase interna debe heredar la clase *SQLiteOpenHelper* de Android que facilita la implementación de las funciones descritas.

SQLiteOpenHelper, contiene los siguientes métodos que debemos implementar:

- **onCreate(SQLiteDatabases db):** método invocado cuando se crea la base de datos, es decir, cando se crean tablas, campos, vistas o triggers. Recibe como parámetro la base de datos que se desea crear.
- **onUpgrade(SQLiteDatabases db, int oldVersion, int newVersion):** método invocado cuando se realizan modificaciones sobre la base de datos, como alterar, borrar o crear nuevas tablas.
- **getReadableDatabase():** método que abre la base de datos en modo lectura.
- **getWritableDatabase():** método que abre la base de datos en modo lectura y escritura.

Otro elemento importante en las bases de datos son los Cursores. Los resultados de las consultan de selección se devuelven en objetos de tipo Cursor. Estos objetos no son más que punteros a conjuntos de datos que proporcionan métodos para navegar entre ellos.

- **boolean moveToNext():** avanza el cursor una posición o registro en la tabla de resultados. Devuelve false si el movimiento sobrepasa el último registro.
- **boolean moveToFirst():** retrocede el cursor hasta el primer registro del conjunto de resultados. Devuelve false en caso de que el conjunto de resultados esté vacío.
- **boolean moveToPosition(int position)** desplaza el cursor a la posición pasada por parámetro dentro de un determinado registro. Devuelve false en caso de que no sea posible alcanzar dicha posición.

Para acceder a los valores almacenados dentro de los objetos Cursor, la clase Cursor incluye métodos como: *getShort(), getString(), getFloat(), getDouble()*..etc

Una vez hayamos obtenido toda la información almacenada en los objetos Cursor, es necesario cerrarlos. Para ello llamamos a su método *close()*, que se encarga de cerrarlo y liberar así los recursos consumidos del sistema.

4.3.1 Base de Datos Helper

La Base de Datos Helper, contiene información relativa a los distintos alimentos y recetas que el usuario puede consultar durante la ejecución de la aplicación.

Ambos tipos de información se encuentra organizados por categorías en orden de más general a más concreta, de modo que el usuario va seleccionando varios tipos de categorías hasta llegar al alimento o receta deseada.

4.3.1.1 Tablas

La Figura 18 muestra el conjunto de tablas de las que se compone esta base de datos y la relación entre ellas.

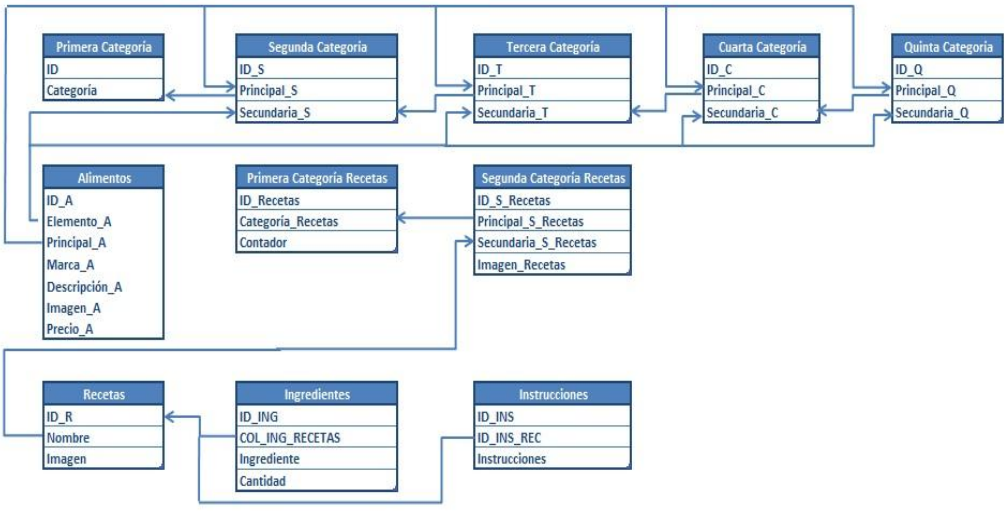

Figura 18: Tablas de Base de Datos Helper

- ### Tabla Primera Categoría
  Esta tabla contiene todas las categorías primarias en las que se organizan los alimentos. Está formada por los campos:
  - **ID:** clave primaria que distingue las diferentes categorías.
  - **CATEGORÍA:** nombre de las categorías primarias .
- ### Tabla Segunda Categoría
  Esta tabla contiene las categorías secundarias correspondientes a cada tipo de categoría primaria. Está compuesta por:
  - **ID_S:** clave primaria para identificar las diferentes categorías secundarias.
  - **PRINCIPAL_S:** clave externa que referencia al campo categoría de la tabla Primera CATEGORÍA, indicando la categoría principal a la que pertenecen los distintos tipos de categorías secundarias.
  - **SECUNDARIA_S:** nombres de las categorías secundarias de los alimentos.
- ### Tabla Tercera Categoría
  Tabla que almacena las distintas categorías terciarias correspondientes a cada tipo de categoría secundaria. Está compuesta por:
  - **ID_T:** clave primaria para identificar las diferentes categorías terciarias.
  - **PRINCIPAL_T:** clave externa que referencia al campo SECUNDARIA_S de la tabla Segunda Categoría, indicando la categoría secundaria a la que pertenecen las distintas categorías terciarias.
  - **SECUNDARIA_T:** nombre de las categorías terciarias de los alimentos.
- ### Tabla Cuarta Categoría
  Tabla que almacena las distintas categorías cuaternarias correspondientes a cada tipo de categoría terciaría. Está compuesta por:
  - **ID_C:** clave primaria que identifica a las diferentes categorías cuaternarias de alimentos.
  - **PRINCIPAL_C:** clave externa que referencia al campo SECUNDARIA_T de la tabla Tercera Categoría, indicando la categoría terciaria a la que pertenecen las distintas categorías cuaternarias.
  - **SECUNDARIA_C:** Nombres de las categorías cuaternarias de los alimentos.
- ### Tabla Quinta Categoría
  Tabla que almacena las distintas categorías quintas correspondientes a cada tipo de categoría cuarta. Contiene los campos.
  - **ID_Q:** clave primaria que identifica a las diferentes categorías quintas de alimentos
  - **PRINCIPAL_Q:** clave externa que referencia al campo SECUNDARIA_C de la tabla Cuarta Categoría, indicando las distintas categorías cuartas a las que pertenecen las distintas categorías quintas.
  - **SECNDARIA_Q:** nombres de las diferentes categorías quintas de los alimentos
- ### Alimentos
  Contiene la información relativa a cada tipo de alimento que el usuario puede consultar y almacenar en la lista de la compra. Contiene los campos:

- **ID_A:** clave primaria que identifica a cada uno de los alimentos.
- **ELEMENTO_A:** clave externa que identifica la categoría última a la que pertenece cada alimento. Debido a que los distintos alimentos se pueden organizar en tres, cuatro o cinco categorías este campo referencia a los campos SECUNDARIA_S, SECUNDARIA_T, SECUNDARIA_C y SECUNDARIA_Q de las tablas Segunda Categoría, Tercera Categoría, Cuarta Categoría y Quinta Categoría respectivamente.
- **PRINCIPAL_A:** clave externa que referencia la categoría penúltima a la que pertenece cada alimento. Debido a que los distintos alimentos pueden tener como penúltima categoría las categorías primarias, secundarias, terciarias y cuaternarias, este campo referencia a los campos CATEGORIA, PRINCIPAL_S, PRINCIPAL_T, PRINCIPAL_C de las tablas Primera Categoría, Segunda Categoría, Tercera Categoría y Cuarta Categoría respectivamente.
- **MARCA_A:** Contiene el nombre de la marca a la que pertenece cada alimento.
- **DESCRIPCION_A:** contiene un texto descriptivo de cada uno de los alimentos.
- **IMAGEN_A:** almacena la referencia a una foto de cada uno de los alimentos.
- **PRECIO_A:** Precio en euros de cada uno de los alimentos

- ### Primera Categoría Recetas

  Almacena las categorías primarias en las que se organizan las diferentes recetas disponibles en la aplicación. Contiene los campos:

  - **ID_RECETAS:** clave primaria que identifica a cada una de las categorías primarias de recetas.
  - **CATEGORIA_RECETAS:** nombres de las diferentes categorías primarias de recetas.
  - **CONTADOR:** número que indica las veces que el usuario ha visualizado cada una de las categorías de recetas.

- ### Segunda Categoría Recetas

  Tabla que almacena los nombres de las diferentes recetas correspondientes a cada tipo de categoría primaria. Contiene los campos:

  - **ID_S_RECETAS:** clave primaria para identificar las diferentes recetas.
  - **PRINCIPAL_S_RECETAS:** clave externa que referencia al campo CATEGORIA_RECETAS de la tabla Primera Categoría Recetas, indicando la categoría primaria a la que pertenece cada una de las recetas.
  - **SECUNDARIA_S_RECETAS:** nombre de cada una de las recetas.
  - **IMAGEN_RECETAS:** referencia a una foto de cada una de las recetas.

- ### Recetas

  Tabla que contiene la información de cada una de las recetas. Contiene los campos:

  - **ID_R:** clave primaria que identifica a cada una de las recetas.
  - **NOMBRE:** clave externa que referencia al campo SECUNDARIA_S_RECETAS de la tabla Segunda Categoría Recetas, indicando el nombre de cada receta.
  - **IMAGEN:** imagen de cada una de las recetas.

- ### Ingredientes

  Almacena los distintos ingredientes y sus cantidades necesarias para la elaboración de cada una de las recetas. Contiene los campos:

- **ID_ING:** clave primaria que identifica a cada ingrediente.
- **COL_ING_REC:** clave externa que referencia al campo ID_R de la tabla Recetas, para indicar la receta a la que pertenece cada ingrediente.
- **INGREDIENTE:** nombre de cada uno de los ingredientes.
- **CANTIDAD:** cantidad de cada uno de los ingredientes.

- **Instrucciones**

  Almacena las distintas instrucciones a seguir para la elaboración de cada receta. Contiene los campos:

  - **ID_INS: clave** primaria que identifica a cada instrucción.
  - **ID_INS_REC:** clave externa que referencia al campo ID_R de la tabla Recetas, indicando la receta a la que pertenece cada instrucción.
  - **INSTRUCCIONES:** cada una de las instrucciones para la elaboración de cada receta.

4.3.1.2 Triggers

Un trigger o disparador, es un procedimiento asociado con una tabla, que automáticamente se ejecuta cuando en la tabla se inserta, se actualiza o se borra una fila.

Son usados para garantizar la integridad referencial y la coherencia entre los datos de las distintas tablas.

Los triggers no pueden ser invocados directamente ya que se ejecutan automáticamente y no reciben ni retornan parámetros.

Para la base de datos Base de Datos Helper se han definido los siguientes triggers:

- **-fk_secundaria:** comprueba que al insertar una nueva categoría secundaria en la tabla Segunda Categoría, la categoría principal a la que pertenece está definida en la tabla Primera Categoría. En caso de no ser así aborta la inserción lanzando un error.
- **-fk_terciaria:** comprueba que al insertar una nueva categoría terciaria en la tabla Tercera Categoría, la categoría secundaria a la que pertenezca esté definida en la tabla Segunda Categoría. En caso de no cumplirse aborta la inserción lanzando un error.
- **-fk_cuaternaria:** comprueba que al insertar una nueva categoría cuaternaria en la tabla Cuarta Categoría, la categoría terciaria a la que pertenece está definida en la tabla Tercera Categoría. En caso de no ser así aborta la inserción lanzando un mensaje de error.
- **-fk_quinta:** comprueba que al insertar una nueva categoría quinta en la tabla Quinta Categoría, la categoría cuaternaria a la que pertenece esté definida en la tabla Cuarta Categoría. En caso de no ser así aborta la inserción lanzando un mensaje de error.
- **-fk_alimentos:** comprueba que antes de insertar un alimento en la tabla Alimentos el campo ELEMENTO_A que referencia a la categoría última del alimento, esté definido en alguna de las tablas Segunda Categoría, Tercera Categoría, Cuarta Categoría o Quinta Categoría.
- **-fk_alimentos_principal:** comprueba que antes de insertar un alimento en la tabla Alimentos, el campo PRINCIPAL_A que referencia la categoría penúltima del alimento, esté definido en alguna de las tablas Primera Categoría, Segunda Categoría, Tercera Categoría y Cuarta Categoría.
- **-fk_secundaria_recetas:** comprueba que antes de insertar una receta en la tabla Segunda Categoría Recetas, el campo PRINCIPAL_S_RECETAS que referencia a la categoría principal a la que pertenece la receta, esté contenido en la tabla Primera Categoría Recetas. En caso de no ser así aborta la inserción y lanza un mensaje de error.
- **-fk_recetas:** comprueba que antes de insertar una receta en la tabla Recetas, el campo nombre que referencia al nombre de cada una de las recetas, esté contenido en la tabla Segunda Categoría Recetas. En caso de no ser así aborta la inserción y lanza un mensaje de error.
- **-fk_ingredientes:** comprueba que antes de insertar un ingrediente en la tabla Tabla Ingredientes, el campo COL_ING_REC que referencia la receta a la que pertenece el ingrediente, esté contenido en la tabla Recetas. En caso de no ser así aborta la inserción y lanza un mensaje de error.
- **-fk_instrucciones:** comprueba que antes de insertar una instrucción en la tabla Instrucciones, el campo COL_INS_REC que referencia la receta a la que pertenece la instrucción, esté contenido en la tabla Recetas. En caso de no ser así aborta la inserción y lanza un mensaje de error.

4.3.1.3 Proceso de Creación de la Base de Datos

Para crear la base de datos seguimos los siguientes pasos:

- Creamos nuestra clase interna que hereda de SQLiteOpenHelper.

- Definimos el constructor que recibe los siguientes parámetros:

- context: contexto asociado a la base de datos.
- DATABASE_NAME: nombre de la base de datos.
- DATABASE_VERSION: versión de la base de datos.

➢ Sobrescribimos el método *onCreate(SQLiteDatabase db*). Recibe como parámetro la base de datos que vamos a crear. Este método es invocado cada vez que se crea la base de datos, es decir, cuando la base de datos no existe en el disco y se ejecuta una única vez en un mismo dispositivo; la primera vez que la aplicación se ejecuta en el dispositivo. Contiene los siguientes métodos:

- **createTables(db):** crea las tablas Primera Categoría, Segunda Categoría, Tercera Categoría, Cuarta Categoría y Quinta Categoría y los triggers fk_seundaria, fk_terciaria, fk_cuaternaria, fk_quinta, fk_alimentos y fk_alimentos_principal.
- **createTablesRecetarioPrimera(db):** crea las tablas Primera Categoría Recetas, Segunda Categoria Recetas, Recetas, Tabla Ingredientes y Tabla Instrucciones y los triggers fk_recetas, fk_secundaria_recetas, fk_alimentos y fk_alimentos_principal. Además rellena parte de los datos que contienen dichas tablas.
- **createTableRecetarioSegunda(db):** finaliza el relleno de los datos de las tablas anteriormente citadas.

➢ Implementamos el método *deleteTables(SQLiteDatabase db)*, que se encarga de borrar todas las tablas y los triggers.

➢ Sobrescribimos el método *onUpgrade(SQLiteDatabase db, int oldVersion, int new Version)*. En este método primeramente borramos todos los elementos de la base de datos llamando al método *deleteTables* y posteriormente los volvemos a crear llamando a los métodos *createTables(db), createTablesRecetarioPrimera(db)* y *createTablesRecetarioSegunda(db)*. *OnUpgrade()* se invoca cada vez que se realizan modificaciones sobre la base de datos como crear nuevas tablas, alterarlas y borrarlas

➢ Sobrescribimos el método *onStop()*, que no recibe ningún parámetro y tiene como misión cerrar el cursor usado para recuperar los registros devueltos en las consultas de selección a la base de datos.

```
super(context, DATABASE_NAME, null, DATABASE_VERSION);}
public void onCreate(SQLiteDatabase db){
createTables(db);
createTablesRecetarioPrimera(db);
createTablesRecetarioSegunda(db);}
private void deleteTables(SQLiteDatabase db){
db.execSQL("DROP TABLE IF EXISTS " + TABLE_PRIMARIA);…}
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
deleteTables(db);
createTables(db);
createTablesRecetarioPrimera(db);
createTablesRecetarioSegunda(db);
public void OnStop(){if (!c.isClosed()){c.close();}}}
```

4.3.1.4 Implementación de la Clase Base de Datos Helper

La clase Base de Datos Helper es la clase que contiene la clase BaseDatosHelperInterna, encapsulando la lógica de la base de datos. Esta clase es la encargada de abrir y cerrar la base de datos y de proporcionar los métodos necesarios para la selección y modificación de los datos.

Como variables, esta clase contiene la versión actual de la base de datos, el nombre de la base de datos, la ruta donde se almacenarán los datos y los nombres de las distintas tablas y campos que componen la base de datos.

El constructor recibe como parámetro un objeto Context.

Dentro de esta clase se definen los siguientes métodos:

- **Métodos para abrir y cerrar la Base de Datos**
  - **public BaseDatosHelper open():** método encargado de abrir la base de datos. Para ello llama al método *getWritableDatabase()* de SQLiteOpenHelper, que proporciona acceso a la base de datos en modo lectura y escritura. Devuelve una referencia a la base de datos.
  - **public void close():** método encargado de cerrar la base de datos.

- **Métodos para la selección de Datos**

- **public Cursor obtenerCategoríaPrincipal():** devuelve dentro de un objeto Cursor los nombres de las categorías primarias de los alimentos.
- **public Cursor obtenerCategoríaPrimariaRecetas():** devuelve dentro de un objeto Cursor los nombres de las categorías primarias de las recetas
- **public Cursor obtenerCategoríaSecundaria(String principal):** devuelve dentro de un objeto Cursor los nombres de las categorías secundarias de alientos asociadas a una determinada categoría primaria que se pasa como parámetro.
- **public Cursor obtenerCategoríaSecundariaRecetas(String principal):** devuelve dentro de un objeto Cursor los nombres de las recetas asociadas a una determinada categoría primaria que se pasa como parámetro.
- **public Cursor obtenerCategoríaTerciaria(String secundaria):** devuelve dentro de un objeto Cursor los nombres de las categorías terciarias de alimentos asociadas a una determinada categoría secundaría que se pasa como parámetro.
- **public Cursor obtenerCategoríaCuaternaria(String terciaria):** devuelve dentro de un objeto Cursor los nombres de las categorías cuaternarias de alimentos asociadas a una determinada categoría terciaria que se pasa como parámetro.
- **public Cursor obtenerCategoríaQuinta(String cuaternaria):** devuelve dentro de un objeto Cursor los nombres de las categorías quintas de los alimentos asociadas a una determinada categoría cuaternaria que se pasa como parámetro.
- **public Cursor obtenerAlimentosSinTerciaria(String secundaria, String principal):** devuelve dentro de un elemento Cursor los alimentos que solo tienen categorías primarias y secundarias, concretamente las que se pasa por parámetro.
- **public Cursor obtenerAlimentosSinCuaternaria(String terciaria, String secundaria):** Devuelve dentro de un elemento Cursor los alimentos que tienen categorías primaria, secundaria y terciaria, concretamente aquellos que pertenecen a las categorías secundarias y terciarias pasadas como parámetro.
- **public Cursor obtenerAlimentosSinQuinta(String cuaternaria, String terciaria):** devuelve dentro de un elemento Cursor los alimentos que tienen categoría primaria, secundaria, terciaria y cuaternaria, concretamente aquellos que pertenecen a las categorías terciarias y cuaternarias pasadas por parámetros.
- **public ArrayList<DatosTablaIngredientes> buscarIngredientes(int clave):** devuelve en forma de array objetos DatosTablaIngredientes, los ingredientes asociados a la receta que tiene como clave el entero pasado como parámetro. Los objetos DatosTablaIngredientes constan de dos campos: un string que almacena la cantidad del ingrediente y otro string que almacena su nombre.
- **public ArrayList<DatosTablaInstrucciones> buscarInstrucciones(int clave):** devuelve en forma de array de objetos DatosTablaInstrucciones las instrucciones asociadas a la receta que tiene como clave el entero pasado como parámetro. Los objetos DatosTablaIngredientes constan de un único campo, un string, que almacena la instrucción.
- **public Cursor obtenerContadores():** devuelve dentro de un elemento Cursor los nombres de las categorías primarias de recetas y los valores de los contadores asociadas a ellas.
- **public String getContador(String receta):** devuelve el valor del contador de la receta que tiene como nombre el String pasado por parámetro.
- **public Cursor mostrarReceta(String nombre):** devuelve dentro de un elemento Cursor la clave y la imagen de la receta que tiene por nombre el string pasado por parámetro.

- **Consultas de Modificación**
  - **public void aumentarContador(String categoría):** aumenta en una unidad el campo contador de TablaPrimariaRecetas asociado a la categoría que tiene como nombre el string pasado por parámetro.

4.3.2 Base Datos Usuarios

La Base de Datos Usuarios simula el comportamiento de una lista de la compra, ya que almacena la información de los alimentos que el usuario desea guardar.

4.3.2.1 Tablas

Está compuesta por una única tabla mostrada en la Figura 19.

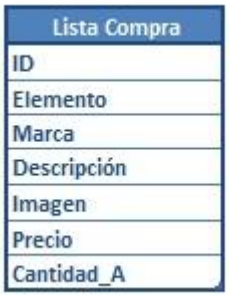

Figura 19: Tabla de Base de Datos Usuarios

La tabla Lista Compra contiene los siguientes campos:

- **ID:** clave primaria que identifica a cada alimento.
- **ELEMENTO:** nombre de cada alimento.
- **MARCA:** marca de cada alimento.
- **DESCRIPCION:** breve descripción de cada alimento.
- **IMAGEN:** referencia a una foto de cada alimento.
- **PRECIO:** precio en euros de cada alimento.
- **CANTIDAD_A:** número de unidades que el usuario desea comprar de cada alimento.

4.3.2.2 Proceso de Creación de la Base de Datos

Para la creación de la base de datos seguimos los mismos pasos ejecutados para la creación de la Base de Datos Helper definidos en el apartado 4.3.1.3.

4.3.2.3 Implementación de la Clase Base de Datos Usuarios

La clase BaseDatosUsuarios contiene la clase interna que hereda de SQLiteHelper encapsulando la lógica de la base de datos. Se encarga de abrir y cerrar la base de datos y de proporcionar los métodos necesarios para la manipulación de los datos.

Como variables, esta clase contiene la versión actual de la base de datos, el nombre, la ruta dónde se almacenará la base de datos y el nombre de la tabla junto con los campos por los que está formada.

El constructor únicamente recibe como parámetro un objeto Context.

Dentro de esta clase se definen los siguientes métodos:

- MÉTODOS PARA ABRIR Y CERRAR LA BASE DE DATOS:
  - **public BaseDatosUsuarios open():** método que abre la base de datos en modo lectura y escritura.
  - **public void close():** método que cierra la base de datos.
- MÉTODOS PARA LA SELECCIÓN DE DATOS
  - **public Cursor obtenerTodosLosElementos():** devuelve dentro de un objeto Cursor toda la información de todos los alimentos almacenados en la base de datos.
  - **public int obtenerNúmeroElementos():** método que devuelve el número de alimentos almacenados en la base de datos.
  - **public ArrayList<String> obtenerClavesAlimentos():** método que devuelve en un array de string las claves de todos los alimentos almacenados en la base de datos.
  - **public float obtenerPrecioAlimentos():** método que devuelve la suma de los precios de todos los alimentos almacenados en la base de datos.
- MÉTODOS PARA LA MODIFICACIÓN DE DATOS
  - **public long introducirDato(DatosAComprar dac):** introduce un alimento en la base de datos. Para ello recibe como parámetro un objeto DatosAComprar que contiene toda la información del alimento que queremos añadir.
  - **public void eliminarRegistro(String clave):**elimina de la base de datos el alimento que tiene como clave el string pasado como parámetro.

4.4 Frames

Los frames son objetos definidos en la aplicación que permiten al módulo de Comprensión del Habla almacenar la interpretación semántica de las distintas intervenciones del usuario.

El uso de frames para la implementación de la estrategia de diálogo permite dotar al sistema de cierto grado de flexibilidad, ya que no es necesario seguir un orden totalmente preestablecido para completar la información requerida por las diferentes funcionalidades que ofrece el sistema.

Tras cada intervención del usuario el módulo de comprensión del habla recibe la sentencia de palabras reconocidas por el módulo de RAH y elabora un objeto frame.

Este objeto es enviado a la clase Gestor de Diálogo que tras recibirlo lo almacena en la pila Registro de Diálogo, permitiendo conocer así la información que hasta un determinado momento ha pronunciado el usuario.

A continuación el Gestor de Diálogo interpreta el frame y establece cuál será el siguiente estado de ejecución dentro del diagrama de estados de la aplicación.

4.4.1 Proceso de generación de Frames

Durante el proceso de generación de frames la clase Comprensión de Habla lleva a cabo los siguientes pasos.

- Consultar el último frame almacenado en el Registro de Diálogo y establecer las palabras claves a buscar.
- Buscar en la sentencia reconocida dichas palabras clave.
- En función del resultado de la búsqueda elaborar los correspondientes frames.

Los tipos de frames pueden dividirse en dos categorías

- **Frames informativos:** recogen una representación semántica válida dentro del contexto de la aplicación de la sentencia pronunciada por el usuario.
- **Frames de error:** creados para informar de un error cuando no ha sido posible interpretar la sentencia pronunciada por el usuario.

Un campo importante presente en algunos frames es el campo error_anterior. Cuando la clase Comprensión del Habla accede al Registro de Diálogo y ve que hay un frame de error, el siguiente frame que manda es el frame que había inmediatamente por debajo de él en la pila y pone el campo error_anterior con el valor "*Si*", para que el Gestor de Diálogo sepa el estado en el que tiene que permanecer para volver a solicitar la información al usuario

4.4.2 Tipos de Frames definidos:

- **Frame Acción General**
  Este frame informa acerca de si el usuario ha decidido iniciar la lista de la compra o el recetario
  - **opción:** contiene le valor "*ListaCompra"* o "*Recetario*", en función de la decisión del usuario.
- **Frame ErrorAcciónGeneral:**
  Informa de que ha habido un error al reconocer la sentencia emitida por el usuario en la que indica si desea añadir o eliminar un alimento
  - **opción:** contiene el valor *"Error"*.
- **Frame AcciónListaCompra**
  Informa de si el usuario desea añadir o eliminar un alimento de la lista de la compra.

- **opción:** contiene el valor *"Añadir"* o "*Eliminar*" en función de la decisión del usuario.
- **error_anterior:** contiene los valores *"Sí"* o *"No*" en función de si la siguiente sentencia que pronuncia el usuario ha sido reconocida con éxito o ha habido algún error.

- ### Frame CategoríaPrincipal
  Este frame recoge la información de las diferentes categorías elegidas por el usuario a la hora de consultar un alimento:
  - **principal:** contiene una de las categorías principales de alimentos.
  - **secundaria:** contiene una de las categorías secundarias de alimentos.
  - **terciaria:** contiene una de las categorías terciarias de alimentos.
  - **cuaternaria:** contiene una de las categorías cuaternarias de alimentos.
  - **quinta:** contiene una de las categorías quintas de alimentos.
  - **error_anterior:** contiene los valores *"Sí"* o "*No"* en función de si la siguiente sentencia emitida por el usuario ha sido reconocida con éxito o por el contrario se ha producido algún error.

- ### Frame ErrorCategoríaPrincipal
  Informa de que ha habido un error al reconocer la categoría principal pronunciada por el usuario
  - **opción:** contiene el valor *"Error"*.

- ### Frame ErrorCategoríaTerciaria
  Informa de que ha habido un error al reconocer la categoría terciaria pronunciada por el usuario
  - **opción:** contiene el valor "*Error"*.

- ### Frame ErrorCategoríaCuaternaria:
  Informa de que ha habido un error al reconocer la categoría cuaternaria pronunciada por el usuario:
  - **opción:** contiene el valor "*Error"*.

- ### Frame ErrorCategoríaQuinta
  Informa de que ha habido un error al reconocer la categoría quinta pronunciada por el usuario
  - **opción:** contiene el valor *"Error"*.

- ### Frame FinCategorías
  Informa de que una categoría seleccionada por el usuario no contiene otras subcategorías.
  - **categoríaFinalizada:** contiene el valor *"Sí"*.

- ### Frame DatosAComprar
  Recoge la información almacenada en la base de datos perteneciente al alimento que el usuario desea añadir a la lista de la compra.

