Análisis y Optimización de Agentes Conversacionales 3D para Sistemas Empotrados

Tesis Doctoral

UNIVERSIDAD
DE MÁLAGA

Marcos Santos Pérez

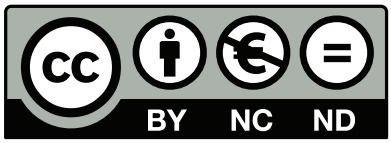
Escuela Técnica Superior de Ingeniería de Telecomunicación

2014

Publicaciones y
Divulgación Científica

AUTOR: Marcos Santos Pérez

EDITA: Publicaciones y Divulgación Científica. Universidad de Málaga



Dña. Eva González Parada, Profesora Titular del Departamento de Tecnología Electrónica de la Universidad de Málaga

Certifico:

Que D. Marcos Santos Pérez, Ingeniero de Telecomunicación, ha realizado en el Departamento de Tecnología Electrónica de la Universidad de Málaga, bajo mi dirección el trabajo de investigación correspondiente a su Tesis Doctoral titulada:

Análisis y Optimización de Agentes Conversacionales 3D para Sistemas Empotrados

Revisado el presente trabajo, estimo que puede ser presentado al Tribunal que ha de juzgarlo.

Y para que conste a efectos de lo establecido en el Real Decreto 1393/2007 regulador de los estudios de Tercer Ciclo-Doctorado, Autorizo la presentación de esta Tesis en la Universidad de Málaga.

Málaga, a 13 de Enero de 2014

Fdo. Eva González Parada
Profesora Titular del Dpto. de Tecnología Electrónica

Departamento de Tecnología Electrónica
E.T.S.I Telecomunicación
Universidad de Málaga

Tesis Doctoral

Análisis y Optimización de Agentes Conversacionales 3D para Sistemas Empotrados

Autor:
Marcos Santos Pérez
Ingeniero de Telecomunicación

Directora:
Eva González Parada
Dra. Ingeniera de Telecomunicación

A Isa, que comparte mi vida, mis sueños y mis manías.
A toda mi familia y amigos de Galicia, que están lejos pero que han sido,
son y serán parte imprescindible de mi vida y de quién soy.
A mi familia y amigos andaluces, que me hacen sentir como en casa.
A todos los que ya no están pero cuyo cariño y afecto todavía llevo dentro.

Agradecimientos

Me gustaría expresar mi más profundo y sincero agradecimiento a todas aquellas personas y entidades que con su ayuda han colaborado en la realización del presente trabajo.

En primer lugar, quiero agradecer a mi directora de tesis Eva González Parada, por su constante labor de orientación, consejo y supervisión, así como sus inestimables accesibilidad y cercanía. Gracias también por darme la oportunidad de desarrollar esta tesis con total libertad en un ámbito tecnológico tan apasionante.

A continuación, no puedo dejar de agradecer a José Manuel Cano García su total disponibilidad y ayuda en los aspectos más tecnológicos, así como su ayuda en la revisión y corrección de publicaciones.

Quiero hacer extensible este agradecimiento a todo el equipo de la empresa IHMAN S.L. por su apoyo y participación decidida en la consecución de esta tesis.

Por último, es necesario agradecer a la Junta de Andalucía y al extinto Ministerio de Educación y Ciencia por la financiación de los proyectos P08-4198 y TEC2009-13763-C02-01, gracias a los cuales he podido disfrutar tanto de una retribución durante el desarrollo de este trabajo, así como de ayudas para material, asistencia a congresos y estancias de investigación.

Resumen

En la actualidad el mundo de los sistemas empotrados, y en concreto el de los dispositivos móviles está siendo objeto de una revolución tecnológica sin precedentes. Cada pocos meses hace su aparición en el mercado un nuevo terminal con una mayor potencia de cómputo que deja prácticamente obsoletos a aquellos de uno o dos años de antigüedad. Debido a ello, desde hace unos pocos años diversos investigadores han comenzado a identificar las ventajas de incorporar a este nuevo escenario de dispositivos móviles los Agentes Conversacionales Corpóreos - *Embodied Conversational Agents* (ECAs).

Se podría decir que a grandes rasgos existen dos enfoques diferenciados a la hora de incorporar en un dispositivo móvil las capacidades de interacción y de asistencia que un ECA puede proporcionar. La primera de las opciones consiste en dotar a los ECAs de una arquitectura distribuida mediante una aplicación que se ejecuta a modo de cliente en el terminal o sistema empotrado y otra que se ejecuta en un servidor externo. En este caso la totalidad del ECA se ejecuta de forma remota, de forma que el servidor recibe y procesa la señal de voz del usuario y, posteriormente, envía la información de audio y vídeo necesarios para reproducir el avatar en el cliente. El segundo de los enfoques tiene el propósito de equipar al dispositivo móvil con el ECA en su totalidad, por lo que resultaría una arquitectura de aplicación empotrada, sin necesidad de un servidor.

A causa de la clásica limitación de recursos tanto a nivel de potencia computacional como de disponibilidad de memoria de los sistemas empotrados, hasta hace poco tiempo solamente era posible obtener ECAs eficientes y prácticos empleando la arquitectura distribuida. Aunque algunos investigadores obtuvieron resultados aceptables bajo esta perspectiva, estos ECAs distribuidos presentan un conjunto de limitaciones inherentes al propio para-

digma de dicha arquitectura. Las dos más recurrentes son el problema de la latencia y el ancho de banda. Aunque las tecnologías de las redes de comunicaciones de datos móviles también han evolucionado con el paso del tiempo, también sería correcto decir que el despliegue y el coste de éstas en la práctica hace que todavía un elevado número de potenciales usuarios de ECAs no cuenten con unos valores de latencia y ancho de banda sostenidos que permita tener la sensación de interactividad deseada. Este hecho puede provocar rechazo en los potenciales usuarios y llegar a crear la sensación de que la tecnología de los ECAs no es lo suficientemente madura como para abandonar el mundo de la academia e introducirse de lleno en la sociedad real. Unido a estos problemas, también se presenta el elevado consumo de energía que se sufre mediante el acceso a las redes de datos desde los dispositivos móviles. Es de sobra conocido que la inmensa mayoría de los *smartphones* y *tablets* actuales todavía adolecen de baterías que sólo excepcionalmente logran mantener el dispositivo encendido más allá de unas pocas horas si se hace un uso exhaustivo de aplicaciones que requieran una conexión continuada a las redes de datos.

A la vista de las dificultades expuestas, y teniendo en cuenta los grandes avances en el desarrollo de sistemas empotrados actuales, algunos investigadores han comenzado a explorar el uso de ECAs totalmente embebidos en el terminal móvil. Los datos que se extraen de sus investigaciones dejan claro que incluso hoy en día, la potencia computacional de este tipo de sistemas todavía puede ser insuficiente. En algunos casos, la literatura plantea la necesidad de volver a delegar alguna de las tareas internas más complejas de los ECAs de nuevo en servidores externos, (típicamente el reconocimiento de la voz), pero manteniendo un porcentaje elevado del agente en el terminal cliente (motor de animación, síntesis de voz, etc.). Este tipo de arquitecturas semi-distribuidas o mixtas reducen la necesidad de comunicación de datos a ráfagas intermitentes en vez de una conexión continua. Aunque con esto se logra una reducción del consumo energético y del ancho de banda requeridos, siguen estando presentes las dificultades de la disponibilidad de la red de datos y de que ésta presente una baja latencia.

El trabajo de investigación de esta tesis sigue la creencia de las bondades de disponer de ECAs que se ejecuten totalmente en el dispositivo móvil. El

enfoque novedoso que aporta frente a otros investigadores reside en la forma de afrontar los mismos problemas. En este caso se decide no renunciar a la arquitectura empotrada y optimizar aquellos aspectos que se vean limitados por la potencia de los actuales sistemas empotrados móviles. Para la consecución de este objetivo el primer paso consiste en el desarrollo de una plataforma de desarrollo de ECAs diseñada específicamente para ser ejecutada en sistemas empotrados. El propósito de esta primera plataforma es doble. El primero de ellos es el de servir de método de estudio de los ECAs en sistemas empotrados reales para analizar su rendimiento real e identificar los aspectos susceptibles de una mayor optimización. Este otro propósito es el de servir de plataforma base para desarrollar y validar las mejoras que se proponen a raíz del primer análisis. Como resultado último, este proceso resulta en la consecución de una plataforma *software* de desarrollo de ECAs optimizada para sistemas empotrados, que logra obtener valores mucho menores de latencia y consumo de energía que los de las arquitecturas distribuidas o mixtas.

Otorgándole una gran relevancia al proceso de difusión de los resultados obtenidos, y gracias al esfuerzo añadido de basar los componentes de la plataforma en *software* libre, se decide en las últimas fases de la investigación portar la plataforma optimizada al sistema operativo Android. La portabilidad que se consigue a través de este esfuerzo añadido permite asegurar futuras investigaciones con ECAs en una gran multitud de sistemas empotrados.

Abstract

Today the world of embedded systems, and in particular that of mobile devices is undergoing an unprecedented technological revolution. Every few months makes its appearance in the market a new terminal with more computing power that leaves virtually obsolete those one or two years old. As a result, since a few years ago several researchers have begun to identify the advantages of incorporating Embodied Conversational Agents (ECAs) to this new scenario of mobile devices.

It could be said that there are broadly two different approaches when incorporating the interaction's and assistance's capabilities an ECA can provide into a mobile device. The first option is to provide ECAs with a distributed architecture by using an application that runs as a client on the terminal or embedded system and another that runs on an external server. In this case the entire ECA runs remotely, so that the server receives and processes the user's voice signal and then sends back the video and audio information necessary to play the avatar on the client. The latter approach is intended to equip the mobile device with the ECA in its entirety, so that it would be obtained an embedded application architecture, without requiring a server.

Because of the classical resource's both in terms of computational power and memory availability in embedded systems, until recently it was only possible to obtain efficient and practical ECAs by means of a distributed architecture. Although some researchers obtained acceptable results following this approach, these distributed ECAs provide a number of limitations inherent in the paradigm of this architecture. The two most frequent are the problem of latency and bandwidth. While networking technologies for mobile data communications have evolved over time, it would be correct to say that the deployment and the cost of these networks in practice still makes a large

number of potential users of ECAs not to have sustained values of latency and bandwidth that allow them to reach the desired interactivity. This may cause rejection in potential users and eventually create the feeling that the technology of ECAs is not mature enough to leave the world of academia and enter fully into the human society. In conjunction with these problems, it also comes to play the high energy consumption incurred by accessing data networks from mobile devices. It is well known that the vast majority of the smartphones and tablets today still suffer from batteries which only rarely manage to keep the device on more than a few hours if there exist applications running that require a continuous data connection .

In view of the above exposed difficulties, and considering the great advances in the development of embedded systems today, some researchers have begun to explore the use of ECAs fully embedded in the mobile terminal. The data drawn from theirs investigations make clear that even today, the computational power of such systems may still be insufficient. In some cases, the literature suggests the need to re-delegate some of the most complex ECAs' inner tasks back to external servers (typically voice recognition), while maintaining a high percentage of the agent on the client terminal (animation engine, speech synthesis, etc..). This type of semi-distributed or mixed architectures reduce the need for data communication to intermittent bursts instead of a continuous connection. Although this fact leads to a reduction of energy consumption and required bandwidth, the difficulties of network availability and that this has a low latency are still present.

The research work of this thesis maintains the belief of the benefits of having ECAs that run completely on the mobile device. The novel approach that provides over other researchers lies in how to address the same issues. In this case it was decided not to waive the embedded architecture and to optimize those aspects that are constrained by the power of today's mobile embedded systems . To achieve this objective the first step consists of the development of a platform for developing ECAs specifically designed to run on embedded systems. The purpose of this first stage is doubled. The first is to provide a method of study of ECAs in real embedded systems to analyze their actual performance and identify areas for further optimization. The other purpose is to serve as a base platform for developing and validating

the proposed improvements following the first analysis. As a final result, this process results in the achievement of a platform software for developing ECAs optimized for embedded systems, which manage to obtain values for latency and energy consumption much lower than the distributed or mixed architectures .

Giving great importance to the process of dissemination of the results, and thanks to the extra effort to have built the base platform components with free software, it was decided in the final stages of research to adapt the existing optimized platform to the Android operating system. The portability that is achieved thanks to this extra effort ensures future research with ECAs in a multitude of embedded systems.

Índice general

Lista de Acrónimos

3D Tridimensional

AMC Consumo Medio de Memoria - *Average Memory Consumption*

AIML Lenguaje de Marcas para Inteligencia Artificial - *Artifical Intelligence Markup Language*

API Interfaz de Programación de Aplicaciones - *Application Programming Interface*

ART Tiempo Medio de Respuesta - *Average Response Time*

ASR Reconocimiento Automático de Voz - *Automatic Speech Recognition*

CE Motor Conversacional - *Conversational Engine*

CMU Universidad Carnegie Mellon

DARPA Agencia de Proyectos de Investigación Avanzada en Defensa - *Defense Advanced Research Projects Agency*

DVM Máquina Virtual Dalvik - *Dalvik Virtual Machine*

ECA Agente Conversacional Corpóreo - *Embodied Conversational Agent*

EDGE Tasas Mejoradas de Datos para la Evolución de GSM - *Enhanced Data Rates for GSM Evolution*

G-LM Modelo de Lenguaje General - *General Language Model*

GMM Modelo de Mezclas Gaussianas - *Gaussian Mixture Model*

GPS Sistema de Posicionamiento Global - *Global Positioning System*

HMM Modelo Oculto de Markov - *Hidden Markov Model*

HTK *Hidden Markov Model Toolkit*

HSPA Acceso de Paquetes a Alta Velocidad - *High Speed Packet Access*

JNI *Java Native Interface*

JVM Máquina Virtual Java - *Java Virtual Machine*

LMS Selección del Modelo de Lenguaje - *Language Model Switching*

MFCC *Mel Frequency Cepstral Coefficients*

MIT Instituto Tecnológico de Massachusetts

NDK *Native Development Kit*

NLTK *Natural Language ToolKit*

ODB Base de Datos Orientada a Objetos - *Object-oriented DataBase*

OPENSL_ES *Open Sound Library for Embedded Systems*

PDA Asistente Digital Personal - *Personal Digital Assistant*

RAM Memoria de Acceso Aleatorio - *Random Access Memory*

RDB Base de Datos Relacional - *Relational DataBase*

SAR Tasa de Segundo Intento -*Second Attempt Rate*

SDK *Software Development Kit*

SL4A *Scripting Layer for Android*

SO Sistema Operativo

SQL Lenguaje de Consulta eStructurado - *Structured Query Language*

SVM Máquina de Vectores de Soporte - *Support Vector Machine*

TD-LM Modelo de Lenguaje Dependiente del Tópico - *Topic-Dependent Language Model*

TTS Síntesis Artificial de Voz - *Text To Speech*

VAD Detección de Actividad Vocal - *Voice Activity Detection*

VHA Animación de Cabeza Virtual - *Virtual Head Animation*

WER Tasa de Error por Palabra - *Word Error Rate*

WLAN Red de Área Local Inalámbrica - *Wireless Local Area Network*

XML Lenguaje de Marcas eXtensible - *eXtensible Markup Language*

xRT Factor de Tiempo Real - *Real Time Ratio*

ZOPE *Zope Object-oriented DataBase*

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Introducción

Los sistemas empotrados están evolucionando a un ritmo imparable. Los recientes avances en microelectrónica han llevado al desarrollo de nuevos procesadores más pequeños, con menor consumo de energía pero a la vez con una gran potencia computacional. A causa de ello, muchos dispositivos de consumo como *smartphones*, *tablets* o consolas de videojuegos poseen un rango de funcionalidades que eran impensables hasta hace poco tiempo. Es de esperar que muchas otras disciplinas como la Domótica, la Inteligencia Ambiental o la Robótica Social también se beneficien de las ventajas del uso de los actuales procesadores empotrados.

Este nuevo escenario sitúa al usuario en el centro de un entorno tecnológico con el que debe aprender a interactuar. Tanto la comunidad académica como la industria deben trabajar conjuntamente para lograr que la interacción de los usuarios con las nuevas aplicaciones y sistemas tecnológicos que los rodean sea lo más simple y sencilla posible. En este sentido, existe un creciente interés en el desarrollo de nuevas interfaces hombre-máquina que no requieran aprendizaje por parte del usuario. Este concepto es conocido como Interacción Natural, e implica que el usuario puede comunicarse con el sistema o aplicación igual que lo haría con otra persona, sin necesitar por tanto un esfuerzo extra por su parte.

El abanico de aplicaciones que se pueden beneficiar de una interfaz de

interacción natural en sistemas empotrados es muy diverso. La industria automovilística apuesta de forma decidida por la introducción en sus vehículos de interfaces más naturales que puedan controlar diferentes sistemas por voz, tales como el teléfono, el climatizador, el sistema de navegación o el reproductor multimedia [1]. Por otro lado, la comunidad académica también ha hecho sus aportaciones en esta materia, proponiendo modelos que mejoran la naturalidad del diálogo entre el usuario y el sistema bajo control [2].

Otro área de aplicación para esta tecnología es el de las casas inteligentes. Existen diferentes estudios que han analizado el grado de aceptación de interfaces vocales para controlar varios elementos de la vivienda (puertas, luces, persianas, temperatura, televisión, sauna, etc.) [3]. Estos estudios concluyen que los usuarios mejoran enormemente sus expectativas respecto a este nuevo tipo de interfaces después de probarlas durante un corto período de tiempo. Otros autores también ponen de relevancia la importancia de la integración de este tipo de tecnología en terminales de reducidas dimensiones y bajo consumo de energía [4]. Este hecho se torna esencial de cara a facilitar su uso y aumentar su grado de aceptación por la población.

Las interfaces conversacionales en sistemas empotrados también pueden llegar a mejorar la calidad de vida de las personas con discapacidad [5] [6]. Recientemente se han propuesto prototipos de navegación asistida para personas invidentes, que les permiten mantener un cierto diálogo con el sistema. Gracias a este prototipo, estas personas pueden guiarse por entornos tanto familiares como desconocidos de una forma independiente y segura [7]. Otra aplicación en un ámbito similar es el de una silla de ruedas inteligente para personas con discapacidad motora [8].

Por último, no se puede ignorar el campo de las aplicaciones para dispositivos móviles. Los terminales actuales permiten el control por voz de diferentes aplicaciones del terminal, como la búsqueda web, la navegación por Sistema de Posicionamiento Global - *Global Positioning System* (GPS), así como la posibilidad de recibir la respuesta del terminal en forma de voz sintética [9].

La mayoría de las aplicaciones comentadas anteriormente emplean tecnologías propietarias y requieren una arquitectura cliente-servidor. En este tipo de arquitecturas el reconocimiento de voz, la comprensión y gestión del

diálogo y la síntesis de voz se realizan en un servidor externo, mientras que el dispositivo móvil actúa solamente como cliente. Esto puede llevar a que el usuario tenga una experiencia negativa cuando el sistema se vea afectado por latencias excesivas de forma aleatoria. Otro aspecto negativo del uso constante de una conexión de datos en los dispositivos móviles es su elevado consumo de energía. En los dispositivos actuales la duración de la batería todavía es una limitación muy importante a la hora de introducir nuevas interfaces de uso habitual.

En el mercado actual, la práctica totalidad de los agentes conversacionales carece de presencia física virtual. Podríamos poner como ejemplos a Siri para iOS o S-Voice para Android. Esta falta de un avatar animado puede ser una de las causas por las que este tipo de aplicaciones no son de uso masivo. En cambio, numerosos estudios han puesto de manifiesto la importancia de dotar de un cuerpo o al menos una cara a los agentes conversacionales. Así, la conversación cara a cara con un agente conversacional virtual en 3 Dimensiones (3D) o ECA ha demostrado que incrementa el deseo de participación con este tipo de sistemas [10].

1.2. Motivación de la tesis

En el mundo académico se han realizado intentos previos para crear ECAs en sistemas empotrados como Agentes Personales Digitales - *Personal Digital Assistants* (PDAs), *tablets* o *smartphones*. En la mayoría de los casos, los investigadores intentan adaptar agentes previamente desarrollados a los nuevos dispositivos móviles. Pero este proceso de adaptación no es inmediato debido principalmente a la gran diferencia de potencia computacional existente entre ambos entornos (escritorio/servidor frente a móvil/empotrado). En la mayoría de las ocasiones, ante la pérdida de rendimiento que se obtiene, se suele renunciar a alguna de las características del ECA (baja calidad del avatar animado, ausencia de sincronización labial, etc.) o se delega su ejecución en un servidor externo. La primera de las alternativas implica una gran degradación en la sensación de naturalidad de la interacción, por lo que suele resultar en el rechazo del usuario a comunicarse con el agente. Delegar ciertas funcionalidades computacionalmente exigentes en un servidor externo

permite mantener las características del ECA original a costa de introducir nuevas limitaciones. La comunicación entre el dispositivo móvil y el servidor a través de redes de telefonía móvil suele presentar valores puntuales de latencia muy elevados que implican una pérdida de interactividad entre el usuario y el agente. Este tipo de arquitectura de comunicación también implica un consumo de energía elevado en el terminal móvil debido a la necesidad de tener activa la interfaz de comunicación radio durante el tiempo que dura la conversación.

Ante el reciente avance de los sistemas empotrados, cada vez más potentes y con capacidad de procesado 3D, parece posible superar la limitación tecnológica y poder comenzar a crear ECAs que se ejecuten completamente en dispositivos móviles. El cambio de escenario de sistemas fijos de escritorio a dispositivos móviles abre las puertas a un nuevo abanico de aplicaciones donde los ECAs pueden resultar de gran importancia (asistencia en entornos abiertos, información turística, interacción social, etc.).

La irrupción de Android en el ámbito de los dispositivos móviles ha supuesto una revolución tecnológica y social. En unos pocos años Android ha conseguido dominar el mercado de *smartphones* y *tablets*, y también está incrementando muy rápidamente su penetración en el mercado de otros tipos de sistemas empotrados, como electrodomésticos inteligentes, domótica y asistentes inteligentes en vehículos. Esta realidad hace que muchos desarrolladores de aplicaciones para sistemas empotrados estén portando sus aplicaciones a este nuevo ecosistema para gozar de una difusión mucho mayor. Por lo tanto, parece acertado tener en cuenta a Android como Sistema Operativo (SO) móvil para la integración de los futuros ECAs empotrados.

1.3. Objetivos de la tesis

Este trabajo se enmarca dentro de una nueva línea de investigación que se ha venido desarrollando en el Departamento de Tecnología Electrónica de la Universidad de Málaga gracias a la concesión por parte de la Junta de Andalucía del proyecto de investigación de Excelencia P08-TIC-04198.

En este proyecto se propone desarrollar "AVATAR", una plataforma empotrada de bajo coste y ultrabajo consumo, cuya interfaz sea un asistente de

realidad virtual, basado en técnicas de animación facial fotorealista interactiva, con capacidad de síntesis y reconocimiento de voz, y soportado por un robot conversacional [11]. El asistente virtual integrado, bajo la filosofía de la interacción natural, tiene como objetivo hacer de puente entre la tecnología y el usuario final, poniendo especial atención en los usuarios con dificultades para acceder a ella.

El propósito final del desarrollo de una plataforma *software* empotrada para el desarrollo de agentes conversacionales con presencia física fotorealista para dispositivos móviles no es una tarea trivial. Una simple integración de componentes independientes posiblemente daría lugar a un agente conversacional con un rendimiento pobre, difícil de mantener y dependiente de librerías o aplicaciones *software* de terceros. Un buen equipo de desarrolladores podría solventar quizá los problemas de mantenimiento y dependencias externas. Pero sin la formación experta y especializada de la teoría, algoritmos y problemática que comprende cada de los elementos internos que forman un ECA completo, el obstáculo del rendimiento sería insalvable. La ejecución de un ECA en un ordenador de escritorio o en un servidor puede permitirse una implementación poco eficiente ya que los procesadores y tarjetas gráficas actuales poseen una potencia de cómputo muy elevada. En cambio, en los sistemas empotrados tanto la capacidad computacional como el uso de memoria dinámica se consideran recursos escasos. Por lo tanto, para lograr ejecutar un agente conversacional 3D en sistemas empotrados sin tener que depender de un servidor externo, se va a necesitar algún tipo de modificación, adaptación u optimización de los componentes del agente.

A la vista de las posibles dificultades que a priori presenta el desarrollo de ECAs en sistemas empotrados, se pueden definir los siguientes objetivos para esta tesis:

- Estudiar a nivel conceptual y teórico los elementos que forman la estructura y arquitectura de comunicación interna de un ECA.
- Realizar un estudio comparativo de las alternativas de implementación existentes y disponibles para cada uno de los componentes del agente, analizando su ventajas e inconvenientes.
- Diseñar una versión de referencia del agente conversacional 3D.

- Analizar el rendimiento de la versión de base del agente e identificar los elementos y aspectos susceptibles de mejora.

- Proponer mejoras sobre la plataforma de base y desarrollar una nueva versión optimizada.

- Analizar el rendimiento de la plataforma optimizada y compararlo con la de referencia.

La consecución de los objetivos expuestos podría haber marcado el fin del trabajo de esta tesis, pero el escenario de los sistemas empotrados, especialmente el de los dispositivos móviles, es de un alto carácter dinámico y cambiante. El mejor ejemplo de este aspecto lo constituye la revolución que Android ha supuesto en este ámbito en los últimos años.

La plataforma Android hizo su aparición en el mercado pocos meses antes del arranque del proyecto de investigación que dio lugar a esta tesis. Su crecimiento exponencial en los años siguientes, haciéndose con una posición claramente dominante del mercado, hizo replantearse el entorno *software* sobre el que se debía ejecutar el agente conversacional 3D. Por esta razón, durante el proceso de desarrollo de esta tesis surgieron nuevos objetivos que es necesario destacar:

- Estudiar la arquitectura de Android para poder plantear una adaptación del agente conversacional 3D a dicho SO.

- Analizar las modificaciones necesarias, tanto a nivel de arquitectura como de implementación, de la plataforma optimizada previamente desarrollada para llevar a cabo la portabilidad del agente.

- Diseñar y desarrollar una nueva versión de la plataforma conversacional 3D optimizada para Android.

- Validar el proceso de adaptación de la plataforma mediante el análisis de su rendimiento en un dispositivo Android real.

1.3.1. Estado de la técnica sobre los objetivos

Hasta hace relativamente unos pocos años, los agentes conversacionales virtuales eran diseñados solamente para sistemas de escritorio o servidores [12] [13] [14] [15]. La baja capacidad computacional de los sistemas empotrados unida a la elevada complejidad de los algoritmos que requieren las funciones internas del agente, hacían imposible pensar en un cambio de escenario a corto plazo.

Gracias a la introducción de acceso a redes de datos en los terminales móviles, algunos grupos de investigación pioneros realizaron los primeros intentos de llevar al entorno móvil la interacción con un agente conversacional virtual [16] [17]. En estos casos la arquitectura del agente es cliente-servidor, y el terminal móvil solo ejecuta una aplicación cliente que captura la señal vocal del usuario y la envía a un servidor externo para ser procesada. También se encarga de recibir el audio de respuesta y la animación del avatar animado, todavía en tecnología 2D. Es en el servidor externo donde se ejecutan las tareas internas del agente conversacional animado.

Aunque en la actualidad la potencia computacional de los sistemas empotrados ha evolucionado mucho a causa de la revolución de los *smartphones* y *tablets*, todavía existen algunas barreras y dificultades que hacen que incluso hoy se mantengan las mismas arquitecturas cliente-servidor sin llegar a dar el paso definitivo [18] [19].

En cambio, en los últimos 2-3 años, coincidiendo con el desarrollo de esta tesis, otros han hecho el intento de obtener un agente conversacional 3D que se ejecute completamente en un dispositivo móvil. La mayoría de ellos son investigadores con una larga carrera en este campo de investigación, por lo que decidieron portar sus agentes existentes de los sistemas de escritorio a los dispositivos móviles [20] [21] [22]. En [21], Danihelka et al. motivan su cambio de escenario de cliente-servidor a empotrado por los elevados consumo de energía y latencia que suponen el *streaming* de vídeo y audio frente al renderizado, reconocimiento y síntesis de voz en el dispositivo móvil. Estiman que el consumo de energía del ECA distribuido es al menos el doble que el del empotrado en el terminal. Además, se mide una latencia media sólo para el procesado gráfico de 400 ms, que afecta a la interactividad. En

[20], Huijie et al. aportan nuevos resultados que refuerzan la ventaja de la arquitectura empotrada. Estiman que el acceso desde un terminal móvil a un ECA ejecutado en un servidor supone 3 veces más consumo de energía y el doble de tiempo de respuesta que su equivalente empotrado en el dispositivo.

Además de validar la portabilidad de los ECAs a sistemas empotrados, estos estudios también identifican las dificultades que supone esta adaptación. Por ejemplo, en [21], Danihelka et al. no consiguen ejecutar el reconocimiento de voz en el dispositivo móvil más que en escenarios extremadamente simples que no permiten una comunicación fluida con el agente (limitan el número máximo de palabras del reconocedor a 50). Además, también reconocen que no alcanzan el rendimiento mínimo necesario si se ejecutan simultáneamente el reconocimiento de voz y el renderizado gráfico del avatar. En cambio, en [22], Klaassen et al. se encuentran con otro problema que no logran resolver; el sistema de síntesis de voz por defecto de Android no ofrece la información necesaria para poder efectuar una sincronización labial correcta del avatar. Finalmente, en [20], Huijie et al. son los únicos que consiguen un agente conversacional 3D completo y funcional, aunque sea a costa de renunciar a cierta flexibilidad y funcionalidades. Deciden usar la implementación del sistema de reconocimiento de voz de Android, lo que les impide ajustar el modelo de dicho sistema a un dominio de aplicación concreto. Además, su gestión de diálogo es un sencillo algoritmo de *pattern-matching* difuso, que carece de capacidad para manejar información de contexto o el historial de la conversación por ejemplo.

1.4. Metodología de trabajo

De los objetivos de la tesis se desprende que el trabajo desarrollado tiene una importante componente aplicada. Debido a esto, la metodología de trabajo sigue un esquema análogo al clásico de desarrollo sistemas, que comprende las etapas de análisis, diseño, desarrollo y validación. Para que los resultados sean más concluyentes y prácticos, se decidió que el desarrollo y validación se efectuasen sobre dispositivos reales y no en un entorno simulado/emulado. Así, las principales fases en las que se puede dividir la realización de la Tesis son las siguientes:

- Búsqueda de información: Recopilación de toda la información posible, tanto bibliográfica como publicaciones en congresos y revistas sobre los aspectos teóricos y de implementación de cada una de las partes que forman el agente conversacional 3D. Para ello se hará uso de las herramientas de búsqueda de información científica y técnica disponibles en Internet.

- Realizar una comparativa de las alternativas disponibles de los componentes de un ECA: Se deben identificar todas las ventajas e inconvenientes de cada una de las implementaciones, siempre desde el enfoque orientado a sistemas empotrados.

- Definir la arquitectura de la plataforma de referencia del agente: La selección de los componentes sigue criterios cualitativos ante la imposibilidad de realizar un análisis exhaustivo de todas las alternativas.

- Desarrollo de una versión de base del agente conversacional 3D: Pensando en lograr una mayor difusión de la plataforma *software* final en el ámbito académico, se deben primar los componentes basados en *software* libre.

- Realizar el análisis de prestaciones objetivas de la arquitectura: Se estudia cada una de las partes que componen el sistema para determinar aquellas cuya optimización suponga un mayor impacto final en el rendimiento global del agente.

- Optimizar el agente conversacional 3D: Se desarrollan mejoras para cada una de las partes susceptibles de optimización sustancial identificadas en el punto anterior.

- Realizar un segundo análisis de prestaciones: Estas pruebas validan las mejoras propuestas para optimizar el rendimiento del agente.

- Proponer una adaptación del agente a Android: Se analizan las posibles dificultades de este proceso y se proponen soluciones que mantengan las funcionalidades y flexibilidad del agente conversacional.

- Validar la portabilidad a Android: Un nuevo análisis de prestaciones debe confirmar que el proceso de adaptación a Android no ha mermado las características de la versión optimizada anterior.

1.5. Contenidos de la tesis

La estructura de la tesis pretende reflejar con la mayor fidelidad todo el proceso de documentación, aprendizaje, desarrollo e investigación que el autor ha vivido durante los últimos años. De esta forma se pueden exponer los contenidos de la tesis de la siguiente forma:

El Capítulo 1 comienza con la presentación del marco conceptual de la tesis dentro del ámbito de los agentes conversacionales 3D para sistemas empotrados. A continuación expone las dificultades y limitaciones encontradas en los esfuerzos investigadores previos. Esta problemática lleva a encontrar la motivación para el desarrollo de la tesis y proponer unos objetivos realistas y alcanzables. También se tratan aquí las fases y metodología a seguir durante todo el proceso investigador. Por último, se describen los contenidos del presente documento y se establece una guía de lectura para ayudar al lector a seleccionar los pasajes que puedan ser de su mayor interés.

El Capítulo 2 se encarga de exponer toda la carga teórica que existe detrás de la tecnología de agentes conversacionales virtuales. Constituye una primera descripción detallada del estado del arte en cuanto a los modelos y algoritmos que hacen realidad cada una de las partes que componen un agente conversacional virtual en su totalidad. Por lo tanto, inicialmente se hará una descripción de la arquitectura genérica de un agente conversacional virtual para, a continuación, describir los enfoques teóricos más relevantes que se aplican en cada una de las partes del agente global.

El Capítulo 3 detalla la tarea del investigador que debe realizar una versión de referencia de la plataforma de desarrollo de agentes conversacionales 3D. Por lo tanto, aquí se enumeran las diferentes implementaciones existentes de los componentes necesarios para construir el agente conversacional 3D. Para cada una de ellas se analizan de forma cualitativa las ventajas e inconvenientes que presentan. Finalmente, se expone la selección inicial que dará lugar al primer prototipo en base a los criterios perseguidos y al contexto

donde se va a ejecutar el agente, sistemas empotrados y móviles.

El Capítulo 4 describe el primer proceso de análisis del prototipo de agente conversacional virtual. Para desarrollar este análisis, se presentan y dan detalles del sistema empotrado y las pruebas a las que se someten los diferentes módulos del agente. Como resultado de esas primeras pruebas, se identifican los elementos de la plataforma software con mayor capacidad de optimización.

El Capítulo 5 cubre el proceso de optimización de la plataforma software inicial. En este proceso se explican con detalle los diferentes objetivos sujetos a evolución del agente conversacional. Para ello, se razona el propósito de cada una de las mejoras propuestas, se describe un conjunto de pruebas que validen la optimización realizada y se da cuenta de los resultados obtenidos.

El Capítulo 6 se muestra el último paso dado en el desarrollo del agente conversacional 3D, la portabilidad de toda la plataforma software al SO de uso mayoritario en sistemas empotrados y móviles actuales. Este proceso de portabilidad resulta en modificaciones a nivel de arquitectura funcional y lógica debido a las restricciones del entorno de ejecución fijado. Por lo tanto, en este capítulo también se analiza el rendimiento del nuevo prototipo y se presentan los resultados comparándolos en caso de ser posible con otras alternativas disponibles por investigadores y desarrolladores.

El Capítulo 7 cumple el objetivo de resumir las contribuciones aportadas por la tesis, extraer las conclusiones y desarrollar las múltiples líneas futuras del trabajo expuesto.

1.6. Guía de lectura de la tesis

A continuación se realizan diversas recomendaciones en cuanto a la lectura de esta tesis en función de los diferentes perfiles que pueden aprovechar el contenido en ella expuesto.

- Perfil académico/investigador: Resultará principalmente de interés el Capítulo 2 como base teórica y bibliográfica de los diferentes enfoques que se consideran el estado del arte a la hora de abordar cada una de las partes constituyentes del agente conversacional virtual. En este caso

también se recomienda la lectura del Capítulo 5 debido a su contenido específico de optimización y ajuste de diferentes aspectos relevantes de un agente conversacional 3D en el contexto de sistemas con limitaciones de recursos computacionales y de memoria.