- **cantidad:** número de unidades que el usuario ha especificado que desea comprar.
- **clave:** contiene la clave primaria asociada al alimento en la base de datos.
- **descripción:** contiene la descripción del alimento almacenado en la base de datos.
- **elemento:** contiene el nombre del alimento almacenado en la base de datos.
- **imagen:** contiene la referencia a la imagen del alimento almacenada en la base de datos.
- **marca:** contiene la marca del alimento almacenada en la base de datos.
- **precio:** contiene el precio del alimento almacenado en la base de datos.
- **error_anterior:** contiene los valores *"Sí"* o *"No"*, en función de si la siguiente sentencia que pronuncia el usuario ha sido reconocida con éxito o ha habido algún error.

- ### Frame ConfirmaciónAñadir
  Informa de si el usuario ha confirmado añadir el alimento o por el contrario ha preferido no añadirlo
  - **opción:** contiene las palabras clave *"Sí"* o *"No"* en función de la decisión del usuario

- ### Frame ErrorConfirmaciónAñadir
  Informa de que se ha producido un error en el reconocimiento de la sentencia emitida por el usuario cuando la aplicación le solicita la confirmación del alimento
  - **opción:** contiene el valor *"Error"*.

- ### Frame ConfirmarEliminarAlimento:
  Informa de si el usuario ha confirmado eliminar el alimento o por el contrario ha preferido no eliminarlo
  - **opción:** contiene las palabras *"Sí"* o *"No"* en función de la decisión del usuario.

- ### Frame OpciónRecetario
  Este frame informa acerca de si el usuario ha decidido consultar alguna receta o ha preferido que el sistema le proponga alguna
  - **opción:** contiene los valores "*Consultar"* o *"Proponer*" en función de la opción indicada por el usuario.
  - **error_anterior:** contiene los valores "*Sí"* o "*No*" en función de sí la siguiente sentencia que pronuncia el usuario ha sido reconocida con éxito o ha habido algún error.

- ### Frame ErrorOpciónRecetario
  Informa de que ha habido un error al reconocer la sentencia emitida por el usuario en la que indica si desea consultar una receta o que el sistema le proponga alguna
  - **opción:** contiene el valor "*Error"*.

- ### Frame CategoríaPrincipalRecetas
  Informa sobre la categoría y el nombre de la receta que ha decidido consultar el usuario.

- **principal:** contiene le nombre de la categoría principal de la receta.
- **secundaria:** contiene el nombre de la receta almacenado en la base de datos.
- **error_anterior:** contiene los valores "*Sí*" o "*No*" en función de si la siguiente sentencia emitida por el usuario en la que indica la categoría de la receta que desea consultar, ha sido reconocida con éxito o no.

Capítulo 5

Módulo Lista de la Compra

En este capítulo se realiza un análisis detallado de los diferentes submódulos que integran el Módulo Lista de la Compra, describiendo las funcionalidades que ofrece al usuario, su arquitectura, los diagramas de flujo y los distintos escenarios de uso.

5.1 Submódulos de Ejecución

5.1.1 Submódulo Inicio

5.1.1.1 Funcionalidad

En este submódulo, se implementa el primer diálogo entre el usuario y la aplicación, que tras un saludo inicial, permite al usuario seleccionar cuál de las dos funcionalidades que ofrece la aplicación (lista de la compra o recetario) desea iniciar.

5.1.1.2 Diagrama de Flujo y Arquitectura

La Figura 20 ilustra el diagrama de flujo entre las diferentes clases de la aplicación.

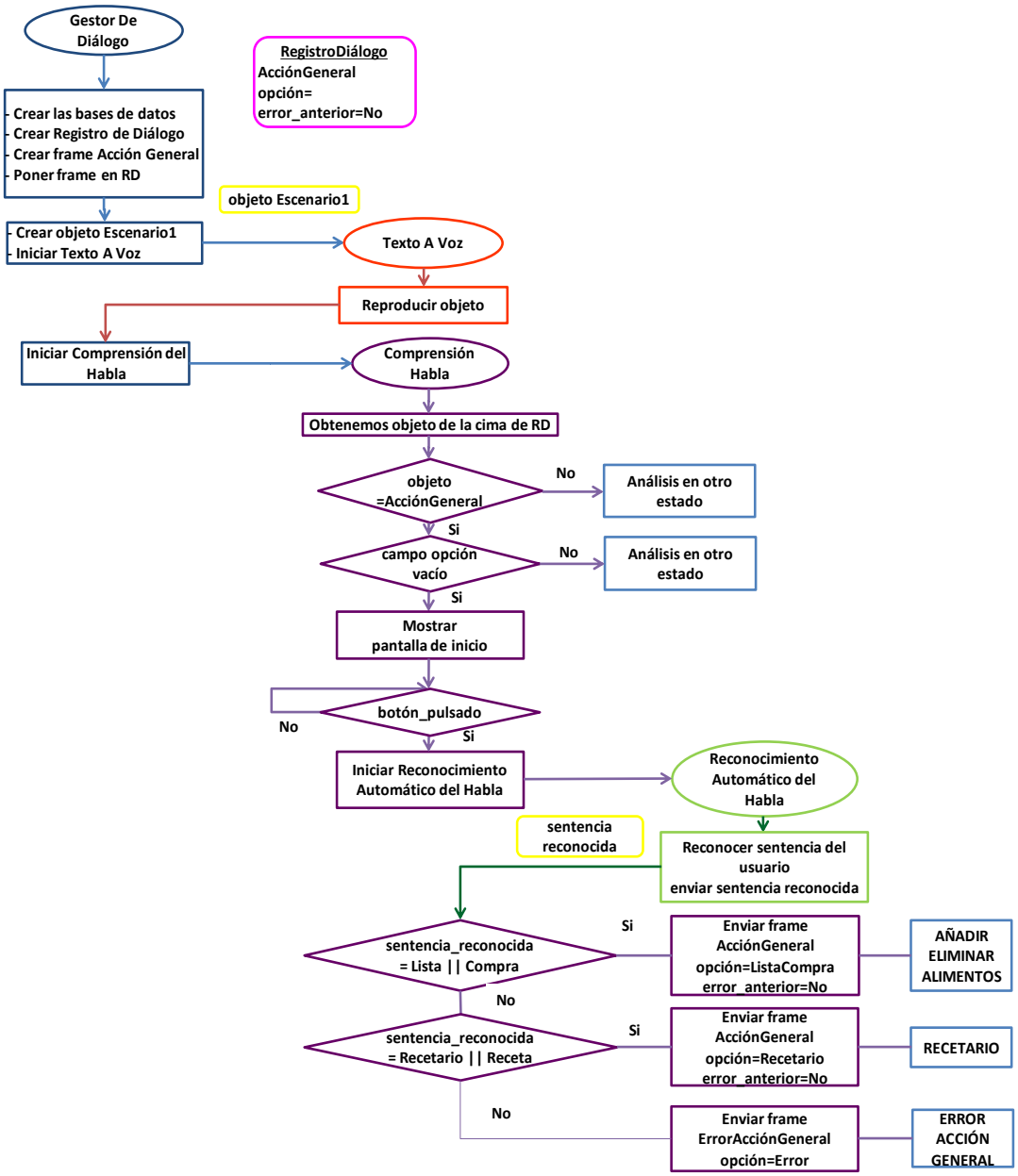

Figura 20: Diagrama de Flujo Submódulo Inicio

❖ La actividad Gestor de Diálogo, actividad principal de la aplicación, crea e inicializa las dos bases de datos desarrolladas para esta aplicación.

```
public static BaseDatosHelper bdh;
public static BaseDatosUsuarios bdu;
```

- **BaseDatosHelper:** almacena toda la información relativa a los alimentos y recetas que el usuario puede consultar.
- **BaseDatosUsuarios:** almacena la información relativa a los alimentos que el usuario desea añadir a la lista de la compra

A continuación, el Gestor de Diálogo crea la clase Registro de Diálogo, que emula el funcionamiento de una estructura pila.

```
static RegistroDialogo rd;
rd = new RegistroDialogo(100);
```

La funcionalidad del Registro de Diálogo es la de almacenar los distintos frames que se generan a lo largo de la aplicación, permitiendo al resto de componentes conocer en cada momento en que punto se encuentra el diálogo. Finalmente, el Gestor de Diálogo crea el frame AcciónGeneral y lo coloca en el Registro Diálogo.

Una vez inicializados los componentes, el Gestor Diálogo construye un objeto de tipo Escenario1, que dará la bienvenida al usuario y le pedirá que indique si desea iniciar la lista de la compra o el recetario. Este mensaje es enviado a la actividad Texto a Voz

- La actividad Texto a Voz reproduce la sentencia recibida haciendo uso del motor de síntesis de voz que proporciona Android. Seguidamente retorna el control a la actividad Gestor de Diálogo.

- El Gestor inicia la actividad Comprensión Del Habla.

- La actividad Comprensión del Habla, obtiene el frame de la cima del Registro de Dialogo. Al tratarse de AcciónGeneral, con el campo opción a null, configura los elementos gráficos en la forma en la que se ilustra en Figura 21.

Figura 21: Elementos gráficos submódulo Inicio.

El usuario, para indicar la funcionalidad que desea iniciar, pulsará sobre el botón de voz mostrado en pantalla, sobre el que se registra un evento, de tipo *setOnClickListener*, de modo que cuando el usuario pulse sobre él se ejecutará el código que aparece dentro del método *onClick()*. Este método lo que hará será iniciar la actividad Reconocimiento Automático del Habla.

```
private ImageButton accion_general_micro;
```

```
accion_general_micro =
(ImageButton)findViewById(R.id.accion_general_microfono);
accion_general_micro.setOnClickListener(new OnClickListener() {
public void onClick(View arg2) {
startVoiceRecognitionActivity(VOICE_ACCION_GENERAL_OPCION);}});
```

- La actividad Reconocimiento Automático del Habla, almacena la sentencia pronunciada por el usuario y la envía a la actividad Comprensión del Habla.

- La actividad Comprensión del Habla, analiza dicha sentencia realizando una comparación con el conjunto de palabras claves definidas para este estado, generando un nuevo frame, que enviará al Gestor de Diálogo, determinando el siguiente submódulo de ejecución.

  - → Si sentencia_reconocida contiene los valores "*Lista"* o *"Compra*": se envía el frame Acción General con el valor *"Lista Compra*" en el campo opción, por lo que el siguiente submódulo que ejecutará el Gestor de Diálogo será Añadir/Eliminar Alimentos.
  - → Si sentencia_reconocida contiene el valor *"Recetario*" enviamos el frame Acción General con el valor *"Recetario*" en el campo opción, por lo que el siguiente submódulo que ejecutará el Gestor de Diálogo será el submódulo Recetario.
  - → Si sentencia_reconocida no contiene ninguna de las opciones anteriores: enviamos el frame Acción General con el valor *"Error"* en el campo opción, por lo que el siguiente submódulo que ejecutará el Gestor de Diálogo será el submódulo Error Acción General

5.1.2 Submódulo Añadir Eliminar Alimentos

5.1.2.1 Funcionalidad

El Gestor de Diálogo inicia la ejecución de este submódulo cuando recibe el frame AcciónGeneral, con el valor "*ListaCompra*" en el campo opción. La funcionalidad principal de este submódulo es preguntar al usuario si desea añadir o eliminar un alimento. Para ello, si la lista de la compra contiene alimentos, éstos se muestran en pantalla junto con el importe total en euros de todos ellos; en caso de que la lista de la compra esté vacía, se informa al usuario de que no dispone de alimentos almacenados en la lista de la compra, y solo se le da la opción de añadir nuevos alimentos.

5.1.2.2 Diagrama de Flujo y Arquitectura

La Figura 22 ilustra el diagrama de flujo entre las diferentes clases de la aplicación

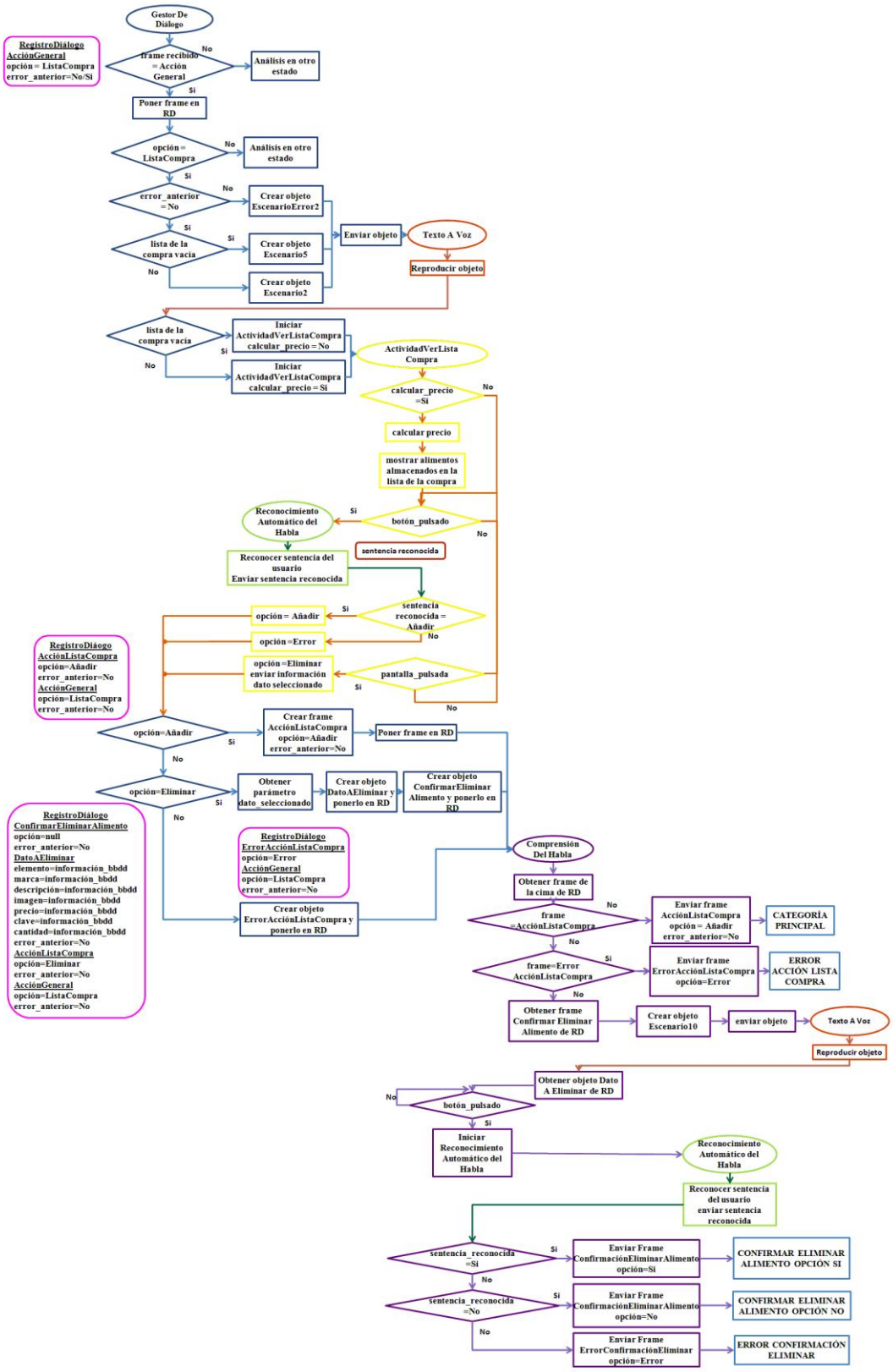

Figura 22: Diagrama de Flujo Submódulo Añadir Eliminar Alimentos.

- La actividad Gestor de Diálogo, lo primero que hace es determinar si el usuario se encuentra en este estado porque anteriormente ha indicado que desea iniciar la lista de la compra, o bien porque anteriormente ha especificado que desea añadir un alimento y se ha producido algún tipo de error en el reconocimiento. Para ello analiza el campo error_anterior del frame AccionGeneral:

  - → Si el campo error_anterior contiene el valor *"Si"*: genera un objeto de tipo EscenarioError2 que informa al usuario de que se ha producido un error en el reconocimiento y que debe volver a iniciar la opción que desea.
  - → Si el campo error_anterior contiene el valor "*No"*: comprueba el número de alimentos almacenados en BaseDatosUsuarios llamando al método *obtenerNumeroElementos()*, que hace uso del método *getCount()* de la clase Cursor que proporciona Android, y que devuelve el número de elementos almacenados en una base de datos.

```
int elementos =GestorDialogo.bdu.obtenerNumeroElementos();
public int obtenerNumeroElementos(){
Cursor c = new Cursor();
c = sqldb.query(TABLE_LISTA, new String[] {ID,ELEMENTO, MARCA,
DESCRIPCION, IMAGEN, PRECIO,CANTIDAD_A}, null, null, null, null, null);
if(c.getCount()==0){return 0;}
else{return 1;}}
```

  En función del número de alimentos almacenados, la actividad Gestor de Diálogo procede del siguiente modo:

  - → Si hay alimentos almacenados: construye un objeto de tipo Escenario2 que da al usuario la opción de añadir alimentos o la opción de eliminar algún alimento almacenados previamente, para lo cuál le indica que pulse sobre la pantalla el alimento que desea eliminar.
  - → Si no hay alimentos almacenados: construye un mensaje de tipo Escenario5 para informar al usuario de que la lista de la compra está vacía y darle la opción de añadir nuevos alimentos.

  En ambos casos, el objeto escenario se envía a la actividad Texto a Voz.

- La actividad Texto a Voz reproduce el objeto recibido y devuelve el hilo de la ejecución al Gestor de Diálogo.

- A continuación la actividad Gestor de Dialogo, inicia la Actividad Ver Lista Compra, pasando como parámetro el objeto calcular_precio, para indicar si debe o no calcular el precio de los alimentos, en función de si la lista de la compra está vacía o no.

- La Actividad Ver Lista Compra, analiza el parámetro recibido y en el caso de que contenga el valor *"Si"*, calcula el precio de los alimentos invocando al método *obtenerPrecioAlimentos()* de la clase Base de Datos Usuarios.

```
float precio = GestorDialogo.bdu.obtenerPrecioAlimentos();
```

Dicho método, obtiene las columnas precio y cantidad, asociadas a cada alimento almacenado en la base de datos y con estos valores calcula el precio final de la lista de la compra.

Una vez calculado el precio, éste es mostrado en pantalla, junto con la información de todos los alimentos guardados hasta el momento. Para ello, como se describe en el siguiente fragmento de código, la Actividad Ver Lista Compra invoca al método *obtenerTodosLosElementos()* de la clase Base de Datos Usuarios, que devuelve todas las columnas que contienen información referente a los alimentos almacenados. Esta información se almacena en un vector de objetos DatosAComprar.

Una vez obtenida la información, el método *VerBaseDatos()* instancia un objeto de la clase AdaptadorVerListaCompra que como ya hemos comentado anteriormente, la clase Adapter proporciona una interfaz común para cada modelo de datos. El objeto adaptador, se asocia al elemento gráfico lista, de tipo List View.

```
Vector<DatosAComprar> vector = new Vector();
private ListView lista;
lista=     (ListView)findViewById(R.id.listaDatosCompra);
Cursor c =GestorDialogo.bdu.obtenerTodosLosElementos();
if(c.moveToFirst()){do{
verbase_clave=c.getString(0);
verbase_elemento = c.getString(1);
verbase_marca = c.getString(2);
```

Con todo ello, la apariencia final de la aplicación es la mostrada en la Figura 23.

Figura 23: Elementos gráficos submódulo Añadir Eliminar Alimentos

Una vez los alimentos aparecen en pantalla, el usuario puede pulsar tanto el botón de voz para indicar que desea añadir un alimento, o la pantalla, para seleccionar el alimento que desea eliminar.

Para ello, tanto en la lista de datos como en el botón de voz, se registran escuchadores de eventos.

```
lista.setOnItemClickListener(new OnItemClickListener(){
public void onItemClick(AdapterView<?> a, View v,int position, long id){
dac = (DatosAComprar)a.getItemAtPosition(position);
dae = new DatoAEliminar(dac.getElemento(), dac.getMarca(),
dac.getDescripcion(), dac.getImagen(),
dac.getPrecio(),dac.getClave(),dac.getCantidad());}});
micro.setOnClickListener(new OnClickListener(){
public void onClick(View arg1){
```

- → Si se pulsa el botón de voz, se inicia la actividad Reconocimiento Automático Del Habla,
- → Si se pulsa sobre un elemento de la lista, se obtiene la información relativa al alimento seleccionado y se genera un objeto opción con el valor "*Eliminar*" como atributo

❖ En el caso de que el usuario desee añadir un alimento, la actividad Reconocimiento Automático Del Habla, reconoce la sentencia pronunciada por el usuario y envía el resultado a la Actividad Ver Lista Compra.

❖ La Actividad Ver Lista Compra, analiza dicha sentencia procediendo del siguiente modo:

- → Si la sentencia reconocida contiene la palabra clave "*Añadir*": se almacena dicho término en el objeto opción.
- → Si la sentencia reconocida contiene la palabra clave *"Error":* se almacena dicho término en el objeto opción.

El objeto opción generado (que recordemos que en el caso de que el usuario haya decidido eliminar el alimento contiene el valor "*Eliminar"*), es enviado al Gestor de Diálogo.

❖ El Gestor de Diálogo analiza el valor del objeto opción, plasmando la elección del usuario en forma de frame, que introducirá en la cima del Registro de Diálogo:

- → Si opción contiene el valor "Añadir": genera el frame AcciónListaCompra con el valor "*Añadir"* en el campo opción
- → Si opción contiene el valor "*Error"*: genera el frame ErrorAcciónListaCompra
- → Si opción contiene el valor *"Eliminar*": genera los frames DatoAEliminar con la información del alimento que se desea eliminar y el frame ConfirmarEliminar Alimento con el valor "*Eliminar"* en el campo opción.

- La actividad Comprensión del Habla, obtiene el frame de la cima del Registro Diálogo y en base a él genera un nuevo frame que enviará a la actividad Gestor de Diálogo y determinará el siguiente submódulo de ejecución:

  - → Si el frame de la cima del Registro de Diálogo es AcciónListaCompra: genera el frame AcciónListaCompra con el valor "*Añadir*" en el campo opción, por lo que el siguiente submódulo que ejecutara el Gestor de Diálogo será el Submódulo Categoría Principal
  - → Si el frame de la cima del Registro de Diálogo es ErrorAcciónListaCompra: genera el frame ErrorAcciónListaCompra con el valor "*Error"* en el campo opción, por lo que el siguiente submódulo que ejecutara el Gestor de Diálogo será el submódulo Error Acción Lista Compra.

  En el caso de que el frame obtenido sea ConfirmarEliminarAlimento, la actividad lleva a cabo una función extra, que es la de solicitar al usuario que confirme si el dato que ha pulsado para eliminar es correcto. Para ello genera un objeto de tipo Escenario10 que solicita al usuario que confirme la eliminación del alimento.

- La actividad Texto a Voz reproduce el objeto generado por la clase anterior, y cuando finaliza el devuelve el control de la aplicación a la actividad Comprensión del Habla.

- La actividad Comprensión del Habla, haciendo uso de la información almacenada en el frame DatoAEliminar, muestra la información del alimento como puede apreciarse en la Figura 24.

Figura 24: Elementos gráficos para eliminar el alimento

Cuando el usuario pulsa el botón de voz, que como hemos comentado anteriormente lleva registrado un escuchador de eventos, se inicia la actividad Reconocimiento Automático del Habla.

- La actividad Reconocimiento Automático del Habla reconoce la sentencia pronunciada por el usuario y devuelve el resultado a la actividad Comprensión del Habla

- Finalmente, como en casos anteriores, la actividad Comprensión del Habla analiza la sentencia reconocida, generando el correspondiente frame que enviará al Gestor de Diálogo y que determinará el siguiente submódulo de ejecución:

  - → Si sentencia_reconocida contiene el valor "*Si*": genera el frame Confirmar Eliminar Alimento con el valor "*Si*" en el campo opción, por lo que el siguiente submódulo que ejecutará el Gestor de Diálogo será Confirmar Eliminar Alimento Opción Sí
  - → Si sentencia_reconocida contiene el valor "*No*": genera el frame Confirmar Eliminar Alimento con el valor "*No*" en el campo opción, por lo que el siguiente submódulo que ejecutará el Gestor de Diálogo será Confirmar Eliminar Alimento Opción No
  - → Si sentencia_reconocida no contiene ninguna de las opciones anteriores: genera el frame ErrorConfirmaciónEliminar, por lo que el siguiente submódulo que ejecutará el Gestor de Diálogo será Error Confirmación Eliminar

5.1.3 Submódulo Categoría Principal

5.1.3.1 Funcionalidad

La ejecución de este submódulo se realiza cuando el Gestor de Diálogo recibe el frame AcciónListaCompra con la palabra clave *"Añadir"* en el campo opción.

La funcionalidad principal es la de mostrar las diferentes categorías principales de alimentos, de manera que el usuario pueda visualizarlas en pantalla y seleccionar la correspondiente al alimento que desea elegir.

5.1.3.2 Diagrama de Flujo y Arquitectura

La Figura 25, muestra el diagrama de flujo definido para este submódulo.

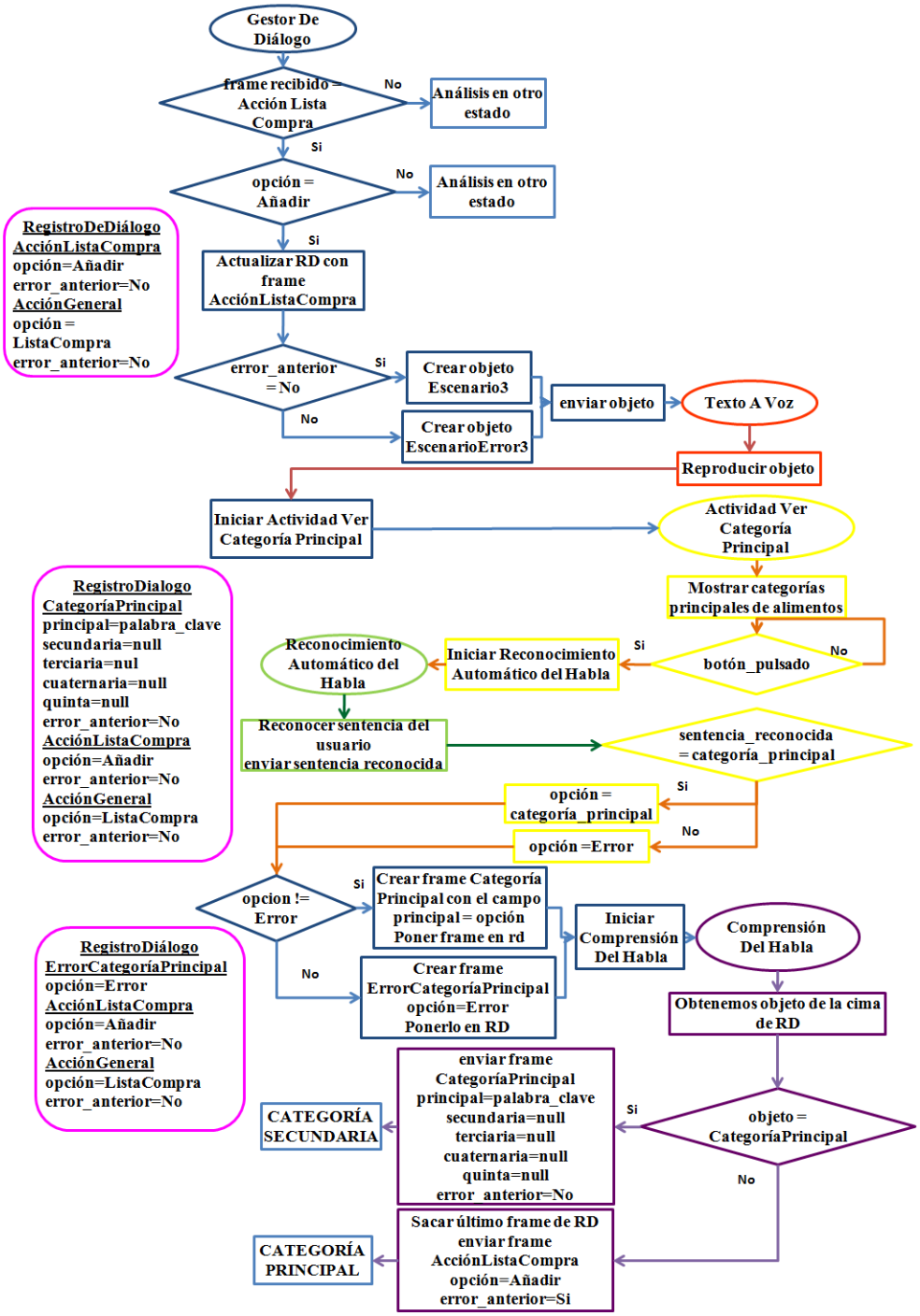

Figura 25: Diagrama de Flujo Submódulo Categoría Principal

- La actividad Gestor de Diálogo, cuando recibe el frame AcciónListaCompra, debe determinar si el usuario se encuentra en este submódulo porque proviene del submódulo añadir/eliminar y ha especificado que desea añadir el alimento, o bien porque ya estaba en la ejecución de este submódulo y se ha producido algún tipo de error en el reconocimiento de la categoría principal seleccionada.