- Perfil docente: El ámbito de esta tesis se ajusta a un nivel teórico de cursos de Máster y Doctorado. Por lo tanto, el Capítulo de mayor importancia en este caso es el 2 por su descripción del estado del arte y abundancia de referencias bibliográficas donde profundizar más sobre el tema. Son también recomendables los Capítulos 3 y 6 por su profundo carácter aplicado y numerosas referencias a implementaciones de los componentes que forman un agente conversacional 3D. De esta forma puede tener un equilibrio teórico-práctico sobre el que orientar posibles contenidos docentes.

- Perfil desarrollador: Este tipo de lector puede obviar claramente los Capítulos 2 y 5 y centrarse solamente en los Capítulos 3, 4 y 6 donde se proporcionan detalles de implementación, entorno *hardware* de desarrollo y prueba y metodología de análisis de rendimiento del agente conversacional virtual. También puede encontrar muy útiles las referencias que estos capítulos contienen sobre implementaciones (la mayoría de código abierto) de las partes que forman un agente conversacional 3D.

Capítulo 2

Agentes conversacionales 3D

2.1. Introducción

Los agentes conversacionales se caracterizan por su habilidad para comunicarse con los usuarios en lenguaje natural con el objetivo de realizar una tarea especifica. Debido a que el medio de comunicación principal de los agentes conversacionales es el habla y no el texto, también se les conoce como sistemas de diálogo hablado.

Este tipo de sistemas se han implementado con éxito en distintos ámbitos, como pueden ser la asistencia técnica, tanto vía web como telefónica y la realización de tareas complejas en dominios concretos, por ejemplo la consulta de horarios y reserva de vuelos o trenes [23] o el control de elementos del entorno doméstico [24]. Estos sistemas de gestión del diálogo permiten al usuario completar una determinada tarea de forma secuencial. Este tipo de conversaciones resulta ciertamente incómoda para un gran número de usuarios debido a que las conversaciones naturales entre seres humanos no se desarrollan de una forma lineal y determinista. Por lo tanto, para lograr una mayor sensación de naturalidad y disminuir el rechazo del usuario, es necesario que el sistema pueda manejar cambios inesperados de contexto y tener capacidad para lograr diversos objetivos de forma asíncrona [25]. Con el fin de conseguir una experiencia más natural, los investigadores intentan dotar de inteligencia a los agentes conversacionales.

Este Capítulo trata los detalles conceptuales y teóricos sobre los agentes

conversacionales en general. Para ello, de inicio se describen brevemente los aspectos que han resultado de interés en la evolución de los agentes conversacionales. A continuación, se describe una arquitectura típica de agentes conversacionales virtuales, para posteriormente profundizar sobre las técnicas y métodos comunes que afectan a cada una de las partes que forman dicha arquitectura.

2.2. Aspectos generales sobre agentes conversacionales

En el ámbito de la Inteligencia Artificial existen dos grandes posturas que marcan el objetivo de las investigaciones en todos los campos en los que participa. La Inteligencia Artificial Fuerte persigue la creación de máquinas conscientes, una verdadera "vida artificial", mientras que la Inteligencia Artificial Débil descarta esta posibilidad y sus pretensiones son más modestas; consideran suficiente el que una máquina pueda actuar como si fuera inteligente. Estas dos visiones tan distantes han impregnado la investigación sobre sistemas conversacionales. La primera aproximación sería la de la percepción de los sistemas conversacionales como una simulación del lenguaje humano, con el objetivo de lograr una teoría completa e implementable computacionalmente. La segunda aproximación, con un perfil más ingenieril, es la construcción de interfaces hombre-máquina más naturales y fáciles de usar debido a la incorporación de la comprensión y gestión del diálogo hablado [26].

Otro aspecto relevante que afecta al diseño e implementación del sistema conversacional es la metáfora que el propio sistema representa de cara al usuario. La metáfora "interfaz" considera que el sistema se percibe como una interfaz con la máquina, mientras que la metáfora "humana" hace que el sistema sea percibido como un ser humano. Durante las fases de diseño del agente conversacional se debe ser coherente con el tipo de metáfora que representa el sistema, ya que el tipo de diálogo que el usuario mantiene y espera de él es diferente en ambos casos [27].

Gracias a este reciente desarrollo en la tecnología de los agentes conversacionales, están surgiendo investigaciones que abordan la incorporación de

aspectos "humanos" a estos sistemas, como pueden ser la personalidad [28] y el uso de la intención [29] o las emociones durante los diálogos [30] [31].

Otro aspecto importante que se introdujo en tiempos relativamente recientes es la idea de dotar al agente conversacional de una representación física virtual, una encarnación. Son los denominados ECAs [15]. Estos agentes pueden consistir en simples caras animadas que hablan o ser mucho más complejas, con representaciones gráficas 3D de cara y cuerpo, incluyendo variaciones gestuales, sincronización labial, etc.. Entre otras aplicaciones, se están presentando estudios prometedores acerca del uso de este tipo de agentes como nueva forma de interacción en el área de los ambientes inteligentes [32].

2.3. Arquitectura de agentes conversacionales 3D

La arquitectura clásica de un agente conversacional virtual típico se muestra en la Figura 2.1 y consiste en los siguientes componentes:

- Detector de Actividad Vocal - *Voice Activity Detection* (VAD): Captura, digitaliza y segmenta la señal de audio en frases. La salida de este módulo es un segmento digitalizado de audio que contiene una frase completa.

- Reconocedor Automático de Voz - *Automatic Speech Recognition* (ASR): Recoge cada frase de audio, y la procesa para obtener la mejor secuencia de palabras que han generado esa entrada. La salida de este módulo es una cadena de texto con la frase reconocida.

- Motor Conversacional - *Conversational Engine* (CE): Procesa la cadena de texto reconocida para extraer toda la información conceptual necesaria para interactuar con las aplicaciones que definen su entorno (ejecutar acciones, solicitar información, etc.) y poder continuar el diálogo con el usuario humano (comunicar las acciones realizadas, solicitar la aclaración de algún concepto, etc.). La salida de este módulo es el texto con la respuesta que debe recibir el usuario.

- Sintetizador Artificial de Voz - *Text To Speech* (TTS): Realiza la tarea de convertir el texto de respuesta en una señal de audio para cerrar el bucle de comunicación entre el sistema y el usuario. También se encarga de generar la información necesaria para sincronizar la animación con el audio.

- Animación de la Cabeza Virtual - *Virtual Head Animation* (VHA): Representa la encarnación del agente conversacional. Normalmente ofrece características avanzadas como sincronización labial y capacidad para expresar emociones o gestos.

2.4. Detector de actividad vocal (VAD)

El término detector de actividad vocal se refiere a un tipo de procesado de señal que detecta si fragmentos cortos de una señal acústica contienen información vocal o no (generalmente silencio o ruido ambiente). La detección de estos periodos de voz o silencio dentro de la señal acústica de entrada es un proceso importante en la construcción de sistemas reales que apliquen las tecnologías del habla. Por ejemplo, en la codificación de voz de los sistemas de comunicaciones móviles el VAD permite reducir la potencia consumida, reducir la interferencia cocanal y aprovechar mejor el ancho de banda disponible [33] [34]. En otras aplicaciones donde las características del entorno degradan el rendimiento del sistema, la función del VAD será la reducción y compensación del ruido ambiente [35]. En sistemas de reconocimiento de voz, el VAD cumple en general dos funciones claramente diferenciadas. La primera es la segmentación de las frases del usuario, función imprescindible para poder ofrecer sistemas conversacionales que interactúen en tiempo real con el usuario. La segunda función del VAD en este tipo de aplicaciones es la de mejorar la robustez global del sistema mediante la técnica conocida como "*frame-dropping*" [36]. De esta forma el VAD elimina a la entrada del reconocedor de voz los segmentos de la señal que son detectados como silencio o ruido, logrando reducir los errores de inserción.

El principio básico del VAD reside en la extracción de características medidas de la señal de audio de entrada, que es dividida en *"frames"* de 5-40

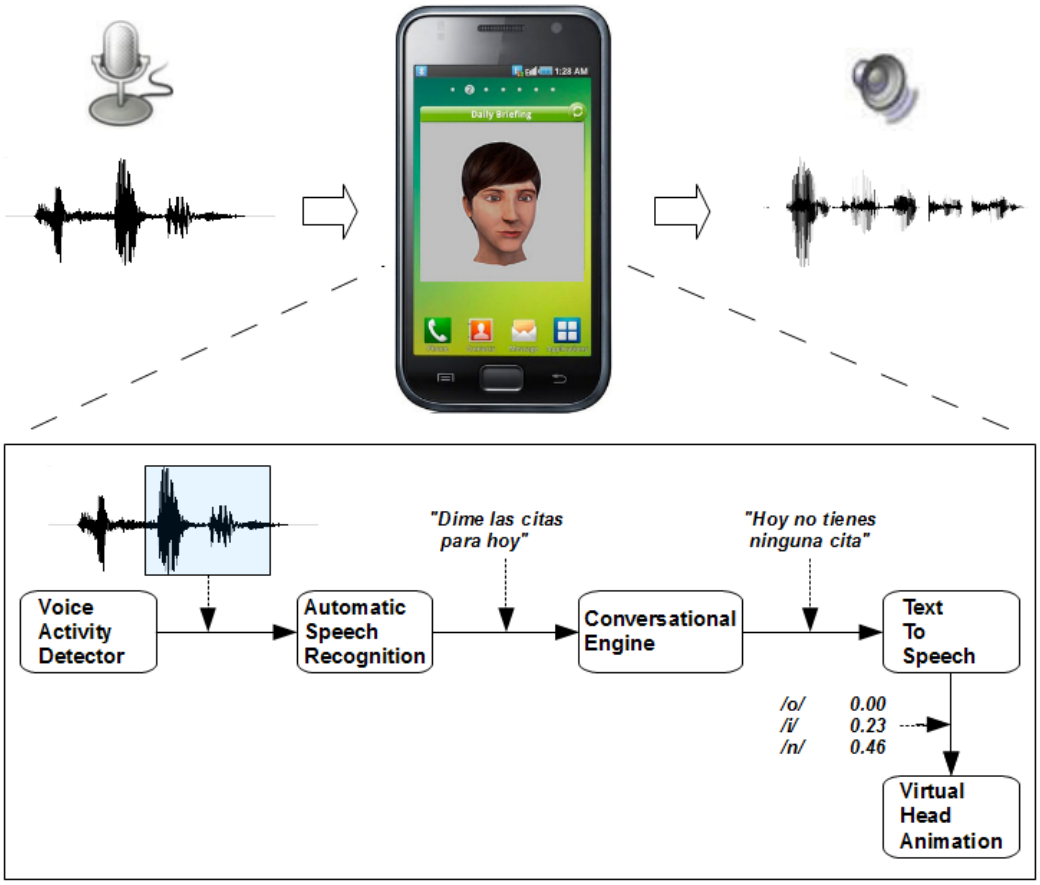


Figura 2.1: Arquitectura genérica de un agente conversacional virtual.

ms de duración. Estas características medidas de la señal se comparan con unos umbrales límite que normalmente se obtienen por estimación durante los periodos de ruido de la señal de entrada. Si la característica del segmento de entrada excede el valor del umbral estimado, el VAD decide que existe presencia de voz en dicho segmento. En caso contrario el VAD decide que no existe voz en el segmento de audio. El diagrama de bloques básico del VAD se muestra en la Figura 2.2.

A través de los años se han propuesto diferentes enfoques para la detección de segmentos de voz en la señal acústica de entrada. Los algoritmos

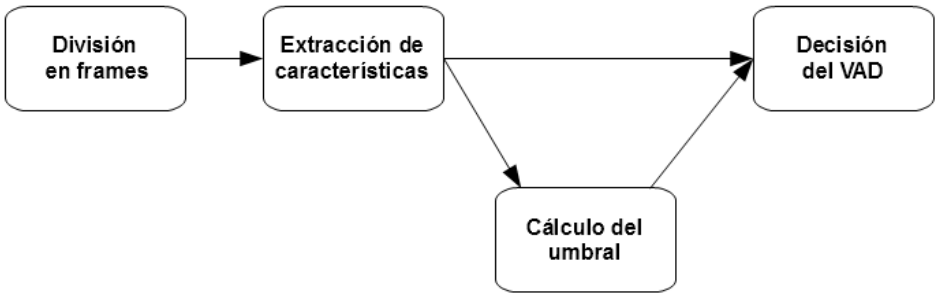


Figura 2.2: Esquema del detector de actividad vocal.

VAD clásicos se basan en la extracción de características localizadas como la energía y módulo de la señal, la tasa de cruces por cero, la predicción lineal [37] y el análisis de la frecuencia fundamental [38]. En los métodos más modernos la clasificación se basa en coeficientes cepstrales [39], transformadas *wavelet* [40], medidas de periodicidad [41] y modelos estadísticos [42].

2.5. Reconocimiento automático de voz (ASR)

El reconocimiento de la voz es una parte crucial de los sistemas conversacionales, y su funcionalidad es esencial para para permitir la comunicación hablada entre los seres humanos y las máquinas. El reconocimiento automático de voz es el proceso de conversión de una señal de voz a una secuencia de palabras de texto mediante un algoritmo implementado para ser ejecutado de forma automática en un sistema informático.

La investigación en ASR ha sido motivada por el deseo humano de construir modelos mecánicos que emulen las capacidades de la comunicación verbal humana, dado que el habla es la forma más natural de comunicación entre personas. El objetivo principal en el ámbito de ASR es el desarrollo de técnicas y sistemas para que sirvan de interfaz de entrada de la voz humana a las máquinas. Por razones que van desde la curiosidad metodológica acerca de los mecanismos causantes de la voz humana hasta el deseo de automatizar tareas simples que requieren interacción hombre-máquina, la investigación

en ASR ha sido un gran foco de atención durante los últimos 60 años [43]. Aunque cualquier tarea que requiera interacción entre personas y máquinas puede ser adecuada para el uso de reconocimiento de voz, actualmente, las aplicaciones con interfaz por voz más comunes son el procesado automático de llamadas telefónicas, sistemas de búsqueda de información, consulta de información meteorológica, dictado automático de texto por voz, acceso a sistemas que ofrecen información de viajes, transcripción de texto o ayuda para personas con algún tipo de discapacidad.

La complejidad de este tipo de sistemas radica en la diversidad de los factores que incorpora el habla humana (acústica, fonética, fonología, léxico, semántica) y la posibilidad de existencia de ambigüedades y errores para poder obtener una interpretación aceptable del mensaje recibido. En muchos casos, la sensación de naturalidad del interfaz conversacional depende en gran medida de la robustez del reconocimiento de la voz [44].

Pese a todas estas dificultades, actualmente se ha avanzado mucho en este ámbito de investigación y es posible lograr sistemas ASR con unos niveles de error aceptables para un gran número de aplicaciones [45]. Además del requisito de una baja tasa de error, para la construcción de sistemas que no produzcan rechazo en los usuarios es necesario que la tarea de reconocimiento se realice en un tiempo limitado. Los progresos hechos para optimizar la tarea de reconocimiento del habla unidos a la mejora de prestaciones de los ordenadores actuales han conseguido que sea posible cumplir estos requisitos de tiempo real en estaciones de sobremesa sin mayor problema e incluso comienzan a ser una realidad en sistemas empotrados [46] [47].

Internamente, un sistema ASR típico se puede dividir en 2 partes: *front-end* y *back-end*. Ambos componentes cumplen funciones bien diferenciadas. El *front-end* realiza un preprocesado de la señal de audio para extraer los vectores de características. Estos vectores de características representan de forma compacta los aspectos más relevantes de la voz humana necesarios para su reconocimiento. El *back-end* o decodificador es el responsable de la tarea específica de reconocimiento de voz. Para ello, implementa un complejo modelo probabilístico que incluye típicamente modelos acústico y de lenguaje y un diccionario de pronunciación. Esta estructura de sistemas ASR se puede ver en la Figura 2.3.

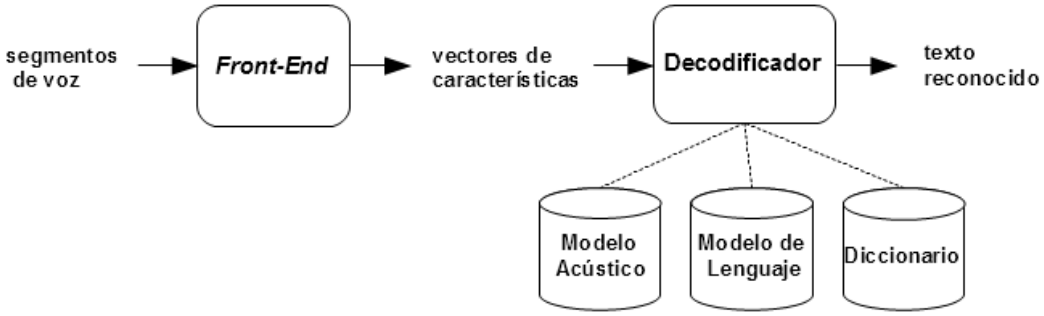


Figura 2.3: Esquema general de un sistema ASR.

2.5.1. Extracción de características

La extracción de características de la señal es un proceso complejo que busca enfatizar los aspectos más relevantes de la señal acústica con el fin de facilitar la discriminación durante la fase de reconocimiento. Aunque existen varios formatos de representación de estas características, la más popular es la denominada *Mel Frequency Cepstral Coefficients* (MFCC). A continuación se describen brevemente los pasos de los que consta este preprocesado de la señal vocal y que también se muestran en la Figura 2.4.

- Preénfasis: Con este primer paso se busca incrementar la energía de las frecuencias altas de la señal vocal. La voz humana presenta un espectro de energía mayor en las frecuencias bajas y es necesario un filtro que efectúe una compensación de este efecto.

- Enventanado: Aunque la señal vocal no es estacionaria en un sentido amplio, se suele asumir que en periodos de tiempo cortos (20-30 ms) llamados $frames$ sí se puede considerar estacionaria. Por lo tanto, el enventanado permite obtener segmentos de señal donde las propiedades estadísticas son prácticamente constantes.

- Obtención de coeficientes MFCC: Los coeficientes MFCC se derivan de la Transformada de Fourier, pero a diferencia de ésta, las frecuen-

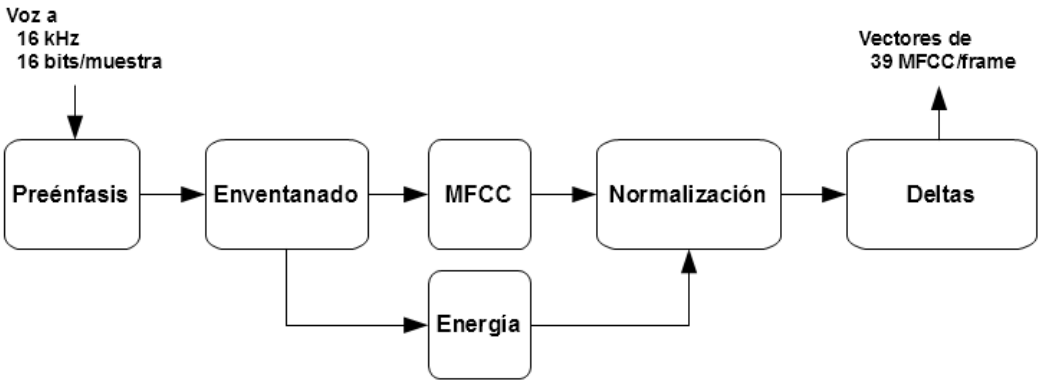


Figura 2.4: Esquema de extracción de características de la señal vocal.

cias presentan una escala logarítmica, llamada escala de Mel [48]. Es habitual representar cada $frame$ con 12 valores MFCC.

- Cálculo de energía: Para cada $frame$ se calcula el valor de energía de ese segmento de señal.

- Normalización: Los 13 valores que representan un $frame$ se normalizan restándoles el valor máximo de energía y la media cepstral de toda la señal vocal de la frase de usuario.

- Deltas: Para los 13 valores normalizados que se obtienen de cada $frame$ (12 de MFCC y 1 de energía) se calculan otros 13 valores denominados *delta* que representan su velocidad de cambio. Seguidamente, de los valores *delta* se calculan otros nuevos valores denominados $doble-delta$, que representan la aceleración respecto a los 13 valores originales.

2.5.2. Proceso de reconocimiento

Actualmente el considerado estado del arte en ASR se basa en el modelado estadístico del habla humana. Este modelo probabilístico dice que dadas las observaciones acústicas O (vectores de características de la señal), se de-

be seleccionar la secuencia de palabras $\hat{W}$ que maximiza la probabilidad a posteriori $P(W/O)$,

$$\hat{W} = argmax_W P(W/O) \tag{2.1}$$

Empleando el teorema de Bayes para descomponer $P(W/O)$ en términos acústicos y lingüísticos se tiene que,

$$P(W/O) = \frac{P(O/W)P(W)}{P(O)} \tag{2.2}$$

y, finalmente, $P(O)$ es independiente de W y se puede eliminar de la ecuación,

$$\hat{W} = argmax_W P(O/W)P(W) \tag{2.3}$$

donde $P(O/W)$ es la verosimilitud del modelo acústico y $P(W)$ representa el modelo de lenguaje determinado a priori. La Figura 2.3 muestra este modelo genérico de reconocimiento de voz. Para sistemas ASR con un gran vocabulario, es necesario crear modelos acústicos de unidades inferiores a la palabra, por lo que se hace necesario un vocabulario de pronunciación que describa la composición de las palabras.

2.5.2.1. Modelo acústico

Debido a que la unidad fundamental del lenguaje es la palabra, parece lógico crear modelos acústicos para cada palabra del vocabulario que se desea reconocer. Pero a medida que el número de palabras crece, se hace muy difícil obtener suficientes muestras para entrenar dichos modelos y el número de modelos se torna demasiado grande. Este problema se resuelve creando modelos acústicos de unidades inferiores a la palabra, como el fonema. Pero los fonemas suenan diferente en función del contexto en que se encuentren. Este fenómeno se denomina coarticulación. Por lo tanto, la unidad en la que se suelen basar los modelos acústicos es el trifonema. Aunque la combinación de posibles trifonemas es muy elevada, en la práctica sólo se observa una pequeña fracción de éstos.

La mayoría de los sistemas ASR actuales modelan la estructura secuencial del habla mediante Modelos Ocultos de Markov - *Hidden Markov Models*

(HMMs) [49]. Los HMM se representan como un conjunto de estados conectados mediante arcos que modelan la probabilidad de transición de un estado al siguiente, denominada a_{ij} para el paso del estado i al estado j. Cada trifonema que se desee modelar cuenta con un estado en el esquema HMM. Además de la probabilidad de transición, los estados también cuentan con una probabilidad de emisión, b_j donde j representa a un estado concreto. La emisión define la probabilidad de que ese estado (trifonema) haya generado una observación acústica concreta (vector de 39 MFCC de un $frame$). Debido a que los vectores de características representan un espacio de posibilidades muy elevado, la probabilidad de emisión no se puede caracterizar por ninguna distribución simple y es necesario recurrir a modelos más complejos, como Modelos de Mezclas Gaussianas - *Gaussian Mixture Models* (GMM). De esta forma, la ecuación 2.4 para la probabilidad de emisión queda como,

$$b_j(O) = \sum_{m=1}^{M} c_{jm} N(O, \mu_{jm}, U_{jm}) \tag{2.4}$$

donde O es el vector de observación/características, c_{jm} son los coeficientes de la mezcla de M gaussianas en el estado j, y $N(O, \mu_{jm}, U_{jm})$ es una distribución normal.

En el caso de los modelos HMM continuos, cada estado j cuenta con su propio conjunto de gaussianas. Debido al gran coste computacional que este hecho implica, se han propuesto modelos alternativos que agrupen estados HMM en *clusters* de forma que se compartan los mismos parámetros. Además de esta técnica de agrupación, también son muy populares los modelos HMM semicontinuos, donde todos los estados comparten en mismo conjunto de gaussianas y solamente se diferencian entre sí por los valores de los coeficientes c_{jm}. Este tipo de modelos reducen mucho la carga computacional del sistema ASR y, por lo tanto, son apropiados para sistemas con limitación de potencia de cálculo.

2.5.2.2. Modelo de lenguaje

Como se deduce de la ecuación 2.3, un sistema ASR necesita un modelo de lenguaje para poder distinguir entre las posibles secuencias de palabras que cuentan con una representación fonética similar. Por lo tanto, el modelo de

lenguaje establece una probabilidad a priori sobre una secuencia de palabras $W = (w_1, w_2, ..., w_n)$ de la siguiente forma:

$$P(W) = P(w_1)P(w_2/w_1)P(w_3/w_1, w_2)...P(w_n/w_1, ..., w_{i-1}) \quad (2.5)$$

$$= \prod_{i=1}^{n} P(w_i/w_1, ..., w_{i-1}) \quad (2.6)$$

Ante la imposibilidad de estimar correctamente las probabilidades de palabras con historias $w_1, ..., w_{i-1}$ demasiado largas, con frecuencia se simplifica el modelo limitando la longitud máxima de la historia que afecta a la palabra actual. Las más comunes suelen ser las historias de longitud 0, $P(w_i)$, llamadas unigramas; las de longitud 1, $P(w_i/w_{i-1})$, llamadas bigramas, y las de longitud 2, $P(w_i/w_{i-1}, w_{i-2})$, llamadas trigramas. Actualmente, los modelos de lenguaje probabilísticos basados en n-gramas se consideran el estado del arte en aplicaciones como reconocimiento de voz, traducción automática o reconocimiento de caracteres. En el caso de un modelo basado en trigramas se tiene que la ecuación 2.5 se simplifica y queda como,

$$P(W) = \prod_{i=1}^{n} P(w_i/w_{i-1}, w_{i-2}) \quad (2.7)$$

Los modelos de lenguaje probabilísticos suelen sufrir el problema de la dispersión de los datos. Debido a ello, se han propuesto diferentes métodos para suavizar los parámetros de los modelos de lenguaje, de forma que la masa de probabilidad se redistribuya de las palabras que cuentan con un mayor número de observaciones en los datos de entrenamiento hacia otras que apenas aparecen. De todas las técnicas propuestas hasta la fecha, estudios comparativos establecen que los mejores resultados se obtienen empleando el algoritmo modificado de Kneser-Ney [50].

La métrica que con mayor frecuencia se utiliza para estimar la bondad de un modelo de lenguaje se denomina perplejidad. La idea que subyace a esta métrica se basa en la idea de medir cuánto se "sorprende" un modelo de lenguaje cuando se enfrenta a unos datos de prueba desconocidos. La definición se muestra en la ecuación 2.8, y se basa en la entropía de la distribución de probabilidad que representa el modelo de lenguaje.

$$Perplejidad \;=\; 2^{H(p)} \tag{2.8}$$

$$H(p) \;=\; -\sum_{x} p(x)\; log_2\; p(x) \tag{2.9}$$

2.5.2.3. Algoritmo de búsqueda

En la ecuación 2.1 se define la idea principal del reconocimiento de voz como la tarea de encontrar la secuencia de palabras más probables en función de las observaciones. Pues bien, el algoritmo de búsqueda más empleado para encontrar esa secuencia de palabras W óptima es el de Viterbi [51].

La búsqueda de Viterbi se implementa como un algoritmo recursivo que recorre de forma lineal en tiempo ($frame$ a $frame$) los posibles estados (HMM) que pueden representar los vectores de características (MFCC) observados. La ecuación 2.10 resume el proceso de actualización de la mejor secuencia de estados que va almacenando el algoritmo a medida que avanza el tiempo,

$$P_j(t) \;=\; max_i(P_i(t-1)\; a_{ij}\; b_j(t)) \tag{2.10}$$

donde, $P_j(t)$ es la probabilidad del estado j en el tiempo t, a_{ij} es la probabilidad de transición del HMM y $b_j(t)$ es la probabilidad de emisión. Las probabilidades del modelo de lenguaje se incorporan como transiciones a estados nulos que se intercalan entre estados HMM existentes que representan los trifonemas.

2.5.3. Tipos de sistemas ASR

Los sistemas ASR se pueden diferenciar en varios tipos diferentes mediante la descripción del tipo de frases que son capaces de reconocer. Esta clasificación es la siguiente:

- Palabras aisladas: Los sistemas ASR de palabras aisladas tienen como requisito principal que la frase de entrada tenga ausencia de voz o ruido antes y después de la señal acústica que va a ser reconocida. Suele aceptar palabras sueltas o incluso secuencias de palabras. Este

tipo de sistemas tienen estados de "escucha" y "espera", por lo que el usuario debe permanecer callado entre 2 frases consecutivas, ya que se debe esperar al procesado completo de cada una de ellas.

- Palabras conectadas: Este tipo de sistemas ASR son similares al anterior pero permiten separar las frases del usuario para ser procesadas en bloque, estableciendo una pausa mínima entre ellas.

- Habla espontánea: Un sistema ASR con capacidad de reconocer habla espontánea puede tratar con una gran variedad de fenómenos vocales naturales que ocurren junto a secuencias de palabras conectadas. Este tipo de fenómenos vocales suelen ser momentos de duda, pausas, tos, etc..

Existen más criterios por los que se puede establecer una clasificación de los sistemas de reconocimiento de voz. Algunos de los más relevantes son los siguientes:

- Entrenabilidad: Determina si el sistema necesita un entrenamiento previo antes de empezar a usarse.

- Dependencia del hablante: Especifica si el sistema debe entrenarse para cada usuario o es independiente del hablante.

- Continuidad: Hay sistemas de reconocimiento de voz que pueden reconocer habla continua y sin embargo otros requieren que el usuario haga pausas entre palabra y palabra.

- Robustez: Un módulo de reconocimiento de voz puede estar diseñado para usarse con señales poco ruidosas o, por el contrario, puede funcionar aceptablemente en condiciones ruidosas, ya sea ruido de fondo, ruido procedente del canal o la presencia de voces de otras personas.

- Tamaño del dominio: Determina si el sistema está diseñado para reconocer lenguaje de un dominio reducido (unos cientos de palabras para reservas de vuelos o peticiones de información meteorológica) o extenso (miles de palabras).

Por último, otra característica relevante para establecer una clasificación de sistemas ASR es el tipo de aprendizaje que se emplea:

- Aprendizaje Deductivo: Las técnicas de Aprendizaje Deductivo se basan en la transferencia de los conocimientos que un experto humano posee a un sistema informático. Un ejemplo paradigmático de las metodologías que utilizan tales técnicas lo constituyen los Sistemas Basados en el Conocimiento y, en particular, los Sistemas Expertos.

- Aprendizaje Inductivo: Las técnicas de Aprendizaje Inductivo se basan en que el sistema pueda, automáticamente, conseguir los conocimientos necesarios a partir de ejemplos reales sobre la tarea que se desea modelizar. En este segundo tipo, los ejemplos los constituyen aquellas partes de los sistemas basados en HMMs o en redes neuronales artificiales que son configuradas automáticamente a partir de muestras de aprendizaje.

En la práctica, no existen metodologías que estén basadas únicamente en el Aprendizaje Inductivo, de hecho, se asume un compromiso deductivo-inductivo en el que los aspectos generales se suministran deductivamente y la caracterización de la variabilidad inductivamente.

2.6. Motor conversacional (CE)

El motor conversacional concentra varias tareas críticas para la construcción de un ECA capaz de entablar un diálogo natural con un usuario humano: la comprensión del lenguaje natural, la gestión del diálogo y la generación de respuestas naturales. Aunque es frecuente encontrar literatura sobre agentes conversacionales que estudia y analiza estas 3 tareas de forma independiente, a la hora de crear ECAs que sirvan de asistente o interfaz para el control de aplicaciones o simplemente puedan mantener una conversación de tipo social, se suele elegir una sola forma de representación de la información dentro del motor conversacional del agente. Por lo tanto, ante la fuerte dependencia que existe entre estas 3 tareas, esta sección se centra en el estudio de la gestión del diálogo como elemento central del motor conversacional, y que suele determinar la representación de la información a nivel semántico del ECA.

2.6.1. Gestión del diálogo

Durante bastantes años, debido a la limitación computacional de los sistemas existentes, la máquina se limitaba a ser una herramienta pasiva, un "esclavo" del usuario. Recientemente, a medida que crecía la capacidad de integración y, por lo tanto, el rendimiento y las posibilidades de los ordenadores, se produjo un cambio de concepción de la máquina. Ahora la interacción hombre-máquina se considera una actividad de cooperación entre 2 agentes: el sistema y el usuario [52]. Por lo tanto, el sistema no debe ser un receptor pasivo de órdenes y comandos, sino que debe interpretar y ser capaz de responder y asistir al usuario humano, haciendo de mediador entre éste y la aplicación o sistema que se esté manipulando. En el caso de los interfaces conversacionales, durante este proceso de mediación, la comunicación se produce a través de un diálogo entre el humano y la máquina. Existen distintos tipos de aproximaciones de gestión de diálogo en función de quién lleva la iniciativa de la conversación o de cómo representa la información que se está intercambiando entre ambos agentes.

En general, existen tres tipos de modelos de iniciativa del diálogo [53]. En el primero de ellos la iniciativa del diálogo recae en el sistema (iniciativa del sistema) y el usuario solo puede responder en orden secuencial a las cuestiones formuladas por la interfaz. Esta estrategia produce diálogos poco flexibles pero muy eficaces de cara a lograr el objetivo perseguido durante la conversación. En la segunda estrategia es el usuario quien lleva la iniciativa y quien responde de forma pasiva (iniciativa del usuario). En este caso la flexibilidad es máxima ya que el usuario no tiene ningún tipo de restricción durante la conversación. Aún así, esta aproximación puede producir una pérdida de eficiencia en la consecución del objetivo, ya que se pierde por completo la guía del sistema y el usuario puede terminar confuso o desorientado. La última estrategia es mixta ya que tanto el sistema como el usuario se turnan la iniciativa del diálogo (iniciativa mixta). Este modelo intenta aprovechar las ventajas de las dos estrategias anteriores de forma que se le da cierta libertad al usuario para permitir un diálogo flexible pero a la vez el sistema intenta conducir la conversación de forma que no se pierda el objetivo final.

2.6.1.1. Representación de la información

Teniendo en cuenta cómo se representa la información que intercambian el usuario y el sistema durante el diálogo, se obtiene la siguiente clasificación [54]:

- Sistemas de estados finitos: El diálogo se representa mediante una máquina de estados (grafo) y, por lo tanto el flujo de navegación entre estados (nodos) se puede conocer de forma determinista. En cada estado, el sistema intenta obtener la información necesaria para continuar avanzando hacia el siguiente estado programado. Las principal desventaja de este tipo de sistemas es la excesiva rigidez que presentan, dado que al usuario no se le permite desviarse de la ruta establecida.

- Sistemas basados en *frames*: En estos sistemas se pretende ofrecer una mayor flexibilidad de forma que cada objetivo de diálogo se represente mediante un *frame* con distintos *slots*. En este caso, el orden en que se vayan rellenando dichos *slots* puede ser arbitrario. De esta forma, se gana flexibilidad con respecto al método anterior, ya que aunque la estructura de los *frames* está fijada de antemano, no se requiere que la información para completarlo se aporte de forma determinista.

- Sistemas basados en planes: En este caso cada intervención del usuario se clasifica como un acto de diálogo (saludo, afirmación, pregunta, etc.) y, a su vez, los actos de diálogo forman parte de un plan u objetivo global. El inconveniente de estos sistemas radica en la dificultad para identificar correctamente el plan del usuario.

- Sistemas basados en agentes: Son sistemas adecuados a tareas complejas que requieran una negociación entre sistema y usuario. El flujo del diálogo se determina mediante algún tipo de razonamiento basado en métodos de Inteligencia Artificial y es común que se separe la gestión del diálogo de las acciones relacionadas con el dominio de aplicación. De forma interna, la tarea global se divide en subtareas, de las que será responsable cada uno de los agentes. Estos agentes deben colaborar entre sí para satisfacer los requisitos que llevan al cumplimiento de

la meta global que requiere el usuario. Así, se consigue una mayor flexibilidad al poder desarrollar agentes específicos para subtareas dentro del diálogo, pudiendo ser reutilizados en sistemas conversacionales para aplicaciones diferentes.

2.6.2. Tecnología de bots conversacionales

Los agentes conversacionales clásicos suelen tender a conversaciones demasiado lineales y rígidas. Para una mayor sensación de naturalizad y evitar el rechazo de los usuarios, es necesario que el sistema pueda manejar cambios inesperados de contexto y ser capaz de manejar varios objetivos de forma asíncrona. Un enfoque de este tipo de sistemas con gran éxito tanto en el ámbito académico como empresarial es el de los *bots* conversacionales.

Un *chatterbot* o *bot* conversacional es un programa que simula una conversación con una persona. Actualmente los *chatbots* se utilizan en diferentes ámbitos de aplicación, como pueden ser el marketing, la educación o el entrentenimiento [55] [56]. Uno de los más destacados es ELIZA [57], que es considerado el pionero de los *chatterbots* actuales.

Debido a la escasa información disponible acerca de su funcionamiento y a la inexistencia de transcripciones públicas de alguna conversación, los *chatterbots* más exitosos parecen ser los basados en el Lenguaje de Marcas para Inteligencia Artificial - *Artifical Intelligence Markup Language* (AIML), un lenguaje basado en el Lenguaje de Marcas eXtensible - *eXtensible Markup Language* (XML) que está considerado como un estándar de facto después de hacerse públicos su composición y funcionamiento.

En 1950, Alan Turing propuso un método para determinar si una máquina puede ser considerada inteligente. Este prueba se denomina Test de Turing y se basa en asumir que si una máquina es capaz de comportarse como un ser humano, entonces dicha máquina es inteligente. El desafío consiste en hacer conversar a un interrogador o juez humano mediante un programa informático de chat con otro agente desconocido. El juez debe adivinar si la identidad del agente desconocido pertenece a un ser humano o a una máquina. En 1990 se creó un concurso llamado el "Premio Loebner" [58] que persigue reproducir el Test de Turing. El reto tiene carácter anual y además

del premio principal, existe un premio de consolación para el programa de ordenador que más se aproxime a la consecución del test. Los programas que han conseguido mejores resultados en esta competición son los denominados *bots* de charla o *chatterbots*. A.L.I.C.E. [59], un *chatterbot* basado en AIML ha ganado el "Premio Loebner", en tres ocasiones: 2000, 2001 y 2004.

2.6.2.1. Lenguaje AIML

La unidad básica de conocimiento en AIML se llama "*category*". Cada "*category*" consiste en un estímulo de entrada, una respuesta de salida y un contexto opcional. La Figura 2.5 muestra la estructura básica que forman estos 3 elementos. El estímulo se denomina "*pattern*" y la respuesta de salida se denomina "*template*". Los dos tipos de elementos de contexto se denominan "*that*" y "*topic*".

```
<category>
        <pattern>
                <!--modelo de enunciación del usuario-->
        </pattern>
        <template>
                <!--reacción del bot al modelo -->
        </template>
</category>
```

Figura 2.5: Estructura básica del lenguaje AIML.

El contenido del elemento "*pattern*" ha de cumplir las siguientes reglas:

- Puede contener solamente palabras, espacios y los caracteres comodín "_" y "*".
- Las palabras pueden contener letras o numerales, pero ningún otro tipo de caracteres.

- El "*pattern*" no es sensible a mayúsculas.

- Las palabras se separan por un espacio simple, y los caracteres comodín se tratan como palabras.

El "*template*" en su forma más simple consiste en texto plano, pero puede complementarse con distintos *tags* AIML que transforman la respuesta en un programa reducido de ordenador que puede almacenar datos, activar otros programas, dar respuestas condicionadas, y recursivamente llamar al reconocedor de patrones para insertar las respuestas de otras categorías. De hecho, la mayor parte de los *tags* AIML pertenecen a este sublenguaje del "*template*".

Las partes de las "*category*" opcionales referentes al contexto consisten en dos variantes, llamadas "*that*" y"*topic*". El *tag* "*that*" aparece dentro de la "*category*", y su patrón debe coincidir con la última respuesta del *bot*. La capacidad de poder recordar la respuesta previa del *bot* es importante si éste ha efectuado una pregunta. El *tag* "*topic*" aparece fuera de la "*category*", y agrupa a un conjunto de éstas. El "*topic*" puede ser modificado dentro de cualquier "*template*".

Pudiera parecer que las categorías AIML forman una sencilla base de datos de preguntas y respuestas y, de hecho, el lenguaje es mucho más simple que el Lenguaje de Consulta eStructurado - *Structured Query Language* (SQL). Pero la característica que le confiere un mayor poder de lógica es la posibilidad de que la salida dependa no sólo de una categoría coincidente, sino también de otras alcanzadas a través de un mecanismo recursivo.

El AIML implementa la recursión con el operador "'*srai*". No existe un acuerdo unánime acerca del significado del acrónimo. Dicha falta de acuerdo refleja la variedad de aplicaciones de "*srai*" en AIML. Las más importantes son:

- Reducción simbólica: Reduce las formas gramaticales complejas a otras más simples.

- División de sentencias: Divide la entrada en dos o más partes, y combina las respuestas a ambas.

- Sinónimos: Enlaza diferentes formas de decir lo mismo hacia la misma respuesta.
- Corrección ortográfica y gramatical.
- Detección de palabras clave.
- Condicionales: Ciertas formas de bifurcaciones se pueden implementar mediante el *tag* "srai".
- Cualquier combinación de las anteriores.

2.7. Síntesis artificial de voz (TTS)

La síntesis artificial de voz consiste en la generación de una voz artificial a partir de texto escrito. Los sistemas de síntesis de voz tienen una gran variedad de aplicaciones. Se usan conjuntamente con los reconocedores de voz en agentes conversacionales, pero también son importantes en aplicaciones no conversacionales pero que también hablan a los usuarios, como por ejemplo aplicaciones que leen contenido para personas invidentes, o en videojuegos para niños. Finalmente, la síntesis de voz se puede usar en sistemas que hablan por pacientes que sufren desórdenes neurológicos, como el astrofísico Steven Hawking. Los sistemas de síntesis de voz más modernos permiten lograr una voz muy natural, aunque incluso los mejores sistemas todavía presentan ciertas limitaciones.

Generalmente, un sintetizador se compone a su vez de dos partes: un *front-end* que realiza un análisis textual y un *back-end* cuya función es la generación de formas de onda [60]. A grandes rasgos, el *front-end* toma como entrada texto y produce una representación lingüística fonética. El *back-end* toma como entrada la representación lingüística simbólica y produce una forma de onda sintetizada. El *front-end* desempeña dos tareas principales. Primero, toma el texto y convierte partes problemáticas como números y abreviaturas en palabras equivalentes. Este proceso se llama a menudo normalización de texto o preprocesado. Entonces, asigna una transcripción fonética a cada palabra, y divide y marca el texto en varias unidades prosódicas, como

frases y oraciones. El proceso de asignar transcripciones fonéticas a las palabras recibe el nombre de conversión texto-fonema o grafema-fonema [61]. La combinación de transcripciones fonéticas e información prosódica constituye la representación lingüística fonética. El *back-end*, toma la representación lingüística simbólica y la convierte en sonido. El *back-end* es referenciado muy a menudo directamente como sintetizador.

La generación de formas de onda puede seguir diferentes enfoques estructurados en tres generaciones principales [62]. Las técnicas de la primera generación requieren una descripción detallada de bajo nivel de lo que el sistema va a pronunciar. La segunda generación usa un enfoque basado en datos para incrementar la calidad de la voz artificial. Por otra parte, en la tercera generación se usan métodos estadísticos o de aprendizaje máquina. En este último caso, se consigue una reducción de los requisitos de uso de memoria y permite variar el modelo para, por ejemplo, convertir la voz original en otra diferente [63].

2.7.1. Síntesis de formantes

La síntesis de formantes fue la primera técnica TTS desarrollada y se mantuvo como la técnica dominante hasta los primeros años de la década de los 80. La síntesis de formantes también se suele denominar síntesis por reglas.

El principio básico de la síntesis de formantes consiste en obtener un modelo de la función de transferencia del tracto vocal mediante la simulación de las frecuencias y amplitudes de las formantes. El tracto vocal presenta unas frecuencias de resonancia principales [64] que varían cuando se modifica la configuración del tracto vocal. Estas frecuencias se observan como "picos-esonantes en la función de transferencia del tracto vocal y se conocen como "formantes".

La síntesis consiste en un filtrado basado en modelos matemáticos del órgano vocal del ser humano. La síntesis de formantes hace uso del modelo acústico de tubos resonantes. En este modelo el sonido se genera de una fuente que es periódica para sonidos considerados de voz y ruido blanco para los demás sonidos. A continuación, la fuente se conecta al modelo del

tracto vocal que consiste en la cavidad oral y nasal. Finalmente existe un componente radiante, que simula las generación de la onda de presión que representa la voz artificial.

La técnica de síntesis de formantes genera una voz artificial y robótica pero inteligible incluso a alta velocidad. Además, no constituye un proceso computacionalmente intensivo, especialmente para los procesadores actuales [65]. Por lo tanto, las ventajas de la síntesis de formantes son su relativa simplicidad y su escaso uso de memoria, lo que la hace especialmente adecuada para sistemas empotrados y dispositivos móviles. Por último, dado que los sistemas basados en formantes tienen un control total sobre todos los aspectos del habla producida, pueden incorporar una amplia variedad de tipos de entonaciones, que no sólo comprendan preguntas y enunciaciones.

2.7.2. Síntesis articulatoria

La síntesis articulatoria genera la voz artificial mediante modelos mecánicos y acústicos de la producción de la voz.

La síntesis articulatoria transforma un vector de parámetros anatómicos o fisiológicos en una señal vocal con propiedades acústicas predefinidas [60]. Produce una onda acústica completa basada en modelos matemáticos de las estructuras del tracto vocal (labios, dientes, lengua, glotis y velo) y los procesos articulatorios (tránsito del flujo de aire a través de las cavidades supraglotales) de la generación de la voz en los seres humanos. El proceso completo convierte cadenas de texto en descripciones fonéticas, ayudado por un diccionario de pronunciación, reglas de caracteres-sonidos y ritmo, y modelos de entonación. Posteriormente transforma las descripciones fonéticas en parámetros de articulación de bajo nivel para el sintetizador, que son usados para generar un modelo de articulación vocal humano produciendo un formato adecuado para los dispositivos de salida de audio. Esta técnica es intensiva computacionalmente.

Existen dos dificultades principales en la síntesis articulatoria: cómo obtener los parámetros de control a partir de las especificaciones y cómo encontrar el equilibrio correcto entre un modelo altamente preciso que represente fielmente la fisiología humana y otra representación más pragmática que sea

más sencilla de diseñar y manejar [66].

2.7.3. Síntesis concatenativa

La síntesis concatenativa genera la voz artificial en base a segmentos pregrabados de uno o varios locutores humanos. La base de datos de segmentos intenta reflejar el mayor número de variedades fonológicas del lenguaje [67].

Las técnicas de concatenación toman unidades cortas de voz, normalmente en forma de onda acústica, y enlazan secuencias de estas unidades simples para producir una onda completa que represente todo el texto de entrada al sintetizador.

En los sistemas concatenativos es muy importante la selección apropiada de las unidades y de los algoritmos que unen esas unidades y también realizan algún tipo de procesado de señal posterior para suavizar las transiciones y ajustarse a los esquemas prosódicos predefinidos.

Generalmente, la síntesis concatenativa produce los resultados más naturales. Sin embargo, la variación natural del habla y las técnicas automatizadas de segmentación de formas de onda resultan en defectos audibles, que conllevan una pérdida de naturalidad.

Existen tres subtipos principales de síntesis concatenativa [68]: la síntesis basada en difonos, la síntesis de dominio específico y la síntesis por selección de unidades.

2.7.3.1. Síntesis de dífonos

La síntesis de difonos es el método más empleado para crear una voz sintética a partir de grabaciones o muestras de una persona particular, que se almacenan en una base de datos nominal [69]. La cantidad de difonos presentes en la base de datos de ser la misma que la existente en el idioma de la voz que se pretende recrear. En tiempo de ejecución, la prosodia de la oración se superpone a estas unidades mínimas mediante procesado digital de la señal, dotando así a la voz artificial de unas pronunciación y entonación más naturales.

La síntesis de difonos presenta problemas en idiomas con una gran cantidad de inconsistencias en la pronunciación. También suelen ocurrir disconti-

nuidades entre sonidos vocálicos adyacentes que pueden llegar a ser percibidos como discontinuidades entre dos difonos consecutivos.

2.7.3.2. Síntesis de dominio específico

La síntesis de dominio específico concatena palabras y frases grabadas para crear oraciones completas [60]. Se usa en aplicaciones donde la variedad de textos que el sistema puede producir está limitada a un dominio particular, como anuncios de salidas de trenes o información meteorológica [70].

Esta tecnología es muy sencilla de implementar, y se ha usado comercialmente durante bastante tiempo. La naturalidad de estos sistemas puede ser elevada en general debido a que la variedad de oraciones está limitada y se corresponde con la entonación y la prosodia de las grabaciones originales. Sin embargo, al estar limitados a unas ciertas frases y palabras de la base de datos, no son de propósito general y sólo pueden sintetizar la combinación de palabras y frases para los que fueron diseñados.

2.7.3.3. Síntesis por selección de unidades

La síntesis por selección de unidades es la técnica de TTS dominante. Esta técnica es la extensión de la segunda generación de síntesis concatenativa y abarca problemáticas más complejas: cómo manejar eficientenemente cantidades elevadas de unidades, cómo extender la prosodia y el control del tiempo en la oración, y cómo reducir las distorsiones causadas por el procesado de la señal [69].

Durante la creación de la base de datos, el habla se segmenta en algunas o todas de las siguientes unidades: fonemas, sílabas, palabras, frases y oraciones. Normalmente, la división en segmentos se realiza usando un reconocedor de voz modificado para forzar su alineamiento con un texto conocido. Posteriormente se corrige manualmente, usando representaciones como la forma de onda y el espectrograma [71]. La especificación de las unidades se describe con una estructura de características, que pueden ser una mezcla de características acústicas y lingüísticas. Durante la síntesis, un algoritmo selecciona la mejor unidad de todas las posibles con el objetivo global de obtener la secuencia de unidades que mejor se ajuste a las especificaciones. Este proceso

se logra típicamente usando un árbol de decisión especialmente ponderado. Debido al hecho de que la selección de unidades no aplica mucho procesado digital de la señal al habla grabada, a menudo hace que el sonido grabado suene menos natural. Para solucionar este inconveniente, algunos sistemas incluyen una etapa de procesado de señal en la concatenación para suavizar las formas de onda [65].

La gran diferencia entre la selección de unidades y la síntesis basada en difonos es la longitud de los segmentos de voz utilizados. En la selección de unidades se emplean como unidades palabras e incluso frases enteras. Esto implica que la bases de datos es varios órdenes de magnitud mayor que en la de difonos [72]. Por lo tanto, mientras que el consumo de CPU es relativamente bajo, el consumo de memoria es muy elevado.

2.7.4. Otros métodos de síntesis

En las secciones previas se han revisado las técnicas más importantes y extendidas de la síntesis de voz, pero otros autores han propuesto nuevos enfoques. Uno de ellos es el de intentar aunar las ventajas de la síntesis concatenativa y la de formantes para minimizar los defectos acústicos en la concatenación de segmentos. Esta técnica se denomina síntesis híbrida.

Los métodos más recientes de síntesis intentan aprovechar modelos probabilísticos basados en aprendizaje supervisado para generar modelos más complejos y naturales. Un ejemplo de ello es el de la síntesis basada en HMMs [73]. En esta técnica el habla, el espectro de frecuencias (tracto vocal), la frecuencia fundamental (fuente vocal) y la duración (prosodia) se modelan simultáneamente mediante HMMs. Finalmente, la forma de onda se genera mediante el criterio de máxima verosimilitud.

2.8. Animación de la cabeza virtual (VHA)

Los avatares representan un punto importante de investigación tanto en el ámbito académico como en el industrial debido a la gran cantidad de aplicaciones potenciales.

Los avatares se pueden dividir de forma general en 2 categorías dependiendo del tipo de interacción que el avatar establece con el mundo exterior. La primera de ellas se enmarca en las comunicaciones hombre-hombre, como la telepresencia. La segunda actúa como un agente inteligente en las interacciones hombre-máquina.

La tarea de crear un avatar natural y realista como los que se pueden encontrar en algunos videojuegos o películas puede ser muy complicada. En cambio, los avatares de tipo *cartoon* son mucho más fáciles de desarrollar. Tradicionalmente, se empleaban caros sistemas de captura de movimiento para registrar los movimientos de una persona. Ahora también es común que existan personas expertas que puedan recrear los avatares mediante técnicas de escultura, modelado y animación en 3D.

En el mundo de la animación facial existen una gran variedad de técnicas basadas en modelos en 3D [74]. En general, estas técnicas crean primero una cara 3D formada por una malla 3D, que define la forma geométrica de la cara. El segundo paso consiste en una proyectar sobre la malla 3D una textura de tipo humano o *cartoon*. Además del modelo 3D, también se deben definir los parámetros que determinan las animaciones posibles del avatar.

Un avatar 3D tradicional requiere un modelo geométrico muy preciso, ya que se deben representar incluso tejidos blandos como los labios, los músculos faciales o la lengua. En esta fase de creación del avatar se debe prestar una especial atención, ya que cualquier deformación extraña del modelo que no encaje con el comportamiento de una cara natural propiciará el rechazo del usuario. Este tipo de fenómeno se denomina "*uncanny valley*" [75].

Las animaciones faciales basadas en imágenes logran un gran realismo en vídeos sintetizados a partir de diferentes partes de la cara obtenidas de imágenes 2D pregrabadas [76] [77]. La desventaja de esta técnica es que es complicado cambiar la pose de la cabeza de forma natural o renderizar diferentes expresiones faciales.

Otro tipo de técnica muy común es el "*blend-shape*". Con este método se crean las expresiones faciales deseadas gracias a la combinación de un conjunto de ejemplos preexistentes. La combinación de estas expresiones previas se suele hacer mediante interpolación lineal aplicada a imágenes [78] o a modelos faciales [79], o puede ser basada en *morphing* de modelos [80].

2.9. Conclusiones

En este Capítulo se ha hecho una revisión de los conceptos generales de los ECAs. Se ha descrito la estructura interna de un ECA y las diferentes arquitecturas de comunicación posibles entre sus elementos (distribuida, empotrada o mixta), destacando las ventajas e inconvenientes que éstas presentan. A continuación se han detallado los módulos del ECA (VAD, ASR, CE, TTS y VHA), estudiando las principales aproximaciones teóricas que constituyen el estado del arte para cada uno de ellos. Con todo esto se cumple el primero de los objetivos indicados en el Capítulo 1.

Una vez satisfecha la aproximación teórica a los ECAs, el siguiente paso es el de realizar un estudio comparativo entre las diferentes alternativas de implementación disponibles para cada módulo interno del agente.

Capítulo 3

Diseño de la plataforma base del agente

3.1. Introducción

La construcción de una plataforma de desarrollo de agentes conversacionales 3D es un proceso tedioso y complejo. Como se ha visto en el Capítulo 2, un agente conversacional 3D está formado por diferentes módulos que deben trabajar conjuntamente para lograr un objetivo global. Además, el contexto de los sistemas empotrados en que se desarrolla esta tesis, establece una limitación estricta de la complejidad global del agente en su conjunto, por lo que cada módulo que forma parte de éste debe seleccionarse bajo criterios de eficiencia computacional y consumo de recursos.

A lo largo del mundo, existe una multitud de grupos de investigación y comunidades de programadores que desarrollan implementaciones de cada uno de los módulos necesarios para construir un agente conversacional 3D. Por lo tanto, este capítulo refleja el proceso incremental y temporal de estudio y selección de los diferentes componentes que forman la plataforma de desarrollo de agentes conversacionales 3D que es objeto de análisis y optimización en esta tesis.

3.2. Descripción de alternativas

La arquitectura funcional que se persigue para la plataforma software de desarrollo de agentes conversacionales 3D es la misma que la vista la Figura 2.1 del Capítulo 2. Por lo tanto, en las secciones que vienen a continuación se hará un estudio de las diferentes implementaciones existentes para cada uno de los módulos que forman la plataforma. Debido al ámbito académico de este trabajo, y también con el propósito de garantizar el futuro de esta investigación y su difusión, se restringe la elección de componentes a aquellos que sean gratuitos y de código abierto.

3.2.1. Detección de actividad vocal

Debido a que muchos sistemas ASR se usan en reconocimiento *offline* de grandes secuencias de audio (por ejemplo para transcripciones de discursos), la mayoría de las implementaciones de código abierto de ASR no contemplan la funcionalidad del VAD. Una de las pocas excepciones es el grupo de sistemas de reconocimiento de voz CMU Sphinx. Se pueden encontrar dos implementaciones diferentes en función del tipo de reconocedor elegido. La librería Sphinxbase [81], en lenguaje C y necesaria para el uso del reconocedor PocketSphinx, implementa un algoritmo de VAD basado en energía que adapta el umbral de ruido ambiente de forma dinámica cada pocos segundos. De la misma familia es el módulo SpeechClassifier perteneciente al reconocedor Sphinx4, siendo éste programado en lenguaje Java. Ambas librerías cuentan con licencia BSD y permiten una integración inmediata del módulo VAD con el ASR.

Otra implementación de algoritmo VAD asociado a un sistemas ASR se puede encontrar en el paquete SHoUT [82] [83]. En este caso, el algoritmo se basa en GMMs. Este tipo de técnicas pueden ofrecer una mayor robustez en entornos con ruido ambiente elevado, pero tiene como gran desventaja que requiere entrenar el modelo del entorno donde el sistema ASR va a ser empleado.

En cambio, sí es más frecuente encontrar implementaciones de algoritmos VAD en librerías de compresión de audio, ya que, como se comentó en el Capítulo 2, el uso principal de este tipo de algoritmos se encuentra en el

ámbito de las comunicaciones telefónicas por red fija y/o móvil. El ejemplo más significativo es la librería Speex [84] [85], cuyo objetivo es crear un códec libre para voz con licencia BSD y libre de patentes de software. Esta librería incluye la funcionalidad de VAD, pero está integrada con la compresión a tasa de bit variable. Esta característica haría necesario modificar el código fuente de la librería para poder aprovechar la funcionalidad VAD sin aplicar sistema de compresión alguno en sistemas empotrados. Es necesario tener en cuenta que el módulo ASR recibe a su entrada las muestras de audio sin comprimir.

3.2.2. Reconocimiento automático de voz

En los últimos años el interés en la investigación de los sistemas ASR ha crecido tanto que actualmente existen un buen número de librerías de ASR de codigo abierto disponibles en el mundo académico. Pero en el momento de inicio de esta tesis eran solamente dos las que gozaban de amplio reconocimiento y difusión: el paquete *Hidden Markov Model Toolkit* (HTK) y la familia de sistemas CMU Sphinx.

3.2.2.1. HTK

El paquete HTK es un software para construir y manipular HMMs [86] [87]. Su uso se enfoca principalmente en el reconocimiento de voz, pero también ha sido utilizado en otro tipo de aplicaciones como la síntesis de voz, el reconocimiento de caracteres y la secuenciación de ADN.

HTK está formado por un conjunto de módulos y herramientas en lenguaje C. Estas herramientas proveen funcionalidades para análisis de la voz humana, entrenamiento de HMMs, ejecución de pruebas y análisis de resultados. Ofrece una gran flexibilidad para construir sistemas complejos basados en HMMs debido a que soporta GMMs discretos y continuos. Su principal desventaja es que Microsoft es el propietario del *copyright* del código fuente, y aunque existe total libertad para su uso y modificación en el ámbito académico, no está permitida la redistribución de ninguna parte del código.

3.2.2.2. CMU Sphinx

CMU Sphinx, es el término general para describir un grupo de sistemas de reconocimiento de voz desarrollado en la Universidad Carnegie Mellon (CMU). Incluye una serie de programas para reconocimiento de voz conocidos como Sphinx 2, 3, 4 y PocketSphinx, y un entrenador de modelos acústicos llamado SphinxTrain.

En el año 2000, el grupo de Sphinx se comprometió a desarrollar varios componentes para reconocimiento de voz, incluyendo Sphinx 2 y más tarde Sphinx 3 en 2001. Posteriormente llegaron las versiones PocketSphinx, dirigido a sistemas empotrados y Sphinx 4, el sistema más reciente desarrollado íntegramente en Java y con un enfoque muy modular y escalable. Aunque su función es la misma, todas las versiones tienen ciertas características específicas que las hacen adecuadas para diferentes situaciones y necesidades, como se puede ver en la Figura 3.1. Los decodificadores de voz vienen con modelos acústicos y aplicaciones de ejemplo. Los recursos disponibles incluyen además el software para el entrenamiento de modelos acústicos, la compilación de un modelo de lenguaje y un diccionario de pronunciación en dominio público llamado "*cmudict*".

La primera versión de Sphinx es un sistema ASR basado en un modelo acústico de HMMs y un modelo de lenguaje estadístico de n-gramas. Fue desarrollado por Kai Fu-Lee [88]. Sphinx interpreta voz hablada en forma continua y reconocimiento de habla de vocabulario amplio.

Sphinx 2 es un sistema de reconocimiento de habla de alta resolución desarrollado originalmente por Xuedong Huang en la CMU [89] [90], quien liberó su código como software libre con una licencia BSD. Sphinx 2 se centra en el reconocmiento de voz en tiempo real y es adecuado para aplicaciones de lenguaje hablado. Incorpora funciones tales como: detección de final de frase, generación de hipótesis parciales y dinámica de modelo de lenguaje. Es utilizado en sistemas de diálogo y los sistemas de aprendizaje de idiomas. Sphinx 2 utiliza una representación semicontinua para el modelado acústico. Es decir, un único conjunto de gaussianas se utiliza para todos los modelos, con los modelos individuales como un vector de peso durante estas gaussianas. Además, también puede ser utilizado en sistemas informáticos basados en

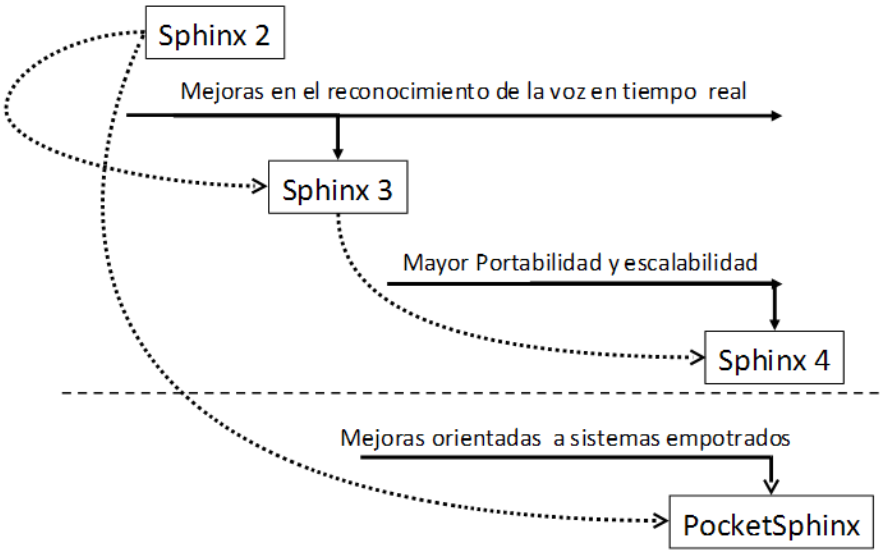


Figura 3.1: Evolución de los reconocedores de la familia CMU Sphinx.

Private Branch eXchange (PBX) como Asterisk.

Así como Sphinx 2 emplea la representación semi-continua para el modelo acústico, Sphinx 3 adopta el más reciente modelo continuo de HMM [91] [92]. De esta forma, esta versión se caracteriza por su elevada precisión a costa de un mayor tiempo de procesado. Aunque los recientes avances en algoritmos y *hardware* han hecho de Sphinx 3 un sistema casi de tiempo real, todavía no es adecuado su uso en aplicaciones cuya interactividad sea un factor crítico. Sphinx 3 sigue estando en desarrollo y en conjunto con SphinxTrain [93], otorga acceso a un gran número de técnicas de procesado actuales que mejoran la tasa de acierto del sistema.

Con Sphinx 4 se llevó a cabo una completa re-escritura del sistema Sphinx con el objetivo de proporcionar un marco más flexible para la investigación en reconocimiento de voz [94] [95]. Está escrito íntegramente en lenguaje de programación Java y Sun Microsystems apoya su desarrollo y contribuye activamente en el proyecto. Al estar escrito en java puede ser utilizado en una gran diversidad de SOs y *hardware*, lo que lo dota de una gran difusión y escalabilidad.

PocketSphinx marcó una nueva línea de investigación y desarrollo dentro de la familia CMU Sphinx, debido a su orientación especifica a sistemas empotrados o con recursos limitados [96] [97]. Parte de la base del reconocedor Sphinx 2 sobre el que se implementaron mejoras a nivel de gestión de memoria, a nivel de máquina y también en los algoritmos implementados. Gracias a todas estas optimizaciones, PocketSphinx permite implementar sistemas de reconocimiento continuo en tiempo real en sistemas empotrados de bajas prestaciones.

3.2.3. Motor conversacional

En el Capítulo 2 se han visto los diferentes enfoques conceptuales que predominan en la literatura de los sistemas conversacionales o de diálogo. Ahora es el momento de describir las implementaciones disponibles que cumplan los criterios de gratuidad y código abierto y que, además, hayan sido utilizados en aplicaciones reales en el ámbito académico o comercial.

3.2.3.1. Galaxy Communicator

En el grupo de sistemas de lenguaje hablado del Instituto Tecnológico de Massachusetts (MIT) desarrolló a finales de los años 90 una arquitectura genérica de referencia para el desarrollo de sistemas conversacionales [98] [97]. Su éxito fue enorme debido a que fue seleccionada por el programa "Communicator" de la Agencia de Proyectos de Investigación Avanzada en Defensa - *Defense Advanced Research Projects Agency* (DARPA). El propósito de este programa era el desarrollo rápido y de bajo coste de sistemas de diálogo hablado [99].

Como se puede ver en la Figura 3.2, la arquitectura del Galaxy Communicator sigue un modelo cliente-servidor. Un módulo *hub* es el encargado de centralizar la comunicación, entre el resto de módulos que componen la arquitectura. La representación de la información se basa en *frames* y la comunicación entre componentes se rige por la programación del *hub* en base a un sistema fijado de reglas. Los elementos constituyentes de los *frames* se denominan *clause*, *topic* y *predicate*. El elemento *clause* representa el objetivo global de la petición de usuario; ejemplos de ello pueden ser "mostrar", "gra-

bar", "repetir", "reservar", etc.. Los *topics* se corresponden generalmente con frases nominales y los *predicate* son típicamente los atributos, que pueden ser expresados como frases verbales, frases preposicionales o frases adjetivas. En la Figura 3.3 se muestra a modo de ejemplo el *frame* que representa la frase "Muéstrame los vuelos de Boston a Denver".

El código fuente del Galaxy Communicator se ofrece como una distribución completa bajo la licencia MIT y cuenta con una extensa documentación de cada uno de las partes de la arquitectura. También cuenta con un *toolkit* que implementa algunos de los servidores de la arquitectura y también dispone de *wrappers* para el uso de $software$ de terceros.

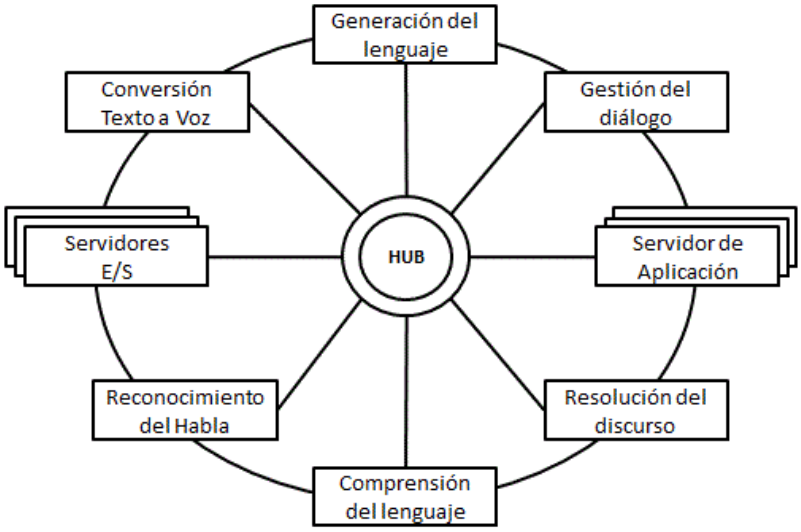


Figura 3.2: Arquitectura del sistema Galaxy Communicator.

3.2.3.2. Olympus/Ravenclaw

Usando como base la arquitectura del Galaxy Communicator, a finales de los 2000 los investigadores de la CMU desarrollaron su propia plataforma para el desarrollo de sistemas conversacionales [100]. También recibió fondos

```
Clause:
   { display
     topic:
      { flight
        number: pl
        predicate:
             { from
               topic: {city name: Boston
             }
        predicate:
             { to
               topic: {city name: Denver
             }
      }
   }
```

Figura 3.3: Ejemplo de frame del sistema Galaxy Communicator.

del programa DARPA Communicator y se empleó con éxito en numerosos proyectos [101].

A nivel de arquitectura, Olympus sigue una estructura similar a la cliente-servidor de la plataforma Galaxy, tal como se muestra en la Figura 3.4. De hecho, la comunicación y el paso de mensajes entre el *hub* y los servidores es exactamente la misma que en el sistema anterior. Como diferencias notables se puede ver que Olympus ofrece una versión diferente de servidores que ofrecen funcionalidades mejoradas con respecto al Galaxy Communicator.

El flujo de ejecución comienza con el servidor de reconocimiento de voz Sphinx (que puede emplear las versiones Sphinx 2 y Sphinx 3), a continuación el módulo Phoenix realiza la comprensión del texto basado en gramáticas semánticas simples que extraen la información mediante la técnica de *slot-filling*. A partir de la información de los anteriores módulos, el servidor Helios calcula una medida de confianza. El gestor del diálogo Ravenclaw es la pieza fundamental de la arquitectura Olympus [102]. La programación de este servidor es dependiente del dominio de aplicación y representa la conversación mediante una estructura en árbol que captura la estructura jerárquica del

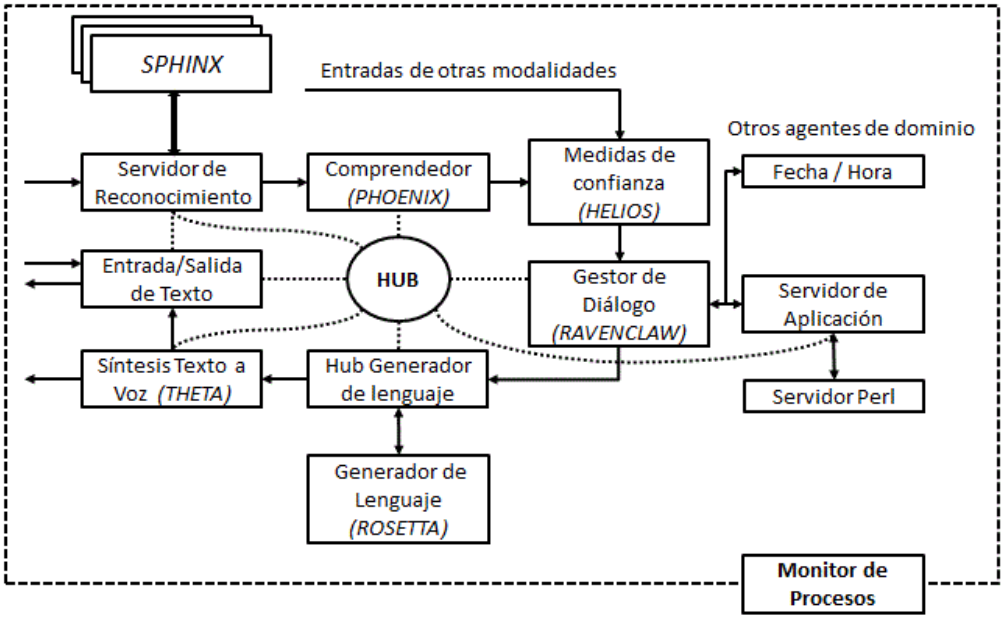


Figura 3.4: Arquitectura del sistema Olympus/Ravenclaw.

diálogo, como se puede ver el la Figura 3.5. La respuesta semántica ofrecida por Ravenclaw se convierte en texto a través del servidor de generación de lenguaje Rosetta mediante el uso de plantillas. Finalmente, el servidor de síntesis de voz Kalliope genera el audio con la respuesta del sistema.