Para ello, la actividad Gestor de Diálogo, analiza el campo error_anterior del frame AccionGeneral, procediendo del siguiente modo:

- → Si error_anterior=No: el gestor construye un objeto de tipo Escenario3, que indica al usuario que debe especificar una de las categorías principales.
- → Si error_anterior=Si: el gestor construye un objeto de tipo EscenarioError3 que informa al usuario de que se ha producido un error en el reconocimiento de la categoría principal y que debe volver a indicarla.

El objeto escenario, es enviado a la actividad Texto A Voz

- ❖ La actividad Texto a Voz, reproduce el objeto escenario, y cuando finaliza, devuelve el hilo de la ejecución al Gestor de Diálogo.

- ❖ A continuación, el Gestor de Diálogo, inicia la Actividad Ver Categoría Principal

- ❖ La Actividad Ver Categoría Principal, muestra las categorías principales de alimentos definidas para esta aplicación.

```
Cursor c =GestorDialogo.bdh.obtenerCategoriaPrincipal();
if(c.getCount()!=0){if(c.moveToFirst()){do{
verCategoria=c.getString(1);
vector.add(verCategoria);
}while(c.moveToNext());}
AdaptadorCategoriaPrincipal adaptador = new
AdaptadorCategoriaPrincipal(this, vector);
lista.setAdapter(adaptador);
```

Como puede apreciarse, la actividad llama al método *obtenerCategoríaPrincipal()*, de BaseDatosHelper, que devuelve en forma de Cursor los nombres de las diferentes categorías principales. Una vez obtenidas, haciendo uso de un objeto ArrayAdapter, denominado AdaptadorCategoríaPrincipal y del elemento gráfico listView denominado lista, las categorías principales se muestran en pantalla, como refleja la Figura 26

Figura 26: Elementos gráficos submódulo Categoría Principal

Cuando el usuario pulsa el botón de voz, la actividad captura el evento, dando paso a la actividad Reconocimiento Automático Del Habla

- La Actividad Reconocimiento Automático del Habla, reconoce la sentencia del usuario y envía el resultado a la clase Actividad Ver Categoría Principal

- La Actividad Ver Categoría Principal, analiza la sentencia reconocida, procediendo del siguiente modo:

  - → Si sentencia_reconocida contiene una de las categorías principales, construye un objeto opción, y almacena en el dicha categoría principal
  - → Si sentencia_reconocida no contiene ninguna de las categorías principales, construye un objeto opción y establece el valor *"Error"* como atributo.

  En ambos casos, el objeto opción es enviado a la actividad Gestor de Diálogo.

- La actividad Gestor de Diálogo, analiza el objeto opción, y en función de su valor, elabora un frame que dará información a la actividad Comprensión del Habla de lo sucedido en el diálogo con el usuario:

  - → Si opción no contiene el valor "*Error"*: genera el frame CategoríaPrincipal con la categoría principal elegida por el usuario en el campo principal
  - → Si opción contiene el valor "*Error*": genera el frame ErrorCategoríaPrincipal con el valor "*Error*" en el campo opción.

- Una vez generado el frame, lo introduce en el Registro de Diálogo, e inicia la actividad Comprensión del Habla

- La actividad Comprensión del Habla, obtiene el frame de la cima del Registro de Diálogo y en base a ello elaborará un determinado frame, que enviará al Gestor de Diálogo, que determinará el siguiente submódulo de ejecución.

  - → Si el frame de la cima del Registro de Diálogo es CategoríaPrincipal: genera el frame CategoríaPrincipal con el valor de la categoría principal en el campo principal, por lo que el siguiente submódulo de ejecución será Categoría Secundaria
  - → Si el frame de la cima del Registro de Diálogo es ErrorCategoríaPrincipal genera el frame AcciónListaCompra con el valor *"Añadir"* en el campo opción, por lo que el siguiente submódulo de ejecución será Categoría Principal

5.1.4 Submódulo Categoría Secundaria

5.1.4.1 Funcionalidad

Este submódulo se ejecuta cuando el Gestor de Diálogo recibe el frame CategoríaPrincipal con únicamente los campos principal y error_anterior rellenos.

Su finalidad es la de mostrar por pantalla las distintas categorías secundarias englobadas dentro de la categoría principal seleccionada en el anterior submódulo y obtener cuál de ellas es la seleccionada por el usuario.

5.1.4.2 Arquitectura y Diagrama de Flujo

La Figura 27 muestra el diagrama de flujo definido para este submódulo.

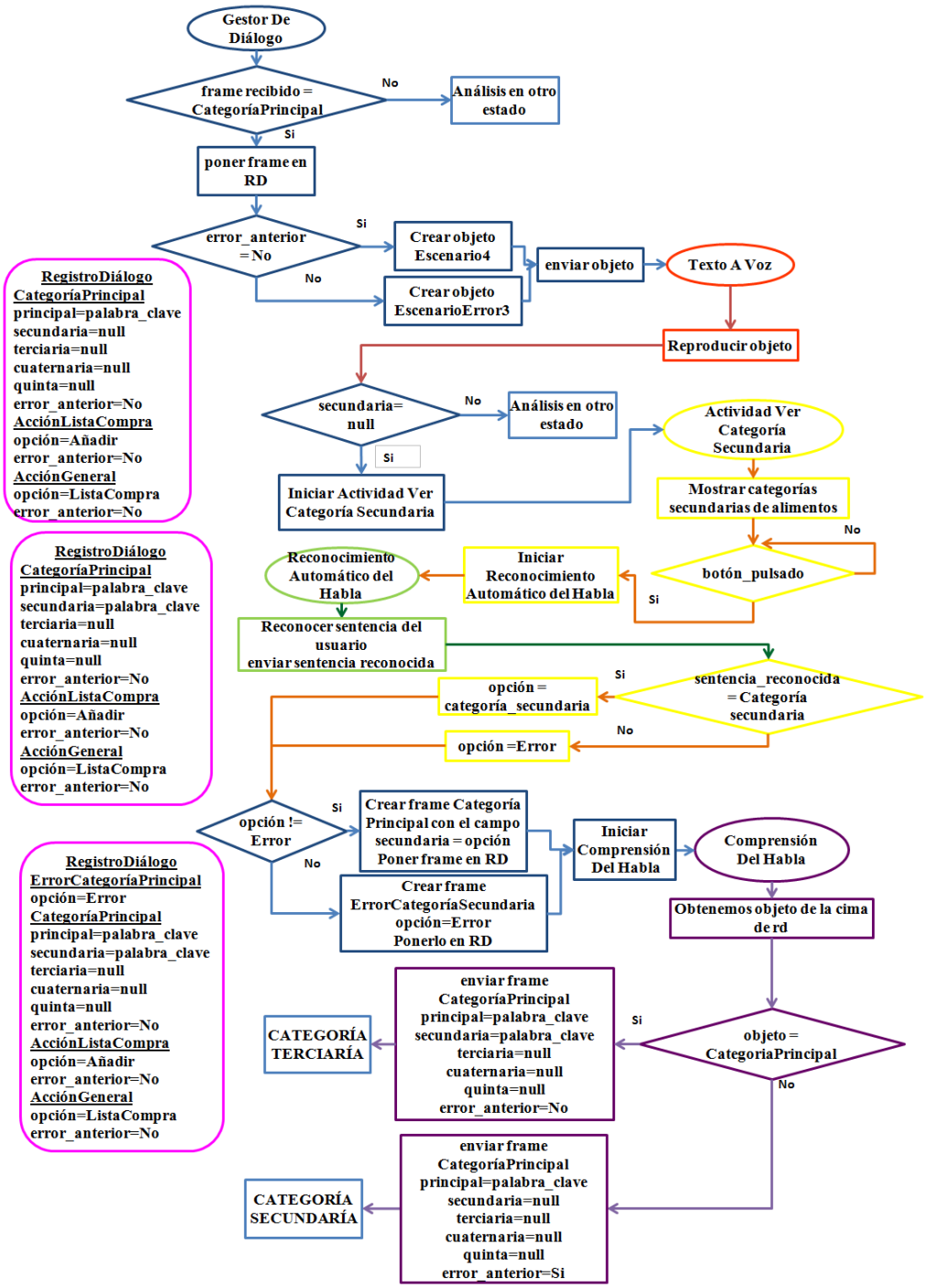

Figura 27: Diagrama de Flujo Submódulo Categoría Secundaria

- Al igual que sucedía en submódulos anteriores, el Gestor de Diálogo debe determinar si el usuario se encuentra en este submódulo porque anteriormente estaba en el submódulo categoría principal o porque ya estaba en este submódulo y se ha producido un error en el reconocimiento de la categoría secundaria que ha indicado.

  Para ello analiza el valor del campo error_anterior del frame CategoríaPrincipal, procediendo del siguiente modo:

- → Si error_anterior = No: construye un objeto de tipo Escenario4 que informa al usuario de que indique una de las categorías secundarias
- → Si error_anterior=Si: construye un objeto de tipo EscenarioError3, que informa al usuario de que se ha producido un error en el reconocimiento, por lo que debe volver a indicar la categoría secundaría que desea.

El objeto generado en ambos casos, es enviado a la actividad TextoAVoz

- ❖ La actividad TextoAVoz, reproduce el objeto recibido y cuando finaliza devuelve el control de la ejecución a la actividad Gestor de Diálogo.
- ❖ Seguidamente, el Gestor de Diálogo inicia la Actividad Ver Categoría Secundaria
- ❖ La Actividad Ver Categoría Secundaria, muestra las categorías secundarías de alimentos englobadas dentro de la categoría principal seleccionada por el usuario.

```
Cursor c =GestorDialogo.bdh.obtenerCategoriaSecundaria(principal);
if(c.moveToFirst()){do{
verCategoria=c.getString(1);
vector.add(verCategoria);
}while(c.moveToNext());}
AdaptadorCategoriaPrincipal adaptador = new
AdaptadorCategoriaPrincipal(this, vector);
lista.setAdapter(adaptador);
```

Para ello, como puede apreciarse en el fragmento de código anterior, la actividad hace uso del método *obtenerCategoríaSecundaria(principal)* de la clase BaseDatosHelper, que recibe como parámetro la categoría principal elegida por el usuario (que se obtiene del frame CategoríaPrincipal) y devuelve en forma de Cursor las categorías secundarias correspondientes. A continuación, la actividad muestra la información en pantalla mediante los objetos AdaptadorCategoríaPrincipal y lista, como ilustra la Figura 28.

Figura 28: Elementos gráficos submódulo Categoría Secundaria

Cuando el usuario pulsa sobre el botón de voz, la actividad captura el evento e inicia la actividad Reconocimiento Automático del Habla

- La actividad Reconocimiento Automático del Habla, reconoce la sentencia pronunciada por el usuario y la envía a la Actividad Ver Categoría Secundaria.

- La Actividad Ver Categoría Secundaria, analiza la sentencia reconocida y procede del siguiente modo:

  - → Si sentencia_reconocida contiene una de las categorías segundarías definidas para la aplicación, elabora un objeto opción que contiene como atributo dicha categoría secundaria.
  - → Si sentencia_reconocida no contiene ninguna de las categorías secundarías defnidas para la aplicación, elabora un objeto opción, con el valor *"Error*" como atributo.

  En ambos casos el objeto opción es enviado a la actividad Gestor de Diálogo.

- La actividad Gestor de Diálogo recibe el objeto opción y en base a él construye un determinado frame, que colocará en la cima del Registro de Diálogo y que dará a la actividad Comprensión Del Habla información sobre el diálogo

  - → Si opción es distinto de *"Error":* construye el frame CategoríaPrincipal con la categoría secundaria seleccionada por el usuario en el campo segunda.
  - → Si opción contiene el valor "*Error"*: construye el frame ErrorCategoríaSecundaría con el valor "*Error*" en el campo opción.

- La actividad Comprensión del Habla obtiene el frame de la cima del Registro de Diálogo y en función de su contenido, genera un nuevo frame que enviará al Gestor, quién determinará cuál será el siguiente submódulo a ejecutar.

  - → Si el frame de la cima del Registro de Diálogo es CategoríaPrincipal: genera el frame CategoríaPrincipal con la categoría secundaría especificada por el usuario en el campo secundaría, por lo que el siguiente submódulo en ejecutarse será el submódulo CategoríaTerciaria
  - → Si el frame de la cima del Registro de Diálogo es ErrorCategoríaSecundaría: genera el frame CategoríaPrincipal con el valor secundaria vacío, por lo que el siguiente submódulo en ejecutarse será el submódulo Categoría Secundaria.

5.1.5 Submódulo Categoría Terciaria

5.1.5.1 Funcionalidad

Este submódulo se ejecuta cuando el Gestor de Diálogo recibe el frame CategoríaPrincipal con los campos principal, secundaría, y error_anterior rellenos.

Su misión es la de mostrar por pantalla las categorías terciarias correspondientes a la categoría secundaria seleccionada por el usuario en el anterior turno de diálogo. La funcionalidad de este submódulo es similar a la de los dos anteriores, salvo por un detalle: hay determinados alimentos que

únicamente poseen categoría primaria y secundaría, y por tanto, la aplicación deberá notificar estos casos, para que el siguiente submódulo de ejecución consista en mostrar la información relativa a estos alimentos en pantalla.

En el caso de que el alimento elegido por el usuario sí que posea categoría terciaría, este submódulo procederá a mostrar en pantalla las correspondientes categorías terciarias, y determinar cuál de ellas es la seleccionada por el usuario.

5.1.5.2 Arquitectura y Diagrama de Flujo

La Figura 29 muestra el diagrama de flujo definido para este submódulo.

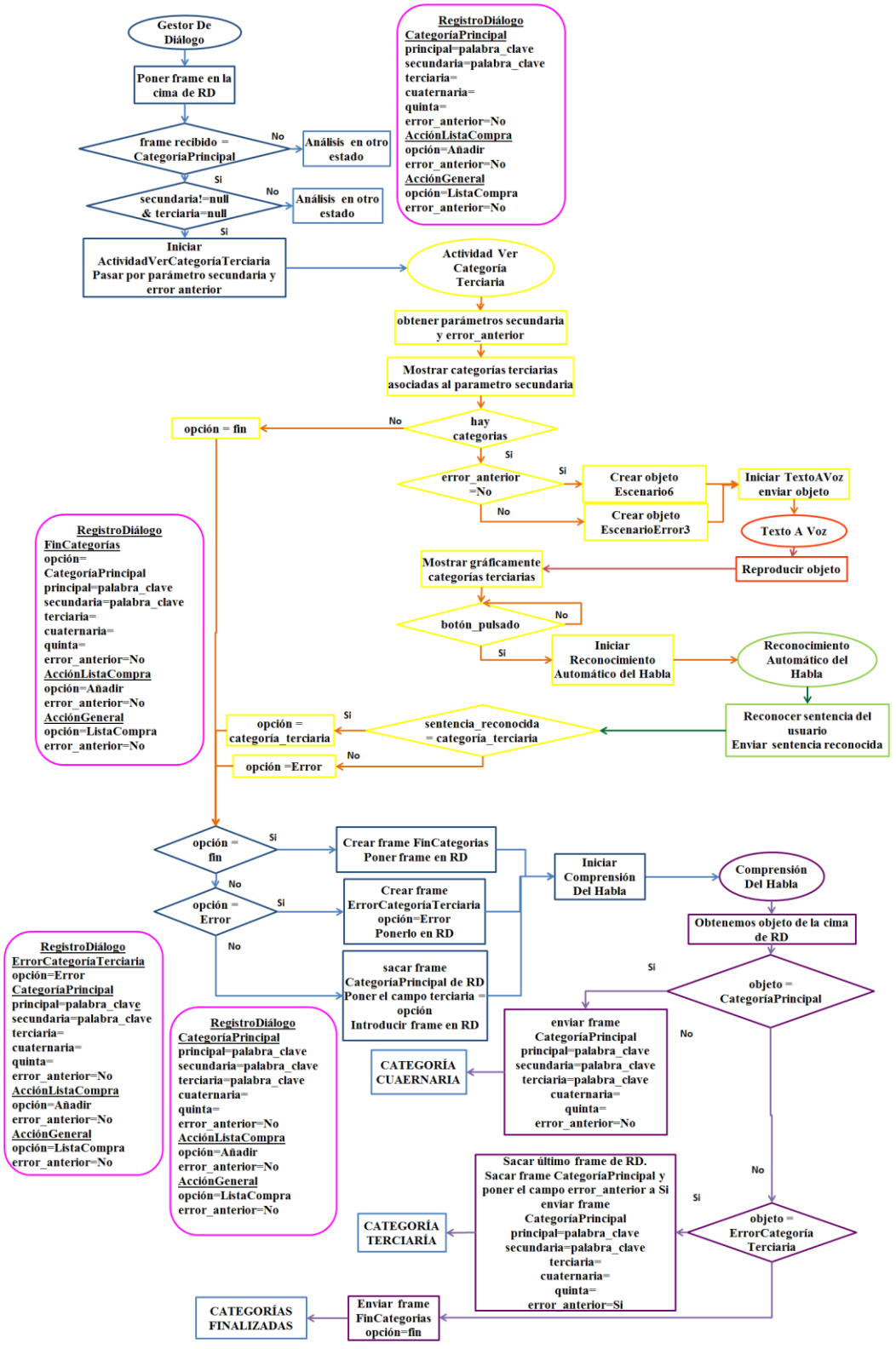

Figura 29: Diagrama de Flujo Submódulo Categoría Terciaria

- El Gestor de Diálogo, cuando recibe el frame CategoríaPrincipal, inicia la Actividad Ver Categoría Terciaria, pasando como parámetros los campos primaria y secundaria recibidos en el frame

- La Actividad Ver Categoría Terciaria, como puede verse en el siguiente fragmento de código, ejecuta el método *obtenerCategoríaTerciaria(String secundaria*) de la clase Base de Datos Helper, que recibe como parámetro un string que corresponde con una determinada categoría secundaria y devuelve en forma de objeto Cursor las correspondientes categorías terciarías asociadas a dicha categoría secundaria.

```
Cursor c =GestorDialogo.bdh.obtenerCategoriaTerciaria(principal);
```

  - → Si el cursor está vacío: no hay más categorías definidas para esa categoría secundaría y lo siguiente que deberá hacer la aplicación es mostrar la información de los alimentos. Para ello la Actividad Ver Categoría Terciaria crea el objeto opción con el valor *"Fin"* como atributo.
  - → Si el cursor no está vacío, significa que existen categorías terciarias que mostrar. Para ello, igual que sucedía en casos anteriores, la actividad debe determinar si el usuario en el anterior turno de diálogo se encontraba en el submódulo categoría secundaría o por el contrario se encontraba en este submódulo y especificó una categoría terciaria que dio lugar a un error en el reconocimiento. Para ello analiza el valor del parámetro error_anterior recibido en el intent.:
    - Si error_anterior = No: crea un objeto de tipo Escenario 6 que informa al usuario de que debe especificar una de las categorías terciarias.
    - Si error_anterior = Si: crea un objeto de tipo EscenarioError3 que informa al usuario de que se ha producido un error en el reconocimiento y que debe volver a indicar la categoría terciaria que desee.

  En ambos casos el objeto generado es enviado a la actividad TextoAVoz.

- La actividad TextoAVoz reproduce el objeto escenario recibido y cuando finaliza devuelve el control a la Actividad Ver Categoría Terciaria

- La Actividad Ver Categoría Terciaria, mediante los objetos AdaptadorCategoríaPrincipal y lista, muestra las categorías terciarias obtenidas anteriormente como se ilustra en la Figura 30.

Figura 30: Elementos gráficos Submódulo Categoría Terciaria

Cuando el usuario pulsa el botón de voz, la actividad captura el evento y lanza la actividad Reconocimiento Automático del Habla.

- La Actividad Reconocimiento Automático del Habla, reconoce la sentencia pronunciada por el usuario y envía el resultado a la Actividad Ver Categoría Terciaria

  - → Si sentencia_reconocida se corresponde con alguna de las categorías terciarias definidas para la aplicación, genera un objeto opción que contiene como atributo dicho valor
  - → Si sentencia_reconocida no se corresponde con ninguna de las categorías terciarias definidas para la aplicación, genera un objeto opción que contiene como atributo el valor "*Error"*

  En los diferentes casos, el objeto opción (que recordemos que en el caso de que no haya categorías terciarias que mostrar contiene el valor "*Fin"*) es enviado al Gestor de Diálogo.

- El Gestor de Dialogo analiza el objeto opción y en base a él construye el correspondiente frame que informará a la actividad Comprensión del Habla de lo sucedido en el turno de diálogo.

- → Si opción contiene el valor "*Fin*": genera el frame FinCategorías
- → Si opción contiene el valor "*Error*": genera el frame ErrorCategoríaTerciaria
- → Si opción contiene una de las categorías terciarias definidas en la aplicación: genera el frame CategoríaPrincipal con el valor de la categoría terciaria en el campo terciaria.

El frame generado se introduce en el Registro de Diálogo y se inicia la actividad Comprensión del Habla

- ❖ La actividad Comprensión del Habla obtiene el frame de la cima del Registro de Diálogo y en función de su contenido elabora un nuevo frame que enviará al Gestor y que determinará el siguiente submódulo de ejecución

  - → Si el frame de la cima del Registro de Diálogo es FinCategorías: genera el frame FinCategorías con el valor "*Si*" en el campo opción, con lo que el siguiente submódulo de ejecución será Categorías Finalizada
  - → Si el frame de la cima del Registro de Diálogo es CategoríaPrincipal: genera el frame CategoríaPrincipal con el valor de la categoría terciaria en el campo terciaria, con lo que el siguiente submódulo de ejecución será Categoría Cuaternaria
  - → Si el frame de la cima del Registro de Diálogo es ErrorCategoríaTerciaria: genera el frame CategoríaPrincipal con el campo terciaría vació, con lo que el siguiente submódulo de ejecución será Categoría Terciaria.

5.1.6 Submódulo Categoría Cuaternaria

5.1.6.1 Funcionalidad

El submódulo categoría cuaternaria se ejecuta cuando el Gestor de Diálogo recibe el frame CategoríaPrincipal con los campos principal, secundaría, terciaría y error_anterior rellenos. Su función consiste en mostrar por pantalla las distintas categorías cuaternarias asociadas a la categoría terciaria que el usuario seleccionó en el anterior turno de diálogo y determinar cuáles de ellas, es la que se corresponde con el alimento que se desea añadir.

La funcionalidad de este submódulo es bastante similar a la del submódulo obtener categoría terciaria debido a que hay alimentos que únicamente tienen categoría principal, secundaria y terciaria. En estos casos la aplicación deberá mostrar directamente estos alimentos en pantalla.

5.1.6.2 Arquitectura y Diagrama de Flujo

La Figura 31 muestra el diagrama de flujo definido para este submódulo.

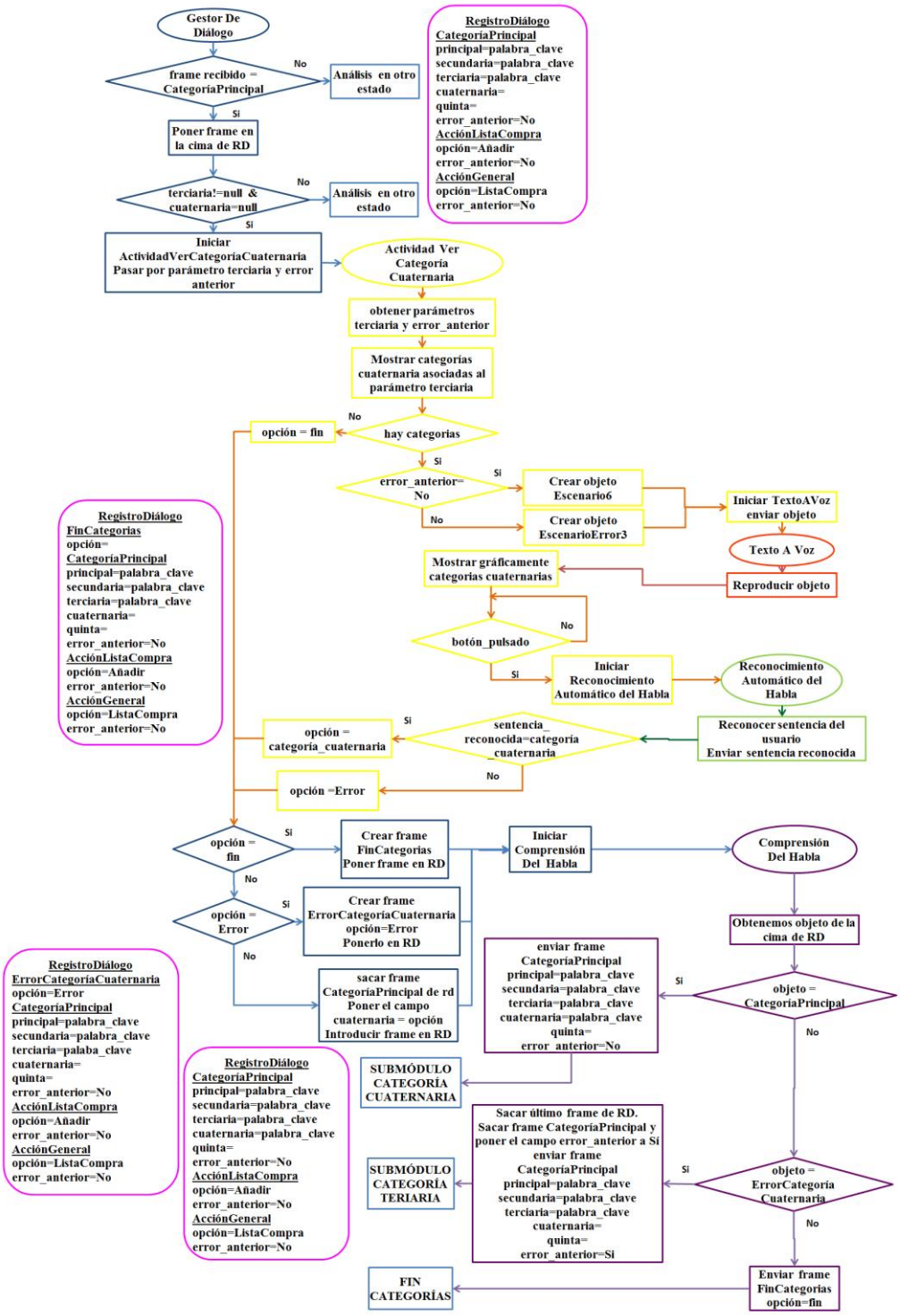

Figura 31: Diagrama de Flujo Submódulo Categoría Cuaternaria

- El Gestor de Diálogo, cuando recibe el frame CategoríaPrincipal, inicia la Actividad Ver Categoría Cuaternaria, pasando como parámetros los valores de los campos terciaria y error_anterior.

- La Actividad Ver Categoría Cuaternaria ejecuta el método *obtenerCategoríaCuaternaria()* de la clase Base de Datos Helper pasando por parámetro la categoría terciaria recibida en el intent. El método devuelve en

forma de objeto Cursor las categorías cuaternarias correspondientes a la categoría terciaria que se pasó como parámetro.

- La Actividad Ver Categoría Cuaternaria procede de la siguiente manera:
  - → Si el cursor está vacío: no hay categorías cuaternarias por lo que en el siguiente paso se debe mostrar la información de los alimentos correspondientes a la categoría terciaria recibida en el Intent. En este caso la actividad crea el objeto opción, con el valor "*Fin"* como atributo
  - → Si el Cursor no está vacío; hay categorías cuaternarias que mostrar. Antes de mostrarlas , la actividad determina si en el anterior turno de diálogo el usuario se encontraba en el submódulo categoría terciaria o por el contrario se encontraba dentro de este submódulo y se ha producido algún tipo de error en el reconocimiento de la categoría cuaternaria.
  - → Para ello la actividad analiza el valor del parámetro error_anterior recibido en el intent:
    - Si error_anterior = No: Crea un objeto de tipo Escenario6 que informa al usuario de que debe especificar una de las categorías cuaternarias
    - Si error_anterior=Si: crea un objeto de tipo EscenarioError3 que informa al usuario de que debe indicar nuevamente la categoría ya que se ha producido un error en el reconocimiento.

En ambos casos el objeto escenario es enviado a la actividad Texto A Voz

- La actividad Texto A Voz reproduce la sentencia recibida y cuando finaliza devuelve el control a la Actividad Ver Categoría Cuaternaria.
- Seguidamente la Actividad Ver Categoría Cuaternaria, como en submódulos anteriores muestra las categorías en pantalla mediante los objetos AdaptadorCategoríaPrincipal y listaCategoríaPrincipal.

```
if(c.moveToFirst()){do{
verCategoria=c.getString(1);
vector.add(verCategoria);
}while(c.moveToNext());}
AdaptadorCategoriaPrincipal adaptador = new
AdaptadorCategoriaPrincipal(this, vector);
lista.setAdapter(adaptador);
```

La apariencia es la mostrada en la Figura 32.

Figura 32: Elementos gráficos Submódulo Categoría Cuaternaria

Cuando el usuario pulsa el botón de voz, la actividad lanza la actividad de Reconocimiento Automático Del Habla

- La Actividad Reconocimiento Automático Del Habla reconoce la sentencia expresada por el usuario y devuelve el resultado a la Actividad Ver Categoría Cuaternaria.

- La Actividad Ver Categoría Cuaternaria analiza la sentencia reconocida:

  - → Si sentencia_reconocida no se corresponde con ninguna de las categorías cuaternarias, se almacena el valor *"Error*" en el objeto opción.
  - → Si sentencia_reconocida sí se corresponde con alguna de las categorías cuaternarias se almacena dicho valor en el objeto opción

  El objeto opción (que contiene el valor "*Fin"* en el caso de que no hay categorías cuaternarias) se envía al Gestor.

- El Gestor de Diálogo, analiza el objeto opción, procediendo del siguiente modo:

  - → Si opción contiene el valor "*Fin*": genera el frame FinCategorías.
  - → Si opción contiene el valor "*Error*": genera el frame ErrorCategoríaCuaternaria
  - → Si opción contiene una de las categorías cuaternarias definidas para la aplicación genera el frame CategoríaPrincipal con esa categoría en el campo cuaternaria.

- La actividad Comprensión del Habla obtiene el frame de la cima del Registro de Dialogo y en función del frame obtenido generará un nuevo frame con la finalidad de determinar el siguiente submódulo de ejecución:

- → Si el frame de la cima del Registro de Diálogo es FinCategorías genera el frame FinCategorías con el valor "*Sí*" en el campo categoría_finalizada, por lo que el siguiente submódulo de ejecución será Categorías Finalizadas
- → Si el frame de la cima del Registro de Diálogo es CategoríaPrincipal genera el frame CategoríaPrincipal con el valor de la categoría cuaternaria en el campo cuaternaria, por lo que el siguiente submódulo de ejecución será Categoría Quinta
- → Si el frame de la cima del Registro de Diálogo es ErrorCategoríaCuaternaría genera el frame CategoríaCuaternaria con el valor cuaternaria vacio y el valor "*Sí*" en el campo error_anterior, por lo que el siguiente submódulo de ejecución será Categoría Cuaternaria.

5.1.7 Submódulo Categoría Quinta

5.1.7.1 Funcionalidad

Este submódulo se ejecuta cuando el Gestor de Diálogo recibe el frame CategoríaPrincipal con los campos principal, secundaría, terciaria, cuaternaria y error_anterior rellenos.

Se encarga de determinar la categoría quinta dentro de la cual se engloba el alimento que el usuario desea añadir.

Debido a que hay alimentos que solamente poseen categoría cuaternaria, este submódulo deberá detectarlos y mostrar la información relativa a estos alimentos o en caso contrario mostrar las categorías de quinto nivel, asociadas a la categoría cuaternaria especificada por el usuario en el anterior turno de diálogo.

5.1.7.2 Arquitectura y Diagrama de Flujo

La Figura 33 muestra el diagrama de flujo definido para este submódulo.

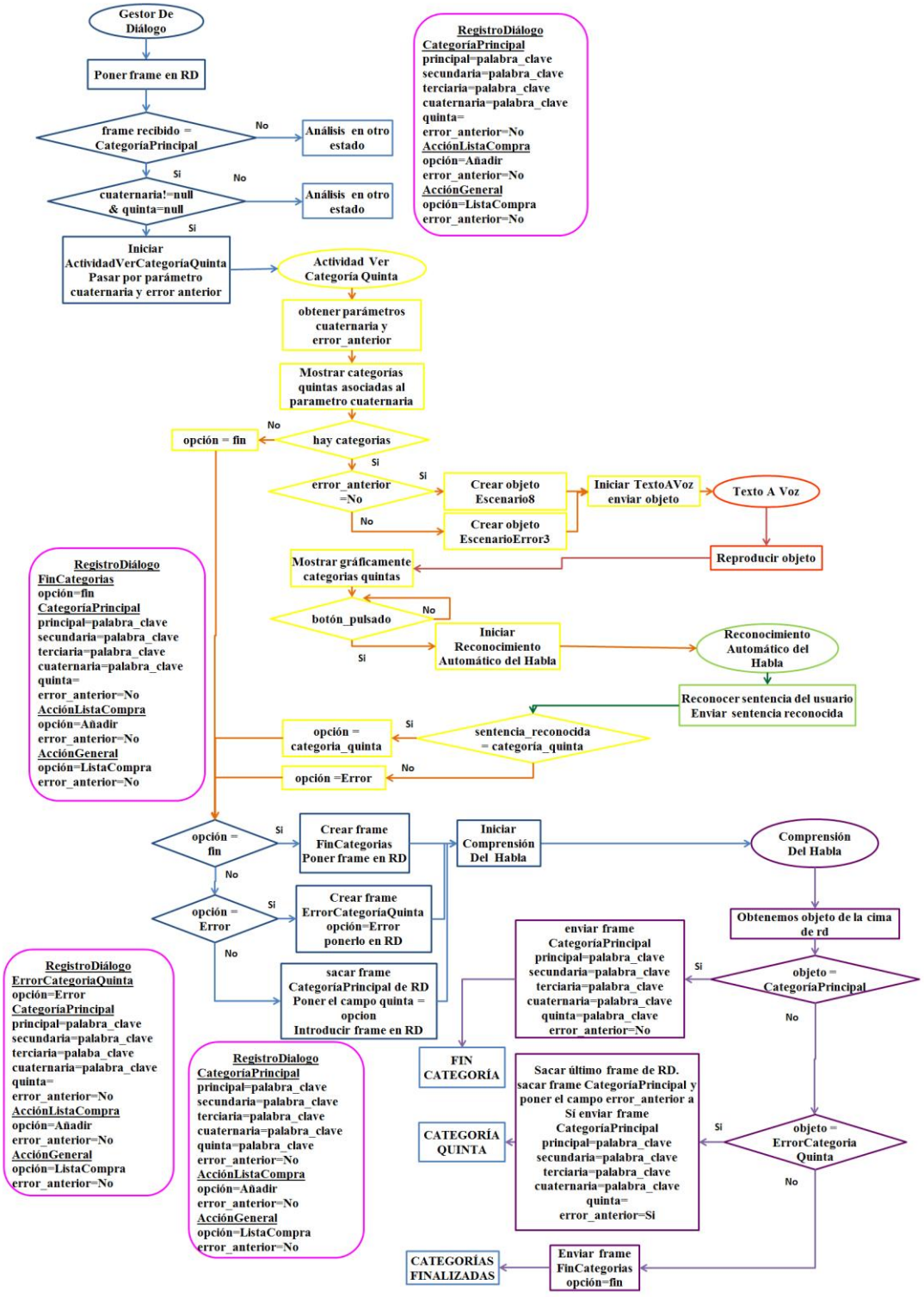

Figura 33: Diagrama de Flujo Submódulo Categoría Quinta.

- El Gestor de Dialogo cuando recibe el frame CategoríaPrincipal, con los valores en los campos anteriormente comentados, inicia la Actividad Ver Categoría Quinta pasando por parámetros los valores de los campos cuaternaria y error_anterior.

- La Actividad Ver Categoría Quinta, en un primer lugar, determina si la categoría cuaternaria recibida, tiene categorías de quinto nivel asociadas. Para ello ejecuta el método *obtenerCategoríaQuinta* de la clase Base de Datos Helper pasando por parámetro la categoría cuaternaria que recibió en el intent. Este método devuelve en forma de objeto Cursor las categorías quintas asociadas a la categoría cuaternaria. La Actividad Ver Categoría Quinta procede del siguiente modo:

  → Si el cursor está vacío: no hay categorías quintas por lo que en el siguiente paso se debe mostrar la información de los alimentos correspondientes a la categoría cuaternaria recibida en el Intent. En este caso la actividad crea el objeto opción, con el valor "*Fin*" como atributo
  → Si el Cursor no está vacío: La actividad procede a mostrar gráficamente las categorías quintas y a lanzar un mensaje al usuario en función del campo error_anterior recibido en el intent
    - Si error_anterior = No: Crea un objeto de tipo Escenario6 que informa al usuario de que debe especificar una de las categorías cuaternarias
    - Si error_anterior=Si: crea un objeto e tipo EscenarioError3 que informa al usuario de que debe indicar nuevamente la categoría ya que se ha producido un error en el reconocimiento.

En ambos casos el objeto escenario es enviado a la actividad Texto A Voz-

- La actividad Texto A Voz reproduce la sentencia recibida y cuando finaliza devuelve el control a la Actividad Ver Categoría Quinta.

- Seguidamente la Actividad Ver Categoría Quinta, como en submódulos anteriores muestra las categorías en pantalla mediante los objetos AdaptadorCategoríaPrincipal y listaCategoríaPrincipal, como se muestra en la Figura 34.

Figura 34: Elementos gráficos Submódulo Categoría Quinta

Cuando el usuario pulsa el botón de voz, la aplicación lanza la actividad de Reconocimiento Automático del Habla.

- La Actividad Reconocimiento Automático del Habla reconoce la categoría quinta que el usuario desea seleccionar y devuelve el resultado a la Actividad Ver Categoría Quinta.

- La Actividad Ver Categoría Quinta analiza la sentencia reconocida:

  → Si sentencia_reconocida no se corresponde con ninguna de las categorías quintas, se almacena el valor error en el objeto opción.
  → Si sentencia_reconocida sí se corresponde con alguna de las categorías quintas se almacena dicho valor en el objeto opción

El objeto opción (que contiene el valor "*Fin"* en el caso de que no haya categorías quintas) se envía al Gestor

- El Gestor de Diálogo analiza el objeto opción y en base a su contenido elabora un determinado frame que colocará en la cima del Registro de Diálogo y que proporcionará a la actividad Comprensión del Habla información acerca del diálogo.

  → Si opción contiene el valor "*Fin*": elabora el frame FinCategorías
  → Si opción contiene el valor "*Error*" elabora el frame ErrorCategoríaQuinta.
  → Si opción contiene el valor de una de las categorías quintas: elabora el frame CategoríaPrincipal con el valor de esa categoría en el campo quinta.

- La actividad Comprensión del Habla obtiene el frame de la cima del Registro De Dialogo y en función del frame obtenido generará un nuevo frame con la finalidad de determinar el siguiente submódulo de ejecución

  → Si el frame de la cima del Registro es FinCategorías: elabora el frame FinCategorías con el valor *"Sí"* en el campo categoría_finalizada, por lo que el siguiente submódulo de ejecución será Categorías Finalizadas
  → Si el frame de la cima del Registro es CategoríaPrincipal: elabora el frame CategoríaPrincipal con el valor de la categoría quinta en el campo quinta, con lo que el siguiente submódulo de ejecución será Fin Categoría
  → Si el frame de la cima del Registro es ErrorCategoríaQuinta elabora el frame CategoríaPrincipal con el valor "*Sí*" en el campo error_anterior, por lo que el siguiente submódulo de ejecución será Categoría Cuaternaria.

5.1.8 Submódulo Fin Categorías

5.1.8.1 Funcionalidad

Este submódulo se ejecuta cuando el usuario ha seleccionado alimentos que poseen hasta cinco categorías, para indicar a la aplicación que el siguiente paso será mostrar los alimentos englobados dentro de esas categorías por pantalla.

La actividad Gestor de Diálogo recibe el frame CategoríaPrincipal con todas las categorías completas, con los valores que el usuario ha ido seleccionado en anteriores turnos de diálogo.

5.1.8.2 Diagrama de Flujo y Arquitectura.

La Figura 35 muestra el diagrama de flujo definido para este submódulo.

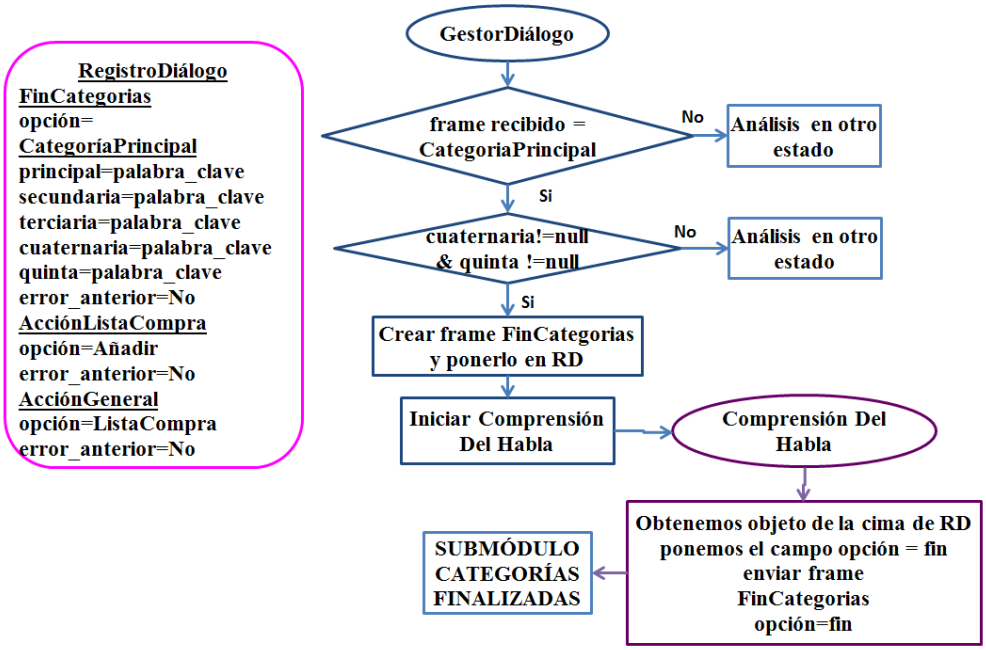

Figura 35: Diagrama de Flujo Submódulo Fin Categorías

- La actividad Gestor de Diálogo, cuando recibe el frame CategoríaPrincipal genera el frame FinCategorias y lo pone en la cima del Registro de Diálogo. A continuación, inicia la actividad Comprensión Del Habla

- La actividad Comprensión del Habla, obtiene el frame de la cima del Registro y lo reenvía a la actividad Gestor de Diálogo para que inicie la ejecución del siguiente módulo que será el de mostrar los alimentos por pantalla.

5.1.9 Submódulo Categorías Finalizadas

5.1.9.1 Funcionalidad

Este submódulo se ejecuta cuando el Gestor Diálogo recibe el frame FinCategorías y su función es la de mostrar los alimentos por pantalla

Para ello debe conocer las categorías escogidas por el usuario, por lo que necesitará al menos la información recogida en los dos últimos turnos de diálogo.

5.1.9.2 Arquitectura y Diagrama de Flujo

La
Figura 36 muestra el diagrama de flujo definido para este submódulo.

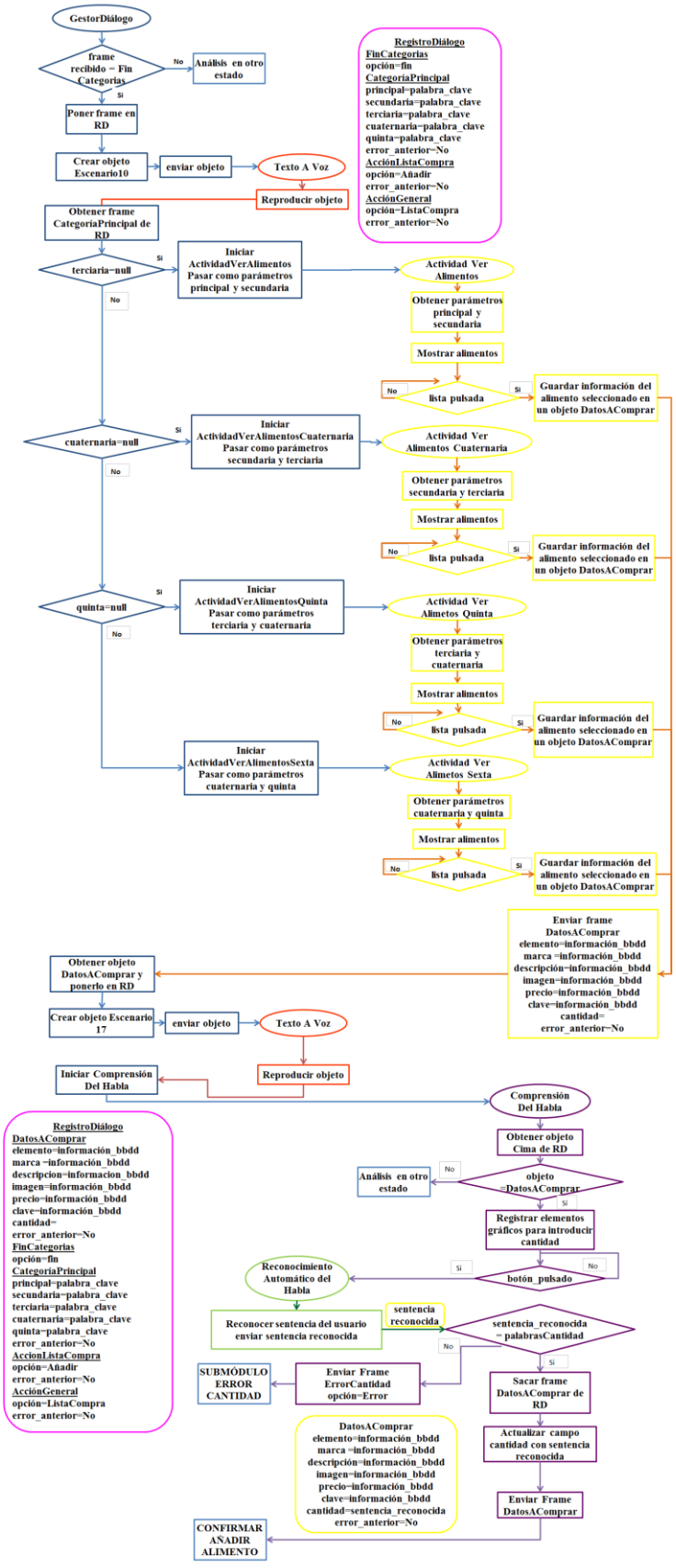

Figura 36: **Diagrama de Flujo Submódulo Categorías Finalizadas**

- El Gestor Diálogo recibe el frame FinCategorías, y construye un mensaje de texto que informará al usuario de que la aplicación mostrará en pantalla la información de los alimentos englobados dentro de la categoría seleccionada. A continuación inicia la actividad Texto A Voz

- La actividad Texto A Voz reproduce la sentencia y devuelve el control a la actividad Gestor de Diálogo

- El Gestor de Diálogo debe proporcionar la información de las categorías seleccionadas por el usuario a la actividad correspondiente de mostrar los alimentos por pantalla. Para ello obtiene el frame Categoría Principal de la cima del Registro de Diálogo y procede del siguiente modo:

  - → Si categoría_terciaria = null: inicia la Actividad Ver Alimentos pasando como parámetros los valores categoría principal y categoría secundaria del frame CategoríaPrincipal
  - → Si categoría_cuaternaria = null: inicia la Actividad Ver Alimentos Cuaternaria pasando como parámetros los valores secundaria y terciaria el frame CategoríaPrincipal
  - → Si categoría_quinta = null: Inicia la Actividad Ver Alimentos Quinta pasando como parámetros los valores terciaria y cuaternaria del frame CategoríaPrincipal
  - → Si todas las categorías están completas: inicia la Actividad Ver Alimentos Sexta pasando como parámetros los valores cuaternaría y quinta del frame Categoría Principal

- Cada una de las actividades debe mostrar la siguiente información relativa al alimento:

  - → Nombre
  - → Marca
  - → Descripción
  - → Imagen
  - → Precio

- Además para cada alimento, la actividad también obtendrá la clave principal con la que se almacena en la Base de Datos. Para ello, cada actividad procede del siguiente modo:

→ Actividad Ver Alimentos: ejecuta el método *obtenerAlimentosSinTerciaria()* pasando como parámetros las categorías primaria y secundaria recibidas en el intent. Este método devuelve en forma de elemento Cursor la información de los alimentos englobados dentro de esas categorías. La actividad recorre el cursor y para cada iteración construye el objeto DatosAComprar, cuyos campos coinciden con los campos de información obtenidos de la base de datos. Los objetos DatosAComprar se almacenan en un objeto vector que es pasado como parámetro a la clase AdaptadorAlimentos, que junto con el objeto de tipo ListView ListaDatosCOmpra, mostrarán por pantalla la información de los alimentos

```
lista=(ListView)findViewById(R.id.listaDatosCompra);
Cursor c =GestorDialogo.bdh.obtenerAlimentosSinTerciaria(secundaria,
principal);
if(c.getCount()!=0){if(c.moveToFirst()){do{
verClave = c.getString(0);
verNombre=c.getString(1);
verMarca = c.getString(2);
verDescripcion = c.getString(3);
verImagen = c.getString(4);
verPrecio = c.getFloat(5);
DatosAComprar dac= new DatosAComprar(verNombre,
verMarca,verDescripcion,verImagen,verPrecio,verClave);
vector.add(dac);}while(c.moveToNext());}
AdaptadorAlimentos adaptador = new AdaptadorAlimentos(this, vector);
```

El resultado es el mostrado en la Figura 37

Figura 37: Elementos gráficos Submódulo Categorías Finalizadas

❖ El resto de actividades presentan un comportamiento similar, a excepción del método invocado y los parámetros pasados para obtener la información relativa a los alimentos.

→ La Actividad Ver Alimentos Cuaternaria, llama al método *obtenerAlimentosSinCuaternaria* de Base de Datos Helper pasando como parámetros las categorías secundaria y terciaria:

Cursor c =GestorDialogo.bdh.obtenerAlimentosSinCuaternaria(terciaria, secundaria);

→ La Actividad Ver Alimentos Sin Quinta llama al método *obtenerAlimentosSinQuinta* pasando como parámetros las categorías terciaria y cuaternaria.

Cursor c =GestorDialogo.bdh.obtenerAlimentosSinQuinta(cuaternaria, terciaria);

→ La Actividad Ver Alimentos Sin Sexta llama al método *obtenerAlimentosSinSexta* de Base de Datos Helper pasando como parámetros las categorías cuaternaria y quinta.

Cursor c =GestorDialogo.bdh.obtenerAlimentosSinSexta(quinta, cuaternaria);

❖ Una vez mostrados los alimentos las actividades permanecen en espera activa hasta que el usuario pulsa sobre el alimento que desea añadir a la lista de la compra. Cuando esto ocurre la actividad construye el frame DatosAComprar con la información del alimento seleccionado y lo envía al Gestor de Diálogo.

❖ La actividad Gestor de Dialogo pone el frame DatosAComprar en la cima del Registro de Diálogo e inicia un segundo turno de diálogo para preguntar al usuario cuántas unidades del alimento desea añadir. Para ello construye una sentencia de tipo Escenario 17 y lo envía a la actividad Texto A Voz.

❖ La actividad Texto A Voz reproduce el mensaje al usuario y devuelve el control al Gestor.

❖ El Gestor inicia la actividad Comprensión del Habla.

❖ La actividad Comprensión del Habla muestra en pantalla los elementos gráficos para que el usuario indique la cantidad de alimento que desea, como se muestra en la Figura 38

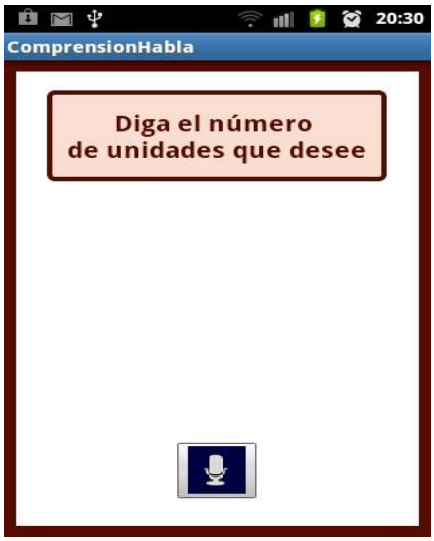

Figura 38: Elementos gráficos para introducir la Cantidad.

- Cuando el usuario pulsa sobre el botón de voz, se inicia la actividad de Reconocimiento del Habla.

- La actividad Reconocimiento Automático del Habla reconoce la sentencia pronunciada por el usuario y envía el resultado a la actividad Comprensión del Habla.

- La actividad Comprensión del Habla analiza la sentencia recibida y en función de ella elaborará un determinado frame que enviará al Gestor para que dé comiendo el siguiente submódulo de ejecución.

  → Si sentencia_recibida contiene una palabra de cantidad: genera el frame DatosAComprar con la información relativa al alimento seleccionado, por lo que el siguiente submódulo de ejecución será Confirmar Añadir Alimento.
  → Si sentencia_recibida no contiene una palabra de cantidad: genera el frame ErrorCantidad, por lo que el siguiente submódulo de ejecución será Error Cantidad.

5.1.10 Confirmar Añadir Alimento

5.1.10.1 Funcionalidad

Este submódulo tiene como finalidad dar al usuario la opción de si el alimento que seleccionó para añadir a la lista de la compra es realmente el que desea añadir o por el contrario desea añadir otro diferente.

Se ejecuta cuando el Gestor Diálogo recibe el frame DatosAComprar con la información del alimento seleccionado

5.1.10.2 Diagrama de Flujo y Arquitectura

La Figura 39 muestra el diagrama de flujo definido para este submódulo.

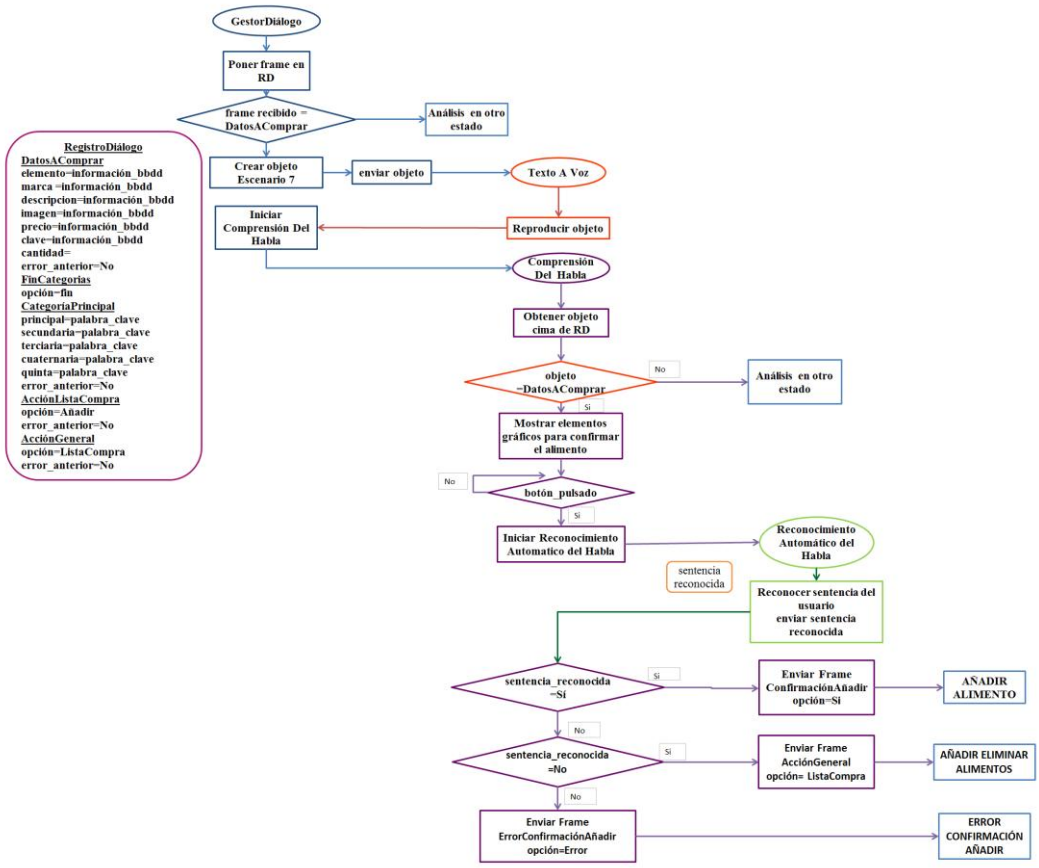

Figura 39: Diagrama de Flujo Submódulo Añadir Alimento.