Olympus se ofrece bajo licencia BSD pero para poder construir un sistema conversacional completo es necesario cumplir con una lista bastante completa de requisitos entre los que se encuentra el SO, que debe ser Microsoft Windows.

3.2.3.3. Jaspis

El éxito del Galaxy Communicator también se extendió al continente europeo y un grupo de investigación de la Universidad de Tampere desarrolló una plataforma propia denominada Jaspis [103] [104]. El objetivo de Jaspis es la implementación de sistemas conversacionales multilíngües, dis-

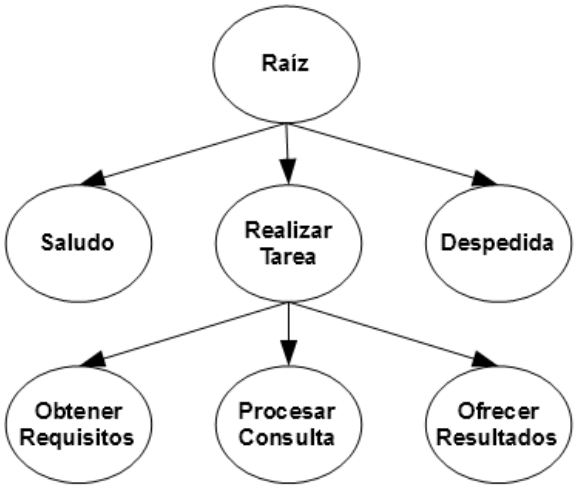


Figura 3.5: Estructura en árbol compuesta por agentes en Ravenclaw.

tribuidos y adaptados al entorno y usuario.

La arquitectura de Jaspis que se muestra en la Figura 3.6 sigue un esquema cliente-servidor en sintonía con el Galaxy Communicator y Olympus. El nivel superior lo forman los "*managers*", conectados en forma de estrella y gestionan la comunicación entre los distintos componentes. Los "*agents*" son los componentes que manejan las situaciones en las que se requiere interacción con el usuario y los "*evaluators*" se encargan de seleccionar los los "*agents*" más adecuados en cada momento. La información del sistema se almacena en bases de datos compartidas que pueden ser consultadas por todos los componentes de la arquitectura mediante el gestor de la información.

La implementación de Jaspis se liberó bajo licencia LGPL y está basado en Java y XML como lenguajes de programación.

3.2.3.4. AIML

Como ya se ha visto en el Capítulo 2, la tecnología de *chatterbots* son una alternativa a los sistemas conversacionales clásicos. En este ámbito el enfoque que más atención ha recibido es el basado en el lenguaje AIML [105].

Cada *chatterbot* consta de dos componentes básicos: la base de conoci-

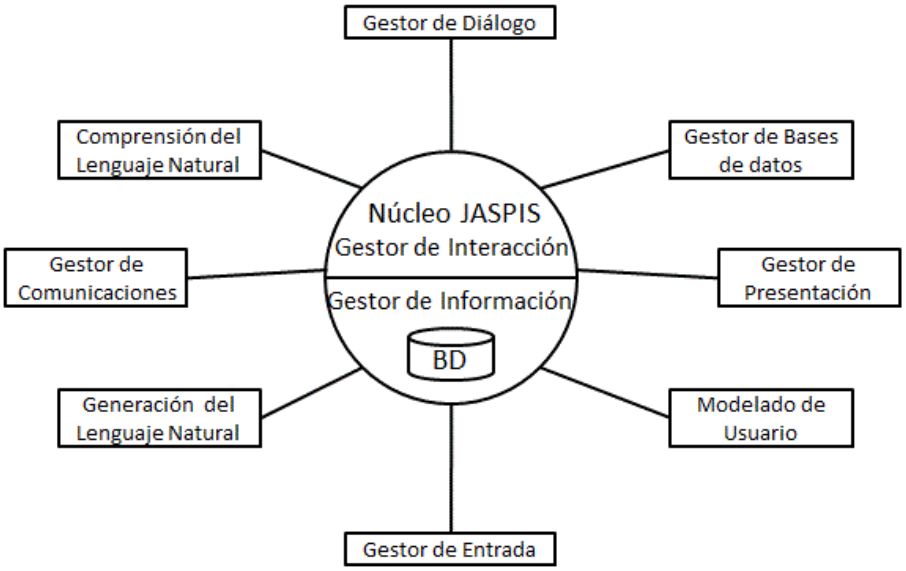


Figura 3.6: Arquitectura del sistema Jaspis.

mientos y el motor de procesamiento de estos datos, el intérprete del lenguaje. Debido al carácter libre y gratuito de las especificaciones del lenguaje AIML, se han desarrollado intérpretes en una multitud de lenguajes de programación: C++, Java, PHP, Python, etc. La gran mayoría tienen licencia GPL o LGPL disponibles para que cualquier desarrollador pueda participar en su evolución o adaptarlo a sus necesidades.

Internamente, un intérprete AIML está formado por 4 módulos principales que se muestran en la Figura 3.7:

- Analizador AIML: Su función es procesar el contenido de los ficheros AIML, incorporando todas las nuevas categorías al "cerebro" del *bot*.

- Sustitución de palabras: Realiza la primera etapa de normalización de sentencias del intérprete, consistente en sustituir palabras que podrían entorpecer el reconocimiento de patrones.

- Buscador de patrones: Implementa el algoritmo de búsqueda que permite al *bot* identificar la entrada del usuario y construir una respuesta adecuada.
- Núcleo: Es el componente que une todas las partes antes mencionadas y cumple también la función de interfaz pública del motor conversacional. Además, incorpora funciones de configuración general y otras funciones auxiliares para, por ejemplo, permitir la persistencia del estado del *bot* entre distintas ejecuciones.

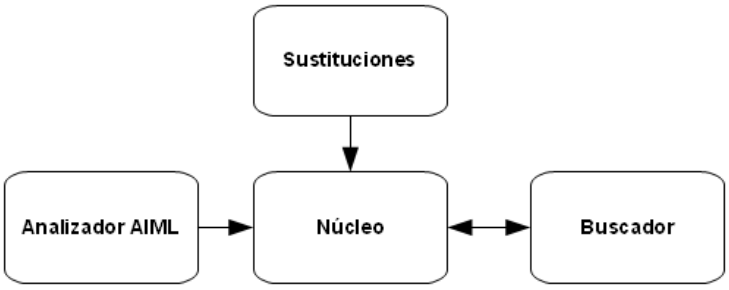


Figura 3.7: Arquitectura de un intérprete AIML.

3.2.4. Síntesis artificial de voz

Para poder describir las diferentes alternativas existentes para el módulo TTS es conveniente recordar a grandes rasgos la composición interna de un sistema de síntesis artificial de voz. La síntesis de voz consiste en dos fases principales: análisis textual (el *front-end*) y generación de formas de onda (el *back-end*). En la primera, el texto de entrada se transcribe a una representación fonética o lingüística mediante el uso de reglas fijas. La segunda fase convierte la representación anterior en la onda acústica que contiene la voz artificial.

Aunque existen diversos sintetizadores comerciales que gozan de gran difusión como los de Loquendo [106] o Verbio [107], también la comunidad de software libre ha avanzado mucho en este campo y ofrece actualmente varias implementaciones de gran calidad como eSpeak, FreeTTS, Festival, GNUSpeech, MBROLA o Flite. A continuación se describen aquellas más relevantes.

3.2.4.1. eSpeak

Es un sintetizador de código abierto y gratuito basado en formantes [108] [109], lo que le permite mantener un tamaño reducido en memoria y ofrecer un amplio número de idiomas a costa de presentar una voz con sonoridad robótica. Además, soporta un gran número de plataformas como Windows, Linux, MAC OS, RISC OS, etc..

3.2.4.2. MBROLA

Aunque también es un sintetizador gratuito, se ofrece en formato binario sin el código fuente [110] [111]. Se basa en la concatenación de difonemas y presenta una calidad mayor a otros programas basados en la misma técnica. Dado que MBROLA requiere como entrada el conjunto de fonemas a sintetizar, no se puede utilizar de forma aislada. Por lo tanto, es necesario su uso concatenado al de otra librería que actúe como *front-end* para la generación de fonemas y prosodia para, a continuación, ser MBROLA quien realice la síntesis final.

3.2.4.3. Festival

Es un sintetizador de código abierto y ofrecido gratuitamente desde la universidad de Edimburgo [112] [113]. De hecho, se ofrece bajo un tipo de licencia similar a la licencia MIT, que permite su uso libremente, tanto para utilización comercial como de otro tipo. El proyecto incluye la documentación completa para desarrollar sistemas de síntesis de voz con varias Interfaces de Programación de Aplicaciones - *Application Programming Interfaces* (APIs), siendo un entorno ideal para el desarrollo e investigación de las técnicas de síntesis de voz. Su núcleo está programado en C++ aunque se puede configurar utilizando Scheme (lenguaje de programación similar a LISP). Festival ofrece síntesis concatenativa con un algoritmo robusto basado en concatenación de difonemas, y varias opciones alternativas como “Clunits”, basado en selección de unidades, “HTS” basado en HMM o “ClusterGen”. Actualmente soporta los idiomas inglés (británico y americano) y español.

3.2.4.4. Flite

Flite ofrece una implementación de síntesis TTS ligera y eficiente [114] [115]. Su diseño se enfoca a su uso en sistemas empotrados y servidores de carga elevada y que deben servir síntesis a muchos puertos simultáneamente. Flite es parte de un conjunto de herramientas de síntesis gratuitas creado en la CMU.

Flite fue desarrollado principalmente para solucionar los problemas de la librería Festival. Festival es una librería pesada, compleja y lenta. Además está escrita en C++ con una capa superior en Scheme, lo que la hace poco portable. Para solucionar esto, Flite está escrita en ANSI C, quedándose solo con lo imprescindible para realizar la función de TTS y fue diseñada para ser portable a prácticamente cualquier plataforma, incluyendo sistemas empotrados sencillos.

3.2.5. Animación 3D

Para la parte visual del agente conversacional el objetivo es seleccionar un sistema de renderizado o *renderer* 3D gratuito, de código abierto y sencillo, ya que la encarnación virtual del agente conversacional suele ser sencillamente una cabeza parlante. Por lo tanto, se deben descartar los llamados *game engines* puesto que, además del *renderer*, aportan muchas funcionalidades no necesarias como la gestión de la inteligencia artificial de los personajes o la detección de colisiones entre otras. A continuación se describen los *renderers* más relevantes: Irrlicht y Ogre3D.

3.2.5.1. Irrlicht

Irrlicht es un motor de gráficos 3D de código abierto y multiplataforma [116] [117]. Está escrito en C++ y soporta otros lenguajes de programación mediante *bindings*. Su potencia recae sobre las reconocidas Direct3D y OpenGL aunque también ofrece su propio *software* de renderizado. Irrlicht tiene una licencia basada en la zlib/libpng y fue diseñado para ser un motor gráfico ligero pero potente y con una curva de aprendizaje corta comparada con la de otros motores.

3.2.5.2. Ogre3D

Ogre3D es un motor de renderizado 3D orientado a escenas, de código abierto y escrito totalmente en lenguaje C++ [118] [119]. Su diseño evita al desarrollador la necesidad de acceder directamente a las APIs de OpenGL y Direct3D. Se distribuye bajo la licencia MIT y presenta una comunidad muy activa, con mucha documentación y la experiencia de haber sido empleado en juegos comerciales.

3.3. Selección de componentes para la plataforma base

En la Sección anterior se ha podido comprobar que para cada parte que compone un agente conversacional 3D existen varias alternativas de implementación, por lo que el número de combinaciones posibles es elevado y requeriría demasiado tiempo de documentación, adaptación e integración de componentes.

Para el trabajo de esta tesis se ha seguido el criterio de realizar una primera selección de aquellos componentes que, desde su concepción y diseño, están orientados a obtener prestaciones en sistemas con potencia computacional reducida.

Dado que la tarea de reconocimiento de voz es posiblemente la más compleja de todo el agente conversacional, cabe comenzar con la elección de los módulos ASR del agente conversacional. En este apartado parece clara la selección de la librería PocketSphinx para implementar el módulo ASR. PocketSphinx está diseñado y optimizado específicamente para ser ejecutado en sistemas empotrados y es capaz de obtener tiempos de respuesta significativamente inferiores que las alternativas Sphinx 2, Sphinx 3 y HTK [120].

Por mantener el diseño del agente lo más sencillo posible, parece lógico elegir la librería SphinxBase como implementación del módulo VAD del agente conversacional. SphinxBase se integra perfectamente con PocketSphinx y ofrece un algoritmo VAD robusto y que se adapta de forma dinámica a los cambios del entorno. El paquete SHoUT no parece adecuado debido a que requiere el entrenamiento del modelo en un entorno de idénticas característi-

cas al que donde va a ser usado el sistema. Por otra parte, la funcionalidad VAD de la librería Speex requiere modificar sensiblemente el código fuente para poder hacerla independiente de la codificación del audio.

En cuanto al módulo CE, se ha visto que los sistemas Galaxy Communicator, Olympus/Ravenclaw y Jaspis siguen una arquitectura muy parecida, de tipo cliente-servidor con un nodo *hub* donde se centraliza el control y el flujo de información del sistema conversacional. Los 3 sistemas parecen realizan un desacoplamiento de todas las funciones que lleva a cabo el sistema conversacional y parecen adecuadas a grandes sistemas de información que residen en servidores y deben atender a una cantidad elevada de usuarios. Su gran modularidad les hace muy flexibles pero a la vez incrementa la curva de aprendizaje y el tiempo necesario para el desarrollo de un sistema conversacional completo, incluso para tareas de baja complejidad. En cambio, los intérpretes de lenguaje AIML son tremendamente sencillos a la hora de su configuración y programación, lo que los hace más adecuados para sistemas empotrados. La complejidad y riqueza de las aplicaciones desarrolladas solamente requiere el dominio del lenguaje AIML. Además, su implementación se centra en las tareas necesarias de comprensión, gestión del diálogo y generación de respuestas, porque no se presupone el uso de otras funcionalidades como ASR o TTS. Por todas estas razones, se ha elegido como base para el motor conversacional un intérprete AIML. Dado que existen una gran variedad de intérpretes AIML de código libre disponibles, se ha optado por seleccionar uno de ellos como punto de partida, PyAIML [121], incorporando las mejoras y correcciones necesarias para obtener la funcionalidad pretendida. PyAIML está escrito completamente en Python, un lenguaje excelente para la integración de otras funcionalidades en otros lenguajes y su implementación se ciñe exclusivamente a la referencia del lenguaje AIML 1.0.1, sin funcionalidades adicionales.

La elección del módulo TTS es la más dificil de realizar sin un análisis cuantitativo debido a que el rendimiento de la síntesis depende en gran medida de la implementación de la voz en castellano en cada una de las alternativas estudiadas en la Sección anterior. Aunque Flite sea aparentemente una opción muy interesante debido a su portabilidad y diseño orientado a sistemas de bajas prestaciones, el paquete de esta librería no ofrece ninguna

voz en castellano. Aunque se base en la plataforma Festival, las voces deben ser recodificadas a lenguaje C y compiladas para poder ser utilizadas en Flite. Debido a estas razones, se contará con eSpeak, MBROLA y Festival como módulos TTS y se hará un análisis cuantitativo del rendimiento de cada una de las alternativas en el siguiente capítulo, con el fin de establecer qué implementación obtiene mejores prestaciones.

Finalmente, en la elección del módulo VHA, aunque la sencillez, el diseño ligero y la rápida curva de aprendizaje de Irrlicht son características muy atractivas, debe ser descartado ya que solamente tiene un soporte experimental de OpenGL ES, la versión de OpenGL para sistemas empotrados. En cambio, Ogre3D cuenta con esta característica de forma oficial y ofrece versiones "*demo*" que logran muy buenas prestaciones. La desventaja en este caso es la mayor complejidad en la integración y el aprendizaje de Ogre3D con respecto a Irrlicht.

Una vez seleccionadas las implementaciones para construir la plataforma base del agente conversacional 3D, conviene visualizar de nuevo la arquitectura en la Figura 3.8 con las librerías que ofrecen las funcionalidades requeridas.

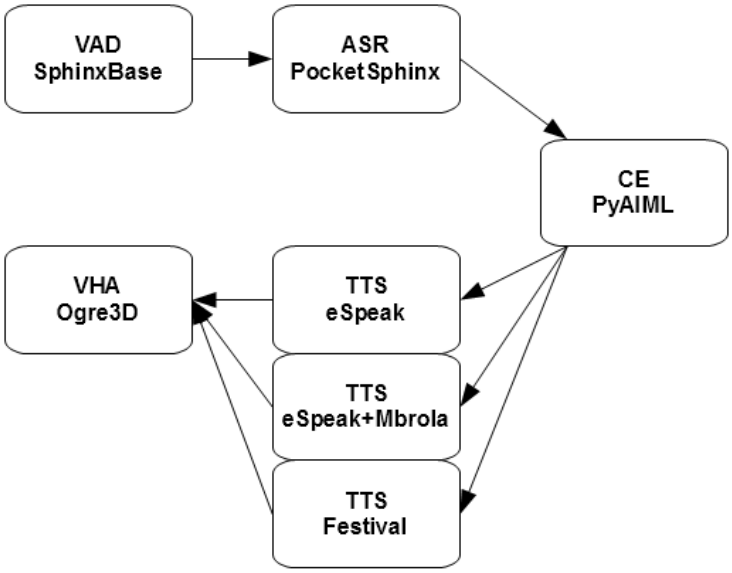


Figura 3.8: Arquitectura de base del agente conversacional 3D.

3.4. Conclusiones

En este Capítulo se ha realizado un análisis cualitativo de los componentes que forman un agente conversacional 3D, dando lugar a una selección de librerías que han servido de base para crear una versión de referencia de la plataforma de desarrollo de ECAs. Esta selección se ha hecho siguiendo en todo momento el criterio principal de orientación a sistemas empotrados móviles. A lo largo de todo el Capítulo se han incluido enlaces bibliográficos de las alternativas de implementación analizadas de cara a ofrecer una guía de referencia a posibles desarrolladores de este tipo de sistemas.

En el Capítulo siguiente se procede a analizar cuantitativamente los componentes de un ECA con el objetivo de determinar los aspectos susceptibles de optimización para, a continuación, ofrecer mejoras que resulten en un incremento del rendimiento.

Capítulo 4

Análisis de la plataforma base

4.1. Introducción

Una vez que se concreta en el Capítulo 3 la arquitectura e implementación de la plataforma base del agente conversacional 3D, llega el momento de analizar su rendimiento cuantitativo. El propósito de este análisis es el de identificar aquellas partes del agente conversacional susceptibles de una mejora u optimización. Las mejoras deben resultar en un incremento de la flexibilidad, sencillez de uso o portabilidad, mientras que las optimizaciones deben llevar a un mayor rendimiento o eficacia del agente conversacional. En este sentido, los resultados del análisis del agente también cumplen la función de medida de referencia para contrastar las futuras optimizaciones.

En este Capítulo se definen tanto el entorno *hardware* de prueba de la plataforma de base como las pruebas de rendimiento para los diferentes casos de uso de interés en el desarrollo de agentes conversacionales 3D para sistemas empotrados.

Aprovechando el carácter modular de la arquitectura del agente conversacional, se ha procedido a analizar sus prestaciones de forma independiente para poder detectar las partes susceptibles de una mayor optimización posterior. Debido a la relación estrecha entre los módulos VAD y ASR del agente, el análisis de estos se ha llevado a cabo de forma conjunta. El módulo VHA no ha sido sujeto a pruebas de rendimiento en este capítulo debido a que el diseño y la integración de la cabeza animada del agente forma parte del

Capítulo 5 al haber sido un trabajo completo e independiente basado en la librería de renderizado Ogre3D. También se definen en este capítulo las métricas de evaluación para estas pruebas de rendimiento. Por último, se analizan los resultados de las pruebas de referencia correspondientes a la integración del agente conversacional.

4.2. Sistema hardware de pruebas - BeagleBoard

El primer sistema empotrado de uso general empleado para implementar la plataforma base de desarrollo de agentes conversacionales es la BeagleBoard. La BeagleBoard es una plataforma de desarrollo de aplicaciones para sistemas empotrados basado en los *chips* de la familia OMAP de Texas Instruments. Su principal ventaja es su coste reducido y que presenta un consumo de potencia bajo. En su revisión B7, la BeagleBoard consta de un procesador OMAP 3530 a 600 MHz, una variante de la arquitectura ARM Cortex A8 con gran difusión y que está presente en muchos dispositivos portátiles y teléfonos móviles, como los Nokia N y E, el Samsung Omnia HD, el Sony Ericsson Vivaz y Satio o el Motorola Droid. Además, la BeagleBoard consta de 128 MBytes de Memoria de Acceso Aleatorio - *Random Access Memory* (RAM) y 256 MBytes de memoria Flash. El SO distribuido con la placa de desarrollo es una distribución Debian Linux 5.0 (también denominada Lenny). La Figura 4.1 muestra la placa de desarrollo.

4.3. Análisis de los módulos VAD+ASR

Hasta hace relativamente poco tiempo los sistemas de reconocimiento de voz no podían trabajar en tiempo real y se empleaban mayormente en modo *offline*. En este tipo de sistemas, la captura, segmentación y preprocesado del audio de entrada se realizan previamente a la tarea de reconocimiento. Se puede decir que los módulos VAD y ASR de este tipo de sistemas funcionan en modo serie, tal como muestra la Figura 4.2. Realizar el preprocesado de la señal de forma independiente al reconocimiento permite optimizar el

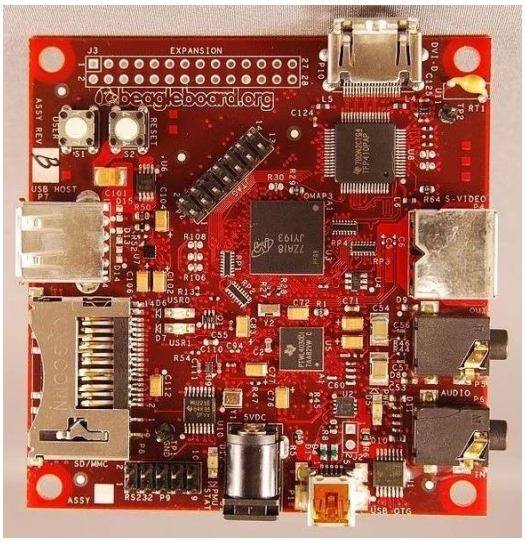


Figura 4.1: Vista cenital de la plataforma hardware de pruebas BeagleBoard.

proceso de extracción de características de la señal acústica y obtener así un rendimiento óptimo. La principal desventaja de este enfoque es que secuencializar completamente el funcionamiento del VAD y del ASR suele resultar en un tiempo de respuesta del sistema prohibitivo para ser integrado en un agente conversacional para sistemas empotrados.

Para comprobar esta hipótesis, se ha realizado un análisis de los módulos VAD y ASR funcionando en modo serie en la plataforma de desarrollo Beagleboard (ver detalles en Sección 4.2).

4.3.1. Métricas de evaluación

En la literatura sobre reconocimiento de voz es habitual encontrar como factor de rendimiento temporal el denominado Factor de Tiempo Real - *Real Time Ratio* (xRT), y se define como en la ecuación 4.1. Este factor compara el tiempo requerido para el reconocimiento de una frase con respecto a la duración de la propia frase. Un valor de xRT igual a 1 indica que se tarda lo mismo en procesar una frase que la duración de la señal de audio. Cuanto menor es el valor de xRT, mejores son las prestaciones.

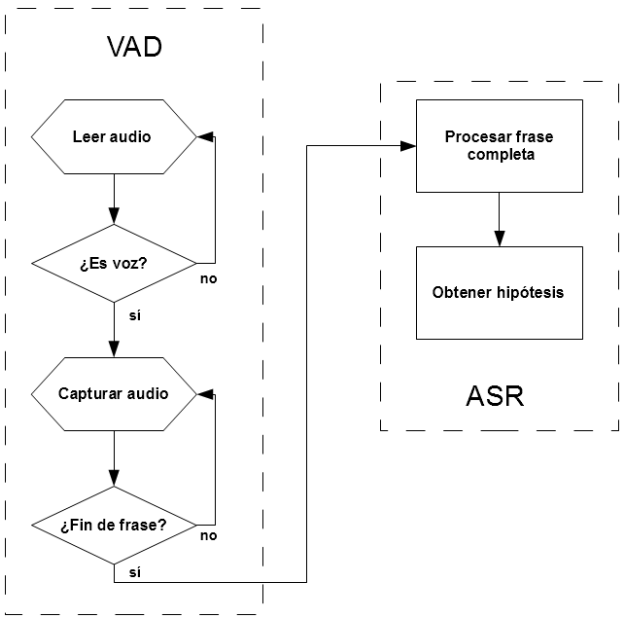


Figura 4.2: Esquema del funcionamiento de los módulos VAD y ASR en serie.

Además de la velocidad de proceso, también es importante tener una medida de la eficacia del reconocedor. El factor de rendimiento que se suele utilizar para este propósito se denomina Tasa de Error por Palabra - *Word Error Rate* (WER). Su definición se muestra en la ecuación 4.2 y mide el porcentaje de palabras fallidas en el reconocimiento. Al igual que en el caso del factor xRT, un valor menor del WER implica un mejor comportamiento del sistema de reconocimiento.

Normalmente la reducción del xRT implica una mayor WER, así que es bastante frecuente fijar una relación de compromiso entre estas dos métricas en función de los requisitos de la aplicación final del sistema.

$$xRT = \frac{Tiempo\ en\ reconocer\ la\ entrada}{Duracion\ de\ la\ entrada} \tag{4.1}$$

$$WER = \frac{Sustituciones + Borrados + Inserciones}{Palabras\ Totales} \tag{4.2}$$

4.3.2. Pruebas y resultados

Las pruebas de los módulos VAD y ASR consistieron en la medida de los valores de xRT y WER obtenidos con la configuración en serie de ambos módulos. Tanto los datos de pruebas como los modelos necesarios para el módulo ASR se crearon a partir del corpus acústico VoxForge [122].

VoxForge ofrece de forma gratuita y bajo licencia GPL audio en distintos idiomas para la creación de modelos acústicos genéricos. Aunque existe una versión gratuita de modelo acústico para español creado por la misma comunidad de VoxForge, éste es un modelo HMM continuo con 1500 senones. Debido a la complejidad computacional que presentan los modelos continuos [120], fue necesario realizar una nueva tarea de entrenamiento del modelo acústico. En este caso, el nuevo modelo es semicontinuo con 1500 senones, 256 gaussianas y 5 estados por HMM [123]. Como modelo de lenguaje se decidió usar el mismo proporcionado para el entrenamiento del modelo acústico, que consta de 473 unigramas, 855 bigramas y 755 trigramas y un diccionario de pronunciación de más de 22000 palabras. Un modelo de lenguaje de esta complejidad puede representar suficientemente la variabilidad de un diálogo con un agente conversacional para realizar una tarea de complejidad media-baja. Para el entrenamiento del modelo acústico se utilizaron más de 3700 frases de usuarios diferentes.

Los datos de entrada para las pruebas corresponden a 420 frases del corpus español de VoxForge separadas previamente de las utilizadas para el entrenamiento de los modelos. Para los módulos VAD y ASR funcionando en modo serie se han hecho 2 conjuntos de pruebas de rendimiento con parámetros diferentes. La primera configuración de parámetros es la que trae por defecto la librería PocketSphinx, que denominaremos *DEF*. El segundo conjunto de valores de los parámetros viene dado por las recomendaciones para sistemas empotrados que hacen los desarrolladores de la librería PocketSphinx, *OPT* [124]. Estos parámetros optimizados responden a estudios previos de los que se puede extraer que la mayor reducción en tiempo de reconocimiento se obtiene al modificar los valores de submuestreo [125] y número de gaussianas [126] que se emplean. Para cada configuración de parámetros se realizaron 10 repeticiones de toma de valores de xRT y WER, con las que se calcularon

las medias aritméticas que se ofrecen como resultados.

Como se muestra en la Tabla 4.1, el módulo ASR basado en la librería PocketSphinx consigue un valor de xRT inferior a 1 para un modelo de lenguaje de complejidad baja-media basado en trigramas. Empleando los parámetros *DEF* se obtiene un valor de xRT de 0.81 y una WER de 5.9 %. La configuración optimizada de parámetros *OPT* hace que el valor de xRT se reduzca a 0.55 y la WER aumente hasta 10.5 %.

xRT *DEF*	**xRT *OPT***	**WER *DEF***	**WER *OPT***
0.81	0.55	5.9 %	10.5 %

Tabla 4.1: Valores de xRT y WER para las configuraciones *DEF* y *OPT* del módulo ASR.

4.4. Análisis del módulo CE

En la Sección 3.2.3.4 se define el uso de un intérprete de lenguaje AIML como elemento que cumple la función de la comprensión y gestión del diálogo con el usuario. El objetivo de este módulo es de servir de "cerebro" del agente conversacional. Por lo tanto, los objetivos a tener en cuenta durante su integración en el conjunto del agente son su tiempo de respuesta, su uso de memoria dinámica y su facilidad de uso.

Como prueba de análisis para tomar una medida de referencia se ha propuesto el uso de la base de conocimiento en español "ALEXIA", perteneciente al Proyecto Galaia [127]. Esta base de archivos AIML pretende reflejar la personalidad y conocimientos de cultura general de un agente conversacional para interacción social. Para ello, consta de más de 8500 "*categories*" AIML. A esta base de conocimiento general se le han añadido otras 260 "*categories*" AIML creadas específicamente para un dominio de aplicación de control de dispositivos inteligentes de una casa domótica.

4.4.1. Métricas de evaluación

En el caso del motor conversacional el principal factor de rendimiento es el Tiempo Medio de Respuesta - *Average Response Time* (ART) definido en la ecuación 4.3. Debido a que estamos restringiendo el uso de agentes conversacionales a sistemas empotrados, también se debe definir como métrica de rendimiento el uso de memoria dinámica del sistema, que denominaremos Consumo Medio de Memoria - *Average Memory Consumption* (AMC).

$$\begin{aligned} ART &= Tiempo\ en\ obtener\ el\ significado\ de\ la\ entrada\ + \\ &+ Tiempo\ en\ procesar\ la\ entrada\ + \\ &+ Tiempo\ en\ generar\ una\ respuesta \end{aligned} \tag{4.3}$$

4.4.2. Pruebas y resultados

Las pruebas para el módulo CE constan sencillamente de la medida de tiempo de respuesta y consumo de memoria. El tiempo de respuesta puede variar significativamente en función de la complejidad de la frase de entrada al módulo CE. Por lo tanto, se ha hecho una primera serie de medidas con una frase sencilla, "Abrir", que se ha nombrado ART *SENC*. Posteriormente, se ha una segunda fase de pruebas con una frase más compleja, "Por favor abrir la puerta uno de la cocina", que se ha denominado ART *COMP*. Para la medida de tiempo de respuesta, se repitió el experimento 10 veces, ofreciendo como resultado su media aritmética. El consumo de memoria siempre es el mismo al no variar la base de conocimiento y por lo tanto solo se refleja un valor. Se usó la herramienta "*top*" incluida en todas las distribuciones comunes GNU/Linux para obtener el valor de AMC.

La Tabla 4.2 muestra los resultados de las pruebas. El valor de tiempo de respuesta ART *SENC* es muy pequeño, tan sólo 18ms, mientras que si la frase de entrada se complica ART *COMP*, el valor aumenta significativamente hasta los 90ms. El consumo de memoria, AMC, se mantiene en un valor discreto de 19MBytes.

ATR *SENC*	ATR *COMP*	AMC
18 ms	90 ms	19 MBytes

Tabla 4.2: Valores de ATR y AMC del módulo CE.

4.5. Análisis del módulo TTS

Como se explica en la Sección 2.7, la complejidad computacional, el consumo de memoria y la calidad de la síntesis varían enormemente en función de la técnica de síntesis que emplea el módulo TTS del agente conversacional.

En este caso se ha comparado el rendimiento que se obtiene con 3 sistemas diferentes de síntesis de texto a voz. El primer sistema es eSpeak que realiza una síntesis basada en formantes y, por lo tanto, será la que demande un menor número de recursos. A continuación se comprobará el rendimiento de eSpeak conjuntamente con MBROLA. Mediante eSpeak se creará la lista de fonemas a construir y MBROLA será el encargado de realizar la síntesis; en este caso basada en difonos. Por último se probará Festival TTS en su versión más simple, usando también síntesis basada en difonos.

4.5.1. Métricas de evaluación

De forma análoga al motor conversacional, el rendimiento de la síntesis de voz se mide a través de su Tiempo Medio de Respuesta (ART) y de su consumo de memoria. También es usual ofrecer valoraciones sobre la calidad de la voz sintética producida, teniendo en cuenta aspectos como la naturalidad (voz fluida o monótona) y la sonoridad (voz robótica o humana).

4.5.2. Pruebas y resultados

De la misma forma que en el motor conversacional, el tiempo de respuesta del módulo TTS depende de la complejidad de la entrada. En este caso, la complejidad viene dada por la longitud del texto de entrada a sintetizar. Entonces, se ha realizado una primera toma de medidas con una frase de 3 palabras de longitud que simula un saludo del agente, "Hola qué tal", que se referencia como *SENC*. La segunda de las pruebas ha tomado como

entrada un texto de 28 palabras que se corresponde con una petición de clarificación por parte del agente, "En la casa solo existen las habitaciones salón y cocina y tú has elegido una incorrecta; te agradecería que elijas una de ellas la próxima vez que hablemos", que se referencia como *COMP*. Cada una de ambas configuraciones de entrada se han probado con los 3 sistemas TTS, por lo que cada medida tendrá como sufijo el nombre del sintetizador usado. Al igual que en las pruebas de los módulos anteriores, los resultados se corresponden con la media aritmética obtenida después de 10 repeticiones de cada experimento.

La Tabla 4.3 muestra que usando eSpeak y MBROLA, se obtienen valores próximos a 200 ms independientemente de la longitud del texto de entrada. En cambio, Festival alcanza los 300 ms para el texto de entrada corto y más de 2.5 segundos cuando tiene que sintetizar un texto largo.

eSpeak *SENC*	**Mbrola *SENC***	**Festival *SENC***
200ms	210ms	310ms
eSpeak *COMP*	**Mbrola *COMP***	**Festival *COMP***
205ms	220ms	2700ms

Tabla 4.3: Valores de ART para las configuraciones DEF y OPT del módulo TTS.

4.6. Discusión de los resultados

Para poder hacer una valoración más práctica de la métrica xRT del módulo ASR, se supone una duración media de una frase del usuario al agente conversacional de 3 segundos de duración. Teniendo esto en cuenta, se obtiene un tiempo de respuesta de 2.43 segundos empleando los parámetros por defecto y 1.65 segundos si se hace uso de los optimizados para sistemas empotrados. Como se ha visto anteriormente, esta reducción del tiempo de respuesta se consigue a costa de un incremento de la tasa de error. Los resultados indican que el módulo ASR puede ser un elemento clave en la sensación de interactividad del agente conversacional y es susceptible de mejora.

El módulo CE obtiene valores inferiores a los 100 ms de respuesta incluso teniendo que procesar comandos complejos del usuario. Por lo tanto, el tiempo de respuesta de este módulo es muy pequeño y no se considera necesario optimizar este factor de rendimiento. En cambio, su uso de memoria de casi 20 MBytes puede ser un problema en sistemas empotrados de gama media-baja, ya que a este valor hay que sumar el del resto de módulos del agente. Por lo tanto, es conveniente ofrecer alguna solución para estos casos.

Los resultados del módulo TTS reflejan que el sintetizador Festival obtiene valores de más de 2.5 segundos para frases largas, lo que lo descarta para su uso en agentes conversacionales que pretendan ofrecer sensación de interactividad. Al contrario, tanto eSpeak como Mbrola se presentan como sistemas válidos para su uso en sistemas empotrados.

4.7. Conclusiones

En este Capítulo se ha efectuado un primer análisis cuantitativo de la versión de referencia de la plataforma de desarrollo de agentes conversacionales. Dicho análisis se ha efectuado en una placa *hardware* de desarrollo de sistemas empotrados. Debido a la heterogeneidad de los módulos internos del agente, se han diferenciado las pruebas y medidas para cada uno de ellos por separado.

Los resultados han servido para identificar los aspectos que más afectan al rendimiento e interactividad del ECA: el tiempo de respuesta del módulo ASR y el consumo de memoria del módulo CE. Para tratar de mejorar el primero, parece necesario profundizar en la estructura y funcionamiento interno de la librería PocketSphinx en la que se basa el módulo ASR. En el caso del módulo CE, las opciones más lógicas para aliviar el consumo de memoria RAM seguirán la idea de reducir la compleja estructura AIML que reside completamente en memoria RAM. En el siguiente Capítulo se describen en detalle todas las propuestas de mejora y optimización.

Capítulo 5

Optimización de la plataforma

5.1. Introducción

Después de identificar en el Capítulo 4 aquellos aspectos críticos de la plataforma base del agente conversacional que son susceptibles de mejora, este Capítulo describe las contribuciones que se han llevado a cabo para lograr un rendimiento más eficiente del agente.

En la Sección 5.2 se exponen las mejoras ideadas para reducir el tiempo de respuesta del módulo ASR del agente conversacional. Este fue identificado como uno de los parámetros a optimizar ya que su elevado valor impide una comunicación interactiva efectiva entre usuario y máquina.

La Sección 5.3 describe las modificaciones cuyo objetivo es reducir el uso de memoria del módulo CE del agente, ya que el tiempo de respuesta de este módulo era ya de por sí reducido en la plataforma base y no se consideró necesaria su optimización.

Por último, la Sección 5.4 trata el diseño de la animación facial con el objeto de lograr una sincronización labial eficiente. Este aspecto es clave para obtener un agente conversacional 3D realista y que no produzca rechazo en el usuario.

5.2. Mejoras de los módulos VAD+ASR

5.2.1. Configuración de VAD+ASR en paralelo con medida de confianza

En un agente conversacional o sistema de diálogo hablado típico, el módulo ASR debe esperar a que el VAD dictamine el final de cada frase del usuario para comenzar el proceso de reconocimiento. Es decir, trata cada frase del usuario como un bloque de muestras de audio completo. En este caso, se podría decir que los módulos ASR y VAD funcionan en modo serie.

Una de las características de la librería de reconocimiento de voz PocketSphinx es que puede procesar segmentos de una frase completa de forma secuencial y avanzar en el la tarea de búsqueda sin necesidad de tener previamente el audio completo de la frase. Esta característica no está exenta de inconvenientes, ya que la extracción de características del audio sin tener la frase completa conlleva errores en la normalización de las muestras y, por lo tanto, una degradación del proceso de reconocimiento, reflejado en el aumento de la tasa de error.

5.2.1.1. Configuración en paralelo de VAD+ASR

El módulo VAD lee las muestras del $buffer$ de captura del micrófono cada pocas decenas de milisegundos. Normalmente se entiende que un sistema ASR trabaja en tiempo real si es capaz de procesar un segmento de voz humana en menos tiempo que la duración original del segmento. Si se cumple esta condición, los módulos VAD y ASR pueden trabajar en paralelo. Así, el módulo ASR puede procesar los $frames$ clasificados como voz por el módulo VAD mientras se continua la captura de la voz del usuario. Al final de la frase el ASR ya ha procesado una gran parte de los $frames$ que forman la frase completa y puede ofrecer su hipótesis final en mucho menos tiempo que la configuración serie. Ambos esquemas de funcionamiento se muestran en la Figura 5.1.

Como se ha comentado anteriormente, el funcionamiento paralelo de los módulos VAD y ASR mejora el rendimiento en términos de velocidad de reconocimiento pero puede causar una pérdida en la tasa de acierto. Debido

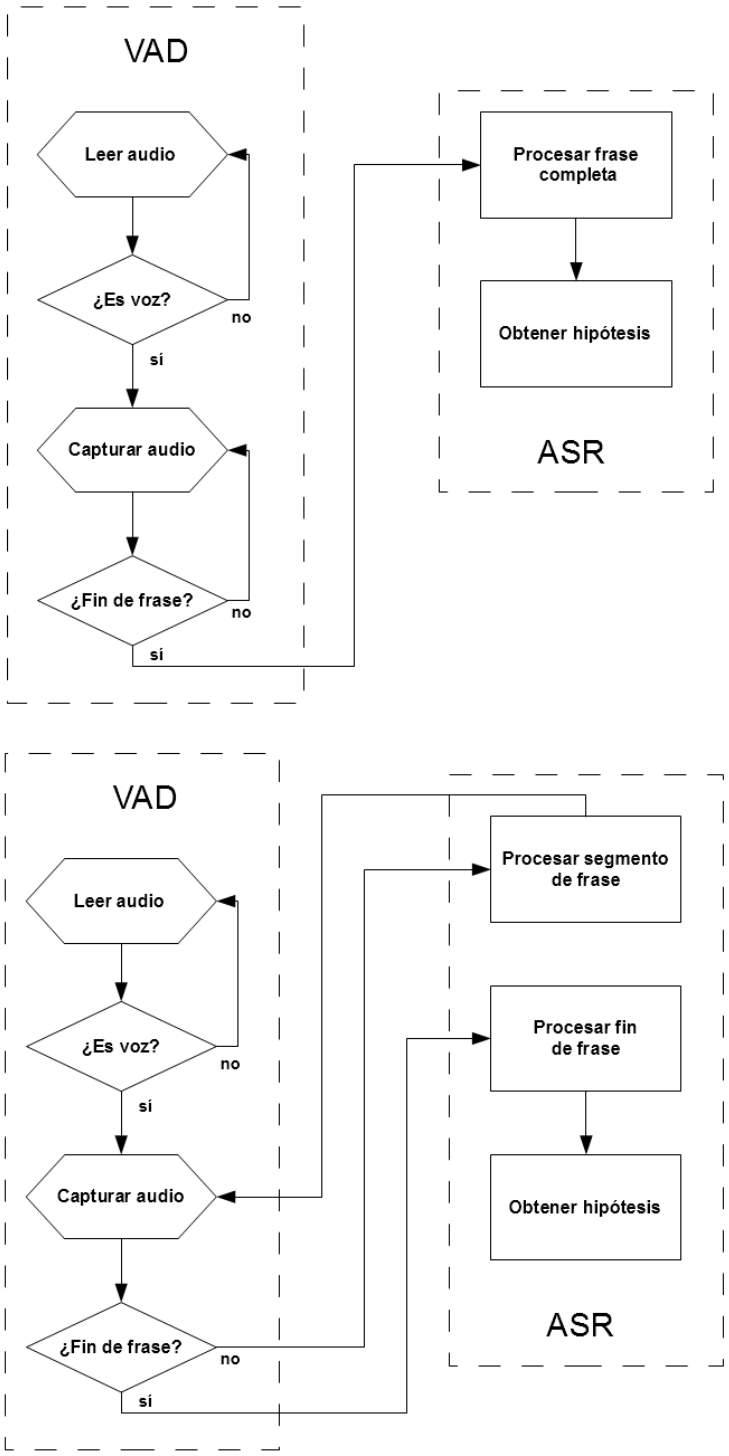


Figura 5.1: Esquema del funcionamiento de los módulos VAD y ASR en serie y paralelo.

a este hecho, se propone a continuación un enfoque híbrido que pretende maximizar los aspectos positivos de ambos esquemas.

5.2.1.2. Configuración híbrida de VAD+ASR con medida de confianza

A continuación se describe un modo de funcionamiento de los módulos VAD y ASR que permita aprovechar al máximo las ventajas de los modos serie y paralelo. Este nuevo esquema se ha denominado modo híbrido.

Cuando el módulo ASR asigna una hipótesis a una frase del usuario también ofrece como resultado una medida de confianza [128], que indica la certeza del módulo ASR de que la hipótesis final represente lo dicho por el usuario. Por lo tanto, es posible tomar ventaja de ello estableciendo un umbral para esta medida de confianza, que determine si se usa el modo serie o paralelo.

Por defecto, el funcionamiento de los módulos VAD y ASR es en modo paralelo para obtener el menor tiempo de respuesta posible. Solo en el caso de que la medida de confianza de la mejor hipótesis obtenida por el módulo ASR esté por debajo del umbral, se fuerza al módulo ASR a realizar una segunda fase de reconocimiento con la frase completa en modo serie. Este modo de funcionamiento se muestra en la Figura 5.2.

Con la idea propuesta antes, se puede establecer un modelo teórico para el tiempo de respuesta del reconocedor en el esquema híbrido:

$$xRT_h(th) \;=\; SAR(th)\; xRT_s \;+\; xRT_p \tag{5.1}$$

donde xRT_h, xRT_s y xRT_p son los tiempos medios de respuesta del módulo ASR en los esquemas híbrido, serie y paralelo respectivamente. Se nota con th al umbral de confianza variable y SAR a la Tasa de Segundo Intento -*Second Attempt Rate*. Esta última representa el porcentaje de las frases totales que obtienen una medida de confianza por debajo del umbral y con las que el módulo ASR debe hacer un segundo intento de reconocimiento.

En el caso de la tasa de error, la ecuación queda como sigue:

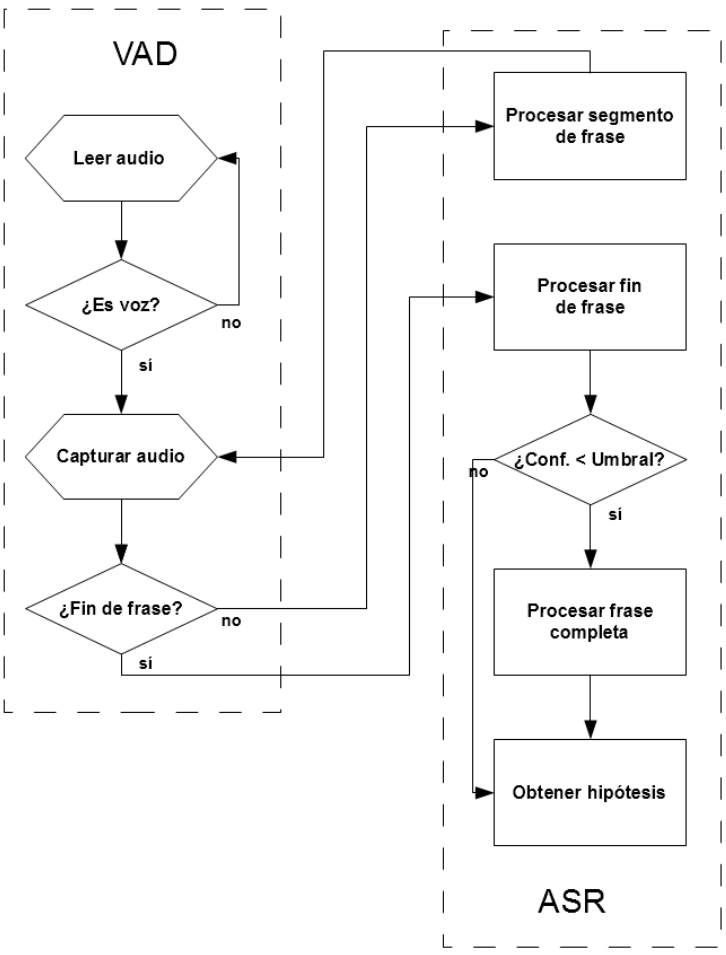


Figura 5.2: Esquema del funcionamiento de los módulos VAD y ASR en modo híbrido.

$$WER_h(th) \;=\; SAR(th)\; WER_s[cs < th] \;+\; (1 - SAR(th))\; WER_p[cs \geq th] \tag{5.2}$$

donde WER_h es la WER media del esquema híbrido y SAR tiene un significado análogo al de la ecuación 5.1. Ahora $WER_s[cs < th]$ y $WER_p[cs \geq th]$ son también la WER media en los modos serie y paralelo, pero la primera se mide con respecto al conjunto de frases que obtienen una confianza inferior

al umbral y el segundo con respecto a las que la obtienen igual o superior.

El umbral de confianza adecuado puede variar dependiendo del dominio de aplicación del agente conversacional. Por ejemplo, si se usa para el control domótico de una vivienda, el umbral debería ser alto para reducir la WER a expensas de un mayor xRT. En este tipo de entornos las frases se reducen a unos pocos comandos, por lo que suelen ser simples y de corta duración. Por lo tanto, el incremento en tiempo de respuesta puede ser aceptable mientras que se mantiene la WER en niveles bajos que permitan maximizar la comprensión de los comandos del usuario. En otro tipo de escenario donde el agente conversacional se emplee para interacción social con el usuario, el umbral puede fijarse a niveles más bajos. De esta forma, se reduce el tiempo de respuesta y se incrementa la interactividad y la participación del usuario. En este escenario las frases del usuario pueden ser sensiblemente más largas y complejas que en el ejemplo anterior por lo que se puede permitir una mayor WER para favorecer la fluidez del diálogo.

5.2.1.3. Pruebas y resultados

Para probar esta mejora respecto al sistema base, se empleó el mismo escenario que en el análisis efectuado en la Sección 4.3. Los modelos acústico y de lenguaje se obtuvieron mediante el entrenamiento de más de 3700 frases en español de usuarios diferentes. Los datos de entrada los forman 420 frases del mismo corpus previamente separadas de las empleadas en el entrenamiento de los modelos. Los resultados que se ofrecen se corresponden a los valores medios de xRT y WER medidos en 10 repeticiones del experimento para cada configuración diferente.

En las Figuras 5.3 y 5.4, el valor S de *THRESHOLD %* corresponde al modo serie, y el valor P al modo paralelo. Los valores intermedios *90-10* corresponden al modo híbrido.

El primer hecho que se puede ver en las Figuras 5.3 y 5.4 es que los valores empíricos de WER_e y xRT_e se ajustan casi perfectamente a sus respectivos valores teóricos WER_t y xRT_t.

Como se puede ver el la Figura 5.3, el modo híbrido alcanza valores significantemente inferiores de xRT que el modo serie. Incluso con un umbral

de confianza del 90 %, el valor de xRT se reduce de 0.81 a 0.54 (un 33 %). Esta reducción continua de forma progresiva hasta un valor de xRT de 0.29 (un 64 %) cuando el umbral se fija al 10 %. Esto es debido a que el SAR pasa del 35 % al 4.3 %. Como era de esperar, el WER aumenta desde el 5.9 % en el modo serie hasta el 9.2 % del modo paralelo.

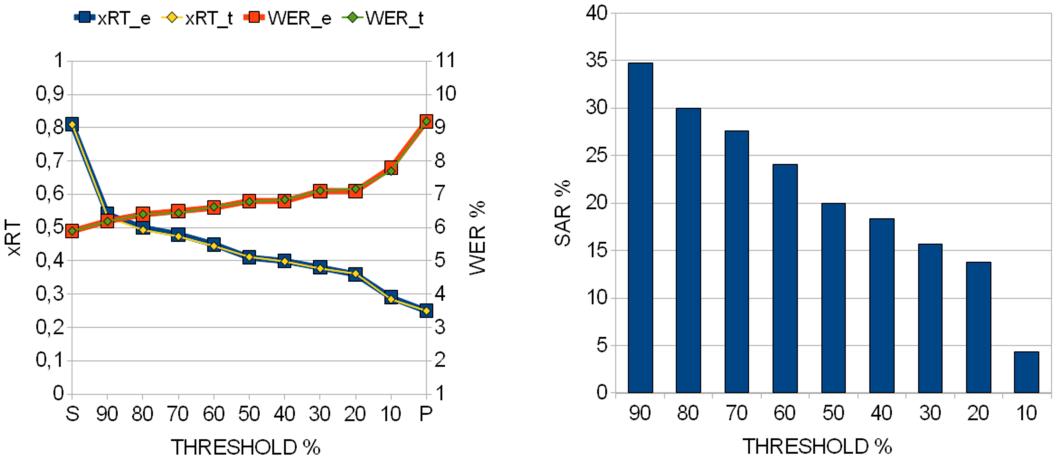
Optimizando los parámetros de configuración del módulo ASR, al igual que en la Sección 4.3, se obtiene como se puede ver en la Figura 5.4 un patrón de valores similares a los mostrados en la Figura 5.3, pero con valores de xRT inferiores al caso anterior y valores más elevados de WER. Los valores de xRT van del 0.55 al 0.33 con un umbral de confianza del 90 %, lo que implica una reducción del 40 % con respecto al modo serie. Todavía en el modo híbrido, se alcanza un valor de xRT de 0.15 con el umbral fijado al 10 %, esto es una reducción del 73 %. La tasa SAR mantiene unos valores similares al caso anterior y el WER toma valores desde el 7.2 % del modo serie hasta el 10.5 % del modo paralelo.

Comparando las configuraciones entre sí, se puede extraer que el modo serie con los parámetros optimizados obtiene un 32 % menos de xRT que con los valores por defecto. En cambio, la WER se ve incrementada un 22 %. En el caso del modo paralelo con la configuración optimizada, se obtiene una reducción del xRT del 58 % comparada con la por defecto y la WER aumenta un 14 %.

5.2.2. Selección automática del modelo de lenguaje

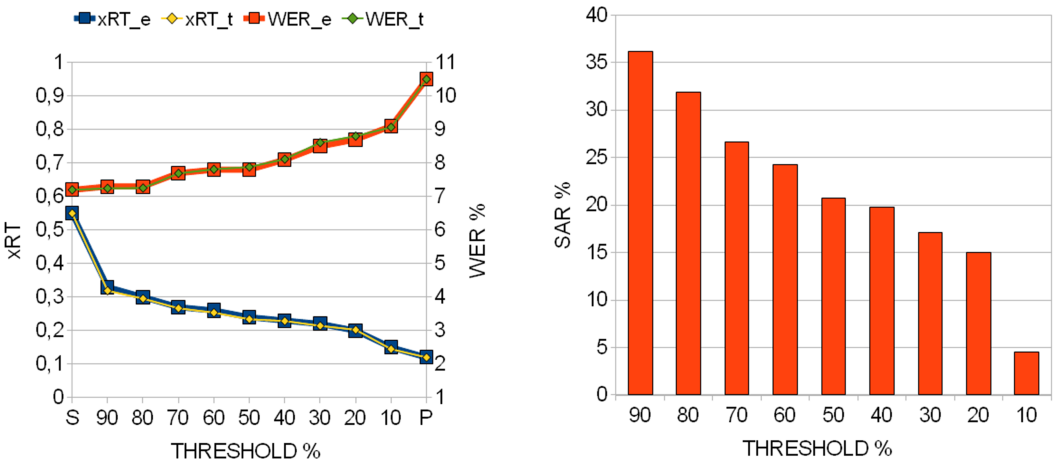
Mientras que varios de los procesos internos que conlleva la tarea de reconocimiento de voz se han optimizado específicamente para su ejecución en sistemas empotrados [96], la complejidad del modelo de lenguaje continúa siendo uno de los factores que más afecta tanto a la tasa de error como al tiempo de respuesta de un sistema ASR.

Una posible aproximación para limitar la complejidad del modelo de lenguaje consiste en seleccionar en tiempo de ejecución un modelo específico del tópico/tema de la conversación que el usuario mantiene con el agente conversacional. Este tipo de técnicas se denominan Selección del Modelo de Lenguaje - *Language Model Switching* (LMS). Normalmente, la LMS implica



a) xRT and WER results b) SAR results

Figura 5.3: Valores de xRT, WER and SAR con parámetros por defecto.



a) xRT and WER results b) SAR results

Figura 5.4: Valores de xRT, WER and SAR con parámetros optimizados.

el uso de varios sistemas ASR en varias etapas. Otros autores han empleado esta técnica para reducir la WER, mediante configuraciones en paralelo o en serie de varios sistemas ASR [129], como se muestra en la Figura 5.5. En la configuración serie, se emplea un Modelo de Lenguaje General - *General Language Model* (G-LM) para obtener una primera hipótesis. Con este primer resultado se identifica el tópico y se selecciona un Modelo de Lenguaje

Dependiente del Tópico - *Topic-Dependent Language Model* (TD-LM). Este modelo se emplea en una nueva fase completa de reconocimiento. En cambio, la configuración en paralelo presenta la voz del usuario a varios sistemas ASR, donde cada uno cuenta con un TD-LM diferente. Finalmente, un algoritmo externo debe seleccionar la mejor hipótesis de todas. La principal desventaja de este tipo de configuraciones es que, aunque consiguen reducir la WER, la complejidad computacional se incrementa, por lo que aumenta el valor de xRT. Otras alternativas de LMS se basan la interpolación de diferentes modelos de lenguaje [130]. El uso de un modelo optimizado mediante interpolación en una segunda etapa de reconocimiento también implica un aumento de xRT. La otra opción es realizar la interpolación de varios modelos de lenguaje de forma dinámica en la primera etapa ASR, pero la complejidad añadida hace que se obtengan valores de tiempo de respuesta muy elevados (8 xRT según [130]).

Partiendo de las ideas anteriores de técnicas LMS, pero con el objetivo de lograr reducir tanto la WER como el xRT, se ha propuesto integrar la LMS en el interior del reconocedor PocketSphinx, aprovechando su estructura interna multietapa que se describe a continuación.

El reconocedor PocketSphinx se basa en una versión anterior denominada Sphinx 2 desarrollada también por la CMU. Los detalles específicos de esta librería se describen en [96] y [131]. De forma breve, y tal como se puede ver en la Figura 5.6, su proceso de reconocimiento se resume en 3 fases o etapas:

1. Búsqueda "*Forward Tree Viterbi beam*": Los resultados son una primera hipótesis y una *lattice* [1] que contiene todas las palabras reconocidas durante esta búsqueda inicial.

2. Búsqueda "*Forward Flat Viterbi beam*": Esta búsqueda se restringe a las palabras identificadas en el paso anterior. Al igual que la primera etapa, los resultados son una hipótesis y una nueva *lattice* de palabras.

3. Búsqueda "*N-best*": Emplea la *lattice* resultante del paso anterior y ofrece como resultado una lista de las mejores N hipótesis.

[1] Una *lattice* de palabras es un grafo acíclico dirigido con un nodo raíz y aristas etiquetadas con una palabra y un peso o puntuación.

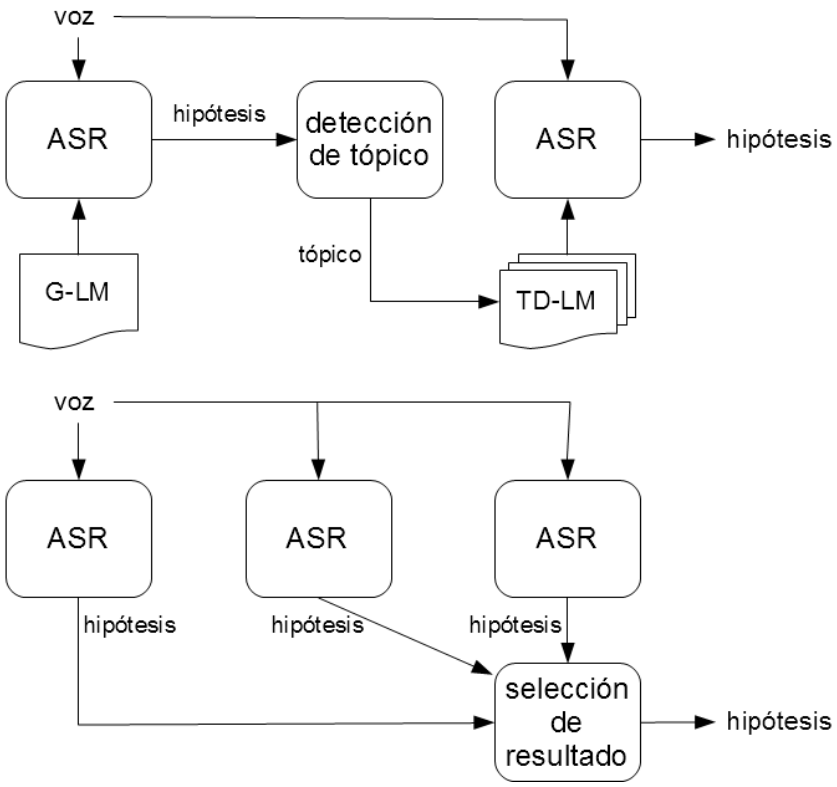


Figura 5.5: Esquemas LMS multi-ASR en serie (arriba) y paralelo (abajo).

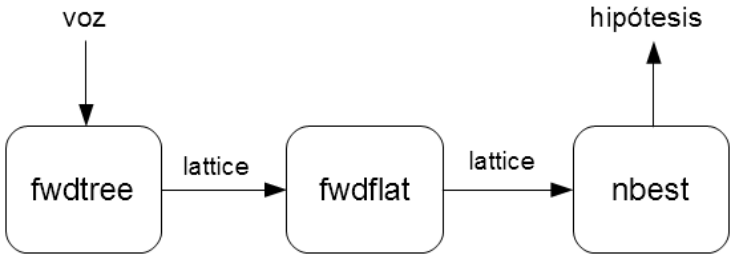


Figura 5.6: Etapas internas del reconocedor PocketSphinx.

PocketSphinx emplea aproximadamente el 70-80 % del tiempo en la primera etapa y solamente el 20-30 % restante en las 2 etapas posteriores. Esta librería, tal como se ha visto en la sección anterior, permite paralelizar la

primera etapa con la captura y la detección vocal de forma que el tiempo de respuesta global se ve reducido enormemente. También se ha explicado que esta reducción del tiempo de respuesta implica una estimación de los valores de normalización cepstral y, por lo tanto, conlleva una ligera pérdida de WER.

La idea principal de esta mejora es realizar una técnica de LMS dentro de la estructura de 3 etapas de PocketSphinx, tal como se muestra en la Figura 5.7. Durante la primera fase se emplea un G-LM para cubrir todos los posibles dominios de aplicación del agente conversacional. Al finalizar esta primera etapa, se realiza una detección del dominio de aplicación y se selecciona un TD-LM, que será empleado en las últimas 2 etapas de reconocimiento.

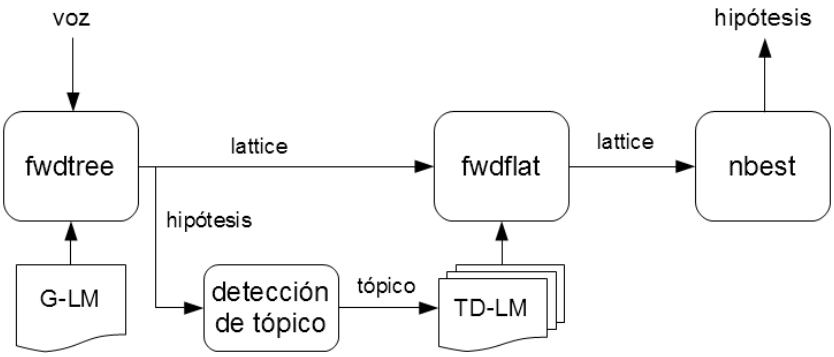


Figura 5.7: LMS integrado en el reconocedor PocketSphinx.

Para esta optimización del agente conversacional se ha elegido como método para la detección del tópico de la conversación las Máquinas de Vectores de Soporte - *Support Vector Machines* (SVMs) [132]. Este tipo de método de clasificación se ha aplicado con éxito en problemas de diferente naturaleza y mejora los resultados logrados por otros métodos bien establecidos en el ámbito académico como las Redes Neuronales Artificiales y las Funciones de Base Radial.

5.2.2.1. Máquinas de Vectores de Soporte

Las SVM son clasificadores binarios que tratan de maximizar el margen de separación entre los datos pertenecientes a clases diferentes. Los problemas

de clasificación de texto presentan un número de dimensiones muy elevado y es común que en este caso los datos no sean separables de forma lineal en el espacio de representación de los datos. Para solventar esta dificultad, las SVM construyen su región de separación mediante una transformación de los datos hacia un espacio de características con un número de dimensiones mayor al espacio de entrada. El objetivo de esta transformación es conseguir que los datos sean separables fácilmente mediante un hiperplano en este nuevo espacio de representación.

La selección del hiperplano de la SVM requiere evaluar numersos productos escalares en el espacio de características. Aunque este cálculo es computacionalmente excesivo, se puede evitar gracias a una representación funcional llamada *kernel*. El *kernel* calcula el producto escalar en el espacio de características como una operación directa de los datos en el espacio de entrada. Empleando un *kernel* efectivo, esta operación se puede realizar sin un incremento significativo del coste computacional.

Si el problema de clasificación tiene más de 2 clases o categorías, se requiere el uso de técnicas de *one-vs-one* o *one-vs-all* para convertir el problema clasificación en varios procesos de clasificación binaria.

5.2.2.2. Pruebas y resultados

Para esta mejora se emplearon 3 conjuntos de frases para las pruebas. Cada conjunto se compone de frases que pertenecen a un dominio de aplicación diferente.

- *CMU Weather limited domain* [133] (conjunto *weather*): Este corpus consiste en 100 frases que cubren aspectos de datos y fenómenos meteorológicos como pueden ser la fecha, el estado del tiempo, pronósticos, la temperatura y la dirección y fuerza del viento.

- *CMU Communicator limited domain* [133] (conjunto *travel*): Esta base de datos consiste en 530 frases que se emplean en el sistema de diálogo CMU Communicator [134], que permite entablar conversaciones telefónicas para acceder a la información de reservas de vuelos.

- *DARPA Resource Management Continuous Speech Corpora* [135] (conjunto *naval*): Este corpus consiste en sentencias que modelan las tareas de gestión de recursos navales y consta de 600 frases distintas.

Debido al idioma inglés de las sentencias que van a ser usadas por el módulo ASR, se procedió al entrenamiento del modelo acústico basado en el corpus en inglés de VoxForge. Existe una versión de modelo acústico HMM continuo con 3000 senones, pero debido a la complejidad computacional que implican los modelos continuos, se decidió entrenar un nuevo modelo. Finalmente se creó un modelo acústico semicontinuo con 1000 senones, 256 gausianas y 5 estados por HMM.

De cada uno de los 3 conjuntos de frases de entrada, *weather*, *travel* y *naval*, se generó un modelo TD-LM basado en trigramas empleando la técnica de suavizado de Kneser-Kney modificado. Para ello, se empleó el conjunto de herramientas que provee el *software MIT Language Modeling Toolkit* [136] [137]. Además, se entrenó un modelo G-LM combinando frases de los 3 conjuntos (3*topics*). La perplejidad de todos los modelos de lenguaje se muestran en la Tabla 5.1.

G-LM *3topics*	**TD-LM *weather***	**TD-LM *travel***	**TD-LM *naval***
4.307	2.699	5.326	3.717

Tabla 5.1: Perplejidad de los modelos G-LM y TD-LM.

Se usaron para las pruebas los parámetros de configuración del módulo ASR que vienen por defecto en los *scripts* de entrenamiento del modelo acústico. La Tabla 5.2 muestra los valores de los diferentes parámetros usados en las 3 etapas del reconocedor.

Para la detección del tópico de la conversación, se representó cada frase (hipótesis) S_i resultado de la primera fase de reconocimiento como un punto de un espacio n-dimensional $(O(w_1), O(w_2), ..., O(w_n))$, donde $O(w_k)$ es el número de ocurrencias de la palabra w_k en S_i y n es el número de palabras totales del corpus resultante de la unión de los 3 conjuntos de frases. Para la clasificación mediante SVM se integró la librería LibSVM [138] [139] en la

lw	**beam**	**wbeam**	**lpbeam**	**lponlybeam**
10	1e-80	1e-40	1e-40	7e-29
fwdlw	**fwdbeam**	**fwdwbeam**	**bestpathlw**	
8.5	1e-64	7e-29	9.5	

Tabla 5.2: Parámetros del módulo ASR.

propia PocketSphinx, de la que se recogieron los porcentajes de acierto como un resultado más de este proceso de pruebas.

En estas pruebas, se compara el rendimiento del reconocedor de base, LMS_no, y el reconocedor que incorpora la propuesta de detección de tópico y selección del modelo de lenguaje entre las etapas 1 y 2, LMS_yes. Para ambos casos se han medido las WER y xRT medias (ecuaciones 4.1 y 4.2) para las frases de entrada de cada dominio, *weather*, *travel*, *naval* y la union de todos ellos. Hay que tener en cuenta que en las medidas de xRT realizadas se ha descontado previamente el tiempo empleado en la primera fase de reconocimiento. Esto es así debido a que se emplea el modo de funcionamiento en paralelo de los módulos VAD y ASR y, por lo tanto, la primera etapa del reconocimiento se realiza mientras el usuario todavía está hablando. Así, el valor que se ha recogido de xRT corresponde al tiempo empleado en el proceso de reconocimiento una vez que el usuario termina de dictar la frase de entrada.

Los valores que se ofrecen como resultados se corresponden a las medias aritméticas de xRT y WER en 10 repeticiones diferentes de cada uno de los experimentos.

Como se puede ver en la Figura 5.8, el reconocedor LMS_yes obtiene una reducción de la tasa de error, WER_dif, del 15.23 % para el conjunto *weather*, un 9.81 % en el caso del conjunto *travel* y un 1.63 % para el conjunto *naval*. Estos resultados combinados conllevan una reducción de WER para la unión de los 3 conjuntos, 3*topics*, de un 3.93 %.

La Figura 5.9 muestra los resultados de xRT para tanto el reconocedor de base como el propuesto como mejora. En el segundo caso, LMS_yes, se mejora el valor de xRT con respecto al LMS_no en un 17.51 % para el conjunto *weather*, un 2.37 % para *travel*, y un 9.76 % para *naval*. Procediendo igual

que antes, el resultado combinado para 3*topics* es una reducción de xRT del 8.77 %.

En la detección del tópico de la conversación, se constata la robustez del clasificador SVM al lograr una tasa de acierto ligeramente superior al 90 %.