- La actividad Gestor de Diálogo cuando recibe el frame DatosAComprar lo pone en la cima del Registro de Diálogo y genera una sentencia de tipo Escenario7 que pregunta al usuario si realmente desea añadir el alimento. La sentencia se envía a la actividad Texto A Voz.

- La actividad TextoAVoz reproduce la sentencia recibida y devuelve el control al Gestor.

- El Gestor de Diálogo inicia la actividad Comprensión del Habla.

- La actividad Comprensión del habla obtiene el frame DatosAComprar de la cima del Registro y muestra la información contenida en él junto con un botón de voz como se muestra en la Figura 40. La actividad permanece en espera activa hasta que el usuario pulsa el botón de voz, momento en el que se inicia la actividad de Reconocimiento del Habla.

Figura 40: Elementos gráficos para confirmar que se desea añadir el alimento

- La actividad Reconocimiento Automático del Habla reconoce la sentencia emitida por el usuario y la envía a la actividad Comprensión del Habla.

- La actividad Comprensión del Habla analiza la sentencia recibida y en función de ella elabora un determinado frame que enviará al Gestor de Diálogo para que determine el siguiente submódulo de ejecución.

  - → Si sentencia recibida contiene el valor "*Sí*" elabora el frame Confirmación Añadir con el valor "*Sí*" en el campo opción., por lo que el siguiente submódulo de ejecución será Añadir Alimento.
  - → Si sentencia recibida contiene el valor "*No*" elabora el frame AcciónGeneral con el valor *"ListaCompra*" para volver al inicio y que el usuario pueda añadir otro alimento, por lo que el siguiente submódulo de ejecución será Añadir Eliminar Alimentos.
  - → Si sentencia_recibida no contiene ninguno de los valores anteriores, elabora el frame ErrorConfirmaciónAñadir con el valor "*Error"* en el campo opción, por lo que el siguiente submódulo de ejecución será ErrorConfirmaciónAñadir.

5.1.11 Añadir Alimento A Base de Datos

5.1.11.1 Funcionalidad

Este submódulo tiene como finalidad añadir el alimento seleccionado por el usuario a la Base de Datos Usuarios.

Antes de añadir el alimento, el submódulo comprueba que el alimento no haya sido ya añadido en anteriores turnos de diálogos, caso en el que lanzará

un mensaje de error al usuario. En caso contrario, el alimento se almacena en la base de datos y se muestran por pantalla todos los alimentos añadidos a la lista de la compra. Se ejecuta cuando el Gestor de Diálogo recibe el frame ConfirmaciónAñadir con el valor "*sí*" en el campo opción.

5.1.11.2 Diagama de Flujo y Arquitectura

La Figura 41 muestra el diagrama de flujo definido para este submódulo.

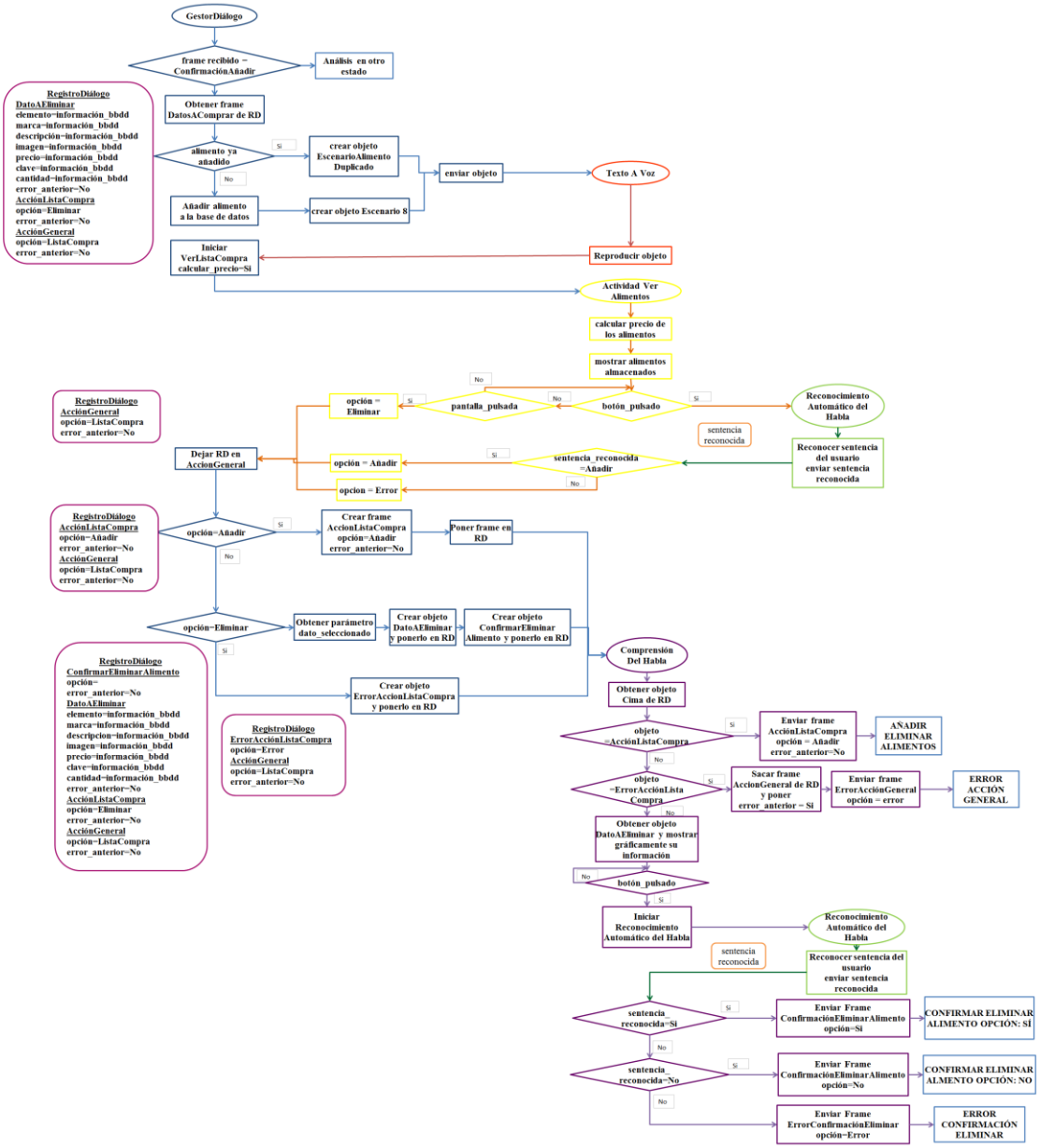

Figura 41: Diagrama de Flujo Submódulo Añadir Alimento a Base de Datos

- La actividad Gestor de Diálogo cuando recibe el frame ConfirmaciónAñadir con el valor "*Sí*" en el campo opción coloca el frame en la cima del Registro Diálogo. A continuación comprueba si el alimento ya ha sido añadido. Para ello ejecuta el método *obtenerClavesAlimentos* de la clase BaseDatosUsuarios que devuelve las claves primarias pertenecientes a los

alimentos almacenados en la base de datos y las compara con la clave primaria del alimento a añadir, almacenada en el frame DatosAComprar del Registro de Diálogo.

Si la clave del alimento a añadir coincide con algunas de las claves de los alimentos almacenados en la Base de Datos Usuarios el Gestor elabora un objeto de EscenarioAlimentoDuplicado, que informará al usuario de que el alimento ya ha sido añadido.

Si por el contrario la clave del alimento a añadir no coincide con ninguna de las claves, el Gestor elabora un objeto de tipo Escenario8, que informa al usuario de que el alimento ha sido añadido correctamente y que a continuación podrá añadir un nuevo alimento diciendo "*añadir*", o "*eliminar*" un alimento ya almacenado pulsando sobre él.

```
String clave = dac.getClave();
duplicado = "No"
claves = bdu.obtenerClavesAlimentos();
for(int i = 0;i<claves.size(); i++ ){
if(claves.get(i).equalsIgnoreCase(clave)){duplicado = "Si";}}
if(duplicado.equalsIgnoreCase("No")){resultado = bdu.introducirDato(dac);
escenario = new Escenario8();}
if(duplicado.equalsIgnoreCase("Si")){ escenario = new
EscenarioAlimentoDuplicado();}
```

- A continuación el Gestor de Diálogo inicia la actividad Texto A Voz pasando como parámetro el objeto generado.
- La actividad Texto A Voz reproduce la sentencia recibida y devuelve el control al Gestor de Diálogo.
- El Gestor de Diálogo inicia la actividad Ver Lista Compra con el valor "*sí*" en el campo calcular_precio, ya que habrá alimentos almacenados en la lista de la compra.

En este punto la actividad Ver Lista Compra realiza los mismos pasos que los comentados en el submódulo añadir/eliminar alimento, tal y como se detalla en elApartado 5.1.2

5.1.12 Submódulo Confirmar Eliminar Alimento: Opción No

5.1.12.1 Funcionalidad

La funcionalidad de este submódulo es volver al inicio de la aplicación, procediendo a mostrar los alimentos almacenados en la lista de la compra en el caso de haberlos, después de que el usuario opta por la opción de no añadir el alimento que había seleccionado en los anteriores turnos de diálogo.

Este submódulo se ejecuta cuando el Gestor de Diálogo recibe el frame ConfirmarEliminarAlimento con el valor "*No*" en el campo opción.

5.1.12.2 Diagrama de Flujo y Arquitectura

La Figura 42 muestra el diagrama de flujo definido para la aplicación.

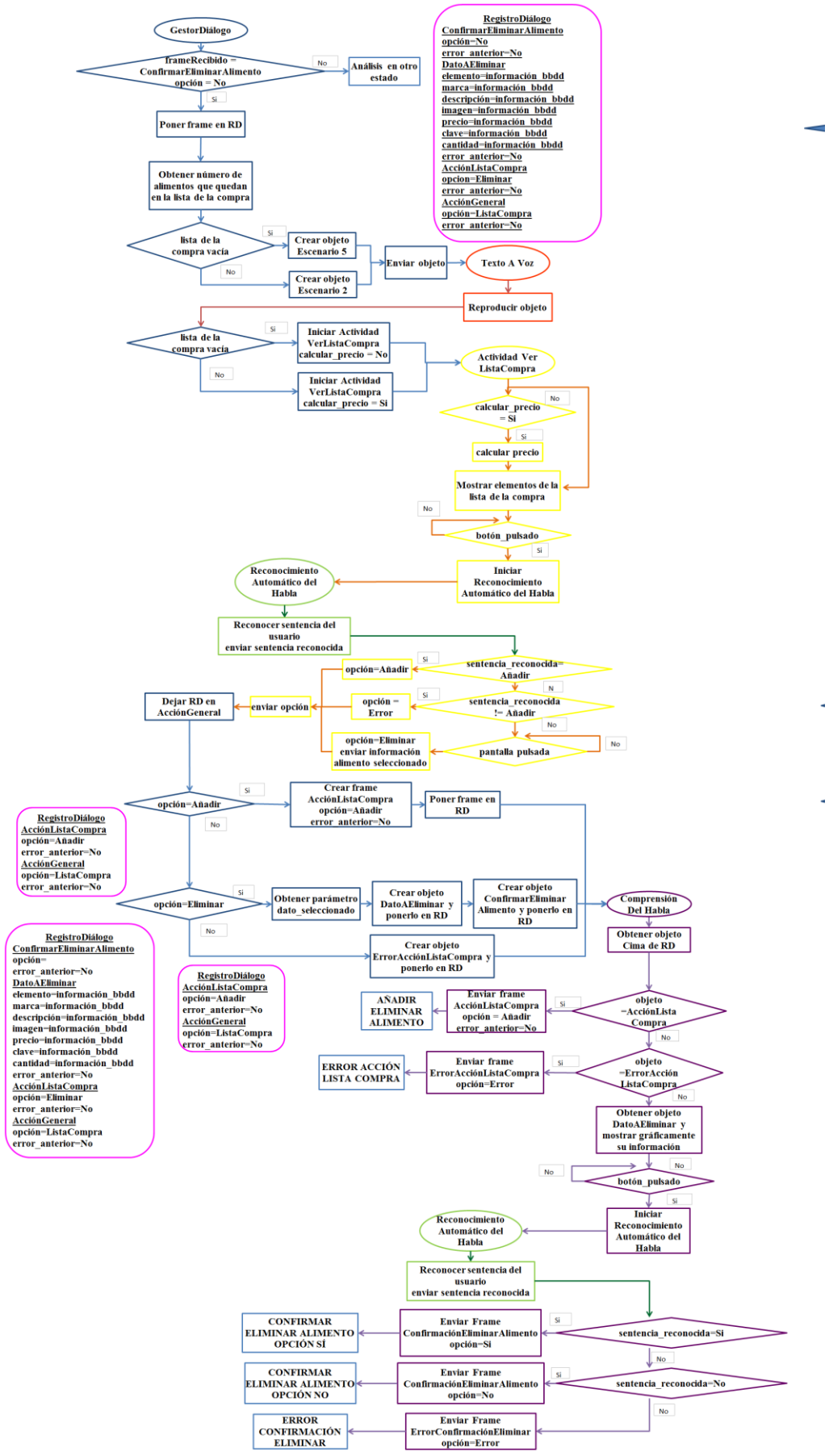

Figura 42: Diagrama de Flujo Submódulo Confirmar Eliminar Alimento Opción No

- La actividad Gestor de Diálogo cuando recibe el frame ConfirmarEliminarAlimento, obtiene el número de alimentos almacenados en la lista de la compra.
  - → Si la lista está vacía: construye un objeto de tipo Escenario5 que ofrece al usuario la posibilidad de añadir un nuevo alimento.
  - → Si la lista contiene alimentos: construye un objeto de tipo Escenario2 que ofrece al usuario la posibilidad de añadir o de eliminar alimentos.
- La actividad TextoAVoz reproduce la sentencia y devuelve el control al Gestor.
- La actividad Gestor de Diálogo lanza la Actividad Ver Lista Compra pasando como parámetro el string calcular_precio, que contendrá el valor *"si*" o "*no*", en función de si hay o no hay alimentos almacenados en la lista de la compra.
- La Actividad Ver Lista Compra, muestra en pantalla los alimentos almacenados y para ello, como puede verse en el diagrama, sigue los mismos pasos que los descritos en el submódulo Añadir Eliminar Alimentos, tal y como detalla el Apartado 5.1.2

5.1.13 Submódulo Confirmar Eliminar Alimento: Opción Sí

5.1.13.1 Funcionalidad

La finalidad de este submódulo es la de obtener la información del alimento que el usuario desea quitar de la lista de la compra y eliminarlo de la Base de Datos Usuarios

5.1.13.2 Diagrama de Flujo y Arquitectura

La Figura 43 muestra el diagrama de flujo definido para este submódulo.

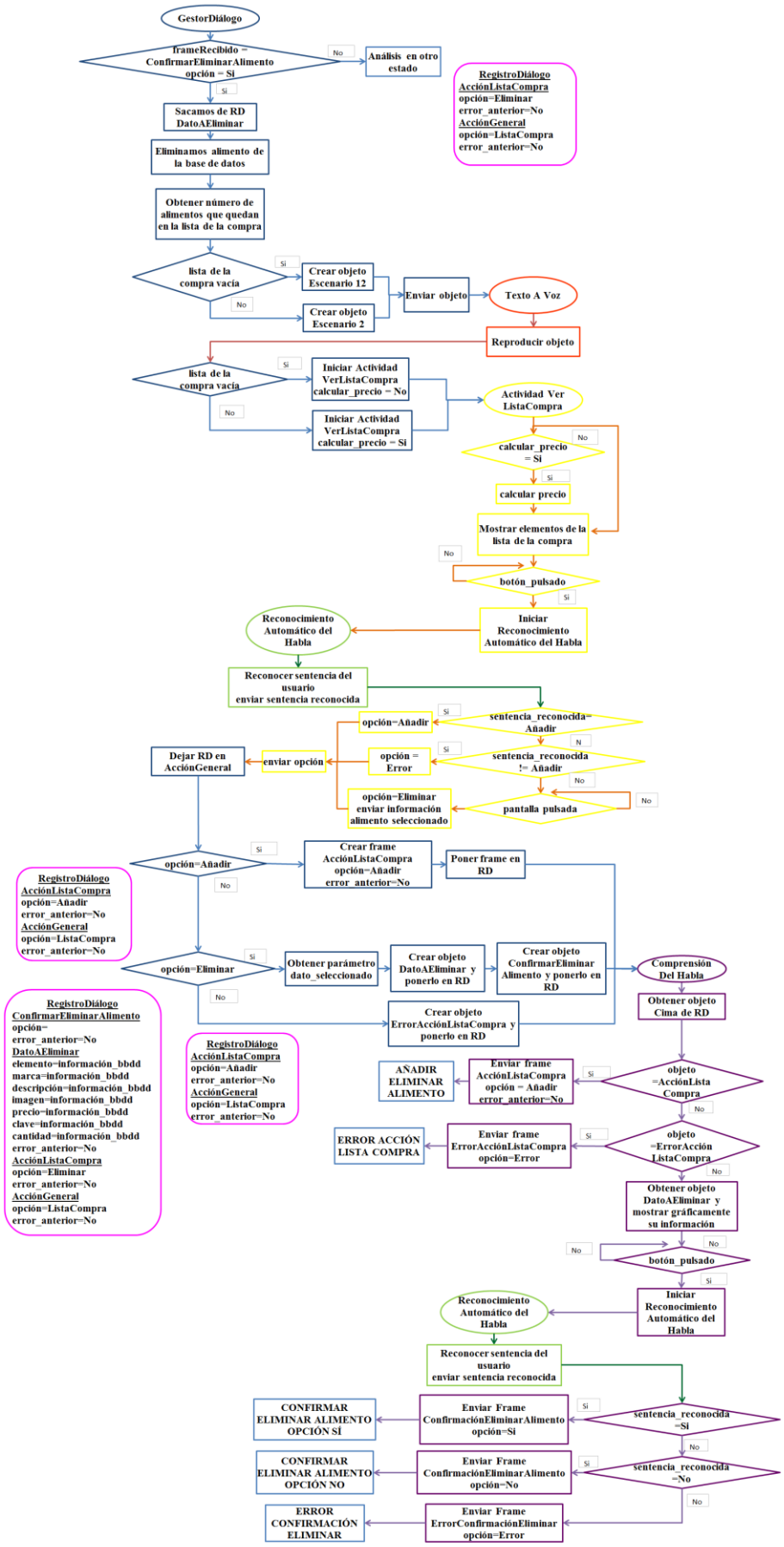

Figura 43: Diagrama de Flujo Submódulo Confirmar Eliminar Alimento Opción Sí

- El Gestor de Diálogo, cuando recibe el frame ConfirmarEliminarAlimento con el valor "*Si*" en el campo opción, extrae del registro de diálogo el frame DatoAEliminar que fue almacenado durante la ejecución del submódulo en el que el usuario pulsó la pantalla para indicar el alimento que deseaba eliminar. Una vez obtenida la información, procede a eliminarlo de la base de datos ejecutando el método *eliminarRegistro* de la clase Base de Datos Usuarios. A este método le pasa como parámetro la clave primaria del alimento a eliminar, que la obtiene del frame DatoAEliminar. Una vez eliminado, el Gestor obtiene el número de alimentos que quedan almacenados en la lista de la compra. En función del resultado, el Gestor procede del siguiente modo:

  - → Si quedan alimentos almacenados : construye una sentencia de tipo Escenario2 que ofrece al usuario la posibilidad de añadir un nuevo alimento o eliminar uno ya almacenado.
  - → Si la lista de la compra está vacía: construye una sentencia de tipo Escenario12 que ofrece al usuario la opción de añadir nuevos alimentos.

```
String clave = dac.getClave();
DatoAEliminar daeliminar = (DatoAEliminar)rd.top();
String clave = daeliminar.getClave();
bdu.eliminarRegistro(clave);
int elementos=GestorDialogo.bdu.obtenerNumeroElementos();
if(elementos==0){ escenario = new Escenario12();}}
else{ escenario = new Escenario2();}
```

- La actividad Texto A Voz reproduce la sentencia generada y devuelve el control al Gestor de Diálogo.

- A continuación como puede observarse en el diagrama de flujo, el Gestor inicia la Actividad Ver Lista Compra, procediendo el mismo modo que el ya comentado en submódulos aneriores, por lo que la explicación del submódulo Confirmar Eliminar Alimento Opción Sí, se da por concluida.

5.2 Submódulos de Errores

Dentro de los diferentes diálogos de la aplicación hay casos en los que la respuesta de los usuarios puede dar lugar a dos tipos diferentes de acciones por parte del Gestor de Dialogo y otros casos en los que en función de la respuesta del usuario, el Gestor debe decidir entre más de dos submódulos de ejecución.

Los submódulos en los que el Gestor Diálogo solamente debe decidir entre dos acciones son:

- Submódulo Categoría Principal

- Submódulo Categoría Secundaria
- Submódulo Categoría Terciaria
- Submódulo Categoría Cuaternaria
- Submódulo Categoría Quinta

En estos submóulos los dos posibles estados entre los que decidirá el Gestor de Diálogo serán, o bien pasar a la siguiente categoría en caso de que la respuesta del usuario sea correcta, o bien, quedarse en el mismo estado, en caso de que la respuesta del usuario sea incorrecta. Debido a la sencillez de la decisión, la gestión de errores se realiza en el propio frame Categoría Principal, rellenando el campo categoría correspondiente en el primer caso o el campo error_anterior con el valor *"si"* en el segundo caso (para más información consultar los Apartados 5.1.3, 5.1.4, 5.1.5, 5.1.6 y 5.1.7)

Para los submódulos en los que el Gestor de Diálogo debe decidir entre más de un submódulo de ejecución, se definen estados auxiliares denominados Submódulos de Errores, que se detallan en las siguientes secciones:

5.2.1 Submódulo Error Acción General

5.2.1.1 Funcionalidad

La ejecución de este submódulo se produce cuando el usuario, que se encuentra en el submódulo inicio, no pronuncia bien las opciones "*lista de la compra*" o "*recetario*", que la aplicación le solicita que indique.

En este submódulo el Gestor de Diálogo recibe el frame ErrorAcciónGeneral con el valor "*Error*" en el campo opción y su funcionalidad es indicarle al usuario que se ha producido un error en el reconocimiento y que especifique nuevamente si desea iniciar la lista de la compra o el recetario.

5.2.1.2 Diagrama de Flujo y Arquitectura

La Figura 44 muestra el diagrama de flujo definido para la aplicación.

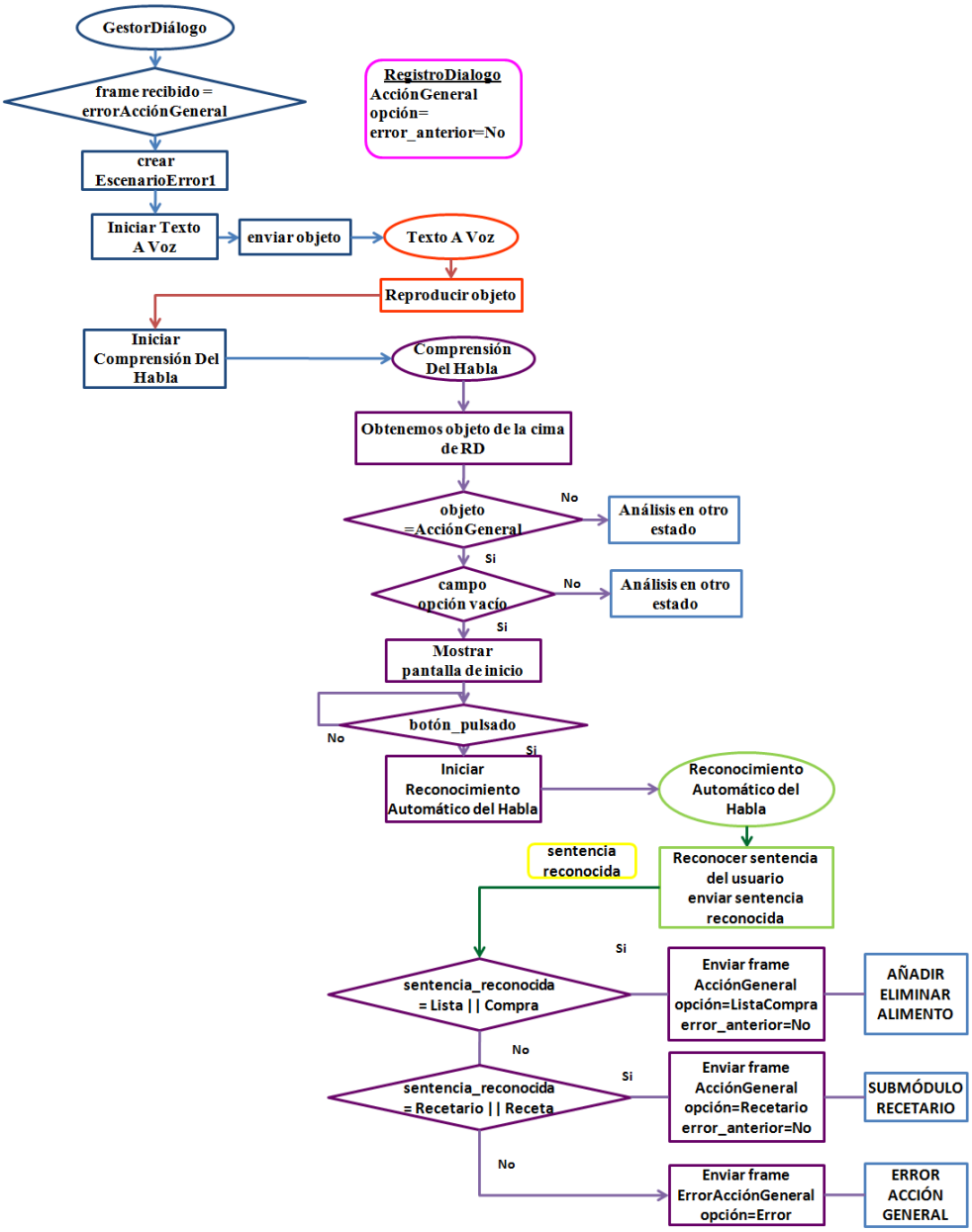

Figura 44: Diagrama de Flujo Submódulo Error Acción General

- El Gestor de Diálogo cuando recibe el frame ErrorAcciónGeneral, genera una sentencia de tipo esenarioError1, que informa al usuario de que se ha producido un error en el reconocimiento y que debe indicar nuevamente si desea iniciar la lista de la compra o el recetario.

- La actividad Texto A Voz reproduce la sentencia y devuelve el control al Gestor de Dialogo.

- El Gestor Diálogo inicia la actividad Comprensión del Habla.

- La actividad Comprensión del Habla, muestra los elementos de la pantalla de inicio, y a partir de este momento la aplicación sigue los mismos pasos

que los realizados en el Submódulo Añadir Eliminar Alimentos , por lo que para más información, consultar el Apartado 5.1.2.

5.2.2 Submódulo Error Confirmación Añadir

5.2.2.1 Funcionalidad

La ejecución de este submódulo se realiza en los siguientes casos:

- Cuando el usuario se encuentra en el Submódulo Añadir Alimento y tras la petición por parte de la aplicación de que confirme si desea añadir el alimento, se produce un error a la hora de reconocer la opción indicada por el usuario.

- Cuando durante la ejecución del submódulo añadir eliminar alimento el usuario pulsa sobre la pantalla para eliminar un alimento y a la hora de que la aplicación le pide que confirme si realmente desea eliminar dicho alimento se produce un error en el reconocimiento.

El Gestor de Diálogo recibe el frame ErrorConfirmaciónAñadir y su función es la de informar al usuario que se ha producido un error en el reconocimiento, por lo que debe indicar nuevamente si desea añadir el alimento.

5.2.2.2 Diagrama de Flujo y Arquitectura

La Figura 45 muestra el diagrama de flujo definido para este submódulo.