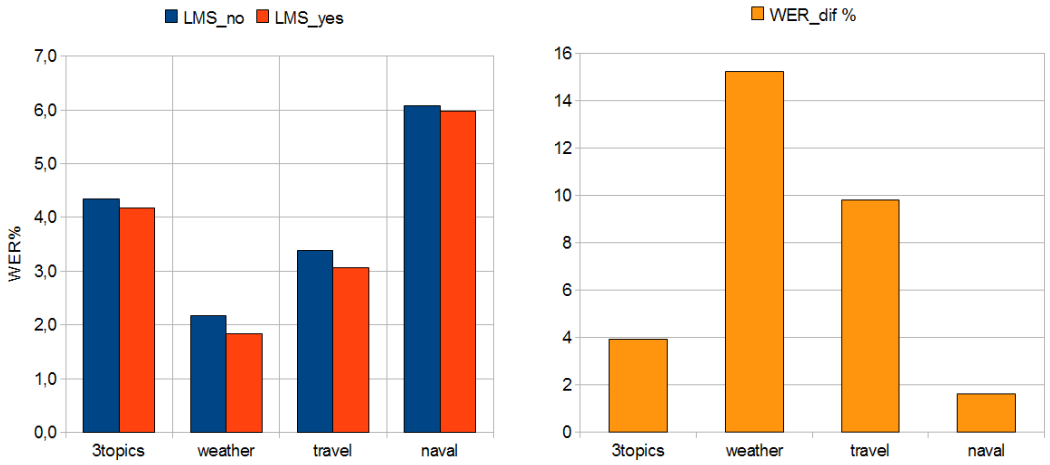


Figura 5.8: Resultados de WER para cada corpus.

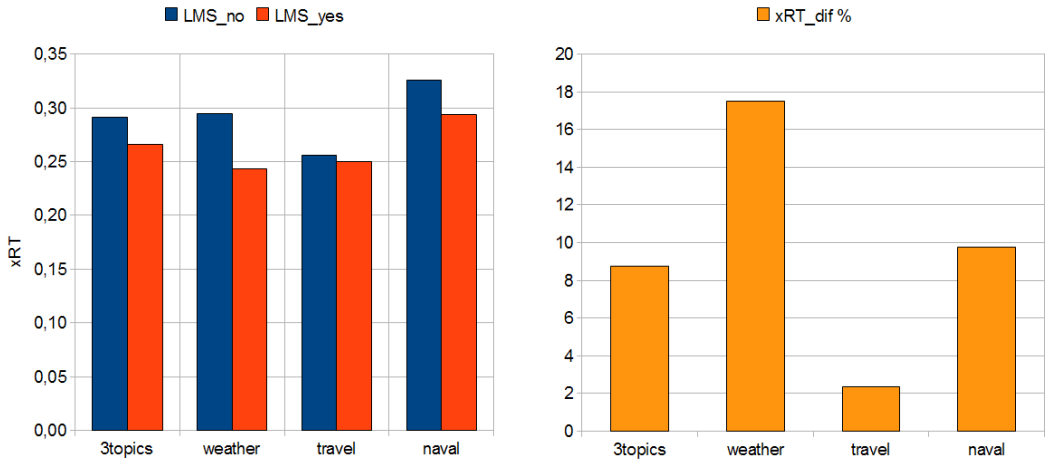


Figura 5.9: Valores de xRT para cada corpus.

5.3. Mejoras del módulo CE

5.3.1. Incorporación de lematizador y base de datos orientada a objetos

Como se comenta en el Capítulo 3, el motor conversacional de la plataforma de desarrollo de agentes conversacionales se basa en el intérprete en Python del lenguaje AIML, PyAIML. Este intérprete AIML es de código abierto y es completamente compatible con las especificaciones AIML 1.0.1. Internamente, es necesario recordar brevemente el funcionamiento de sus elementos internos. El analizador AIML procesa los archivos AIML incorporando todas las nuevas "*categories*" al "cerebro" del *bot*. El submódulo de sustituciones reemplaza palabras que pueden dificultar el reconocimiento de "*patterns*". El elemento más importante es el buscador, que implementa el algoritmo de búsqueda que permite al *bot* extraer la información de la entrada del usuario y construir la respuesta adecuada. Finalmente, el núcleo interconecta todos los componentes descritos además de cumplir el rol de interfaz público del motor conversacional. También incorpora funcionalidades de configuración generales y funciones auxiliares para, por ejemplo, permitir la persistencia del estado del *bot* entre diferentes ejecuciones del *software*.

En esta mejora se han añadido 2 elementos opcionales el intérprete AIML original para facilitar la comunicación en lenguaje natural y para reducir el uso de recursos en sistemas empotrados. Estos elementos se muestran en la Figura 5.10, y son un lematizador y una base de datos orientada a objetos.

5.3.1.1. Lematizador

Mediante el uso del lematizador, el agente conversacional puede afrontar un uso más natural del lenguaje sin tener que contemplar todas las posibles variaciones de forma que puede tener una misma frase para representar un mismo significado.

Como se explica en el Capítulo 2, el conocimiento del *bot* se almacena en estructuras de datos denominadas "*category*", que consisten en al menos 2 elementos, el "*pattern*" y el "*template*". Simplificando enormemente el algoritmo, se puede decir que el intérprete AIML intenta encontrar el "*pattern*"

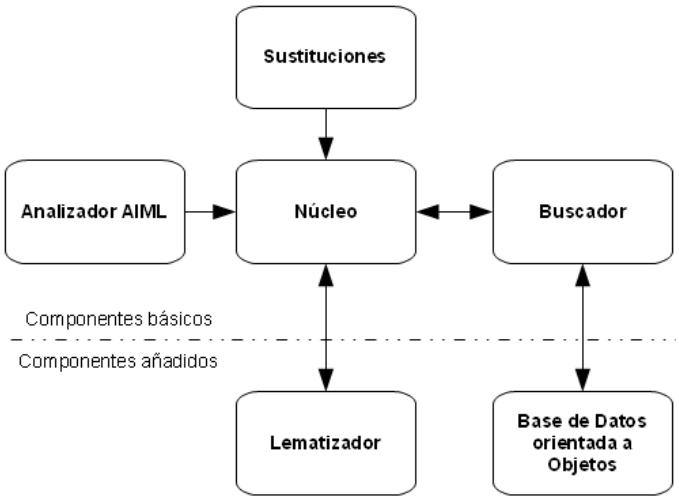


Figura 5.10: Arquitectura del motor conversacional.

que tenga una mayor correspondencia con el texto de entrada del usuario. La respuesta que se provee al usuario se construye con los contenidos del "*template*" de la "*category*" seleccionada. Este tipo de algoritmos basados en emparejamiento de patrones de palabras suele ser apropiado para idiomas poco inflexivos como el Inglés donde, por ejemplo, la forma verbal de una frase apenas sufre variaciones. En cambio, en el caso de un idioma altamente inflexivo como el Castellano, sería necesario reescribir "*patterns*" repetitivos para todas las posibles formas verbales que expresan un mismo significado.

La función primordial de un lematizador es obtener la forma canónica o lema de una frase. Mediante el proceso de reducción de una frase a su forma canónica se pierde cierta información, pero es mucho más sencillo encontrar un "*pattern*" coincidente que guarde relación con su significado original. Los lematizadores se emplean con frecuencia en sistemas de recuperación de información para mejorar la tasa de éxito del sistema de referencia [140] y, especialmente, con idiomas inflexivos [141]. La Tabla 5.3 muestra algunos ejemplos que ilustran la simplificación que se puede obtener gracias al uso del lematizador.

El lematizador empleado en el agente conversacional de esta tesis se basa en FreeLing [142] [143]. Esta es una librería gratuita y de código abierto para análisis lingüístico y soporta múltiples idiomas (Castellano, Italiano, Inglés,

Frase original	Forma canónica	Patrón
Abre la puerta	Abrir el puerta	Abrir puerta
¿Podrías abrir la puerta?	Poder abrir el puerta	Abrir puerta
Quiero que abras la puerta	Querer abrir el puerta	Abrir puerta

Tabla 5.3: Ejemplo de uso del lematizador.

Portugués, etc.).

Como se ha comentado con anterioridad, el conocimiento del agente conversacional se almacena en archivos AIML. Por lo tanto, simplificando los "*patterns*", se reduce el número de "*categories*" necesarias. Así, se logran una menor complejidad y tiempo de creación de dichos archivos AIML y un menor uso de memoria en tiempo de ejecución.

Con el fin de no renunciar a la potencia original del intérprete AIML para reconocer cualquier sentencia en lenguaje natural, el lematizador se usa solo si el intérprete AIML falla en la comprensión de la frase de entrada original. Además, se este proceso se ha implementado para que se pueda aplicar tanto a frases enteras como a partes de ellas. De esta forma, la habilidad de comprensión del motor conversacional se ve maximizada.

Para ilustrar las ventajas del uso del lematizador, se muestra un ejemplo en la Figura 5.11. En este caso el motor conversacional solo tiene 2 "*categories*" almacenadas en su base de conocimiento: una para reconocer la acción "abrir" y otra para reconocer el elemento objeto de dicha acción, "la puerta". Así, la única meta global que puede interpretar el motor conversacional es la de abrir la puerta. Tomando como suposición que la frase de entrada del usuario sea "Abre la puerta", en una primera etapa el intérprete detecta el segmento "la puerta" en la sentencia de entrada y considera que ha identificado el objeto de la acción. Ya que no puede haber un objeto que recibe una acción sin acción seleccionada, el intérprete genera una señal de error que activa el uso del lematizador. A partir de la sentencia original, el lematizador solamente procesa aquellas partes que no activaron ninguna "*category*". Tras el uso del lematizador en las partes de la frase no identificadas, ésta queda como "Abrir la puerta". Con esta nueva sentencia de entrada, el intérprete

AIML puede identificar la acción como "abrir". Ahora se han completado los pasos que definen la meta global y el motor conversacional puede comunicarse con otros módulos de control externos para hacer efectiva la acción de abrir la puerta y generar una respuesta satisfactoria para el usuario, cerrando el bucle de ejecución.

Debido al uso del lematizador como segunda oportunidad de comprensión, su uso se ofrece como una funcionalidad extra opcional del motor conversacional. Por lo tanto, los investigadores y desarrolladores pueden decidir cuando creen conveniente su uso.

5.3.1.2. Base de Datos orientada a Objetos

La base de conocimiento del motor conversacional se almacena en una estructura jerárquica de "*categories*". La implementación original se basa en tablas *hash*. Igual que los *arrays*, las tablas *hash* ofrecen un tiempo de acceso medio de complejidad *O(1)*, independientemente del número de elementos de la tabla. Teniendo en cuenta esta característica, el tiempo de acceso del motor conversacional a su "cerebro" es prácticamente instantáneo y no sufre apenas pérdida de rendimiento al incrementar el número o la complejidad del dominio de aplicación del agente conversacional. La única desventaja que presenta esta técnica es la necesidad de almacenar en memoria la estructura completa de tablas *hash*. Para un ordenador personal moderno con varios Gigabytes de RAM esto no representa ningún problema. Pero para dispositivos móviles y sistemas empotrados, que suelen contar con 512-256 Megabytes en la gama media-baja, puede resultar más conveniente almacenar estas estructuras complejas en disco.

Existen diferentes enfoques para manejar grandes cantidades de datos persistentes. Quizá la más común es el uso de Bases de Datos Relacionales - *Relational DataBases* (RDB). Las RDBs cuentan con un modelo sencillo de organización de datos en tablas y pueden almacenar grandes cantidades de datos de forma eficiente. Este modelo sencillo basado en tablas suele ser fácil de comprender para problemas básicos, pero las RDBs se vuelven demasiado complejas cuando el dominio del problema no se ajusta a una simple organización tabular. En cambio, las Bases de Datos Orientadas a Objetos

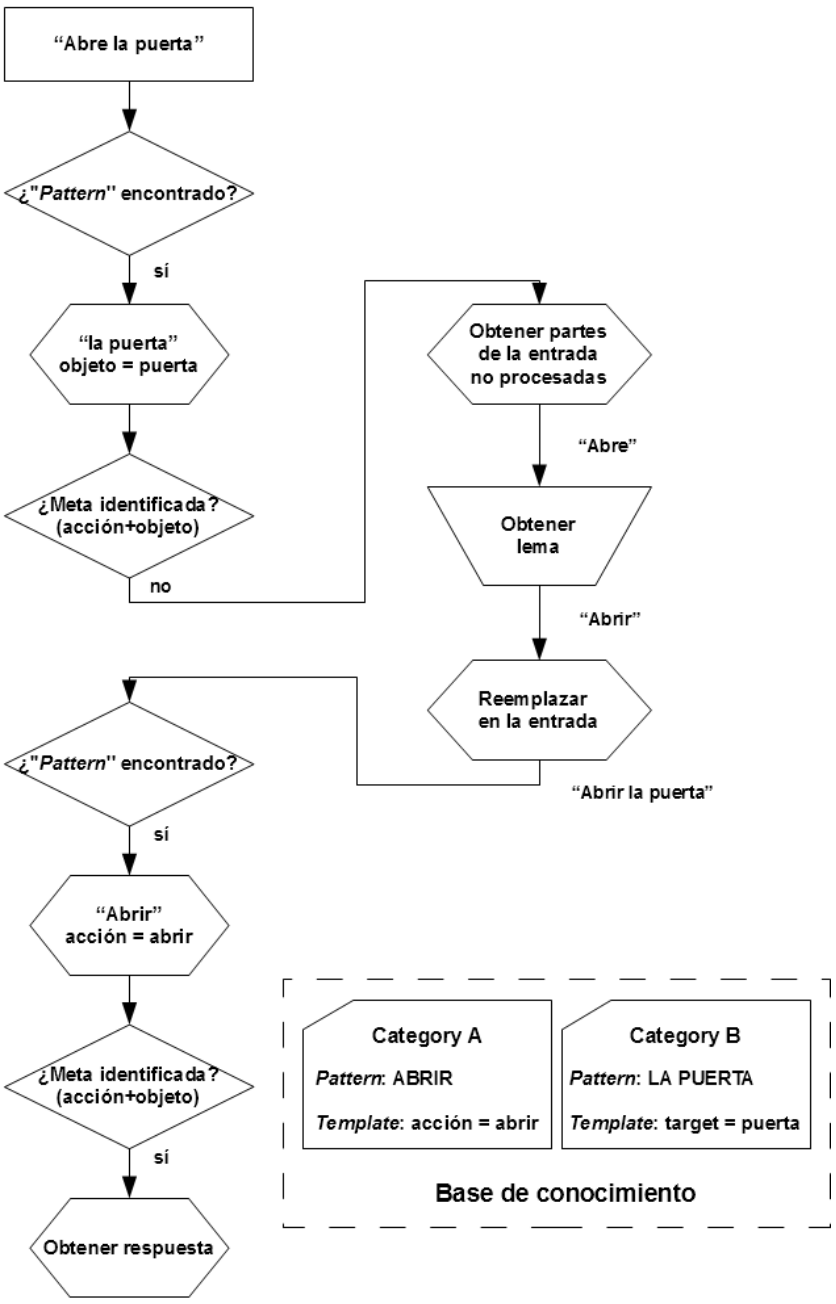


Figura 5.11: Diagrama de flujo del lematizador dentro del intérprete AIML.

- *Object-oriented DataBases* (ODB) suelen resultar en una integración más evidente y sencilla para estructuras y objetos. En este caso los datos no se guardan en tablas sino en formas que reflejen la organización de la informa-

ción en el dominio del problema. Además los desarrolladores de aplicaciones se ven liberados de la necesidad de escribir la lógica de extracción y almacenamiento en disco.

La base de datos elegida para mejorar el almacenamiento de la base de conocimiento del agente conversacional es *Zope Object-oriented DataBase* (ZODB), una ODB escrita en Python [144] [145].

Aunque es perfectamente posible almacenar la base de conocimiento en forma de tablas *hash* anidadas directamente en ZODB, esta no sería una opción práctica ya que en cada acceso a su contenido se cargaría en memoria RAM la estructura entera. Para lograr un acceso eficiente a este tipo de estructuras, ZODB ofrece un tipo de datos con un acceso similar a una tabla *hash* pero basado en árboles B+ llamado BTree [146]. En un árbol B+ todas las claves residen en sus hojas [147]. Los niveles superiores se organizan como árboles binarios que contienen los índices a los niveles inferiores. La Figura 5.12 muestra la estructura de un árbol B+. A diferencia de las tablas *hash*, un árbol B+ requiere que sus elementos estén ordenados. Encontrar un registro en un árbol B+ consume $O(log_b n)$ operaciones en el peor de los casos donde n es el número de registros almacenados y b es el orden del factor de *branching* (número de hijos de cada nodo). Además, en este tipo de estructura el acceso a uno de los registros almacenados no requiere cargar en memoria RAM la estructura completa.

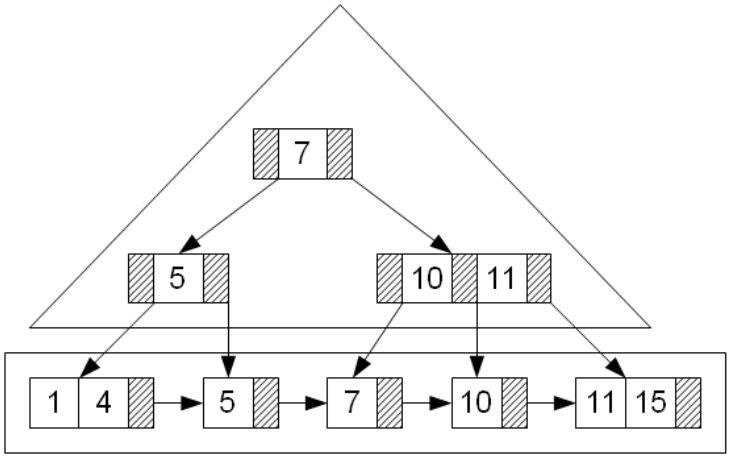


Figura 5.12: Esquema de un árbol B+.

El acceso a una ODB puede incrementar el tiempo de respuesta del motor conversacional, pero cabe analizar su uso teniendo en cuenta el posible factor limitante del uso de memoria RAM para almacenar la base de conocimiento de uno o varios dominios de aplicación del agente conversacional.

5.3.1.3. Pruebas y resultados

El principal factor de rendimiento del motor conversacional es el tiempo medio de respuesta, pero como se ha comentado anteriormente, en el caso de sistemas empotrados el uso de memoria RAM también es un factor limitante del rendimiento del agente conversacional.

La base de conocimiento empleada para estas pruebas ha sido la misma que en la Sección 4.3, la base "ALEXIA" a la que se añadieron "*categories*" extra para un dominio de aplicación de control domótico. También se repitió el proceso de medida de tiempos de respuesta, resultando los valores ofrecidos del cálculo de la media de 10 repeticiones diferentes de cada prueba.

Las pruebas realizadas tenían como objetivo analizar el posible aumento del tiempo medio de respuesta debido al uso de las mejoras para la comprensión (lematizador) y el consumo de memoria RAM (ODB). El test No. 1 constituye el punto de partida para la comparativa, $BASELINE$. En el test No. 2 se introduce el uso del lematizador, $+LMZ$. En cambio, el test No. 3 solamente hace uso de la ODB, $+ODB$. Finalmente, el test No. 4 mide el uso de ambos elementos de mejora del motor conversacional, $+LMZ + ODB$.

Se realizaron 2 rondas de pruebas con diferentes frases de entrada al motor conversacional. La entrada de la primera ronda es una sentencia simple que requiere que el motor conversacional identifique un comando simple. El algoritmo de búsqueda AIML se activa de forma recursiva 3 veces para esta entrada cuando no se hace uso del lematizador (frase de entrada "Abrir") y 4 veces cuando sí se aprovecha este segundo análisis (frase de entrada "Abre"). La sentencia de entrada para la segunda ronda es una frase compleja que requiere la identificación de múltiples elementos. En este caso el algoritmo de búsqueda se activa 14 veces sin el lematizador (frase de entrada "Por favor abrir la puerta uno de la cocina") y 15 veces haciendo uso de él (frase de entrada "Por favor abre la puerta uno de la cocina"). Para las mediciones

del uso de memoria dinámica se empleó la herramienta "*top*" incluida en todas las distribuciones comunes GNU/Linux.

La Figura 5.13 muestra que los tiempos de respuesta para la sentencia corta y sencilla es inferior a 45 ms en todos los casos. En cambio, en la Figura 5.14 se ve que el tiempo de respuesta se incrementa de forma significativa para la frases larga y multi-objetivo. El motor conversacional con su configuración por defecto obtiene respuesta en 90 ms. En caso de utilizar el lematizador, este tiempo se ve aumentado hasta llegar a los 100ms. Pero el mayor incremento viene de la mano de la ODB, que sin lematizador obtiene un tiempo de respuesta del orden de 300 ms y de 350 ms con el uso de ambos elementos de mejora.

Los resultados del uso de memoria dinámica que se ven en la Figura 5.15 indican que el uso del lematizador implica un incremento de unos 12 MBytes, mientras que la ODB logra una reducción de 11.5 MBytes. Por lo tanto, si se usan tanto el lematizador como la ODB como mejoras para la comprensión y el uso de memoria RAM, se obtiene un motor conversacional con una carga de memoria inferior a 20 MBytes independientemente de la complejidad y número de dominios de aplicación que pueda manejar el agente conversacional al completo.

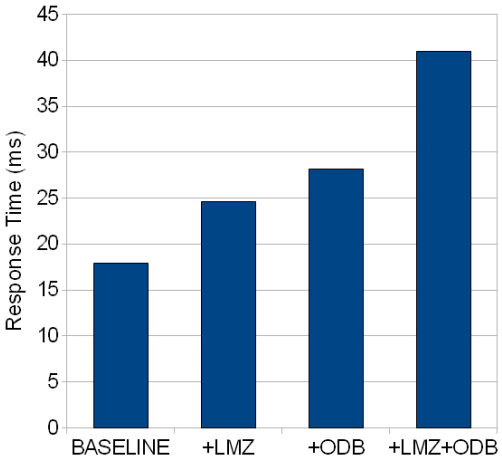


Figura 5.13: Tiempo de respuesta para la frase de entrada simple.

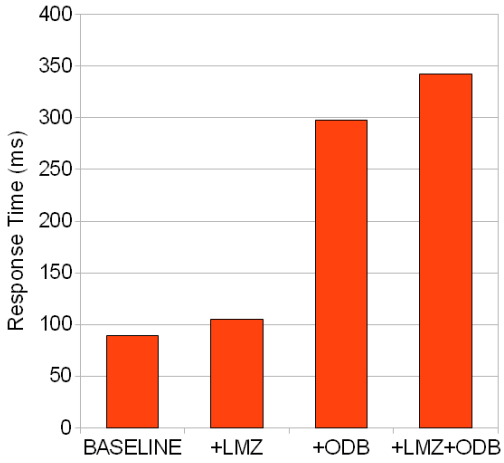


Figura 5.14: Tiempo de respuesta para la frase de entrada compleja.

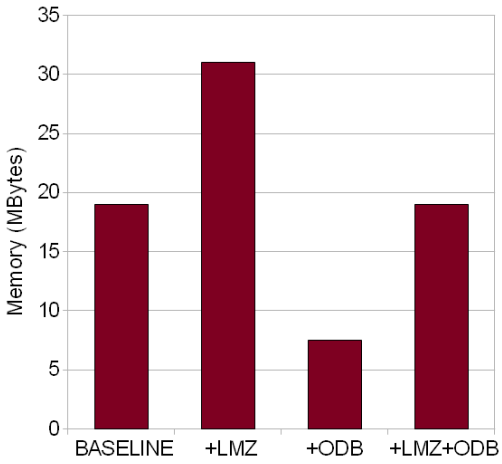


Figura 5.15: Consumo de memoria dinámica.

5.4. Mejoras del módulo VHA

5.4.1. Sincronización labial eficiente

La cabeza virtual es la encarnación del agente conversacional. Debido a la importancia que tiene la parte visual del agente conversacional de cara a mantener el interés del usuario, se intentó dotar al agente conversacional de un gran realismo. El modelado y las animaciones de la cabeza virtual fueron realizados con la herramienta Blender [148] [2]. Blender es una aplicación multiplataforma especializada en el modelado y animación de gráficos 3D y está liberada bajo licencia pública GNU GPL. La cabeza virtual cuenta con 22 huesos diferentes y se han desarrollado más de 30 expresiones diferentes además de los 10 visemas que son necesarios para la simulación del habla. En las Figuras 5.16 y 5.17 se pueden ver el modelo de la cabeza junto a su *wireframe* y las expresiones de "felicidad" y "sorpresa".

[2]Las tareas de modelado y animación de la cabeza 3D del agente conversacional fueron realizadas por personal experto de la empresa Interfaces Hombre Máquina Avanzados S.L. (IHMAN) [149]

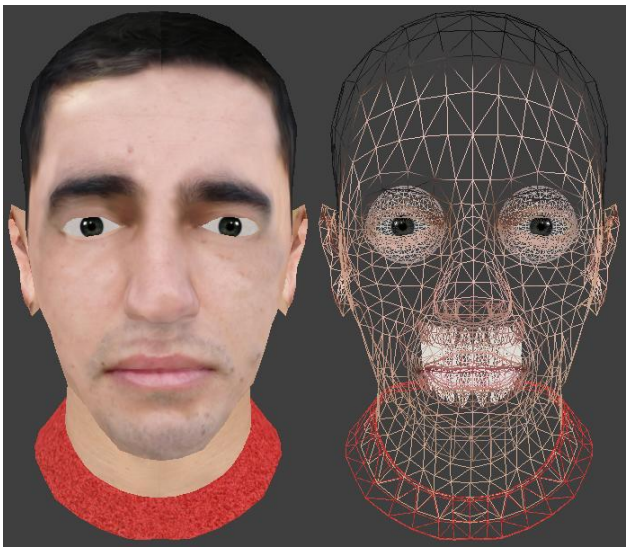


Figura 5.16: Modelo de la cabeza con texturas junto al wireframe.

Figura 5.17: Expresiones de felicidad y sorpresa.

5.4.1.1. Sincronización labial

Una de las técnicas más populares para lograr la sincronía del audio de un sistema TTS y la animación de una cabeza virtual se basa en el uso de interpolación de visemas [150]. Los visemas son la representación visual de los fonemas. Aunque la correspondencia visema-fonema puede ser uno a uno,

es usual emplear un visema para representar varios fonemas. En la Tabla 5.4 se muestra la lista de 10 visemas desarrollados para el agente conversacional objeto de estudio en esta tesis donde el visema *BC* representa una posición de reposo con la boca cerrada. La agrupación de fonemas a visemas se ha realizado en función de la similitud en la articulación de la boca durante la pronunciación de cada fonema.

Visema	**Fonemas**	**Grafías**
A	/a/	a
BC	-	-
BMPV	/B/, /m/, /p/	b, m, p, v
CDGJLNRTZ	/k/, /D/, /G/,/x/, /l/, /y/, /n/, /ñ/, /r/, /t/, /z/	c, k, qu, d, g, gu, j, l, ll, y, n, ñ, r, rr, t, z
E	/e/	e
F	/f/	f
I	/i/	i
O	/o/	o
SCH	/s/, /č/	s, ch
U	/u/	u

Tabla 5.4: Lista de agrupaciones de visemas-fonemas.

Para realizar la animación se parte normalmente de la información de fonemas y tiempo que ofrece el módulo TTS [151]. Esto es una lista de todos los fonemas que serán sintetizados y el tiempo estimado de duración de cada uno de ellos. A partir de esta información, lo habitual es realizar una conversión fonema a visema y ajustar la reproducción de estos a la duración de cada fonema. Esto presenta un problema y es que los visemas suelen ser simplemente $keyframes$ y deben ser interpolados en tiempo real a través del sistema de renderizado. Las técnicas de interpolación también han sido objeto de estudio, así como la dificultad de realizar esta interpolación junto con expresiones faciales [152].

Todas las técnicas de interpolación implican un esfuerzo computacional extra $frame$ a $frame$ para el sistema de renderizado debido al un conjunto de

operaciones matemáticas y la necesidad de tener activos al menos 2 visemas del modelo. Con el objetivo de lograr una animación fluida incluso en sistemas empotrados de gama baja-media, se ha propuesto una modificación de esta técnica que elimina la complejidad de la interpolación de visemas.

La modificación que se propone consiste en extraer el cálculo de la interpolación en tiempo real del *render* y desarrollarla de forma previa en Blender como animaciones. Esto quiere decir que ahora los visemas no son poses estáticas que son interpoladas posteriormente sino que se convierten en animaciones entre 2 de esas poses. Por ejemplo, con el nuevo esquema existe una animación "A_O" para pasar del fonema vocálico "A" al fonema vocálico "O", cuya secuencia de animación se puede ver en la Figura 5.18.

Figura 5.18: Secuencia de animación del visema A_O.

Con esta modificación, la conversión de fonemas a visemas es trivial y solamente se deben concatenar unos detrás de otros. Posteriormente se ajusta la duración de cada visema/animación en función de la información de tiempos que provee el módulo TTS.

5.4.1.2. Pruebas y resultados

El módulo VHA con las modificaciones propuestas se ha ejecutado en la BeagleBoard alcanzando una tasa de reproducción próxima a 20 $frames/s$, suficiente para considerar la animación fluida. Además, se ha probado la fácil adaptación de este módulo optimizado a otro tipo de entornos. Limitando

el $framerate$ del sistema de renderizado a 10 y 15 $frames/s$ el sistema se adapta correctamente y logra mantener la sincronía con el audio.

Algunos sistemas TTS ofrecen una característica para poder variar la velocidad de reproducción del audio sintetizado. Gracias a la sencillez del módulo VHA modificado, es posible aplicar esta variación de reproducción también a la animación simplemente multiplicando por un factor constante la duración normal del visema.

5.5. Discusión de los resultados

Después de haber descrito en sus respectivas Secciones las diferentes propuestas de mejora y optimización realizadas sobre la plataforma base, es necesario realizar una discusión del alcance de los resultados obtenidos.

La primera de las mejoras propuestas para reducir el tiempo de respuesta del módulo ASR consistió en una mejor estructura de comunicación entre los módulos VAD y ASR del agente, aprovechando el carácter interactivo de la conversación entre usuario y ECA. Suponiendo como en la Sección 4.6 una frase típica de usuario de 3 segundos de duración, se puede constatar que se consiguen ahora tiempos de respuesta inferiores a 1 segundo, manteniendo la WER en valores ligeramente superiores al 7 %. Estos valores hacen uso de una configuración conservadora del esquema híbrido con parámetros optimizados, estableciendo el umbral de la medida de confianza al 90 %. Incluso se pueden lograr tiempos de respuesta inferiores a 500 ms bajando el umbral de la medida de confianza al 10 %, a costa de aumentar la WER al 9 %. Estos resultados ya permiten crear ECAs con la plataforma optimizada capaces de mantener una conversación interactiva con el usuario, sobre todo teniendo en cuenta que el *hardware* de pruebas se puede considerar actualmente de gama baja.

La segunda propuesta de mejora del módulo ASR está enfocada a dotar a la plataforma de desarrollo de ECAs de una gran flexibilidad. Normalmente, un mismo agente que se ejecute en un sistema empotrado debe poder asistir al usuario en diferentes dominios de aplicación. Ante el posible incremento de la complejidad del modelo de lenguaje necesario para cubrir diferentes tareas y aplicaciones, se propuso incorporar a la librería PocketSphinx una técnica de

LMS, basada en la clasificación del tópico de la conversación mediante SVMs. Los resultados con tan sólo 3 dominios de aplicación diferentes muestran que es posible obtener una reducción simultánea de los valores xRT y WER. Es de suponer que esta optimización ofrece mejores resultados a medida que se incrementan nuevos dominios de aplicación al ECA.

El objetivo para el módulo CE era una reducción del consumo de memoria RAM, ya que el tiempo de respuesta era muy reducido en la versión de base. Por lo tanto, se ofrecieron 2 alternativas que intentan tratar el problema de la memoria desde perspectivas diferentes. La inclusión de un lematizador al módulo CE permite reducir la estructura AIML simplificando y unificando diferentes "*patterns*". Pero el uso del lematizador presenta una desventaja, y es que incrementa el tamaño en memoria del módulo CE en 10 MBytes. Para poder reducir este consumo de memoria se incluyó una ODB al módulo CE, de forma que la estructura AIML que representa el "cerebro" del ECA pueda residir en disco y ser accedido de forma eficiente en tiempo de ejecución. El uso conjunto del lematizador y la ODB permiten aprovechar las ventajas de ambos limitando el consumo de memoria del módulo CE.

Finalmente, se presentó un diseño eficiente del módulo VHA que consigue lograr una perfecta sincronización de la animación del avatar 3D con la voz sintética generada en el módulo TTS. La sencilla concatenación de visemas/animaciones en la que se basa dicha propuesta permite reducir la complejidad de los cálculos que se deben hacer *frame* a *frame* durante el renderizado del avatar. De esta forma, es posible alcanzar un *framerate* cercano a 20 $frames/s$ en una sistema empotrado de gama baja.

5.6. Conclusiones

En este Capítulo se han propuesto diferentes mejoras que solucionan las limitaciones de rendimiento de ECAs en sistemas empotrados que se habían identificado en el Capítulo 4. Se han tratado por separado el tiempo de respuesta del módulo ASR y el consumo de memoria del módulo CE.

El primero de los problemas se abordó desde dos perspectivas diferentes y complementarias. Por un lado se diseñó una estructura híbrida de comunicación entre VAD y ASR. La otra perspectiva se centra en la complejidad

del modelo de lenguaje interno al sistema ASR, adaptándolo al dominio de aplicación que trate la conversación del usuario con el ECA.

El otro problema era el consumo de memoria del módulo CE. Mediante la inclusión de un lematizador y una ODB en el intérprete PyAIML en que se basa el módulo CE se consiguió limitar el tamaño en memoria a la vez que se ganó en flexibilidad y se simplifica el desarrollo y mantenimiento de la compleja estructura AIML que define el comportamiento del ECA.

Por último, se describió una técnica de sincronización labial entre los módulos VHA y TTS, enfocada a reducir al máximo el número de cálculos que se deben hacer $frame$ a $frame$ para ajustar la animación del avatar a los tiempos de la voz artificial del ECA.

El siguiente Capítulo trata el proceso de adaptación de la plataforma de desarrollo de ECAs optimizada, el último de los objetivos planteados para esta tesis.

Capítulo 6

Adaptación de la arquitectura a Android

6.1. Introducción

El lanzamiento al mercado del primer dispositivo equipado con Android a finales de 2008, supuso un punto de inflexión en el ámbito de los dispositivos móviles. Unos pocos años más tarde, Android domina más del 80 % del mercado de *smartphones* [153]. Este hecho supone que cada día más dispositivos móviles y sistemas empotrados con capacidad de interactuar con el usuario se equipan con Android. Uno de los motivos de esta revolución es el hecho es la portabilidad de las aplicaciones Android.

Con el propósito de asegurar una mayor difusión de la plataforma de desarrollo de agentes conversacionales 3D cuyos detalles se han visto en los Capítulos 3 y 5, se tomó la decisión lógica de portar esta plataforma a Android.

Este Capítulo describe el proceso de la adaptación, tanto a nivel de diseño como a nivel de implementación, del agente conversacional 3D a la plataforma Android. Para ello, primero se describen conceptos generales sobre Android y posteriormente su arquitectura. También se muestra brevemente su modelo de programación de aplicaciones, muy diferente al de los sistemas Linux clásicos. Tras esta introducción de conceptos generales, se describen las modificaciones del diseño e implementación que ha sido necesario realizar en el

agente conversacional. Finalmente, se expone un nuevo análisis del agente conversacional ejecutándose en un dispositivo Android real. Este análisis se centra, al igual que el efectuado en el Capítulo 4, en la latencia o tiempo de respuesta de cada uno de los módulos que forman el agente y también la WER en el caso específico del módulo ASR. Además, se introduce también un estudio del consumo de energía como nueva medida de rendimiento/eficiencia del agente conversacional.

6.2. Sistema Operativo Android

Android es una pila de *software* de código abierto basado en Linux compuesta por SO, *middleware*, aplicaciones móviles nativas junto a un conjunto de librerías que ofrecen una API para crear aplicaciones de terceros. Fue diseñado principalmente para dispositivos móviles con pantalla táctil como *smartphones*, *tablets* y actualmente su uso se ha extendido a otros sistemas empotrados. Android fue anunciado públicamente en 2007 y el primer móvil en incorporarlo salió en 2008.