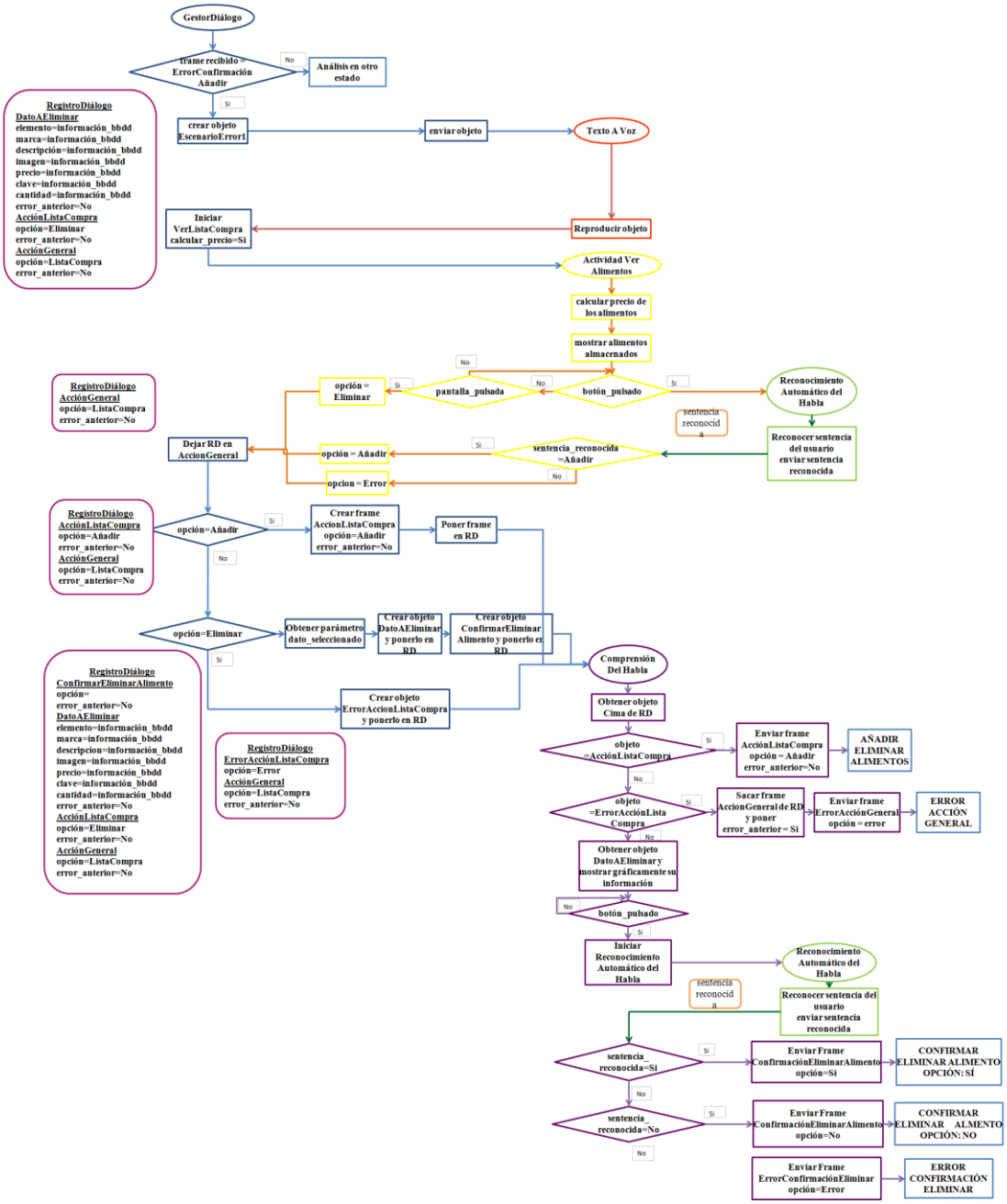

Figura 45: Diagrama de Flujo Submódulo Error Confirmación Añadir

- El Gestor de Diálogo cuando recibe el frame ErrorConfirmaciónAñadir genera una sentencia de tipo EscenarioError1 y lo envía a la actividad TextoAVoz.
- La actividad TextoAVoz reproduce la sentencia y devuelve el control al Gestor de Diálogo.
- El Gestor de Diálogo inicia la actividad Comprensión del Habla.

- La actividad Comprensión del Habla obtiene el frame DatosAComprar de la cima del Registro de Diálogo que fue almacenado en el submódulo anterior y muestra la información por pantalla junto con el botón de voz.

El resto de pasos que se siguen en este submódulo son idénticos a los del submódulo Confirmar Añadir Alimento, por lo que concluimos aquí la explicación.

5.2.3 Submódulo Error Confirmación Eliminar

5.2.3.1 Funcionalidad

La funcionalidad de este submódulo es ofrecer al usuario la opción de no eliminar el alimento seleccionado en caso de que el usuario haya cambiado de opinión, o de que la selección en la pantalla se haya producido por error.

Se ejecuta cuando el Gestor de Diálogo recibe el frame ConfirmarEliminarAlimento con el valor *"Error"* en el campo opción.

5.2.3.2 Diagrama de Flujo y Arquitectura

La Figura 46 muestra el diagrama de flujo definido para la aplicación.

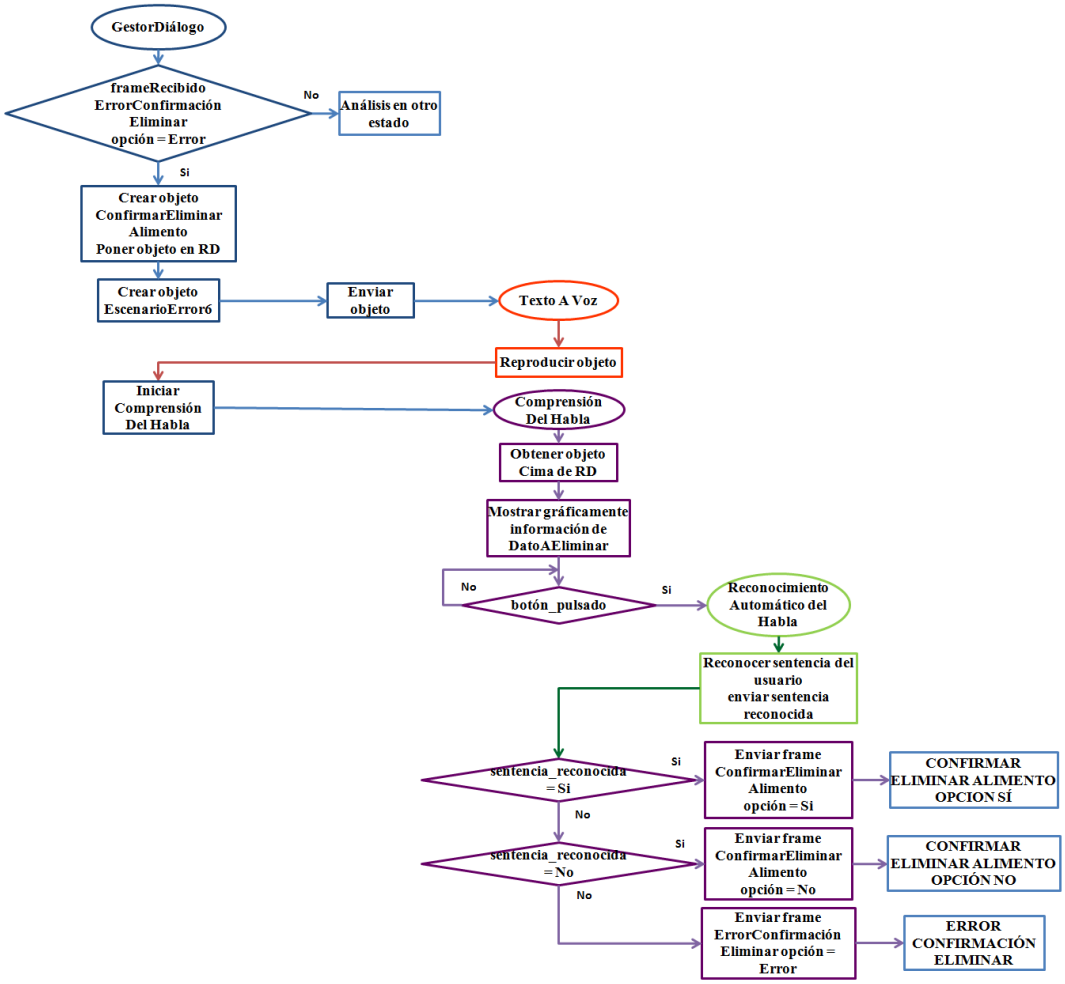

Figura 46: Diagrama de Flujo Submódulo Error Confirmación Eliminar

- Cuando el Gestor recibe el frame ErrorConfirmaciónEliminar, lo coloca en la cima del Registro de Diálogo y genera una sentencia de tipoEscenarioError6 que informa al usuario de que se ha producido un error en el reconocimiento y que debe volver a indicar si desea eliminar el alimento. La sentencia es enviada a la actividad Texto A Voz.

- La Actividad Texto A Voz, reproduce la sentencia recibida por el Gestor y le devuelve el control.

- El Gestor de Diálogo inicia la actividad Comprensión del Habla pasándole en el intent la sentencia reconocida.

- La actividad Comprensión del Habla analiza la sentencia reconocida y en función de ella elabora un determinado frame que envía al Gestor, para que decida el siguiente submódulo de ejecución

- → Si sentencia_reconocida contiene el valor "*si*": genera el frame ConfirmarEliminarAlimento con el valor "*Sí*" en el campo opción, por lo que el siguiente submódulo de ejecución será Confirmar Eliminar Alimento Opción Sí
- → Si sentencia_reconocida contiene el valor "*no"* genera el frame ConfirmarEliminarAlimento con el valor "*no*" en el campo opción, por lo que el siguiente submódulo de ejecución será Confirmar Eliminar Alimento Opción No
- → Si sentencia_reconcida no contiene ninguno de los valores anteriores: genera el frame ErrorConfirmaciónEliminar por lo que el siguiente submódulo de ejecución será ErrorConfirmaciónEliminar.

5.2.4 Submódulo Error Cantidad.

5.2.4.1 Funcionalidad

Este submódulo se ejecuta cuando el usuario se encuentra en el submódulo Categorías Finalizadas y a la hora de especificar la cantidad de unidades que desea añadir del alimento que le solicita el sistema que indique, se produce un error en el reconocimiento. Debido a ello, la finalidad de este submódulo es volver a solicitar al usuario que especifique el número de unidades que desea añadir del alimento.

Se ejecuta cuando el Gestor de Diálogo recibe el frame ErrorCantidad.

5.2.4.2 Diagrama de Flujo y Arquitectura.

La Figura 47 muestra el diagrama de flujo definido para este submódulo.

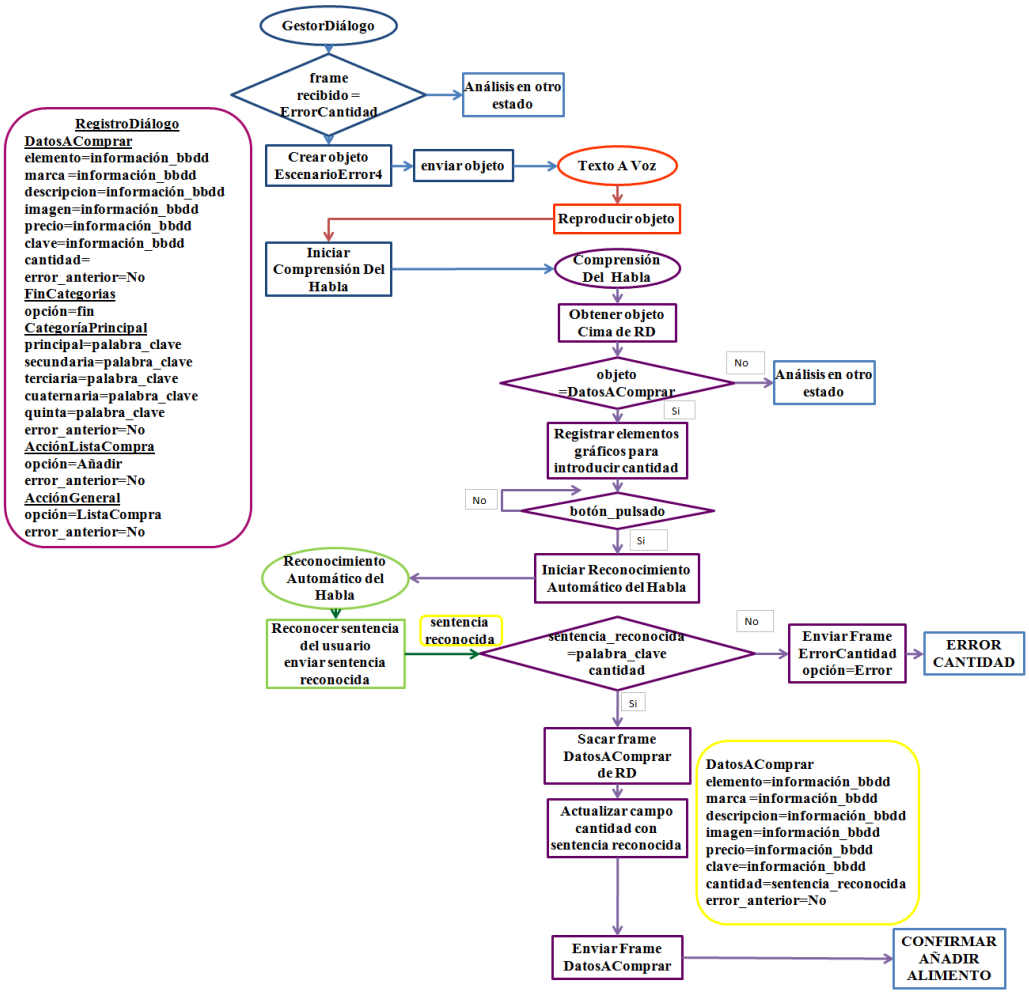

Figura 47: Diagrama de Flujo Submódulo Error Cantidad

- El Gestor de Diálogo, cuando recibe el frame ErrorCantidad, genera una sentencia de tipo EscenarioError4, que informa al usuario de que se ha producido un error en el reconocimiento e inicia la actividad Texto A Voz.

- La actividad Texto A Voz, reproduce la sentencia recibida y devuelve el control al Gestor de Diálogo.

- A partir de aquí, el Gestor de Diálogo realiza los mismos pasos que los comentados en el Submódulo Fin Categoría, para mostrar por pantalla los elementos gráficos que permiten al usuario especificar nuevamente la cantidad.

5.2.5 Submódulo Error Acción Lista Compra

5.2.5.1 Funcionalidad

Este submódulo se ejecuta cuando el usuario está en el Submódulo Añadir Eliminar y se produce un error a la hora de reconocer si el usuario desea añadir un nuevo alimento a la lista de la compra. Cuando el Gestor de

Diálogo recibe el frame ErrorAcciónListaCompra, la aplicación lanza un mensaje de error que informa al usuario de que se ha producido un error en el reconocimiento y que debe indicar de nuevo si desea añadir un alimento o eliminar uno de la lista de la compra almacenado previamente.

5.2.5.2 Diagrama de Flujo y Arquitectura.

La Figura 48 muestra el diagrama de flujo definido para este submódulo.

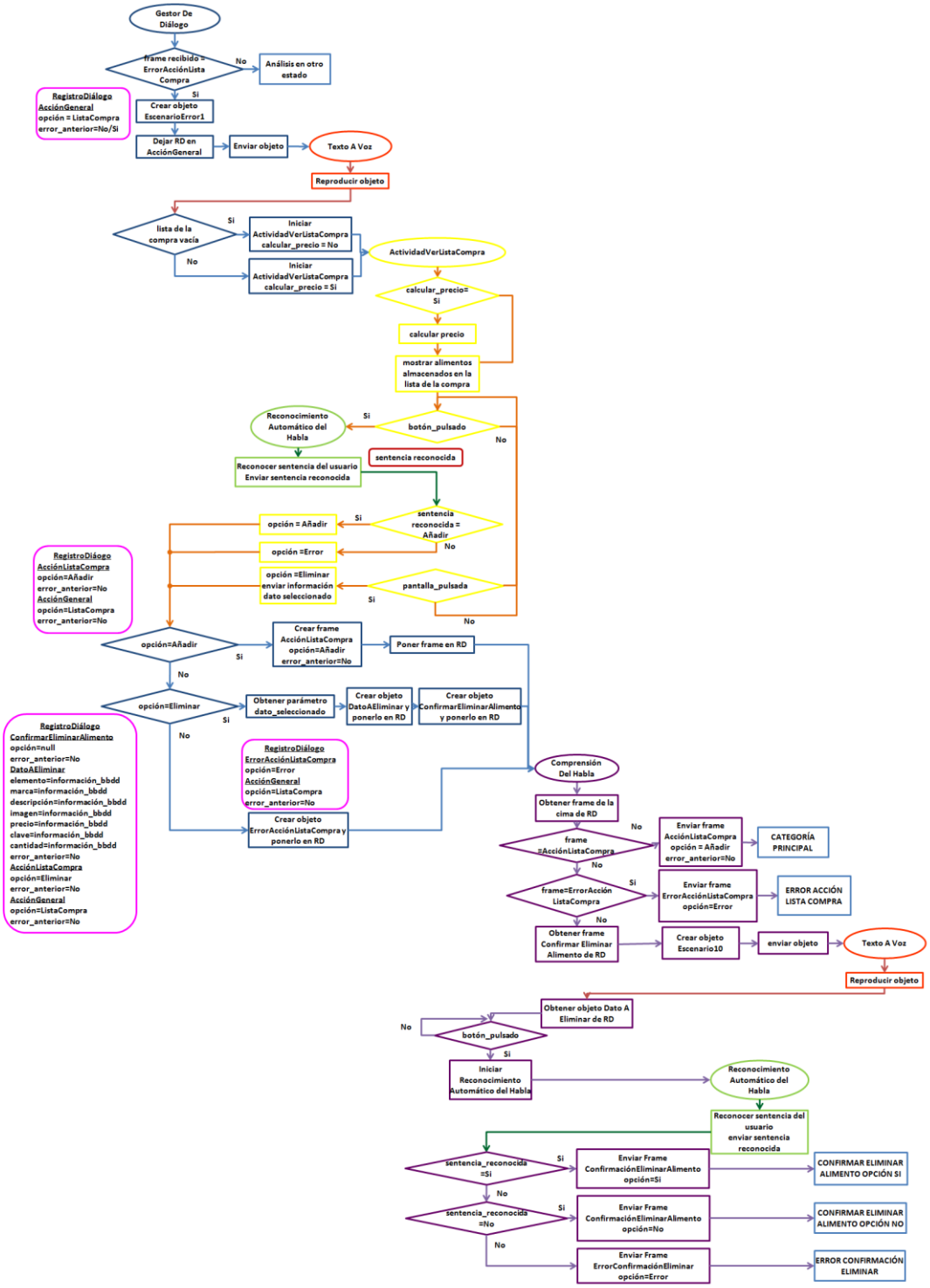

Figura 48: Diagrama de Flujo Submódulo Error Acción Lista Compra

❖ El Gestor de Diálogo cuando recibe el frame ErrorAcciónListaCompra, deja el Registro de Diálogo con el frame AcciónGeneral y genera una sentencia de tipo EscenarioError2 que envía a la actividad Texto A Voz.

❖ La actividad Texto A Voz, reproduce la sentencia recibida que informa al usuario de que se ha producido un error en el reconocimiento y devuelve el control al Gestor.

❖ A partir de este punto, como puede apreciarse en el diagrama de flujo, el Gestor sigue la misma estrategia descrita en el submódulo Añadir Eliminar Alimento, por lo que si desea más información, consulte el Apartado 5.1.2

5.3 Escenarios Conversacionales

Para ilustrar de forma más concreta el diálogo producido entre la aplicación y el usuario, se muestran a continuación algunos escenarios conversacionales que se producen durante la ejecución del Módulo Lista de la Compra

❖ Saludo inicial y elección de añadir alimentos a la lista de la compra vacía

- **SISTEMA**: Bienvenido a la aplicación. ¿Desea iniciar la lista de la compra o el recetario?
- **USUARIO**: Me gustaría iniciar la lista de la compra
- **SISTEMA**: Su lista de la compra está vacía. Diga Añadir para agregar alimentos.

❖ Consulta de la lista de la compra vacía y adicción de alimentos

- **SISTEMA**: ¿Desea iniciar la lista de la compra o el recetario?
- **USUARIO**: Querría iniciar la lista de la compra
- **SISTEMA**: Su lista de la compra está vacía. Diga añadir para agregar nuevos alimentos
- **USUARIO**: Añadir
- **SISTEMA**: Seleccione una de las siguientes categorías principales
- **USUARIO**: Alimentación
- **SISTEMA**: Seleccione una de las siguientes categorías de alimentación
- **USUARIO**: Vinagres
- **SISTEMA**: Seleccione uno de los siguientes tipos de vinagre
- **USUARIO**: Pulsa sobre el elemento a añadir
- **SISTEMA**: Diga el número de unidades que desea añadir del alimento
- **USUARIO**: Dos
- **SISTEMA**: ¿Seguro que desea añadir este alimento?
- **USUARIO**: Sí
- **SISTEMA**: Alimento añadido correctamente. Si desea añadir un alimento diga añadir y si desea eliminar uno de la lista pulse sobre él

❖ Consulta de la lista de la compra y eliminación de un alimento

- **SISTEMA**: ¿Desea iniciar la lista de la compra o el recetario?
- **USUARIO**: Me gustaría iniciar la lista de la compra
- **SISTEMA**: Estos son los alimentos almacenados en la lista de la compra. Si desea añadir un alimento diga Añadir y si desea eliminar uno de la lista pulse sobre él
- **USUARIO**: Pulsa sobre un alimento
- **SISTEMA**: ¿Seguro que desea eliminar este alimento?
- **USUARIO**: Sí
- **SISTEMA**: Alimento eliminado correctamente.

❖ Consulta de la lista de la Compra con errores en el reconocimiento

- **SISTEMA**: ¿Desea iniciar la lista de la compra o el recetario?
- **USUARIO**: Me gustaría iniciar la lista de la compra
- **SISTEMA**: No hemos comprendido su opción. Por favor, especifique de nuevo si desea iniciar la lista de la compra o el recetario
- **USUARIO**: Me gustaría iniciar la lista de la compra
- **SISTEMA**: Estos son los alimentos almacenados en su lista de la compra. Para añadir un alimento diga Añadir y para eliminar uno de la lista pulse sobre él.
- **USUARIO**: Añadir
- **SISTEMA**: No hemos comprendido su opción. Por favor, especifíquelo de nuevo.
- **USUARIO**: Añadir
- **SISTEMA**: Seleccione una de las siguientes categorías principales.

Capítulo 6

Módulo Recetario

En este módulo se realiza un análisis detallado de los diferentes submódulos que integran el módulo Recetario, describiendo las funcionalidades que ofrece al usuario, su arquitectura, los diagramas de flujo y los distintos escenarios de uso:

6.1 Submódulos de Ejecución

6.1.1 Submódulo Inicio

Este submódulo es común para los dos módulos que componen la aplicación, por lo que si desea obtener más información, consulte el Apartado 5.1.1

6.1.2 Submódulo Recetario

6.1.2.1 Funcionalidad

La funcionalidad de este submódulo es preguntar al usuario si desea consultar una receta o por el contrario desea que el sistema del proponga alguna.

Se ejecuta cuando el Gestor de Diálogo recibe el frame Acción General con el valor “Recetario” en el campo opción

6.1.2.2 Diagrama de flujo y Arquitectura

La Figura 49 ilustra el diagrama de flujo definido para este submódulo

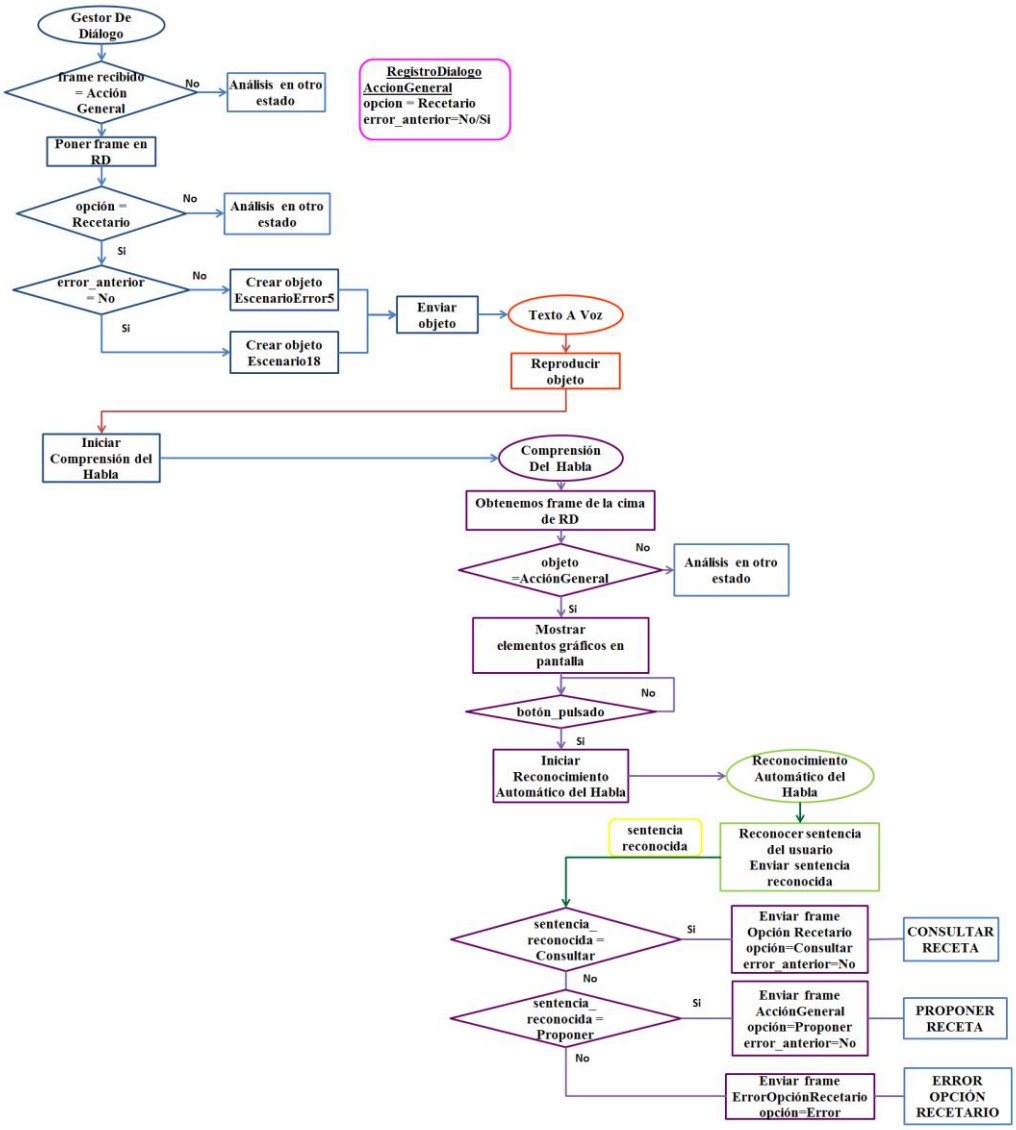

Figura 49: Diagrama de Flujo Submódulo Recetario

- La clase Gestor de Diálogo obtiene el frame Acción General y analiza el campo error_anterior.

  - → Si el campo error_anterior contiene el valor *"no"*: construye un objeto de tipo Escenario18 que pregunta al usuario si desea consultar una receta o que el sistema le proponga alguna.
  - → Si el campo error_anterior contiene el valor "*si*" construye un objeto de tipo EscenarioError5 que informa al usuario de que se ha producido un error en el reconocimiento.

- A continuación el Gestor de Diálogo inicia la actividad Comprensión del Habla.

- La actividad Comprensión del Habla obtiene el frame de la cima del Registro de Diálogo y muestra los elementos gráficos como se muestra en la Figura 50

Figura 50: Elementos gráficos Submódulo Recetario

El usuario para indicar si desea consultar el recetario o que el sistema le proponga nuevas recetas, pulsa sobre el botón de voz, sobre el que se registra el escuchador de eventos. Cuando salta el evento, la actividad Comprensión del Habla inicia el Reconocimiento de Voz.

- La actividad Reconocimiento Automático del Habla, reconoce la sentencia pronunciada por el usuario y la envía a la actividad Comprensión del Habla.

- La actividad Comprensión del Habla analiza la sentencia reconocida realizando una comparación con el conjunto de palabras claves definidas para este estado, generando un nuevo frame que enviará al Gestor de Diálogo para que determine el siguiente submódulo de ejecución.

  - → Si sentencia_reconocida contiene el valor *"Consultar"* : genera el frame OpciónRecetario con el valor "*Consultar*" en el campo opción, con lo que el siguiente submódulo de ejecución será Consultar Receta.
  - → Si sentencia_reconocida contiene el valor "*Proponer*": genera el frame OpciónRecetario con el valor "*Proponer*" en el campo opción, con lo que el siguiente submódulo de ejecución será Proponer Receta.
  - → Si sentencia_reconcida no contiene ninguna de las opciones anteriores: genera el frame ErrorOpciónRecetario, con lo que el siguiente submódulo de ejecución será Error Opción Recetario.