Android es de uso gratuito y está diseñado con múltiples implementaciones *hardware*; además es de código abierto y su código fue liberado bajo la licencia Apache. Es un paquete de *software* que consta no solo de SO, sino de *middleware* y aplicaciones clave también. Es uno de los SO de mayor difusión y uso. Android posee una gran comunidad de desarrolladores de aplicaciones propias, escritas principalmente en una versión adaptada del lenguaje de programación Java. Cualquiera con un conocimiento básico de Java puede comenzar a desarrollar aplicaciones Android. Las versiones del SO Android van desde la 1.0 hasta la 4.4 KitKat. Todas las versiones del SO llevan nombres de postres o dulces y están ordenados alfabéticamente, comenzando por la 1.5 Cupcake hasta la 4.4 KitKat.

El *kernel* del SO Android es Linux, desde la versión 2.6 hasta la 3.x a partir de Android 4.0, por lo que se suele convenir que es un SO basado en Linux. Android cuenta con librerías de código abierto con API pública para desarrollo de aplicaciones incluyendo SQLite, Webkit, OpenGL y un gestor de medios. Embebida en la capa de librerías está el *runtime* de Android, que contiene la Máquina Virtual Dalvik - *Dalvik Virtual Machine* (DVM).

Además, Android también contiene aplicaciones nativas importantes como un gestor de mensajes, un cliente de correo electrónico, un reproductor de música, un navegador *web*, una aplicación de cámara, etc..

La DVM es una máquina virtual comparable a la Máquina Virtual Java - *Java Virtual Machine* (JVM) original. Mientras que Google eligió Java como lenguaje de desarrollo de aplicaciones, decidió sustituir tanto JME como JVM por una plataforma de ejecución diferente, la máquina virtual Dalvik. La DVM opera en *bytecodes* que son convertidos a archivos de clases Java y posteriormente compilados al formato de archivo .dex mediante una herramienta incluida en el *Kit* de Desarrollo de Software Android *Software Development Kit* (SDK) [154]. Los procesos internos de la DVM son optimizados típicamente para ajustarse a los requisitos de bajo uso de memoria necesarios para el desarrollo e implementación de aplicaciones para teléfonos móviles. Además está diseñado para permitir la ejecución simultánea de múltiples máquinas virtuales.

6.3. Arquitectura del SO Android

El SO Android consiste en 4 capas o niveles principales: *kernel*, librerías, *framework* de aplicaciones y aplicaciones y se muestra en el Figura 6.1.

6.3.1. Capa del núcleo/kernel

El SO Android al completo está construido sobre un *kernel* Linux con algunos cambios a nivel de arquitectura hechos por Google. El *kernel* Linux es el responsable de la interacción principal con el *hardware* y contiene los *drivers* esenciales en su interior. Linux fue elegido debido a su demostrada trayectoria en sistemas de escritorio y en muchos casos no es necesario reescribir los *drivers*. Linux también provee características como memoria virtual, entorno de red y gestión de energía.

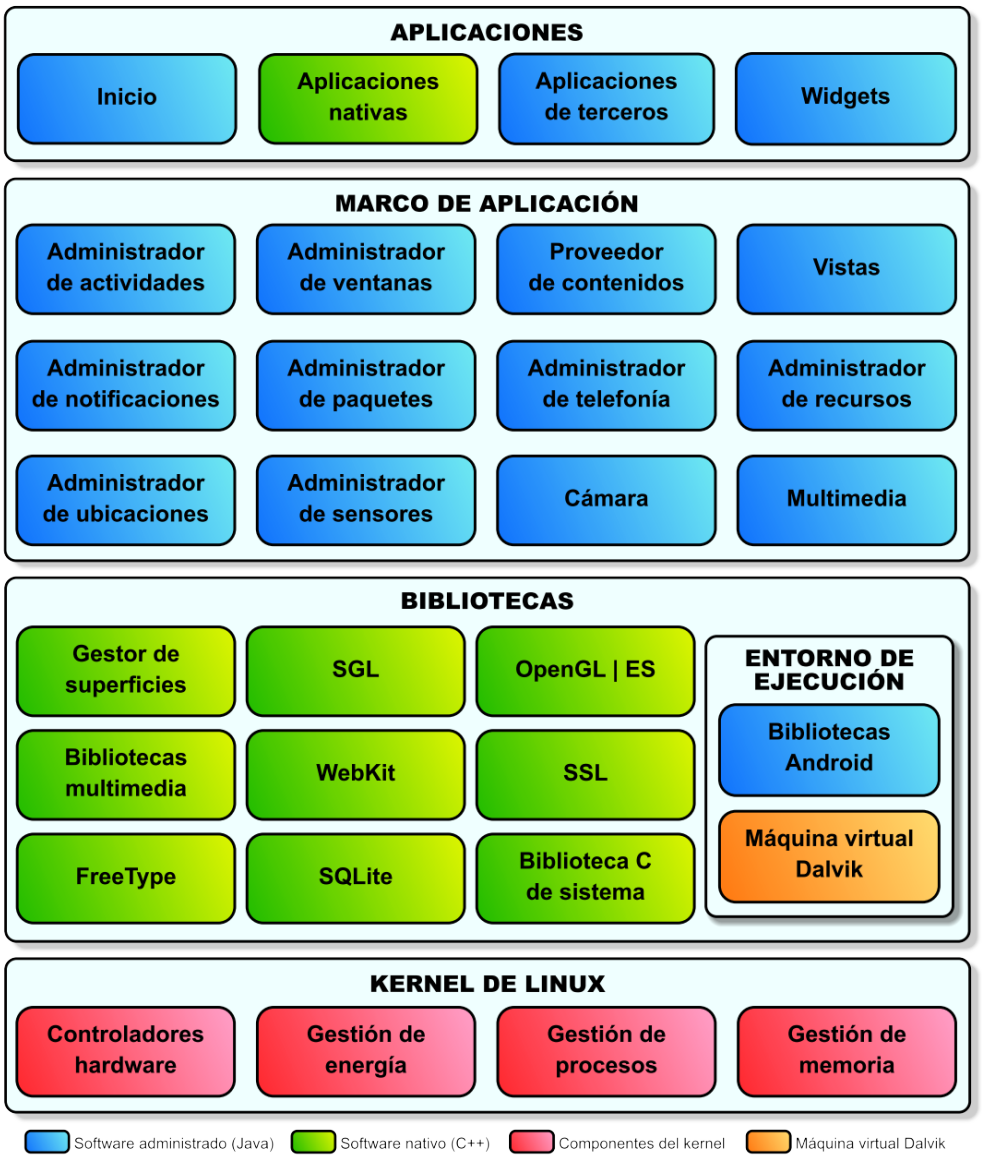


Figura 6.1: Arquitectura software de Android.

6.3.2. Capa de librerías

La capa de librerías nativas provee a Android de las capacidades de sus características principales. Android está equipado con SGL, que actúa como

el sistema de renderizado 2D primario. Su equivalente para soporte de gráficos 3D es OpenGL ES. Para la gestión y almacenamiento de datos incorpora la librería SQLite. El renderizado web corre a cargo de Webkit, que ha sido modificado para adaptar las páginas web a pantallas de menor tamaño. La máquina virtual Dalvik también forma parte de esta capa. Las librerías Dalvik que implementan una API de propósito general están escritas en Java y, tal como se ha explicado anteriormente, están compiladas de forma que su ejecución sea rápida y altamente eficiente en procesadores de baja capacidad computacional.

6.3.3. Capa de framework de aplicaciones

El *framework* de aplicaciones ofrece la mayor parte de las APIs que necesitan las aplicaciones, incluyendo aspectos como intercambio de datos entre aplicaciones, acceso al sistema de telefonía y la recepción de notificaciones. Esta capa y la siguiente superior están escritas completamente en Java.

6.3.4. Capa de aplicaciones

Esta es la capa superior de la arquitectura Android. La capa contiene *software* desarrollado por el equipo de Android así como aplicaciones de terceros que estén instaladas en el dispositivo. Incluso aplicaciones fundamentales como las de telefonía o la gestión de contactos residen en esta capa y cualquier desarrollador puede tener acceso. Un ejemplo de esta flexibilidad es que desarrolladores externos pueden realizar versiones alternativas de dichas funcionalidades básicas del dispositivo.

Android en su conjunto ofrece soporte para servicios al igual que cualquier otro SO moderno, como la memoria virtual y programación concurrente en una plataforma móvil. Muchos de los servicios con los que cuenta Android son el resultado de haber incluido un *kernel* Linux. Por su parte, el equipo de Google introdujo toda la pila de librerías necesarias para la funcionalidad de telefonía.

6.4. Ventajas de Android

Existen muchos otros SOs móviles en el mercado, incluyendo iOS, Windows Phone OS, Java Mobile Version, Symbian, Blackberry OS, etc.. Android es el único SO móvil que combina una arquitectura basada en componentes, ser de código abierto, servicios "*out-of-the-box*", gestión automática del ciclo de vida de la aplicación, buen soporte para gráficos y un amplio soporte de diferentes tipos de *hardware*. Todas estas razones hacen de Android un SO con gran éxito en plataformas móviles. A continuación se describe cómo Android ofrece todas estas características:

- Gestión efectiva del ciclo de vida de la aplicación: Android provee múltiples niveles de seguridad que hacen sencillo aislar programas entre sí. De esta forma el usuario no tiene que preocuparse del ciclo de vida de las aplicaciones ni de tener que cerrar unas para poder ejecutar otras. Todos estos aspectos se manejan de forma efectiva por el SO, lo que le aporta estabilidad. También está optimizado para un uso reducido de memoria y consumo de energía, lo que lo hace adecuado para un amplio abanico de dispositivos.

- Código abierto: Android es una plataforma genuina de código abierto de forma que los desarrolladores obtienen una libertad total para crear y probar sus aplicaciones. Esta es la razón por la que Android cuenta con un número de aplicaciones muy elevado y sigue creciendo cada día.

- Servicios "*out-of-the-box*": Android ofrece un amplio rango de servicios que hacen sencillo el desarrollo de aplicaciones. Las aplicaciones tienen acceso directo a los servicios de localización que hacen uso de las características de red y GPS, a las capacidades de almacenamiento de bases de datos, al servicio de mapas, nevegadores web, proveedores de contenidos y otras muchas características que ayudan a crear aplicaciones estables y eficientes.

- Arquitectura basada en componentes: Se pueden reemplazar los componentes existentes por otros diferentes o de versiones más recientes de forma muy sencilla.

- Soporte para gráficos: Android cuenta con soporte para gráficos vectoriales 2D, gráficos suavizados, animaciones 3D, soporte para Flash y OpenGL. Todas estas características resultan en una experiencia de juegos de alta calidad. Además, posee implentaciones de *códecs* que permiten reproducir prácticamente cualquier tipo de contenido multimedia en el dispositivo.

- Soporte *hardware*: Los programas se escriben en su mayoría en lenguaje Java que se ejecuta en la máquina virtual Dalvik. Por lo tanto, el código es portable a arquitecturas ARM, x86 y muchas otras. Android también soporta una gran variedad de métodos de entrada como teclados, pantallas táctiles, *touchpads* y *trackballs*. Finalmente, la interfaz de usuario se puede adaptar fácilmente a pantallas de todo tipo de resoluciones y tamaños.

6.5. Modelo de programación en Android

Una aplicación Android consiste en un conjunto de recursos empaquetados en un único archivo, un paquete Android o "apk". Esta aplicaciones se suelen escribir en Java, haciendo uso de herramientas de desarrollo Java estándar. Los archivos creados se procesan posteriormente para generar código específico para la DVM. Cada aplicación se ejecuta en un proceso Linux propio (con su instancia propia de la DVM) que protege su código y datos de ser accesibles desde otras aplicaciones.

Una aplicación consiste en un conjunto de componentes que se instancian y ejecutan bajo demanda. Por lo tanto, no existe realmente una punto de entrada específico como la función "main()".

Existen 4 tipos de componentes de aplicación: "*Activities*", "*Services*", "*Broadcast Receivers*" y "*Content Providers*". A continuación se expone brevemente el propósito de cada uno de ellos:

- "*Activity*": Es una unidad funcional de la aplicación que posee algún tipo de interfaz de usuario que puede ser invocada por otra "*Activity*". Aunque una aplicación puede contar con varias "*Activities*", una de

ellas suele ser la principal y es la que se carga primero al lanzarse la aplicación.

- "*Service*": Es similar a una "*Activity*" excepto que suele ejecutarse en segundo plano y no cuenta con interfaz de usuario.

- "*Broadcast Receiver*": Estos componentes responden a mensajes de difusión enviados desde otras aplicaciones o desde el propio sistema.

- "*Content Provider*": Este elemento genera datos desde una aplicación hacia otras bajo demanda. Estos datos pueden estar almacenados en el sistema de archivos, en una base de datos o en cualquier otra implementación.

A la hora de desarrollar una aplicación Android, también es necesario crear una descripción de ésta para el sistema. Esto es lo que se conoce como un "*Manifest File*". Consiste en un archivo XML que se almacena en la raíz del sistema de archivos de la aplicación.

Cuando una aplicación Android requiere una funcionalidad de otra aplicación distinta o del propio sistema, debe generar un "Intent". Esto es un mensaje asíncrono cuya función es provocar la instanciación y ejecución de una "*Activiy*", "*Service*" o "*Broadcast Receiver*".

En las primeras versiones de Android se pudo comprobar que este modelo de programación presentaba limitaciones para desarrollar aplicaciones que tuvieran partes de código complejas desde el punto de vista computacional y se tornaban lentas y poco eficientes desde el punto de vista del usuario. La principal razón de este hecho era la imposición del uso del lenguaje Java, mucho más lento que los lenguajes C o C++. Para solucionar este problema, Google introdujo a partir de su versión 1.5 el *Native Development Kit* (NDK), un *kit* de desarrollo de código nativo. A partir de entonces, las aplicaciones Android pueden implementar las partes críticas de su código en lenguaje C/C++, que es accesible mediante la interfaz *Java Native Interface* (JNI) [155].

6.6. Portabilidad del agente conversacional

Adaptar una plataforma tan heterogénea como la del agente conversacional 3D visto en el Capítulo 3 a Android no es una tarea trivial y requiere un cambio en el diseño. En la Figura 6.2 se pueden observar las diferencias entre ambos ecosistemas. En la primera, las aplicaciones de usuario interaccionan con las librerías de forma sencilla a través de la interfaz del sistema. En la segunda, el sistema de comunicación es más complejo y requiere el paso de información a través de una pila de librerías en Java, la propia DVM y el acceso a las librerías nativas a través de JNI.

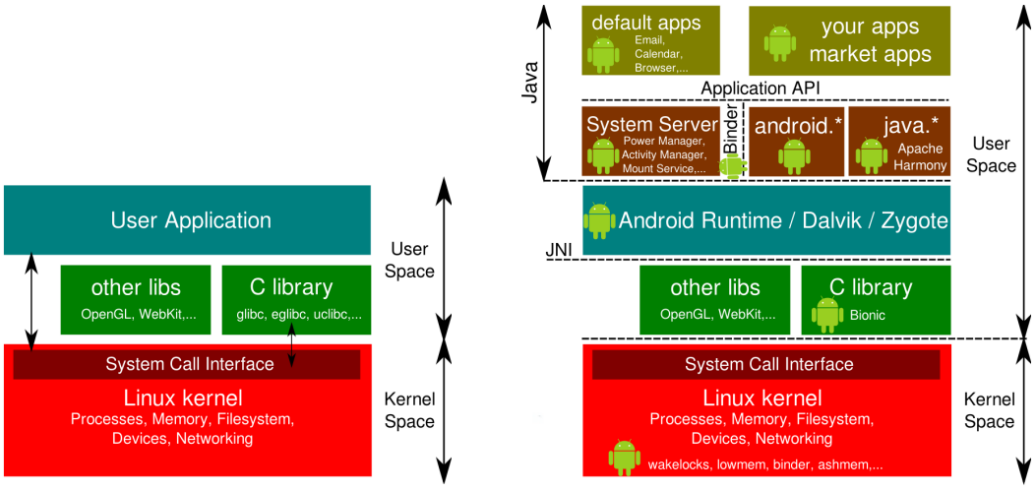


Figura 6.2: Arquitecturas Linux y Android.

En este punto, es necesario recordar que excepto el módulo CE del agente conversacional que está implementado en Python, el resto de componentes (VAD, ASR, TTS y VHA) están escritos en C/C++. Todos ellos incluyen cálculos complejos, por lo que hubiera sido costoso y poco eficiente reescribirlos en Java. Como cada uno de los módulos requirió un tratamiento específico, se decribe en los apartados siguientes la adaptación de todos ellos.

6.6.1. Módulos VAD+ASR

La principal dificultad para convertir los módulos VAD y ASR en librerías de código nativo en Android es que la librería SphinxBase requiere el acceso

a alguna de las librerías de audio típicas de sistemas Linux: OSS, ALSA, ESD o S60. Android no implementa ninguna de ellas ya que Google optó por una implementación alternativa, la *Open Sound Library for Embedded Systems* (OpenSL_ES) [156].

OpenSL_ES es una API de audio escrita en lenguaje C diseñada específicamente para dispositivos con recursos limitados. Fue desarrollada por el grupo Khronos, un consorcio de empresas centrado en la creación de estándares abiertos y APIs de software libre. OpenSL_ES fue concebido originalmente con el objetivo de reducir la fragmentación que existía en las aplicaciones de audio de los dispositivos móviles. Antes de la publicación de OpenSL_ES existían muchas APIs de audio pero eran en su mayoría propietarias.

Ante esta dificultad se decidió adaptar SphinxBase para poder acceder directamente a OpenSL_ES y de esta forma mantener inalterada el resto de la librería. Para ello, se desarrolló un *wrapper* que abstrae a SphinxBase de las particularidades de OpenSL_ES.

Una vez resuelto este problema, se presentaban 3 alternativas para los módulos VAD y ASR que también se representan en la Figura 6.3:

1. VAD y ASR como librerías nativas independientes y uso en modo serie: En este caso cada módulo es una parte de la aplicación en código nativo y se accede a ellas a través de JNI. Esta alternativa mantiene al máximo la modularidad de la arquitectura del agente y requiere una sola llamada JNI a cada uno de los módulos. La principal desventaja es que el uso de ambos módulos en modo serie implica una latencia inasumible (ver Capítulo 4).

2. VAD y ASR como librerías nativas independientes y uso en modo paralelo/híbrido: Al igual que antes cada módulo es una librería nativa accesible mendiante JNI. Esta alternativa también presenta la ventaja de la modularidad, pero implicaría un número muy elevado de llamadas JNI con paso de datos. La alta frecuencia de llamadas JNI podría implicar un aumento global de la latencia significativo [157]. Otra desventaja es que este enfoque resulta en que el control de la comunicación entre ambos módulos habría que realizarlo en Java, por lo que también se podría perder eficiencia por esta causa.

3. VAD y ASR como una sola librería nativa conjunta y uso en modo serie/paralelo/híbrido: La principal desventaja de esta alternativa es que los módulos VAD y ASR pierden la modularidad y se convierten en una sola librería nativa. A cambio, se gana la eficiencia ya que el uso de este nuevo módulo VAD+ASR solamente requiere una llamada JNI y permite configurar la comunicación entre VAD y ASR tanto en modo serie, en paralelo o híbrido de forma transparente desde el punto de vista de la aplicación Java.

Como el objetivo principal del agente conversacional es la interactividad con el usuario, se ha optado por la última de las alternativas, por lo que los módulos VAD y ASR se convierten en un nuevo módulo VAD+ASR implementado como librería nativa en Android, que accede directamente a la librería de audio nativo de bajo nivel OpenSL_ES.

6.6.2. Módulo CE

Podría parecer que al estar escrito en Python, el módulo CE tendría que reescribirse o buscar una implementación alternativa del intérprete AIML. Afortunadamente, el equipo de Google también decidió dar soporte a lenguajes de *script* como Python, Perl, JRuby, Lua, BeanShell, JavaScript, Tcl, y Shell mediante la librería escrita en Java *Scripting Layer for Android* (SL4A) [158].

SL4A ofrece por defecto el acceso a un conjunto elevado de las APIs Android en Java. De esta forma, un desarrollador puede escribir una aplicación que controle la cámara del *smartphone* o acceder a los contactos del usuario a través de una API sencilla en un lenguaje de *script*. Este acceso se realiza mediante módulos de código denominados "*Facades*", que encapsulan varios de los métodos de las APIs de Android y los hacen accesibles de una forma sencilla.

Gracias a la existencia de SL4A, se decidió mantener la implementación del módulo CE en base al intérprete PyAIML. Pero se debe recordar que el módulo CE cuenta con 2 mejoras vistas en el Capítulo 5: el lematizador y la ODB. La ODB se añadió al intérprete PyAIML de forma sencilla ya que también está escrito en Python. En cambio, el lematizador se basaba en la

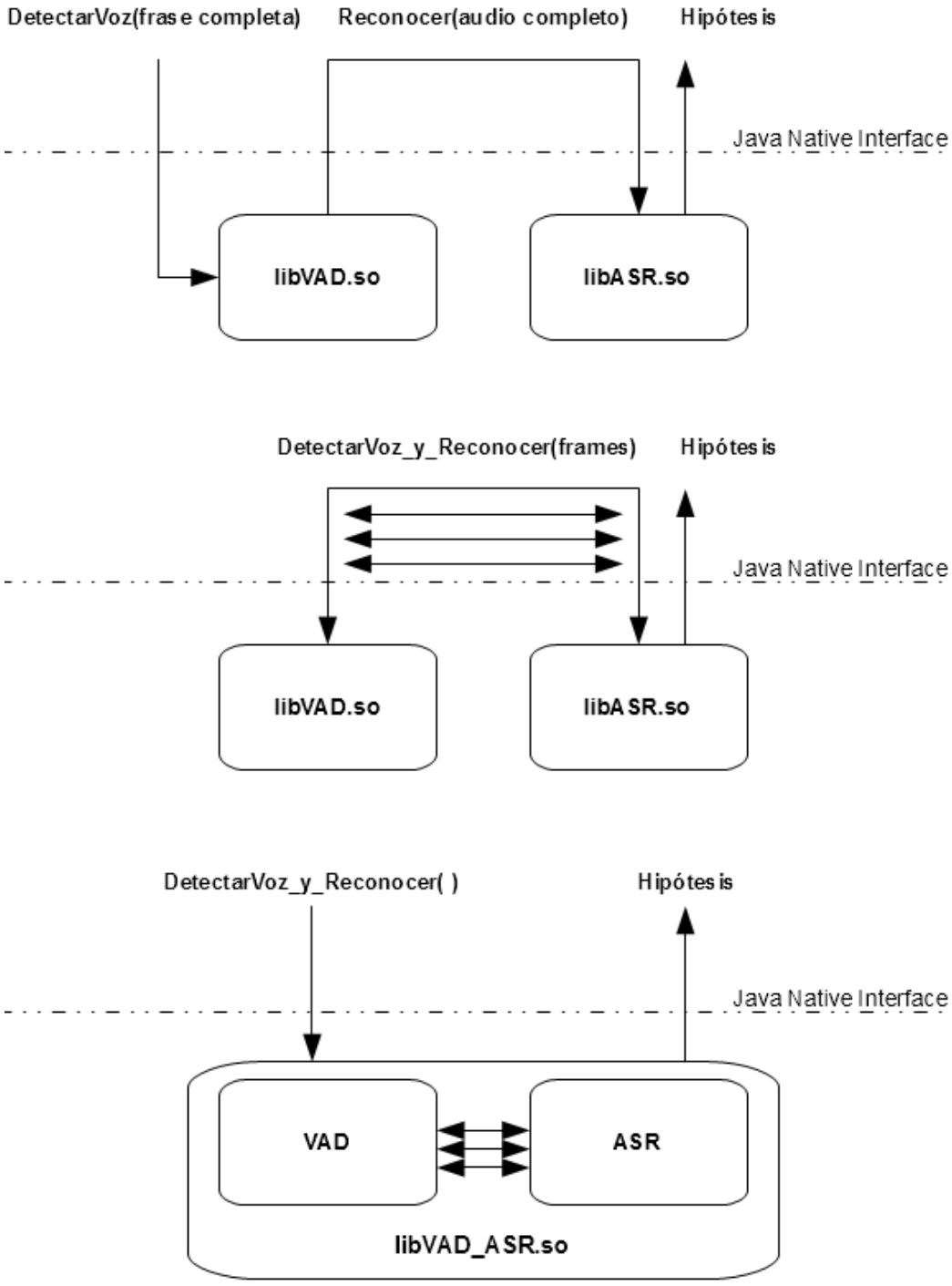


Figura 6.3: Alternativas VAD+ASR.

librería Freeling, escrita en C++. Después de comprobar que existían dependencias e incompatibilidades de esta librería con el conjunto de librerías del núcleo de Android, se decidió reimplementar el lematizador en otro conjunto de librerías de procesado de lenguaje natural, la *Natural Language ToolKit* (NLTK), implementado en Python [159] [160]. El NLTK también cumple los requisitos de código abierto y cuenta con una gran comunidad y documentación disponible.

6.6.3. Módulo TTS

En el Capítulo 5 se vio que las librería eSpeak y MBROLA son las que forman el módulo TTS del agente conversacional. La portabilidad a Android de la primera es sencilla y se convertiría en una librería nativa de la aplicación Android, pero MBROLA se ofrece como un binario ejecutable y no tiene el código liberado. Por esta razón no se puede recompilar para Android y su uso ha sido descartado.

Recordando que eSpeak se basa en la técnica de formantes, la calidad de la voz artificial que genera es muy baja y de sonoridad robótica. Este factor puede provocar un rechazo del usuario al agente conversacional ya que resultaría chocante una voz robótica junto a una cabeza animada con un elevado grado de realismo.

Afortunadamente y coincidiendo con este problema, se consiguió acceso al código de una voz en español compatible con el sistema TTS Flite y basada en difonemas. Este tipo de voces presenta una mayor calidad y naturalidad que las basadas en formantes. Además, como se explica en el Capítulo 3, Flite es una alternativa eficiente y ligera a Festival, y está diseñada específicamente para sistemas empotrados. Por todas estas razones, se decidió cambiar la base del módulo TTS de eSpeak a Flite.

Aunque Flite esté basado en Festival, el código se ha reescrito por completo y no ofrece por defecto la funcionalidad necesaria para obtener la información de tiempos de los fonemas a sintetizar. Esta vez fue necesario modificar la librería Flite para extraer la información de tiempos que sí se mantenía internamente en el sintetizador.

6.6.4. Módulo VHA

En el momento de la decidir la portabilidad del agente conversacional 3D a Android, la comunidad de desarrolladores de Ogre3D también vio una oportunidad en el mundo de los dispositivos móviles y crearon una versión experimental especifica para este SO móvil. Esta versión experimental pretende ser una plataforma de pruebas de todas las funcionalidades que ofrece Ogre3D, por lo que al ser incluida como librería nativa Android en el agente conversacional, resultaba una aplicación muy pesada, con un arranque lento y con un consumo de memoria elevado. En este caso se realizó un análisis del código interno de la librería y se eliminaron la mayoría de las funcionalidades y complementos innecesarios para el funcionamiento del agente conversacional 3D.

6.6.5. Arquitectura del agente conversacional en Android

Después de describir la problemática encontrada y las soluciones aportadas en el proceso de adaptación del agente converacional 3D a la plataforma Android, se hace necesario ofrecer una visión de la nueva arquitectura resultante. Ésta se puede ver gráficamente en la Figura 6.4. A continuación se detallan las diferencias que se pueden encontrar entre la versión del agente conversacional 3D para sistemas empotrados Linux (ver Figura 3.8) y la versión para dispositivos Android:

- Los módulos VAD y ASR en la nueva arquitectura se han transformado en una librería nativa que engloba ambas funcionalidades y accede directamente a la librería nativa de audio OpenSL_ES de Android. El tamaño de la librería dinámica resultante es de tan sólo 518 KBytes.
- Del módulo CE solamente se ha modificado el lematizador, que pasa de estar basado en FreeLing a ser implementado mediante el uso del conjunto de librerías NLTK.
- La base del módulo TTS dejan de ser eSpeak y MBROLA. En cambio, se modifica la librería Flite y se consigue una voz en español de buena

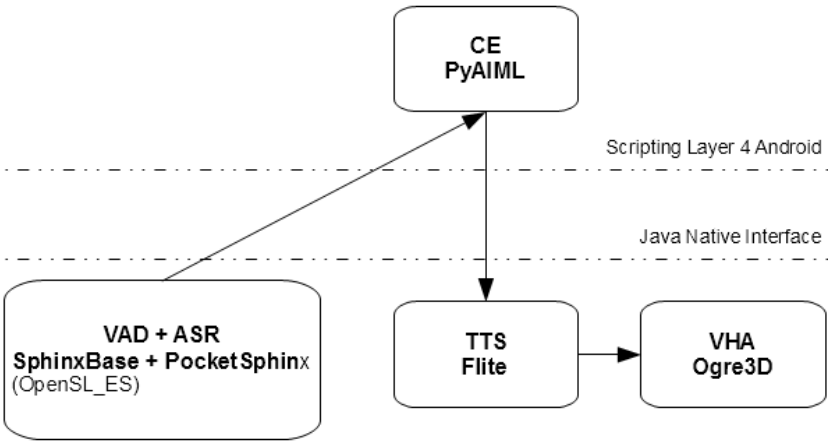


Figura 6.4: Arquitectura del agente conversacional 3D para Android.

calidad compatible con ella. Tanto la nueva versión de Flite como la voz en castellano se compilan como una librería dinámica nativa en Android con un peso de 1.76 MBytes.

- Aunque el módulo VHA sigue empleando Ogre3D como *render*, ésta es una modificación de una versión experimental específica para Android. La librería dinámica nativa resultante cuenta con un tamaño de 8.95 MBytes.

Anteriormente se ha explicado que una de las claves del éxito de Android es la libertad que ofrece a la comunidad de desarrolladores para acceder sin restricciones a todas las funcionalidades de la plataforma móvil. En una línea de progreso similar a la descrita en esta tesis, Google ha ido añadiendo a Android las funcionalidades de VAD, ASR y TTS que también se incluyen en las últimas versiones de la plataforma. Algunas otras compañías con una larga trayectoria en las tecnologías del habla también ofrecen soluciones de ASR y/o TTS para Android. Gracias a la modularidad de la arquitectura del agente conversacional de la Figura 6.4, se ha añadido la posibilidad de emplear alguno de estos módulos alternativos.

6.7. Análisis de la portabilidad

Para validar la calidad de la portabilidad del agente conversacional 3D a Android, se debe garantizar que sigue siendo posible un nivel adecuado de interactividad y también que la autonomía del agente como aplicación que debe estar ejecutándose en un dispositivo móvil es suficientemente alta como para ser práctica para el usuario. Por lo tanto, las 2 métricas principales que se establecieron para medir el rendimiento del agente en Android fueron la latencia y el consumo de energía. Además, para el módulo ASR también se hizo una medida de la tasa de error de palabra. Para poder medir la latencia y de cada módulo que conforma el agente conversacional 3D se emplearon las trazas del *log* de depuración que ofrece Android como plataforma. El consumo de energía se midió con la ayuda de la aplicación PowerTutor, disponible para uso gratuito en Android [161]. Para poder extraer una valoración cualitativa de la bondad de la implementación y mejoras del agente conversacional 3D, se tomaron medidas de otras alternativas a los módulos del agente conversacional siempre que fue posible.

6.7.1. Sistema hardware de pruebas - Samsung Galaxy I8190

En la etapa final de desarrollo de esta tesis, ante el éxito y la difusión incontestables de la plataforma de software para sistemas móviles Android, se decidió portar el agente conversacional a dicha plataforma para poder garantizar su uso en una gran variedad de dispositivos (plataformas de desarrollo, *tablets*, *smartphones*, etc.) y dotar así de una gran versatilidad de aplicaciones y proyectos futuros al agente conversacional 3D desarrollado. El *smartphone* Samsung Galaxy I8190 S3 mini se consideró un dispositivo con las prestaciones suficientes para poder realizar las pruebas del nuevo prototipo del agente conversacional en Android. Este *smartphone* se basa en el *chipset* NovaThor U8420, que consta de un procesador ARM Cortex A9 de doble núcleo a 1.0 GHz y un procesador gráfico Mali400. Incorpora de fábrica la versión de Android 4.1 (también llamada JellyBean). La Figura 6.5 muestra el agente conversacional 3D ejecutándose en el teléfono aquí descrito.

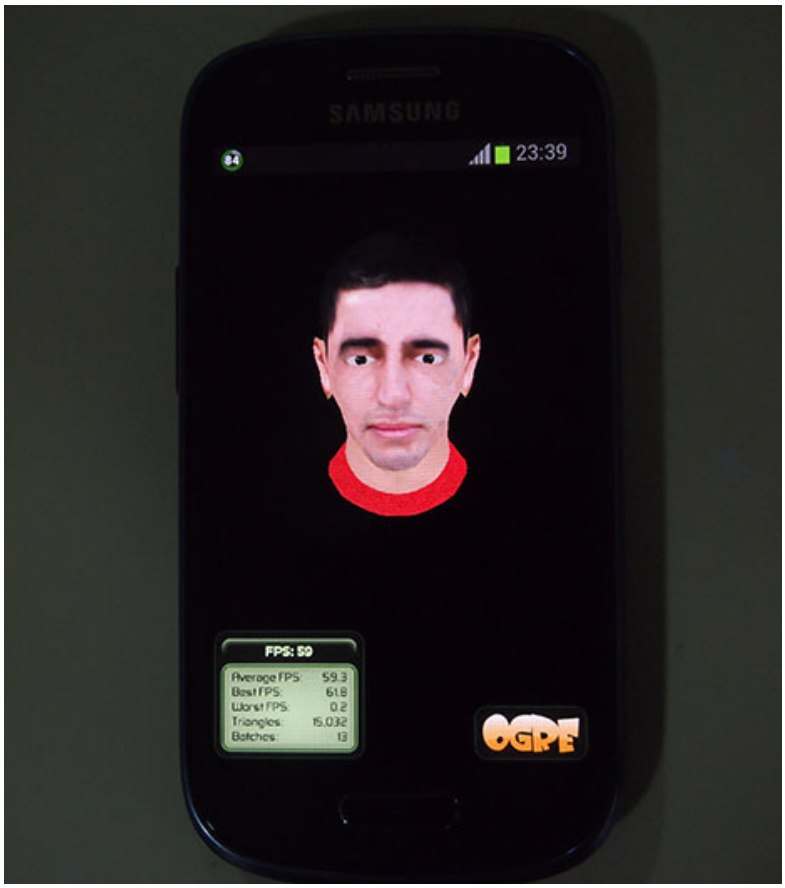


Figura 6.5: Samsung Galaxy S3 mini ejecutando el agente conversacional 3D.

6.7.1.1. Módulos VAD y ASR

El módulo adaptado basado en PocketSphinx y OpenSL_ES se comparó con el sistema de reconocimiento de voz de Android. El ASR de Android se ejecuta normalmente en modo *online*, por lo que se conecta a los servidores de Google donde se realiza el reconocimiento de voz. Además, desde la versión de Android 4.1 también se puede ejecutar *offline* sin necesidad de conexión a la red. A pesar de la doble opción *online/offline*, los usuarios y desarrolladores no pueden adaptar el modelo acústico ni el modelo de lenguaje de esta librería Android. Para el módulo ASR basado en PocketSphinx, se

hizo uso del modelo acústico visto en la Sección 4.3 y de un modelo de lenguaje de dominio limitado creado a partir de unos cientos de transcripciones de usuarios de un sistema de control domótico muy sencillo. Este modelo de lenguaje consta de 76 unigramas, 221 bigramas y 399 trigramas.