6.1.3 Submódulo Consultar Receta

6.1.3.1 Funcionalidad

La funcionalidad de este submódulo es la de mostrar las categorías principales de recetas en las que se engloban las recetas almacenadas en la

aplicación y reconocer la categoría la que pertenece la receta que el usuario desea consultar.

Este submódulo se ejecuta cuando el Gestor de Diálogo recibe el frame OpciónRecetario con el valor "*Consultar*" en el campo opción.

6.1.3.2 Diagrama de Flujo y Arquitectura

La Figura 51 muestra el diagrama de flujo definido por la aplicación para este submódulo.

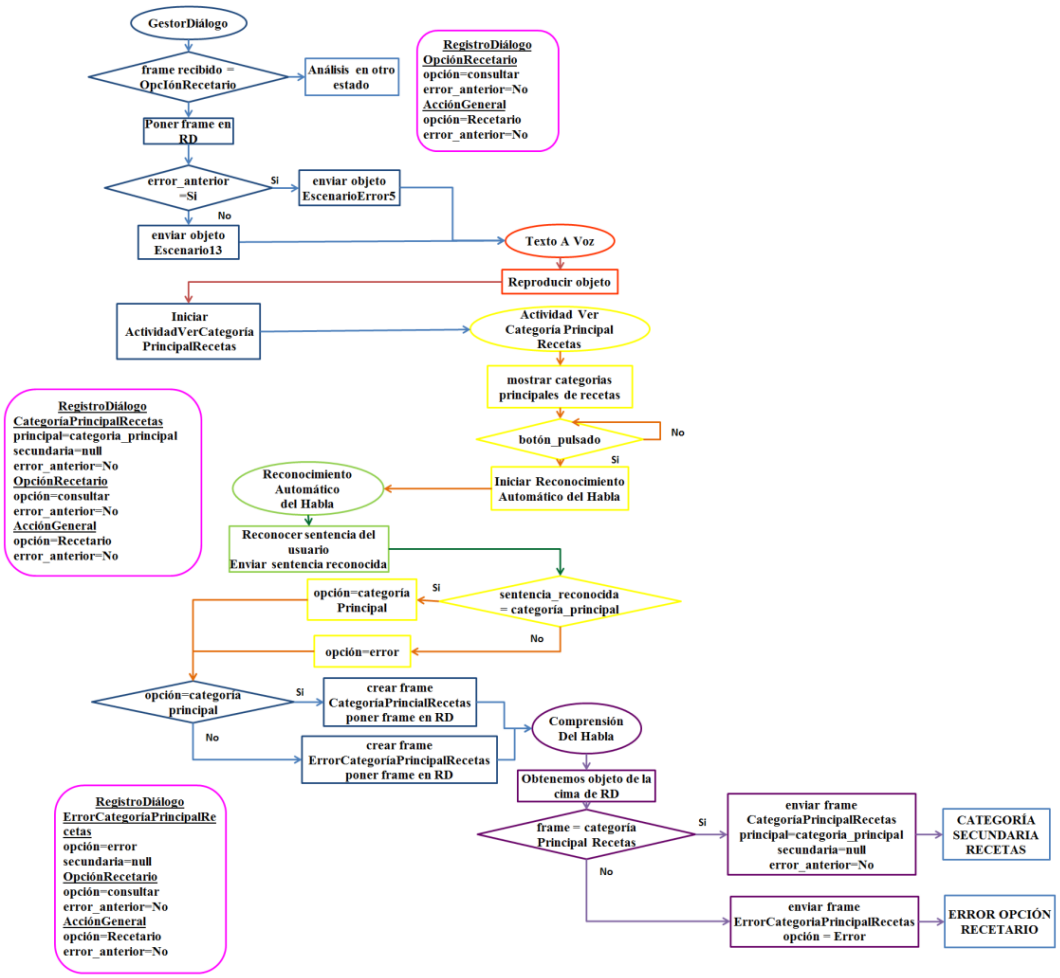

Figura 51: Diagrama de Flujo Submódulo Consultar Receta

❖ La clase Gestor Diálogo, cuando recibe el frame OpciónRecetario, analiza el valor el campo error_anterior:

→ Si el campo error_anterior contiene el valor "*no*" genera una sentencia de tipo Escenario 13 que pide al usuario que seleccione una de las categorías principales de recetas.

→ Si el campo error_anterior contiene el valor "*si*" genera una sentencia de tipo EscenarioError5, que informa al usuario que se ha producido un error en el reconocimiento y debe volver a indicar la categoría de receta que desea.

A continuación inicia la actividad Texto A Voz, pasando como parámetro la sentencia generada.

- La actividad Texto A Voz, reproduce la sentencia recibida y devuelve el control a la clase Gestor de Diálogo.

- El Gestor de Diálogo inicia la actividad Ver Categoría Principal Recetas.

- La Actividad Ver Categoría Principal Recetas muestra en pantalla las categorías principales de recetas definidas para esta aplicación. A continuación se muestra el fragmento de código definido para esta función.

```
lista=      (ListView)findViewById(R.id.listaCategoriaPrincipal);
Cursor c =GestorDialogo.bdh.obtenerCategoriaPrincipalRecetas();
if(c.getCount()!=0){if(c.moveToFirst()){do{
verCategoria=c.getString(1);
vector.add(verCategoria);}while(c.moveToNext());}
AdaptadorCategoriaPrincipal adaptador = new
AdaptadorCategoriaPrincipal(this, vector);
lista.setAdapter(adaptador);
```

  La actividad llama al método *obtenerCategoriaPrincipalRecetas* de la clase BaseDatosHelper, que devuelve en forma de objeto Cursor los nombres de las categorías principales. Una vez obtenidas, son almacenadas en un vector de string. A continuación, se instancia un objeto de tipo AdaptadorCategoriaPrincipal, clase que hereda de ArrayAdapter que junto con el elemento gráfico listaCategoriaPrincipal, mostrarán la información en pantalla como puede verse en la Figura 52.

Figura 52: Elementos Gráficos Submódulo Consultar Receta

- Cuando el usuario pulsa sobre el botón de voz, se inicia la actividad de Reconocimiento de Voz.

- La Actividad Reconocimiento Automático del Habla reconoce la sentencia del usuario y la envía a la Actividad Ver Categoría Principal Recetas.

- La actividad Ver Categoría Principal Recetas analiza la sentencia reconocida, procediendo del siguiente modo.

  - → Si sentencia_reconocida contiene una de las categorías principales de recetas, construye un objeto opción y almacena en él la categoría principal.
  - → Si sentencia_reconocida no contiene ninguna de las categorías principales, construye un objeto opción con el valor "*Error*" como atributo.

  En ambos casos el objeto opción es enviado a la actividad Gestor de Diálogo.

- La actividad Gestor de Diálogo analiza el objeto opción y en función de su contenido envía un determinado frame que colocará en la cima del Registro de Diálogo.

  - → Si opción es distinto de *"Error":* la actividad genera el frame CategoríaPrincipalRecetas con la categoría seleccionada por el usuario en el campo principal.
  - → Si opción contiene la palabra *"Error"*: la actividad genera el frame ErrorCategoríaPrincipalRecetas con el valor "*Error*" en el campo opción.

- La Actividad Comprensión Del Habla, obtiene el frame de la cima del Registro de Diálogo y en base a ello elabora un determinado frame que enviará al Gestor para que este pueda determinar el siguiente submódulo de ejecución:

  - → Si el frame de la cima del Registro de Diálogo es CategoríaPrincipalRecetas: genera el frame CategoríaPrincipalRecetas con el valor de la categoría seleccionada por el usuario en el campo principal, por lo que el siguiente submódulo de ejecución será Categoría Secundaria Recetas.
  - → Si el frame de la cima del Registro de Diálogo es ErrorCateogoríaPrincipalRecetas genera el frame CategoríaPrincipalRecetas con el campo principal a null y el valor "*Si*" en el campo error_anterior, por lo que el siguiente submódulo de ejecución será Error Opción Recetario.

6.1.4 Submódulo Categoría Secundaria Recetas

6.1.4.1 Funcionalidad

Este submódulo se ejecuta cuando el Gestor recibe el frame CateogoríaPrincipalRecetas.

6.1.4.2 Diagrama de Flujo y Arquitectura

La Figura 53 muestra el diagrama de flujo definido para este submódulo.

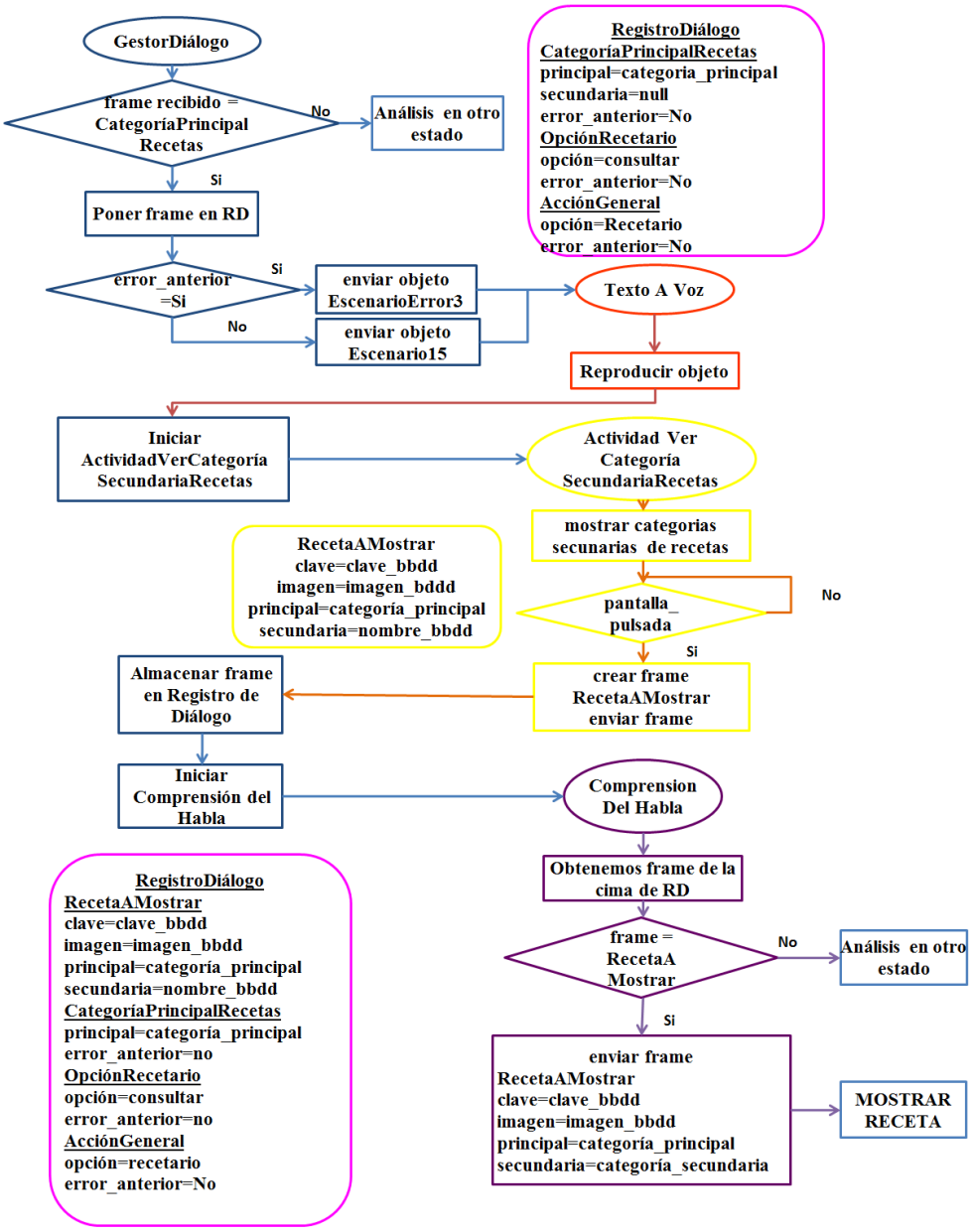

Figura 53: Diagrama de Flujo Submódulo Categoría Secundaria Recetas

- El Gestor de Diálogo recibe el frame CategoríaPrincipalRecetas y analiza el valor del campo error_anterior.

  → Si error_anterior= "no", construye un objeto de tipo Escenario15 que pedirá al usuario que indique una de las categorías secundarias de recetas.

  → Si error_anterior="si", construye un objeto de tipo EscenarioError3, que informará al usuario que se ha producido un error en el reconocimiento y debe volver a indicar la categoría secundaria de recetas.

  El objeto se envía a la actividad TextoAVoz.

- La actividad TextoAVoz reproduce la sentencia recibida y retorna el control al Gestor.

- El Gestor de Diálogo inicia la Actividad Ver Categoría Secundaria Recetas.
- La Actividad Ver Categoría Secundaria Recetas muestra las categorías secundarias de recetas englobadas dentro de la categoría principal que indicó el usuario. A continuación se muestra el fragmento de código definido para esta funcionalidad.

```
lista=(ListView)findViewById(R.id.listaCategoriaPrincipal);
Cursor c =GestorDialogo.bdh.obtenerCategoriaSecundariaRecetas(principal);
do{
verClave = c.getString(0);
verImagen = c.getString(1);
verCategoriaPrincipal=c.getString(2);
verCategoriaSecundaria =c.getString(3);
receta_a_mostrar = new
RecetaAMostrar(verClave,verImagen,verCategoriaPrincipal,verCategoriaSecun
daria);
vector.add(receta_a_mostrar);
AdaptadorRecetasAMostrar adaptador = new AdaptadorRecetasAMostrar(this,
vector);
lista.setAdapter(adaptador);
```

La actividad llama al método *obtenerCategoriaSecundariaRecetas* de la clase Base de Datos Helper, pasando como parámetro la categoría principal que fue almacenada en el frame CategoríaPrincipalRecetas. Este método devuelve en forma de cursor, la siguiente información relativa a las recetas englobadas dentro de la categoría principal seleccionada por el usuario:

- Clave = clave primaria con la que la receta se almacena en la base de datos.
- Imagen = imagen descriptiva de la receta.
- Categoría primaria = categoría dentro de la cuál se engloba la receta.
- Categoría secundaria = nombre con el que la receta se almacena en la base de datos.

La información es almacenada en el frame RecetaAMostrar. Para mostrar esta información por pantalla, se instancia un objeto de tipo AdaptadorRecetaAMostrar, clase que hereda de ArrayAdapter, que junto con el elemento gráfico listaCategoríaPrincipal, mostraran al usuario la información en pantalla, tal y como puede apreciarse en la Figura 54.

Figura 54: Elementos Gráficos Submódulo Categoría Secundaria Recetas.

Sobre la lista de elementos, se registra un escuchador de eventos, de modo que la aplicación permanece en espera activa hasta que el usuario pulsa sobre la pantalla. Cuando esto sucede, el frame RecetaAMostrar es enviado al Gestor de Diálogo.

- El Gestor de Diálogo almacena el frame en la cima del Registro e inicia la actividad Comprensión del Habla.
- La actividad Comprensión del Habla, obtiene el frame de la cima del Registro y lo enviará al Gestor para que lleve a cabo la ejecución del siguiente submódulo, que será Mostrar Receta.

6.1.5 Submódulo Mostrar Receta

6.1.5.1 Funcionalidad

La funcionalidad de este submódulo es la de mostrar en pantalla la información de la Receta que el usuario desea consultar. Este módulo ofrece además la posibilidad de que el usuario pueda escuchar las instrucciones a seguir para elaborar una receta, pudiendo repetir la reproducción de cada una de ellas las veces que lo desee. Se ejecuta cuando el Gestor de Diálogo recibe el frame RecetaAMostrar, con la información específica de la receta.

6.1.5.2 Diagrama de Flujo y Arquitectura

La Figura 55 muestra el diagrama de flujo definido para este submódulo.

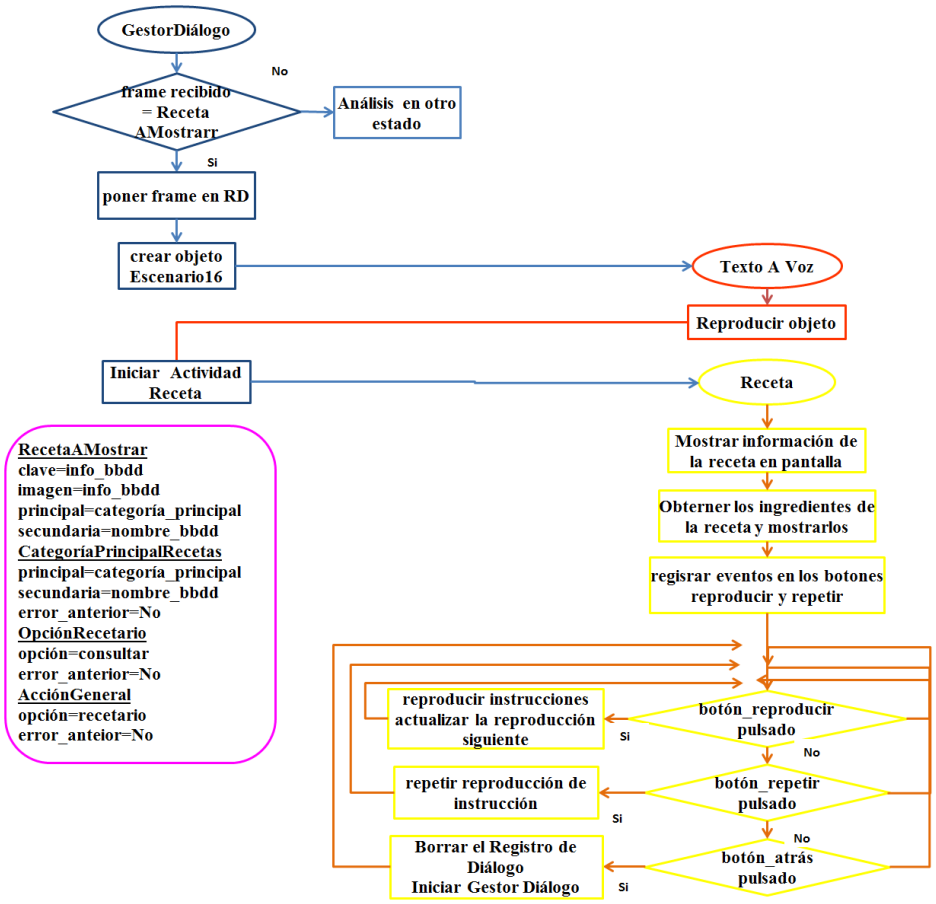

Figura 55: Diagrama de Flujo Submódulo Mostrar Receta

- El Gestor Dialogo recibe el frame RecetaAMostrar y lo coloca en la cima del Registro de Diálogo. Crea un objeto de tipo Escenario16, que informa al usuario de que los datos de la receta se mostrarán en pantalla y que si desea reproducir una instrucción, pulse sobre el botón reproducir y si desea volver a escuchar una instrucción pulse sobre el botón repetir. La sentencia es enviada a la actividad TextoAVoz.

- La actividad TextoAVoz, reproduce la sentencia recibida por el usuario y devuelve el control a la clase Gestor de Diálogo.

- El Gestor de Diálogo, inicia la actividad Receta.

  - La actividad Receta, muestra la siguiente información de la receta:

    - Nombre.
    - Imagen.
    - Ingredientes.
    - Instrucciones.

Para obtener los ingredientes, como puede verse en el siguiente fragmento de código, se ejecuta el método *buscarIngredientes* de la clase Base de Datos Helper, pasando como parámetro la clave primaria de la receta que fue almacenada en el frame RecetaAMostrar.

```
public static ArrayList<DatosTablaIngredientes> ingredientes;
public Vector<DatosTablaIngredientes> vectorIngredientes = new Vector();
ingredientes = GestorDialogo.bdh.buscarIngredientes(c);
ListView lista = (ListView)findViewById(R.id.lista_ingredientes);
for(int i =
0;i<ingredientes.size();i++){vectorIngredientes.addElement(ingredientes.get(i))
;}
AdaptadorIngredientes adaptador = new
AdaptadorIngredientes(this,vectorIngredientes);
lista.setAdapter(adaptador);
```

Este método devuelve cada uno de los ingredientes necesarios para elaborar la receta, con la correspondiente cantidad en gramos. Para recoger esta información, en la aplicación se define el objeto DatosTablaIngredientes que contiene dos atributos:

- string ingrediente: almacena el ingrediente obtenido en la consulta a la base de datos.
- string cantidad: almacena la cantidad correspondiente al ingrediente.

Para cada par ingrediente-cantidad, se crea un objeto DatosTablaIngredientes, que es almacenado en un vector denominado vectorIngredientes.

A continuación se instancia un objeto adaptador de tipo AdaptadorIngredientes, clase definida para la aplicación, que junto con el elemento gráfico lista de tipo ListView, mostrarán en pantalla la información de los ingredientes como se muestra en la Figura 56

Figura 56: Elementos gráficos para mostrar los Ingredientes

Para obtener las instrucciones, como puede verse en el siguiente fragmento de código, se ejecuta el método *buscarInstrucciones* de la clase Base de Datos Helper, pasando como parámetro la clave primaria de la receta que fue almacenada en el frame RecetaAMostrar.

```
public static ArrayList<DatosTablaInstrucciones> instrucciones;
public Vector<DatosTablaInstrucciones> vectorInstrucciones = new Vector();
ListView lista = (ListView)findViewById(R.id.lista_instrucciones);
instrucciones = GestorDialogo.bdh.buscarInstrucciones(c);
for(int i =
0;i<instrucciones.size();i++){vectorInstrucciones.addElement(instrucciones.ge
t(i));}
AdaptadorInstrucciones adaptador = new
AdaptadorInstrucciones(this,vectorInstrucciones);
lista.setAdapter(adaptador);
```

Este método devuelve las instrucciones a seguir para elaborar la receta. Para recoger esta información, en la aplicación se define el objeto DatosTablaInstrucciones que contiene el atributo de tipo string instrucciones. Para cada instrucción obtenida de la base de datos, se instancia un objeto DatosTablaInstrucciones, que es almacenado en un vector denominado instrucciones.

A continuación, se instancia un objeto adaptador de tipo AdaptadorInstrucciones, clase definida para la aplicación, que junto con el elemento gráfico lista de tipo ListView, mostrarán en pantalla la información de los ingredientes como se muestra en la Figura 57.

Figura 57: Elementos gráficos para mostrar las instrucciones

6.1.6 Submódulo Proponer Receta

6.1.6.1 Funcionalidad

La funcionalidad de este submódulo es la de seleccionar una receta entre todas las definidas en la aplicación, obtener su información correspondiente de la base de datos y almacenarla en el frame RecetaAMostrar.

Este submódulo posee cierta "inteligencia", debido a que la selección de la receta se realiza teniendo en cuenta la categoría que más veces ha consultado el usuario.

Se ejecuta cuando el Gestor de Diálogo recibe el frame OpciónRecetario con el valor "*proponer"* en el campo opción.

6.1.6.2 Diagrama de Flujo y Arquitectura

La Figura 58 muestra el diagrama de flujo definido por la aplicación para este submódulo.

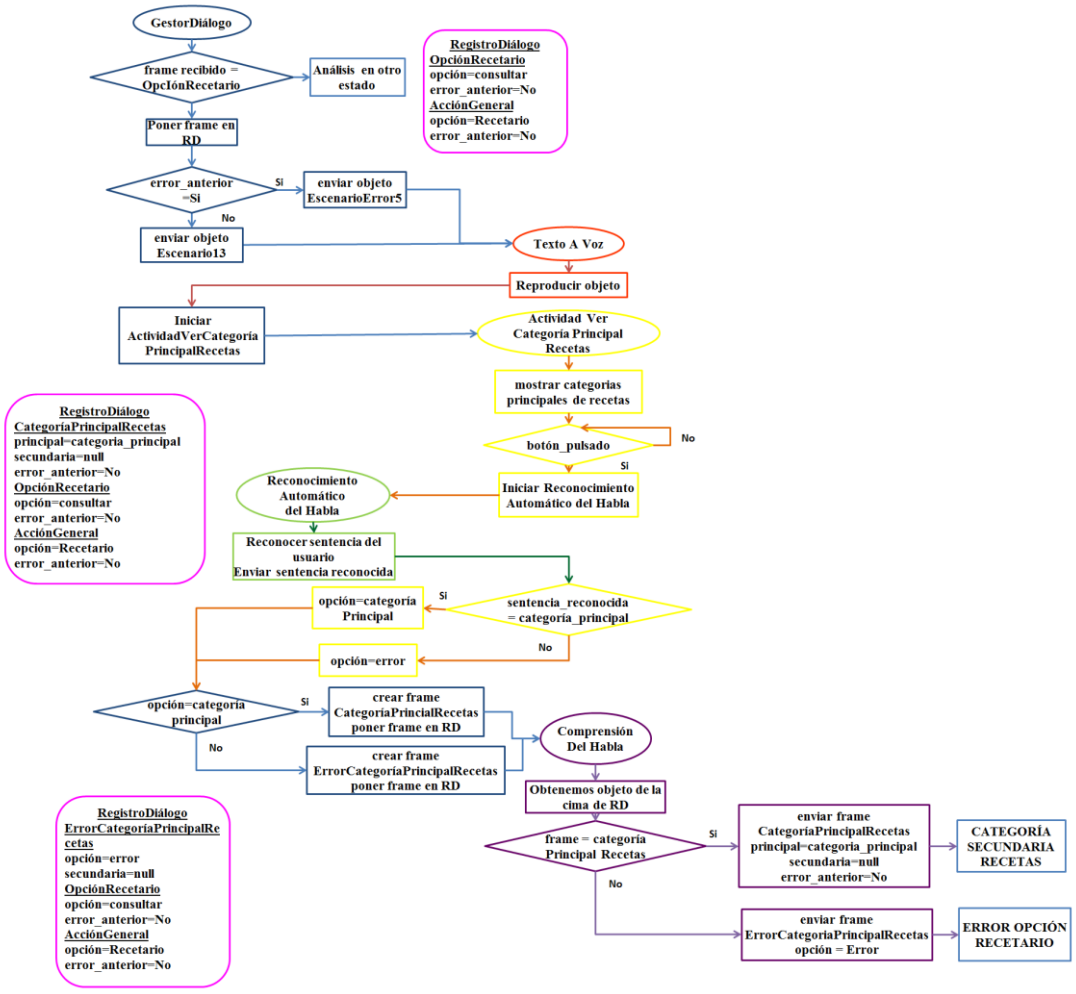

Figura 58: Diagrama de Flujo Submódulo Proponer Receta

El Gestor de Diálogo cuando recibe el frame OpciónRecetario con el valor "*proponer*" en el campo opción, analiza el valor del campo error_anterior.

- Si error_anterior = si, construye un objeto de tipo EscenarioError5 que informa al usuario de que se ha producido un error en el reconocimiento.
- Si error_anterior = no, construye un objeto de tipo Escenario15, que directamente informa al usuario de que el sistema procederá a mostrar la receta.

En ambos casos el objeto es enviado a la actividad TextoAVoz.

- La Actividad Texto A Voz, reproduce el objeto recibido y devuelve el control al Gestor.
- El Gestor de Diálogo inicia la Actividad Proponer Receta.
- La Actividad Proponer Receta es la encargada de seleccionar la receta y obtener su información de la base de datos. Para ello primeramente ejecuta el método

obtenerContadores() de la clase Base de Datos Helper, como puede apreciarse en el siguiente fragmento de código.

```
Cursor c = GestorDialogo.bdh.obtenerContadores();
if(c.moveToFirst()){do{
nombre=c.getString(0);
contador = c.getString(1);
ContadorRecetas cr = new ContadorRecetas(nombre,contador);
cont_rec.add(cr);
}while(c.moveToNext());
return cont_rec;
```

Este método devuelve los valores de los campos "contador" de la Tabla Primera Categoría Recetas, asociados a cada una de las categorías de recetas. El campo contador, se va incrementando en una unidad, cada vez que el usuario consulta una receta de esa categoría.

A continuación, como recoge el siguiente fragmento de código, la aplicación ejecuta el método *obtenerNombreRecetaPreferida*, que recibe como parámetro el array de contadores obtenido en el método anterior y devuelve la categoría que contiene el contador con el valor más alto.

```
String obtenerNombreRecetaPreferida(ArrayList<ContadorRecetas> cr){
String nombre = "";
int numero = 0;
int n = 0;
for(int i = 0; i<cr.size();i++){
ContadorRecetas aux = cr.get(i);
n = Integer.parseInt(aux.getContador());
if(n>=numero){
numero = n;
nombre = aux.getNombre();}}
return nombre}
```

Acto seguido, como muestra el fragmento de código siguiente, la aplicación ejecuta el método *obtenerSubcategoríasRecetas()*, que recibe como parámetro la categoría obtenida en el método anterior, y con dicho parámetro ejecuta el método *obtenerSubCategoríasRecetas* de Base de Datos Helper, que devuelve en forma de objeto Cursor los nombres de las recetas englobadas dentro de esa categoría. Esta información es almacenada en un array de objetos SubcategoriasRecetas.

```
public ArrayList<SubcategoriasRecetas> obtenerSubcategoriasRecetas(String
nombre){
ArrayList<SubcategoriasRecetas> sub_rec = new
ArrayList<SubcategoriasRecetas>();
int contador = 1;
Cursor c = GestorDialogo.bdh.obtenerSubcategoriasRecetas(nombre);
if(c.moveToFirst()){do{
principal=c.getString(0);
secundaria = c.getString(1);
numero = contador;
contador++;
```

```
SubcategoriasRecetas sr  = new SubcategoriasRecetas(principal, secundaria,
numero);
sub_rec.add(sr);
}while(c.moveToNext());}
GestorDialogo.bdh.close();
return sub_rec;
```

Los objetos SubcategoríasRecetas, son objetos auxiliares definidos en la aplicación que constan de los siguientes atributos:

- String principal : almacena la categoría principal en la que se engloba la receta.
- String secundaria: almacena el nombre de la receta.
- Int contador: número inicial asociado a cada objeto SubcategoríaRecetas.

A continuación, entre todas las recetas obtenidas, se selecciona una de manera aleatoria. Para ello, como puede verse en el siguiente fragmento de código, la aplicación llama al método *obtenerNombreRecetaAleatoria*, pasando como parámetro el array de objetos SubcategoríaRecetas obtenido en el método anterior. Este método genera un número aleatorio comprendido entre 1 y el tamaño del vector pasado como parámetro. Este número se compara con el valor el atributo *"contador*" de cada uno de los objetos SubcategoríaRecetas hasta encontrar el que coincide. Cuando eso sucede, el método devuelve el atributo secundaria, que como hemos comentado anteriormente, contendrá el nombre de la receta elegida aleatoriamente.

```
public String
obtenerNombreRecetaAleatoria(ArrayList<SubcategoriasRecetas> sr){
int n = (int)(Math.random()*10+1);
String receta_aleatoria="";
boolean encontrado = false;
do{for(int i = 0;i<sr.size();i++){
SubcategoriasRecetas aux = sr.get(i);
if(aux.getNumero()==n){
receta_aleatoria = aux.getSecundaria();
encontrado = true;}}
n = (int)(Math.random()*10+1);}
while(encontrado==false);
return receta_aleatoriea;}
```

Finalmente, la actividad genera el frame RecetaAMostrar, obteniendo de la base de datos la información relativa a la receta aleatoria seleccionada y lo envía al Gestor Diálogo, por lo que el siguiente submódulo a ejecutar será el submódulo Mostrar Receta.

6.2 Submódulos de Errores

Durante la ejecución del Módulo Recetario, los reconocimientos por parte de la aplicación de las sentencias emitidas por el usuario pueden fallar, por lo que es necesario llevar a cabo una gestión de errores.

En los submódulos Consultar Receta, Categoría Secundaría Receta y Mostrar Receta, la gestión de errores se realiza mediante el campo error_anterior de los frames intercambiados, debido a que el Gestor de Diálogo, solamente deberá decidir entre dos opciones de ejecución:

- Pasar al submódulo siguiente lanzando un mensaje informativo al usuario si error_anterior contiene el valor "*no*".
- Permanecer en el mismo submódulo lanzando un mensaje de error al usuario y pidiéndole que vuelva a repetir la sentencia pronunciada, si error_anterior contiene el valor "*si*".

En el submódulo OpciónRecetario, el Gestor de Diálogo debe decidir en función de la sentencia recibida, si iniciará la fase de consulta de recetas, la fase de proponer recetas o la gestión de errores.

Para simplificar el proceso, se crea un submódulo de gestión de errores que se describe en el siguiente apartado.

6.2.1 Submódulo Error Opción Recetario

6.2.1.1 Funcionalidad

Este submódulo se ejecuta cuando el Gestor de Diálogo recibe el frame ErrorOpciónRecetario y su finalidad es la de informar al usuario que se ha producido un error en el reconocimiento, por lo que debe volver a indicar si desea consultar una receta o que el sistema le proponga alguna.

6.2.1.2 Diagrama de Flujo y Arquitectura.

La Figura 59 muestra el diagrama de flujo definido para la aplicación.

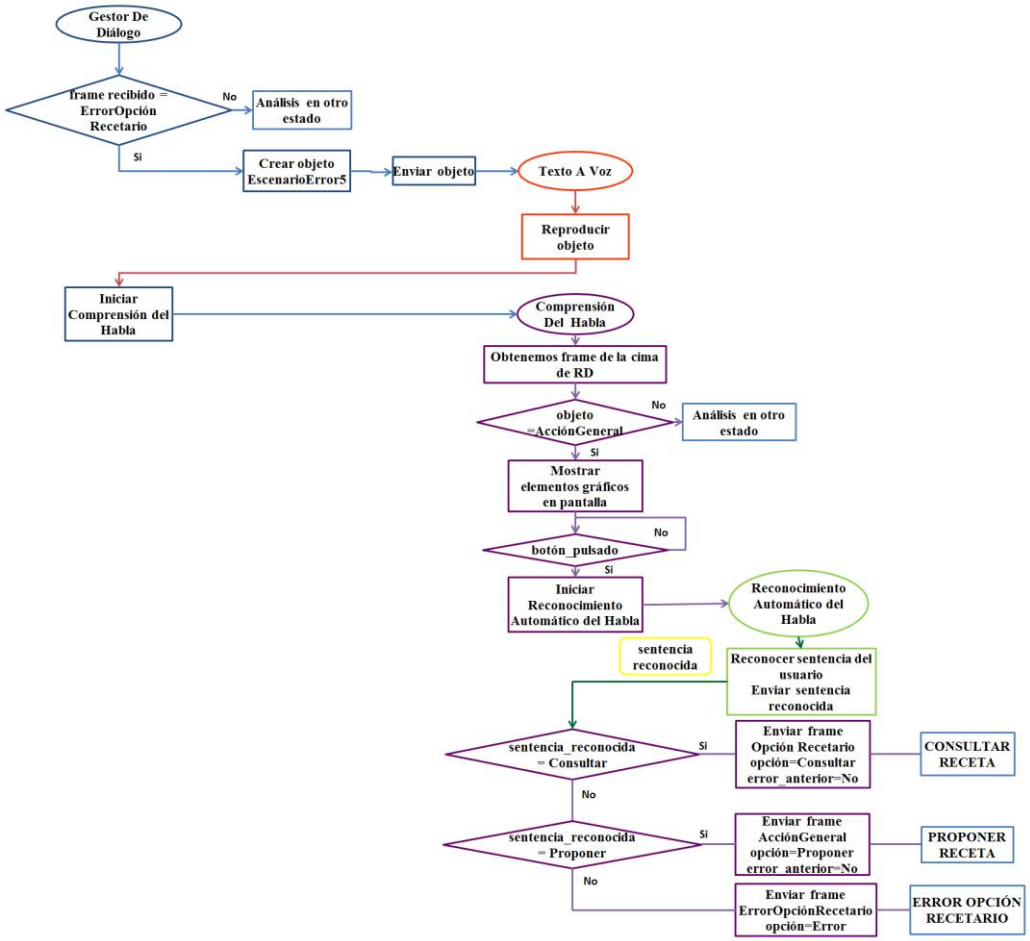

Figura 59: Diagrama de Flujo Submódulo Error Opción Recetario

- El Gestor de Diálogo cuando recibe el frame ErrorOpciónRecetario genera una sentencia de tipo EscenarioError5 que informa del error al usuario, e inicia la actividad Texto A Voz.
- La actividad Texto A Voz reproduce la sentencia recibida y devuelve el control al Gestor.
- A partir de este punto, el Gestor de Diálogo sigue la misma estrategia que la comentada en el submódulo Recetario (ver Apartado 6.1.2).

6.3 Escenarios Conversacionales

Para ilustrar de forma más concreta el diálogo producido entre la aplicación y el usuario, se muestran a continuación algunos escenarios conversacionales que se producen durante la ejecución del Módulo Recetario.

- Saludo inicial mas consulta de recetas.

- **SISTEMA:** Bienvenido a la aplicación. ¿Desea iniciar la lista de la compra o el recetario?
- **USUARIO:** Me gustaría iniciar el Recetario.
- **SISTEMA:** ¿Desea consultar una receta o desea que el sistema le proponga alguna?
- **USUARIO:** Me gustaría consultar una receta.
- **SISTEMA:** Seleccione una de las siguientes categorías de recetas.
- **USUARIO:** Arroces
- **SISTEMA:** Pulse sobre la receta que desee visualizar
- **USUARIO:** Pulsa sobre la receta.
- **SISTEMA:** Procedemos a mostrar la receta. Si desea activar la reproducción de instrucciones pulse el botón reproducir y si desea volver a oir una instrucción pulse sobre el botón repetir.

❖ Saludo inicial más proponer receta.

- **SISTEMA:** Bienvenido a la aplicación. ¿Desea iniciar la lista de la compra o el recetario?
- **USUARIO:** Querría iniciar el recetario.
- **SISTEMA:** ¿Desea consultar una receta o desea que el sistema le proponga alguna?
- **USUARIO:** Me gustaría que el sistema me proponga una receta.
- **SISTEMA:** Le proponemos la siguiente receta. Procedemos a mostrar la receta. Si desea activar la reproducción de instrucciones pulse el botón repetir y si desea volver a oir una instrucción pulse el botón repetir

❖ Saludo inicial y consulta de receta con errores.

- **SISTEMA:** Bienvenido a la aplicación. ¿Desea iniciar la lista de la compra o el Recetario?
- **USUARIO:** Me gustaría iniciar el Recetario
- **SISTEMA:** No hemos comprendido su opción. Por favor especifique de nuevo si desea iniciar la lista de la compra o el recetario.
- **USUARIO:** Me gustaría iniciar el Recetario
- **SISTEMA:**¿Desea consultar una receta o desea que el sistema le proponga alguna?
- **USUARIO:** Me gustaría consultar una receta.
- **SISTEMA:** No hemos comprendido su opción. Por favor diga de nuevo si desea consultar una receta o desea que el sistema le proponga alguna.
- **USUARIO:** Me gustaría consultar una receta.
- **SISTEMA:** Seleccione una de las siguientes categorías de recetas.

Capítulo 7

Evaluación, Planificación y Presupuesto

En la primera parte de este capítulo se recogen los resultados obtenidos en la evaluación que se hizo a diez usuarios que probaron la aplicación. En la segunda parte se detalla la planificación de las distintas fases del desarrollo del proyecto. Finalmente el capítulo concluye con el presupuesto en euros de la aplicación desarrollada.

7.1 Evaluación

Con el fin de evaluar la aplicación desarrollada, se realizó un pequeño cuestionario a 10 usuarios que probaron la aplicación en sus dispositivos móviles.

Para medir el grado de satisfacción y las ventajas que aporta la implementación de estrategias de diálogo en aplicaciones móviles, 4 de los usuarios seleccionados son personas mayores de 60 años con desventajas visuales.

Los resultados obtenidos en la evaluación junto con las conclusiones obtenidas en cada una de las preguntas, se recogen en las Tablas 1 a 9.

1. Puntúe de 1 a 5 su grado de familiarización previo con aplicaciones móviles siendo 1 el grado máximo de familiarización y 5 el grado mínimo

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
1				X	X	X				
2		X	X				X	X		X
3	X								X	
4										
5										

Tabla 1: Valoración del grado de familiarización con aplicaciones móviles

- **Conclusión**: La mayoría de los usuarios presenta un alto grado de familiarización con las aplicaciones móviles

2. ¿Anteriormente había utilizado aplicaciones basadas en interfaces de voz?

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
Muchas veces										
De vez en cuando										
Pocas veces		X	X			X	X	X	X	
Nunca	X			X	X					X

Tabla 2: Valoración del uso de aplicaciones basadas en interfaces de voz

- **Conclusión**: Todos los usuarios encuestados estaban muy poco familiarizados con el uso de aplicaciones basadas en interfaces de voz.

3. Como valoraría al eficiencia de la aplicación en lo referente al reconocimiento de voz

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
Excelente				X	X					
Bien	X		X			X	X	X		X
Regular		X							X	
Mala										
Muy Mala										

Tabla 3: Valoración de la eficiencia del reconocimiento de voz

- **Conclusión**: El 80% de los usuarios de los encuestados considera que el reconocimiento de voz se realiza de forma eficiente.

4. Cómo valoraría la eficiencia de la aplicación en lo referente a la síntesis de voz

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
Excelente				X	X					
Bien	X	X	X				X	X		X
Regular						X			X	
Mala										
Muy Mala										

Tabla 4: Valoración de la eficiencia de la síntesis de voz

- **Conclusión**: El 80% de los usuarios encuestados considera que la síntesis de voz se realiza de forma eficiente

5. Cómo valoraría el tiempo de respuesta de la aplicación:

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
Muy rápido										
Rápido	X			X	X		X	X		
Normal		X	X			X			X	X
Lento										
Muy Lento										

Tabla 5: Valoración del tiempo de respuesta de la aplicación

- **Conclusión**: Un 50% de los encuestados considera que el tiempo de respuesta de la aplicación es rápido mientras que el otro 50% considera aceptable.

6. En su opinión, valore la sencillez de uso de la aplicación

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
Muy Sencilla		X	X				X			
Sencilla	X			X	X	X			X	X
Normal								X		
Poco Sencilla										
Muy poco sencilla										

Tabla 6: Valoración de la sencillez de uso de la aplicación

- **Conclusión**: Un 30% de los encuestados considera que el uso de la aplicación es bastante sencillo, un 60% considera que es sencillo y un 10% califica la sencillez de uso de la aplicación como aceptable.

7. ¿Considera que el reconocimiento de voz ha facilitado la interacción con la aplicación? Puntúe del 1 al 5 siendo,1 la mejor puntuación.

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
1		X	X							X
2	X			X	X	X	X		X	
3								X		
4										
5										

Tabla 7: Valoración de las ventajas del reconocimiento de voz

- **Conclusión**: Un 95% considera que el reconocimiento de voz ha contribuido a facilitar la interacción con la aplicación.

8. ¿Considera que la síntesis de voz ha facilitado la interacción con la aplicación? Puntúe del 1 al 5, siendo 1 la mejor puntuación.

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
1			X						X	
2	X	X		X	X	X	X	X		X
3										
4										
5										

Tabla 8: Valoración de las ventajas de la síntesis de voz

- **Conclusión**: Todos los usuarios consideran que la síntesis de voz ha contribuido a facilitar la interacción con la aplicación.

9. Valore globalmente la aplicación

	Usuario 1	Usuario 2	Usuario 3	Usuario 4	Usuario 5	Usuario 6	Usuario 7	Usuario 8	Usuario 9	Usuario 10
Excelente					X				X	
Buena	X	X	X	X			X	X		X
Normal						X				
Mala										
Muy Mala										

Tabla 9: Valoración global de la aplicación

- **Conclusión**: En líneas generales, un 20% de los encuestados considera que la aplicación es excelente, un 70% considera que es buena y un 10% valora la aplicación como aceptable.

7.2 Planificación

La Tabla 10: **Evolución Temporal de las Distintas Fases de Desarrollo del Proyecto**Tabla 10 recoge un desglose de las tareas realizadas para el desarrollo e implementación de la aplicación. Para cada tarea, se especifica la fecha de inicio, la fecha de fin y la duración en semanas.

TAREA	FECHA INICIO	FECHA FIN	DURACIÓN
Fase de Estudio	**07/07/2014**	**31/08/2014**	**7**
Estudio de la plataforma Android	07/07/2014	03/08/2014	4
Estudio de las estrategias de diálogo	11/08/2014	24/08/2014	2
Planteamiento y estructura de la aplicación	25/08/2014	31/08/2014	1
Fase de Preparación de las herramientas	**06/10/2014**	**12/10/2014**	**1**
Instalación y configuración del entorno de desarrollo	06/10/2014	12/10/2014	1
Fase del desarrollo Módulo Lista de la Compra	**03/11/2014**	**14/06/2014**	**29**
Inicio	03/11/2014	16/11/2014	2
Añadir Eliminar Alimentos	17/11/2014	30/11/2014	2
Categoría Principal	01/12/2014	14/12/2014	2
Categoría Secundaria	15/12/2014	01/01/2015	2
Categoría Terciaria	12/01/2014	25/01/2015	2
Categoría Cuaternaria	26/01/2015	08/02/2015	2
Categoría Quinta	09/02/2015	22/02/2015	2
Fin Categorías	23/02/2015	01/03/2015	1
Categorías Finalizadas	09/03/2015	29/03/2015	3
Confirmar Añadir Alimento	30/03/2015	12/04/2015	3
Añadir Alimento a BBDD	13/04/2015	19/04/2015	1
Confirmar Eliminar Alimento Opción: No	27/04/2015	03/05/2015	1
Confirmar Eliminar Alimento Opción: Sí	04/03/2015	10/05/2015	1
Error Acción General	11/05/2015	17/05/2015	1
Error Confirmación Añadir	18/05/2015	24/05/2015	1
Error Confirmación Eliminar	25/05/2015	31/05/2015	1
Error Cantidad	01/06/2015	07/06/2015	1
Error Acción Lista Compra	08/06/2015	14/06/2014	1
Fase del desarrollo Módulo Recetario	**15/06/2015**	**27/09/2015**	**9**
Recetario	15/06/2015	28/06/2015	2
Consultar Receta	29/06/2015	12/07/2015	2
Categoría Secundaria Receta	13/07/2015	19/07/2015	1
Mostrar Receta	20/07/2015	26/07/2015	1
Proponer Receta	27/07/2015	09/08/2015	2
Error Opción Recetario	10/08/2015	16/08/2015	1
Memoria	**24/08/2015**	**27/09/2015**	**5**

Tabla 10: Evolución Temporal de las Distintas Fases de Desarrollo del Proyecto

	Memoria	Error Opción Recetario	Proponer Receta	Mostrar Receta	Categoría Secundaria Receta	Consultar Receta	Recetario	Error Acción Lista Compra	Error Cantidad	Error Confirmación Eliminar	Error Confirmación Añadir	Error Acción General	Confirmar Eliminar Alimento Opción Sí	Confirmar Eliminar Alimento Opción No	Añadir Alimento a BBDD	Confirmar Añadir Alimento	Categorías Finalizadas	Fin Categorías	Categoría Quinta	Categoría Cuaternaria	Categoría Tercíaria	Categoría Secundaria	Categoría Principal	Añadir Eliminar Alimentos	Inicio	Instalación y configuración del entorno de desarrollo	Planteamiento y estructura de la aplicación	Estudio de las estrategias de codigo	Estudio de la plataforma Android
FECHA INICIO	24/08/2014	10/08/2014	27/07/2014	20/07/2014	13/07/2014	29/06/2014	15/06/2014	08/06/2014	01/06/2014	25/05/2014	18/05/2014	11/05/2014	04/05/2014	27/04/2014	13/04/2014	30/03/2014	09/03/2014	23/02/2014	09/02/2014	26/01/2014	12/01/2014	29/12/2014	01/12/2014	17/11/2014	03/11/2014	06/10/2014	29/08/2014	11/08/2014	07/07/2014
DURACIÓN	33	7	14	7	7	14	14	7	7	7	7	7	7	7	7	21	21	7	14	14	14	14	14	14	14	7	7	14	28

Tabla 11: Diagrama de Gantt con la planificación temporal del PFC

7.3 Presupuesto

En este apartado se refleja el presupuesto del proyecto, teniendo en cuenta los recursos necesarios para su ejecución y el tiempo requerido para su diseño y desarrollo.

Desarrollador: Marina Malo Malo

Categoría. Ingeniera Técnica de Telecomunicaciones

Tiempo Trabajado. 357 días (12 meses)
Salario mensual: 1.200 €/mes

→ **Total personal: 14.400 €**

Software de desarrollo: 750 €
Hardware de desarrollo: 200€

→ **Total material: 950 €**

→ **Suministros eléctricos: 750 €**

→ **Desplazamientos: 300 €**

El presupuesto total de este proyecto asciende a la cantidad de 16.400 € sin IVA

Leganés, a 29 de Septiembre de 2015

El ingeniero proyectista: Marina Malo Malo

Capítulo 8

Conclusiones y Trabajo Futuro

En la primera parte de este Capítulo se realiza un balance de la aplicación desarrollada para el presente Proyecto Fin de Carrera, en base a los objetivos fijados en el Capítulo 1.

En la segunda parte se estudian las líneas de trabajo futuras que se podrían desarrollar para hacer mejorar el diseño de la aplicación y el sistema de diálogo implementado en ella.

8.1 Conclusiones

Como se comentó en el apartado 1.5, el objetivo fundamental de este Proyecto Fin de Carrera era el desarrollo de una aplicación en el sistema operativo Android, que basase su ejecución en una estrategia de diálogo. Como hemos comentado en los Capítulos 5 y 6, el usuario va seleccionando las distintas funcionalidades que ofrecen los módulos Lista de la Compra y Recetario mediante la voz. Al mismo tiempo, la aplicación va guiando al usuario informando de las distintas opciones que ofrecen los módulos mediante la síntesis de voz.

Volviendo a los objetivos marcados en el Apartado 1.5, analizaremos a continuación los resultados obtenidos:

- Naturalidad en la conversación: a pesar de que el sistema de diálogo implementado se corresponde con un modelo de lenguaje restringido, con un vocabulario determinado, la estrategia de reconocimiento implementada se basa en la búsqueda de palabras clave, dando al usuario una mayor naturalidad a la hora de expresarse.

Comentaremos a modo de ejemplo la elección del módulo lista de la compra. En la sentencia emitida por el usuario, se comparan cada una de las palabras reconocidas

con los valores "*Lista*", "*Compra*", "*Listas*" y "*ListaCompra*", proporcionando al usuario una mayor libertad a la hora de expresarse:

- **SISTEMA:** ¿Desea iniciar la Lista de la compra o el Recetario?
- **USUARIO:** Me gustaría iniciar la Lista de la Compra

- **SISTEMA:** ¿Desea iniciar la Lista de la Compra o el Recetario?
- **USUARIO:** Querría iniciar la Lista de la Compra

- **SISTEMA:** ¿Desea iniciar la Lista de la Compra o el Recetario?
- **USUARIO:** La Lista de la Compra por favor.

❖ Facilitar la interacción con las aplicaciones móviles a personas con discapacidad. La conclusión a este objetivo, se ve reflejada en las Tablas 7 y 8, que reflejan los datos de la evaluación realizada a los usuarios. Los usuarios 3, 4,9 y 10, son los que presentaban desventajas visuales y como puede apreciarse en las tablas, todos ellos consideraron que tanto el reconocimiento de voz como la síntesis de voz, facilitaron la interacción con la aplicación.

❖ Ampliar los ámbitos de uso de estrategias de diálogo a entornos en los que resulta complicado interactuar con el dispositivo móvil: Teniendo en cuenta este objetivo, se añadió a la aplicación la síntesis de instrucciones implementada en el módulo Recetario, para ejemplificar la ventaja que supone la implementación de sistemas de diálogo, en situaciones en las que el usuario no dispone de total libertad para manipular el dispositivo móvil, como puede ser una persona cocinando a la vez que tiene que consultar los siguientes pasos a realizar para elaborar una receta.

8.2 Trabajo Futuro

Como líneas futuras de trabajo para mejorar el diseño y la funcionalidad de la aplicación desarrollada, hemos considerado las siguientes ampliaciones:

❖ En lo referente a la estrategia de diálogo:

- Mejorar la robustez del reconocedor: para ello sería necesario ampliar el número de palabras claves definidas en la aplicación, para las posibles sentencias emitidas por el usuario en cada turno de diálogo.

- Variedad en el diálogo: para mejorar este punto, sería necesario aumentar el número de respuestas que la aplicación proporciona al usuario, implementando un algoritmo de selección aleatoria, de modo que el diálogo no sea tan monótono.

- En lo referente a la funcionalidad de la aplicación:
  - Implementación de un sistema de pago que permita al usuario realizar la compra de los productos almacenados en la lista.
  - Desarrollo de un submódulo dentro del módulo Recetario que permita al usuario la búsqueda de nuevas recetas que puedan ser almacenadas en la aplicación
- En lo referente al diseño gráfico de la aplicación:
  - Implementar la parte gráfica de la aplicación con la herramienta Material Design, la nueva estrategia de diseño que presento Google en la conferencia del pasado año 2014 [28].

Bibliografía

[1] Oracle. Octubre de 2011. Consultado en 2015: http://www.oracle.com/us/industries/communications/oracle-communications-future-mobile-521589.pdf.

[2] Serafín, Zig. Microsoft.Noviembre de 2009. Consultado en 2015: http://www.microsoft.com/es-es/.

[3] Sejnoha, Vlad. El Diario. Junio de 2014. Consultado en 2015: http://www.eldiario.es/turing/nuance-reconocimiento-de-voz_0_267373270.html.

[4] Google. Google play. Septiembre de 2013. Consultado en 2015: https://play.google.com/store/apps/details?id=com.aridev.vita&hl=es.

[5] Apple. 2014. Consultado en 2015: http://www.apple.com/es/ios/siri/?cid=wwa-es-kwg-features-com.

[6] Wikipedia. *lenguaje XHTML*. [En línea] 09 de 2015. https://es.wikipedia.org/wiki/XHTML.

[7] García, Víctor Alvarez. Estánder Voice XML.

[8] Los Modelos de diálogo y sus aplicaciónes en sistemas de diálogo Hombre-Máquina, Dyna2009.

[9] Carrión, Zoraida Callejas. Desarrollo de Sistemas de Diálogo Oral Adaptativos y Portabes: Reconocimiento de Emociones, Adaptación al Idioma y Evaluación de Campo. Granada : s.n., Abril de 2008.

[10] DARPA. Speech and Natural Language Workshop. San Mateo, USA : s.n., 1994.

[11] W.Wahlster. SmartKom:Foundations of Multimodal Dialogue. 2006.

[12] Wikipedia.Consultado en 2015 https://es.wikipedia.org/wiki/Tel%C3%A9fono_inteligente.

[13] Glass, J., Flammia, G., Goodine, D., Phillips, M., Polifroni, J., Sakai, S., Seneff, S., Zue, V. "Multilingual spoken-language understanding in the MIT Voyager system". Speech Communication, 17 (1-2), pp. 1-18, 1995.

[14] Zue, V., Seneff, S., Glass, J., Polifroni, J., Pao, C., Hazen, T., Hetherington, L. "Jupiter: A telephone-based conversational interface for weather information". IEEE Trans. on Speech and Audio Proc., 8(1), pp. 85-96, 2000.

[15] Torres, F., Sanchis, E., Segarra, E. "Learning of stochastic dialog models through a dialogue simulation technique". Proc. Interspeech, pp. 817-820, 2005.

[16] Seneff S., Polifroni J. "Dialogue management in the Mercury flight reservation system". Proc. ANLPNAACL Satellite Workshop, pp. 1-6, p. 2000.

[17] Cine entradas. Consultado en 2015: http://www.cervantes.es/seg_nivel/lect_ens/oesi/Estudios%20de%20caso/Cinentradas.PDF
[18] Servicio BBVA, Consultado en 2015: http://www.naturalvox.com/
[19] Aplicación itSpoke. Consultado en 2015: http://www.linguee.es/ingles-espanol/traduccion/it+spoke+to.html
[20] Carlos Vaquero, Vocaliza. Consultado en 2015: http://www.researchgate.net/publication/228664464_VOCALIZA_An_application_for_computer-aided_speech_therapy_in_spanish_language
[21] Haya, P. A., Montoro, G., Alamán, X. "A prototype of a context-based architecture for intelligent home environments", Proc. International Conference on Cooperative Information Systems (CoopIS 2004). pp. 477-491, 2004.
[22] Cuota de mercado de sistemas Operativos en España. Consultado en 2015: http://es.kantar.com/tech/m%C3%B3vil/2015/marzo-2015-cuota-de-mercado-de-sistemas-operativos-en-espa%C3%B1a-hasta-enero/
[23] ITU. 2013. Consultado en 2015: https://www.itu.int/en/ITU-D/Statistics/Documents/publications/mis2013/MIS2013-exec-sum_S.pdf.
[24] Wikipedia. 2009. Disponible en: https://es.wikipedia.org/wiki/Sistema_operativo_m%C3%B3vil.
[25] Oracle. Consultado en 2015: http://www.oracle.com/technetwork/java/javase/downloads/index.html.
[26] Eclipse.Consultado en 2014 http://www.eclipse.org/downloads/.
[27] Android. Consultado en 2014: http://developer.android.com/sdk/index.html.
[28] Google. Material Design. 2014. Disponible en: https://www.google.com/design/spec/material-design/introduction.html.