6.7.1.2. Módulo CE

Este módulo del agente conversacional 3D es especifico del dominio de aplicación y, por lo tanto, no existe una alternativa disponible con la que establecer una comparación cualitativa. A pesar de ello, también se hicieron medidas de latencia y consumo de energía de este módulo. Los ficheros AIML creados para la aplicación de control domótico consistieron en unas 250 entradas diferentes incluyendo recursiones en el algoritmo de búsqueda.

6.7.1.3. Módulos TTS y VHA

Como se ha visto en el Capítulo 5, el módulo VHA requiere de la información de tiempos de los fonemas de la respuesta sintetizada para poder realizar la sincronización labial de la animación del agente conversacional. Actualmente, no existen alternativas al módulo TTS basado en Flite que cuenten con esta característica. Aún así, para poder comparar el rendimiento del módulo TTS basado en Flite se compararon sus medidas de latencia y consumo de energía con los de otras librerías TTS disponibles en Android; una versión *online* de Nuance y una versión *offline* ofrecida por la propia plataforma Android.

6.7.2. Resultados

Como se menciona en la sección anterior, tanto la latencia como el consumo de energía fueron medidos para cada componente del agente conversacional 3D. Cada medida fue repetida 10 veces para reducir la variabilidad de los resultados y obtener un valor medio preciso. Cuando uno de los componentes funciona en modo *online* y requiere conexión de datos, se obtuvieron medidas de latencia y consumo de energía separadas en función del tipo de conexión de datos seleccionada en el dispositivo: Red de Área Local Inalámbrica - *Wireless Local Area Network* (WLAN), Acceso a Datos de Alta Velocidad - *High*

Tipo/Conexión	Latencia Media
Android/WLAN	190 ms
Android/HSPA	337 ms
Android/EDGE	453 ms
Android/Offline	278 ms
PocketSphinx/Offline	279 ms

Tabla 6.1: Latencia de ASR.

Speed Data Access (HSPA) o Tasas Mejoradas de Datos para la Evolución de GSM - *Enhanced Data Rates for GSM Evolution* (EDGE).

6.7.2.1. Módulos VAD + ASR

Se puede ver en la Tabla 6.1 que el sistema ASR de Android obtiene el valor más bajo de latencia, 190 ms, cuando está conectado a una WLAN. Los mejores valores siguientes de latencia se consiguen con el sistema ASR de Android en modo *offline* y el módulo PocketSphinx con 278 ms y 279 ms respectivamente. Los peores datos de latencia los presenta el sistema ASR de Android con conexiones de tipo HSPA y EDGE, con 337 ms y 453 ms.

En cuanto a las medidas de consumo de energía que se muestran en la Tabla 6.2 el claro vencedor es el módulo PocketSphinx con un valor de tan solo 320 mJ de media. El sistema de Android consume más de 3 veces esa cantidad, 1120 mJ, solamente en modo *offline*, y se incrementa sensiblemente en modo *online* con conexiones WLAN, HSPA o EDGE, obteniendo 1990 mJ, 6570 mJ y 3950 mJ respectivamente.

Los valores de WER se han tomado para 3 posibles escenarios: ASR de Android *online* (WLAN), ASR de Android *offline* y PocketSphinx *offline*. El primero obtiene un valor de WER de 9.1 %; 22.5 % el segundo y tan solo un 2.5 % el tercero. Estos resultados reflejan el problema que supone no poder adaptar el modelo de lenguaje al dominio de aplicación del agente. Si el vocabulario que se necesita reconocer es reducido, se puede obtener una gran mejora mediante el uso de un módulo ASR adaptable como el que ofrece la plataforma optimizada adaptada a Android.

Tipo/Conexión	Cons. Medio de Energía
Android/WLAN	1990 mJ
Android/HSPA	6570 mJ
Android/EDGE	6950 mJ
Android/Offline	1120 mJ
PocketSphinx/Offline	320 mJ

Tabla 6.2: Consumo de Energía de ASR.

Tipo/Conexión	WER Media
Android/WLAN	9.1 %
Android/Offline	22.5 %
PocketSphinx/Offline	2.5 %

Tabla 6.3: Tasa de error por palabra de ASR.

6.7.2.2. Módulo CE

El intérprete PyAIML logra una latencia media de 166ms mientras que su consumo de energía se reduce a 40mJ de media. Estos valores no tienen en cuenta el posible incremento de latencia y consumo cuando se establece la comunicación entre el agente conversacional y el entorno inteligente con el que se comunica.

6.7.2.3. Módulos TTS + VHA

La Tabla 6.4 muestra las latencias medias de los sistemas TTS. La librería TTS de Android obtiene un sorprendente valor de 10 ms, seguido por el módulo basado en Flite TTS con 166 ms. Con valores muy alejados de estos, resultando en latencias de más de 1 s inútiles para aplicaciones interactivas, se encuentra la librería Nuance TTS en cualquiera de los 3 tipos de conexiones de datos disponibles (WLAN, HSPA y EDGE).

Los resultados de consumo de energía de la Tabla 6.5 reflejan que el módulo basado en Flite TTS está a la cabeza en este apartado, con un valor medio de 159 mJ. La librería *offline* de Android TTS obtiene un consumo de

más del doble, 411 mJ, mientras que el caso de la librería Nuance TTS se ve afectada por el consumo extra de la conexión de datos, obteniendo valores de 340 mJ, 3683 mJ y 4663 mJ para cada uno de las 3 modalidades de conexión.

Por su parte, el módulo VHA obtiene un consumo de potencia de 1350 mW con el avatar ejecutándose en primer plano y estableciendo al máximo el brillo de la pantalla. Con el brillo al mínimo su valor de potencia de 0.96 mW.

6.7.2.4. Resultados agregados

Teniendo en cuenta los resultados expuestos anteriormente, se puede concluir que el agente conversacional 3D optimizado y portado a Android obtiene la mejor relación entre latencia y consumo de energía. Este hecho responde a múltiples factores, siendo alguno de los más relevantes la implementación en código nativo de las tareas más complejas y de uso intensivo de CPU y la habilidad para adaptar los componentes que forman el agente conversacional 3D al dominio de aplicación específico requerido por el usuario.

Finalmente, se puede realizar una estimación de la latencia y consumo de energía del agente conversacional al completo, simplemente mediante la suma de los valores medios obtenidos para sus diferentes partes. Para la latencia, este cálculo resulta en la suma de 279 ms + 166 ms + 166 ms, obteniendo una latencia global de 611 ms. En el caso del consumo de energía, la suma análoga es 320 mJ + 40 mJ + 159 mJ, con un valor de consumo global de 519 mJ. Es necesario resaltar que este último valor, el del consumo de energía se verá aumentado si se añade el valor resultante del módulo VHA. A partir de su valor de potencia se puede estimar que para 611 ms su consumo de energía es de 825 mJ con el brillo al máximo y de 587 mJ con el brillo al mínimo. Por lo tanto, sumado a los 519 mJ anteriores del resto de módulos del agente, resulta un valor final máximo de 1344 mJ y mínimo de 1106 mJ por cada turno completo de conversación del agente conversacional 3D.

Tipo/Conexión	Latencia Media
Nuance/WLAN	1379 ms
Nuance/HSPA	1634 ms
Nuance/EDGE	2212 ms
Android/Offline	10 ms
Flite/Offline	166 ms

Tabla 6.4: Latencia de TTS.

Tipo/Conexión	Cons. Medio de Energía
Nuance/WLAN	340 mJ
Nuance/HSPA	3683 mJ
Nuance/EDGE	4663 mJ
Android/Offline	411 mJ
Flite/Offline	159 mJ

Tabla 6.5: Consumo de Energía de TTS.

6.8. Conclusiones

En este Capítulo se describe el proceso de adaptación a Android de la plataforma optimizada para desarrollo de agentes conversacionales 3D. Para ello, se hizo primero una revisión de los aspectos más importantes de dicho SO móvil, siendo los más relevantes su arquitectura interna y su modelo de programación. A continuación, se comentaron las particularidades que conllevó la adaptación de cada uno de los módulos internos del agente, resultando en un nuevo diseño de la plataforma de desarrollo de ECAs. Con el fin de validar el proceso de portabilidad a Android, se ofrecieron y analizaron los resultados obtenidos por la plataforma en parámetros de latencia y consumo de energía ejecutándose en un dispositivo móvil real. También se han comparado estos resultados con los de otras alternativas existentes para algunos de los módulos del agente, pudiéndose constatar la eficiencia que logra la plataforma final en Android.

La decisión de portar la plataforma a Android tiene 2 ventajas muy im-

portantes para garantizar la continuación del trabajo desarrollado en esta tesis. La primera se debe a la gran difusión de Android, lo que permitirá realizar experimentos con ECAs en una elevada variedad de dispositivos. La segunda es que Android expone al desarrollador una gran cantidad de aplicaciones y servicios sobre los que un ECA puede servir de interfaz de acceso y control.

Capítulo 7

Conclusiones y líneas futuras

7.1. Introducción

En este Capítulo se realiza una breve retrospectiva de todas las fases que forman esta tesis doctoral desde que en el Capítulo 1 se plantease la problemática que se pretende resolver. Durante el transcurso de este resumen, se hacen notar las aportaciones más relevantes y cómo éstas modifican el escenario anterior existente. Finalmente, se proponen una serie de líneas futuras de investigación que ayudan a valorar no sólo el trabajo desarrollado, sino también el potencial y calidad de los estudios que pueden llevarse a cabo a posteriori.

7.2. Conclusiones

Esta tesis enmarca su objetivo principal, el diseño de una plataforma para el desarrollo de agentes conversacionales virtuales optimizada para sistemas empotrados, dentro de un escenario tecnológico muy cambiante y en constante evolución. Por esta razón, la capacidad de adaptación del trabajo de investigación y desarrollo que se ha presentado en este documento puede considerarse una dificultad añadida.

El Capítulo 2 estableció el complejo y heterogéneo marco conceptual y algorítmico al que debe estar dispuesto a enfrentarse cualquiera que quiera llegar a considerarse experto en el fascinante ámbito de los ECAs. En él se

realizó una descripción de técnicas específicas para cada una de las partes críticas que forman un agente conversacional virtual, concentrando los mayores niveles de detalle solamente en aquellos aspectos necesarios para la mejor comprensión del trabajo posterior.

Después de contemplar la ingente cantidad de tiempo y esfuerzo que podría suponer llegar a dominar todos los campos que definen un ECA, se antojó necesario establecer las posibles alternativas disponibles en cuanto a la implementación de una plataforma de base que aúne todas las funcionalidades típicas de los ECAs. Con este propósito, en el Capítulo 3 se hizo un estudio de las ventajas e inconvenientes de cada una de las pártes de un ECA, sin olvidar nunca que el objetivo final es el rendimiento óptimo en sistemas empotrados. Todo ello llevó al diseño de una plataforma *software* inicial basada en un conjunto de librerías gratuitas y de código abierto: SphinxBase, PocketSphinx, PyAIML, eSpeak/Mbrola/Festival y Ogre3D. El interés desde un punto de vista aplicado de esta primera versión de la plataforma de desarrollo de ECAs para sistemas empotrados ha quedado recogido en [162].

Una vez obtenida una primera versión de la plataforma de desarrollo de ECAs para sistemas empotrados, el siguiente paso siguiendo la metodología de trabajo definida en el Capítulo 1 consistió en realizar un análisis de rendimiento de la plataforma en un sistema empotrado real. Para ello se eligió la placa *hardware* de desarrollo BeagleBoard como entorno para las pruebas de prestaciones. Sus modestas características, principalmente su procesador Cortex-A8 a 600MHz junto a sus 128MB de memoria dinámica fueron suficientes como para arrojar luz sobre los aspectos críticos de la plataforma de base que podían limitar el desarrollo de ECAs completos en sistemas empotrados.

Los resultados del proceso de análisis identificaron al módulo ASR como posible cuello de botella del agente conversacional 3D en términos de tiempo de respuesta. Con una configuración de complejidad media-baja en cuanto a los modelos acústicos y de lenguaje, y haciendo uso de los parámetros por defecto del sistema ASR, éste obtuvo una latencia de casi 2.5 segundos para procesar comandos simples de usuario de aproximadamente 3 segundos, con una WER de aproximadamente el 6 %. Incluso optimizando los parámetros del sistema ASR para obtener una mayor velocidad a costa de degradar la

WER hasta valores cercanos al 10 %, no se consiguió reducir la latencia a menos 1.65 segundos, siendo este valor todavía muy alto para conseguir un ECA interactivo. Tanto el módulo CE de la plataforma base como el módulo TTS con el uso de eSpeak y Mbrola obtuvieron unos valores de tiempo de respuesta de no más de 100ms para el primero y 200ms para el segundo, valores un orden de magnitud más bajo que el sistema ASR. Además de analizar los valores de latencia de los diferentes módulos, también se vio que el consumo de memoria del módulo CE, cercano a 20MBytes, podía hacer plantearse una mejora para tratar de reducirlo.

El Capítulo 5 se centró exclusivamente en ofrecer soluciones a los problemas encontrados en proceso de análisis de la plataforma base. El conjunto de mejoras expuesto en este Capítulo se añadieron a dicha plataforma base, dando como resultado final una plataforma de desarrollo de ECAs optimizada para sistemas empotrados.

Los mayores esfuerzos de optimización de la plataforma base se centraron en el módulo ASR. Con el objetivo de reducir la latencia del ASR hasta valores que aseguren el mantenimiento de la sensación de interactividad, se propusieron 2 mejoras que siguen enfoques complementarios.

La primera de las optimizaciones que se propuso aprovecha la capacidad de la librería PocketSphinx para poder intercalar la primera fase de reconocimiento con el módulo VAD. Como este modo de funcionamiento puede degradar la tasa de error principalmente por errores en la estimación de los parámetros de la normalización cepstral, se propuso el uso de la medida de confianza final que ofrece PocketSphinx para dar lugar a un esquema VAD+ASR híbrido. Replicando el mismo ejemplo de comandos de usuario de 3 segundos de duración ahora con el esquema híbrido, fijando el umbral de la medida de confianza al 50 %, se consigue pasar de una latencia de 1.65 segundos a un valor de 720ms (un 56 % de reducción con respecto a la plataforma base) con un valor de WER del 7.8 %. Esta aportación se recoge en [163].

Una segunda propuesta de mejora del módulo ASR consistió en realizar una adaptación del modelo de lenguaje del ASR en función del tópico de la conversación que el ECA mantiene con el usuario. El objetivo de esta idea es lograr reducir el tiempo de respuesta y la tasa de error simultáneamente.

De forma interna en la librería PocketSphinx, antes de iniciar las 2 etapas finales de reconocimiento, se realiza una detección de tópico mediante SVM. A partir de ese momento, se selecciona el modelo de lenguaje que mejor representa el tópico del diálogo para el resto del proceso de reconocimiento. Los resultados para un conjunto de datos de entrada repartidos en 3 tópicos arroja una reducción global del xRT de casi un 9 % y de un 4 % en para la WER. Esta aportación se recoge en [164].

El módulo CE también fue objeto de mejoras con los objetivos de mejorar la flexibilidad y reducir el uso de memoria. La introducción de un lematizador en el intérprete PyAIML mejoró su capacidad para comprender los idiomas altamente flexivos como el castellano pero implicó un incremento del uso de memoria. Para contrarrestar este efecto se introdujo también a PyAIML una base de datos orientada a objetos, ZODB. Mediante el uso combinado de ambas funcionalidades extra se consigue un módulo CE con un uso de memoria dinámica inferior a 20MBytes y un tiempo de respuesta acotado a 300-350ms como caso peor. Esta aportación se recoge en [165].

Como última mejora del Capítulo 5 se ha descrito la técnica empleada en módulo VHA para lograr una sincronización labial eficiente y de gran realismo. La idea que hay detrás de esta técnica es la misma que la interpolación lineal clásica de visemas, pero en este caso se realizan de forma $offline$ a modo de animaciones de transiciones entre fonemas. Ahora, la sincronización se logra con una sencilla concatenación de animaciones mediante un simple mapeo y modulación de los visemas que representan la secuencia de foenmas que se van a sintetizar. Esta aportación se recoge en [162].

Debido al reciente posicionamiento de Android como SO dominante en el mercado de dispositivos móviles, se consideró necesario adaptar la plataforma optimizada a este nuevo escenario. En el Capítulo 6 se describió este proceso de adaptación para cada una de las partes que forman el ECA. Esta aportación se ha recogido en [166]. También se ofrecen en este Capítulo los resultados de unos nuevos análisis de latencia y consumo de energía de la plataforma portada a Android corriendo sobre un *smartphone* real. Los resultados de estas pruebas se compararon con los obtenidos mediante otras alternativas disponibles para algunos de los módulos del ECA. Como corolario final, se concluyó que la latencia global de la plataforma en Android es de

611ms, con un consumo de energia en el rango de 1106-1344 mJ, batiendo en este aspecto a todas las alternativas incluidas en el estudio. Esta aportación se encuentra recogida en [167].

Una vez habiendo expuesto las conclusiones de los capítulos que forman esta tesis, es necesario hacer una valoración final sobre el trabajo desarrollado. Para ello es necesario tener en cuenta algunas consideraciones. La primera de ellas es que se partía de un objetivo extremadamente exigente, lograr una plataforma para el desarrollo de agentes conversacionales 3D que pueda ejecutarse completamente en sistemas empotrados. Este objetivo surge al analizar los problemas que presentan las arquitecturas de ECAs distribuida y mixta: latencia, requisitos de ancho de banda y consumo de energía elevados además de la propia dependencia de una conexión de datos. Otro aspecto a resaltar es la dificultad que ante el mismo reto han experimentado otros investigadores. Por todo ello, se puede considerar un éxito la consecución final de una plataforma de desarrollo de ECAs para dispositivos Android que obtiene resultados altamente competitivos en términos de latencia y consumo de energía. En consecuencia, se considera asegurada la continuidad de esta línea de investigación y otras complementarias que se asocian con la temática de los ECAs.

7.3. Líneas de trabajo futuras

A lo largo del desarrollo de esta tesis han surgido algunas ideas para continuar y ampliar los estudios que han sido presentados, y que se relacionan en los siguientes puntos:

- Continuar con la optimización del módulo ASR: La idea de la adaptación del modelo de lenguaje se puede llevar a la primera de las fases de reconocimiento. En este caso, se sustituye el modelo de lenguaje global y estático por un modelo basado en la interpolación de los diferentes modelos de lenguaje dependientes del tópico. De forma dinámica y en tiempo de ejecución se procedería a la actualización de la distribución de pesos siguiendo un algoritmo EM (Expectation-Maximization).
- Desarrollar capacidades emotivas: Aprovechando que el módulo VHA

de la plataforma permite la expresión de estados emocionales se podría integrar esta funcionalidad con la capacidad de los módulos TTS de modular la voz artificial para dotarla de carga emocional.

- Realización de estudios experimentales con usuarios: Sería interesante desarrollar algún ECA que cumpla algún propósito concreto (informador virtual, asistente personal, etc.) que permita validar aspectos subjetivos de la plataforma, como la naturalidad del diálogo, la sensación de naturalidad, el grado de satisfacción, etc.

- Validación de aspectos de la plataforma dependientes de la aplicación: El desarrollo de ECAs que sirvan de interfaz o asistente para alguna aplicación concreta dan lugar al análisis de factores de rendimiento que dependen fuertemente de la aplicación, como por ejemplo: reducción de uso de memoria gracias al uso del lematizador, tasa de compleción de tareas, número de turnos por diálogo, etc.

Capítulo 8

Conclusions and Future Research

8.1. Introduction

This Chapter provides a brief retrospective of all the phases that form this thesis, since it arose in Chapter 1 the problem to be solved. During the course of this summary, the main contributions are presented and how they modify the existing previous scenario. Finally, a number of future lines of research are proposed to help to assess not only the work done, but also the potential and quality of the studies can be carried out *aposteriori*.

8.2. Conclusions

This thesis frames its main objective, the design of a platform for the development of virtual conversational agents optimized for embedded systems within a rapidly changing and evolving technology landscape. For this reason, the adaptability of the research and development that has been presented in this paper can be considered an added difficulty.

Chapter 2 established the complex and heterogeneous conceptual and algorithmic framework that should be ready to face anyone who wants to come to be considered an expert in the fascinating field of ECAs. In it, a description of specific techniques for each of the critical parts that form a

virtual conversational agent was performed, concentrating the highest levels of detail on only those aspects necessary for the better understanding of the later work.

After contemplating the enormous amount of time and effort that could lead to master all fields that define an ECA, it seemed necessary to establish the the available alternative possibilities for the implementation of a base platform that combines all the typical features of ECAs. To this end, a study of the advantages and disadvantages of each of the parts of an ECA was done in Chapter 3, without ever forgetting that the ultimate aim is the optimal performance in embedded systems. This led to the design of an initial software platform based on a set of free and open source libraries: SphinxBase, PocketSphinx, PyAIML, eSpeak/Mbrola, Festival and Ogre3D. The interest from a practical point of view this first version of the development platform for ECAs aimed at embedded systems has been included in [162].

After obtaining a first version of the platform for developing ECAs, the next step following the methodology of work defined in Chapter 1 was to conduct a performance analysis on a real embedded system. The BeagleBoard development board was chosen as the environment for performance testing. His modest features, mainly its Cortex-A8 600MHz processor with 128MB of dynamic memory, were enough to shed light on the critical aspects of the base platform that could limit the development of complete ECAs in embedded systems.

The results of the analysis identified the ASR module as possible bottleneck of the 3D conversational agent in terms of response time. With a medium setting of low complexity in terms of the acoustic and language models, and using the default parameters of the ASR system, it scored a latency of about 2.5 seconds to process simple user commands of about 3 seconds, and a WER of about 6 %. Even optimizing the parameters of the ASR system to get a higher speed at the cost of degrading the WER to values close to 10 %, did not manage to reduce latency within 1.65 seconds, and this is still very high value to get an interactive ECA. Both the CE module of the base platform as the TTS module using eSpeak and Mbrola obtained low values of response time of no more than 100 ms for the first and 200ms for the second, values an order of magnitude smaller than that of the ASR system. In addition to

analyzing the latency values of the different modules, it also was shown that the memory consumption of the CE module, close to 20 MBytes, should be tried to be reduced.

Chapter 5 focused exclusively on providing solutions to problems encountered during the analysis of the base platform. The set of improvements described in this Chapter were added to the base platform, the end result being a platform for developing ECAs optimized for embedded systems.

Most efforts to optimize the base platform focused on the ASR module. Aiming to reduce the latency of the ASR values that ensure the maintenance of a feeling of interactivity, two complementary approaches were proposed.

The first of the proposed optimizations exploits the ability of the PocketSphinx library to interleave the first phase of recognition with the VAD module. As this scheme can degrade the error rate mainly due to errors in the estimation of the parameters of the cepstral normalization, it was proposed the use of the confidence measure that PocketSphinx offers to get a VAD+ASR hybrid scheme. Replicating the same example of user commands of 3 seconds now with the hybrid scheme, and setting the threshold of confidence measure to 50 %, it is feasible to achieve latency of 720 ms instead of the baseline value of 1.65 seconds (56 % of reduction relative to the baseline), being 7.8 % the new WER. This contribution is included in [163].

A second proposal to improve the ASR module consisted of an adaptation of the ASR language model based on the topic of the conversation that the ECA maintains with the user. The objective of this idea is to reduce the response time and error rate simultaneously. Internally, within the PocketSphinx library and before starting the final two stages of recognition, topic detection is performed using SVMs. From that moment, the language model that best represents the topic of the dialog for the rest of the recognition process is selected. The results for a set of input data divided into 3 topics shown an overall reduction of xRT of almost 9 % and 4 % for the WER. This contribution is included in [164].

The CE module was also improved with the objectives of getting a better flexibility and reducing the memory usage. The addition of a lemmatizer to the PyAIML interpreter improved their ability to understand the highly inflected languages like Spanish but implied an increase in memory usage.

To counteract this, an object-oriented database (ZODB) was also added to PyAIML. Through the combined use of both extra functionalities resulted in lower use of dynamic memory, only 20 MBytes, and a response time in the range of 300-350 ms bounded as worst case. This contribution is included in [165].

As a final improvement included in Chapter 5, it was described the technique employed in the VHA module for getting an efficient and highly realistic lip synchronization. The idea behind this technique is the same as the classic linear interpolation of visemes, but in this case the interpolation was performed offline so the visemes are now animations of transitions between phonemes. With this new approach, the synchronization is accomplished with a simple concatenation of animations by a simple mapping and modulation of the visemes representing the phoneme sequence to be synthesized. This contribution is included in [162].

Due to the recent positioning of Android as the dominant operating system in the mobile device market, it was considered necessary the adaptation of the optimized platform to this new environment. In Chapter 6 this adaptation process for each of the parts forming the ECA was described. This contribution has been included in [166] . In this chapter, it were also offered the results of a new analysis of latency and power consumption of the new Android-adapted platform running on a real smartphone. The results of these tests were compared with those obtained by other alternatives available for some of the modules of the ECA. As a final corollary, it was concluded that the overall latency of the Android platform is 611 ms , with a consumption of energy in the range of 1106-1344 mJ, beating in this regard all the alternatives included in the study. This contribution is contained in [167] .

After having discussed the conclusions of the chapters that make up this thesis , it is necessary to make a final assessment of the work done. For this, it is necessary to take into account some considerations. The first one is that this thesis was based on an extremely ambitious target, to achieve a platform for the development of 3D conversational agents that can fully run on embedded systems. This objective arises when analyzing the problems presented by other ECAs architectures (distributed and mixed): latency , bandwidth requirements and high energy consumption in addition to own dependence on

a data connection. Another aspect to highlight is the difficulties that other researchers have experienced when attempting the same challenge. Therefore, it can be considered a success the final achievement of a platform for developing ECAs for Android devices with highly competitive results in terms of latency and energy consumption. Consequently , it is considered secured the continuation of this line of research and other complementary associated with the subject of ECAs .

8.3. Future Research

Throughout the development of this thesis some ideas to continue and expand the studies that have been presented have emerged, which are listed in the following points:

- Continue with the optimization of the ASR module: The idea of adapting the language model can be applied within the first stage of recognition, too. In this case, the static and generic language model is replaced with a model based on the interpolation of the different topic-dependent language models. This new model would be updated at runtime following an EM (Expectation-Maximization) algorithm.

- Develop new emotional skills: Taking advantage of the fact that the VHA module allows the expression of emotional states, it could be joined this functionality with the ability of TTS modules to modulate the artificial voice.

- Conducting experimental studies with users: It would be interesting to develop an ECA that meets a specific purpose (personal assistant, virtual guide, etc.) to validate subjective aspects of the platform as the naturalness of the dialog, the sense of naturalness, the satisfaction, etc..

- Validation of application-dependent aspects of the platform: The development of ECAs that act as an assistant or interface to any specific application leads to the analysis of performance factors which are heavily dependent on the application, such as: memory usage reduction by

using the lemmatizer, task completion rate, number of shifts per dialog, etc..

Bibliografía

[1] S. W. Hamerich, "Towards advanced speech driven navigation systems for cars," in *International Conference on Intelligent Environments*, pp. 247–250, Sept. 2007.

[2] B. Yan, F. Weng, Z. Feng, F. Ratiu, M. Raya, Y. Meng, S. Varges, M. Purver, A. Lien, T. Scheideck, B. Raghunathan, F. Lin, R. Mishra, B. Lathrop, Z. Zhang, H. Bratt, and S. Peters, "A conversational in-car dialog system," in *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, (Rochester, NY, USA), pp. 23–24, Apr. 2007.

[3] H. Soronen, M. Turunen, and H. Hakulinen, "Voice commands in home environment - a consumer survey," in *Proceedings of the 9th Annual Conference of the International Speech Communication Association (INTERSPEECH 2008)*, (Brisbane, Australia), pp. 2078–2081, 2008.

[4] W. Minker, R. López-Cózar, and M. McTear, "The role of spoken language dialogue interaction in intelligent environments," *Journal of Ambient Intelligence and Smart Environments*, vol. 1, pp. 31–36, Jan. 2009.

[5] P. Garrido, A. Sanchez, F. Martinez, S. Baldassarri, E. Cerezo, and F. Seron, "Using 3d virtual agents to improve the autonomy and quality of life of elderly people," in *Ambient Intelligence - Software and Applications*, vol. 219 of *Advances in Intelligent Systems and Computing*, pp. 129–136, Springer International Publishing, 2013.

[6] D. Phan, T. Nguyen, and T. Bui, "A 3d conversational agent for presenting digital information for deaf people," in *Agent Computing and Multi-Agent Systems* (A. Ghose, G. Governatori, and R. Sadananda, eds.), vol. 5044 of *Lecture Notes in Computer Science*, pp. 319–328, Springer Berlin Heidelberg, 2009.

[7] L. Ran, S. Helal, and S. Moore, "Drishti: an integrated indoor/outdoor blind navigation system and service," in *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications. (PerCom 2004)*, pp. 23–30, Mar. 2004.

[8] J. Pineau, A. Atrash, R. Kaplow, and J. Villemure, "On the design and validation of an intelligent powered wheelchair: Lessons from the SmartWheeler project," in *Brain, Body and Machine*, vol. 83 of *Advances in Soft Computing*, pp. 259–268, Springer Berlin / Heidelberg, 2010.

[9] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, ""Your word is my command": Google search by voice: A case study," in *Advances in Speech Recognition*, pp. 61–90, Springer US, 2010.

[10] S. Mulken, E. André, and J. Moller, "The persona effect: How substantial is it?," in *People and Computers XIII* (H. Johnson, L. Nigay, and C. Roast, eds.), pp. 53–66, Springer London, 1998.

[11] "Proyecto AVATAR website." http://www.proyectoavatar.diana.uma.es/.

[12] S. Kopp, L. Gesellensetter, N. C. Kramer, and I. Wachsmuth, "A conversational agent as museum guide: design and evaluation of a real-world application," in *Intelligent Virtual Agents* (T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, and T. Rist, eds.), vol. 3661 of *Lecture Notes in Computer Science*, pp. 329–343, Springer Berlin Heidelberg, 2005.

[13] H. van Welbergen, D. Reidsma, Z. M. Ruttkay, and J. Zwiers, "Elckerlyc - A BML Realizer for continuous, multimodal interaction with

a Virtual Human," *Journal on Multimodal User Interfaces*, vol. 3, pp. 271–284, August 2010.

[14] T. Moir and G. L. Filho, "From science fiction to science fact: A Smart-House interface using speech technology and a photo-realistic avatar," in *15th International Conference on Mechatronics and Machine Vision in Practice (M2VIP 2008).*, pp. 327–333, 2008.

[15] S. Baldassarri, E. Cerezo, and F. Seron, "An open source engine for embodied animated agents," in *Actas del XVII Congreso Español de Informática Gráfica - CEIG 2007*, pp. 91–98, 2007.

[16] R. Malaka, J. Haeussler, and H. Aras, "SmartKom mobile: intelligent ubiquitous user interaction," in *Proceedings of the 9th international conference on Intelligent user interfaces*, (Funchal, Madeira, Portugal), pp. 310–312, ACM, 2004.

[17] M. W. Kadous and C. Sammut, "InCA: a mobile conversational agent," in *PRICAI 2004: Trends in Artificial Intelligence*, pp. 644–653, 2004.

[18] Y. Hayashi and K. Ono, "Embodied conversational agents as peer collaborators: Effects of multiplicity and modality," in *RO-MAN, 2013 IEEE*, pp. 120–125, 2013.

[19] A. Benin, G. R. Leone, and P. Cosi, "A 3d talking head for mobile devices based on unofficial ios webgl support," in *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, (New York, NY, USA), pp. 117–120, ACM, 2012.

[20] H. Lin, J. Jia, X. Wu, and L. Cai, "Talkingandroid: An interactive, multimodal and real-time talking avatar application on mobile phones," in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2013 Asia-Pacific*, pp. 1–4, 2013.

[21] J. Danihelka, R. Hak, L. Kencl, and J. Zara, "3d talking-head interface to voice-interactive services on mobile phones," *IJMHCI*, vol. 3, no. 2, pp. 50–64, 2011.

[22] R. Klaassen, J. Hendrix, D. Reidsma, and H. J. A. op den Akker, "Elckerlyc goes mobile: enabling technology for ecas in mobile applications," in *The Sixth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, Barcelona, Spain*, pp. 41–47, XPS (Xpert Publishing Services), September 2012.

[23] D. Bohus, A. Raux, T. K. Harris, M. Eskenazi, and E. I. Rudnicky, "Olympus: an open-source framework for conversational spoken language interface research," in *HLT-NAACL 2007 workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technology*, 2007.

[24] J. F. Quesada, F. Garcia, E. Sena, J. n. Bernal, and J. G. Amores, "Dialogue management in a home machine environment: linguistic components over an agent architecture.," *Procesamiento del Lenguaje Natural*, vol. 27, pp. 89–96, 2001.

[25] V. C. Hung, A. J. Gonzalez, and R. F. DeMara, "Towards a context-based dialog management layer for expert systems.," in *eKNOW* (A. Kusiak and S. goo Lee, eds.), pp. 60–65, IEEE Computer Society, 2009.

[26] S. Larsson, "Dialogue systems: Simulations or interfaces," in *Dialor'05: Proceedings of the ninth workshop on the semantics and pragmatics of dialogue*, p. 4552, 2005.

[27] J. Edlund, M. Heldner, and J. Gustafson, "Two faces of spoken dialogue systems," in *INTERSPEECH 2006 - ICSLP Satellite Workshop Dialogue on Dialogues: Multidisciplinary Evaluation of Advanced Speech-based Interactive Systems*, 2006.

[28] A. M. Galvao, F. A. Barros, A. M. M. Neves, and G. L. Ramalho, "Persona-aiml: An architecture developing chatterbots with personality," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04, (Washington, DC, USA), pp. 1266–1267, IEEE Computer Society, 2004.

[29] A. M. M. Neves, F. A. Barros, and C. Hodges, "iaiml: a mechanism to treat intentionality in aiml chatterbots.," in *ICTAI*, pp. 225–231, IEEE Computer Society, 2006.

[30] S. Baldassarri, E. Cerezo, and D. Anaya, "Interacción emocional con actores virtuales a través de lenguaje natural," in *Actas de Interacción 2007 - VIII Congreso Internacional de Interacción Persona-Ordenador*, pp. 343–352, 2007.

[31] E. Cerezo, S. Baldassarri, and F. Seron, "Interactive agents for multimodal emotional user interaction," in *Proceedings of IADIS International Conference Interfaces and Human Computer Interaction*, pp. 35–42, 2007.

[32] E. Cerezo, S. Baldassarri, E. Cuartero, and F. Seron, "Agentes virtuales 3d para el control de entornos inteligentes domóticos," in *Actas de Interacción 2007 - VIII Congreso Internacional de Interacción Persona-Ordenador*, pp. 363–372, 2007.

[33] F. Beritelli, S. Casale, and A. Cavallaero, "A robust voice activity detector for wireless communications using soft computing," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 9, pp. 1818–1829, 1998.

[34] K. Li, M. Swamy, and M. Ahmad, "An improved voice activity detection using higher order statistics," *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 5, pp. 965–974, 2005.

[35] R. Le Bouquin-Jeannès and G. Faucon, "Study of a voice activity detector and its influence on a noise reduction system," *Speech Communication*, vol. 16, pp. 245–254, Apr. 1995.

[36] J. Ramirez, J. M. Gorriz, and J. C. Segura, *Voice Activity Detection. Fundamentals and Speech Recognition System Robustness*. InTech, June 2007.

[37] B. Atal and L. Rabiner, "A pattern recognition approach to voiced-unvoiced-silence classification with applications to speech recognition,"

IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 24, no. 3, pp. 201–212, 1976.

[38] A. M. Noll, "Cepstrum pitch determination," *The Journal of the Acoustical Society of America*, vol. 41, no. 2, 1967.

[39] J. A. Haigh and J. Mason, "Robust voice activity detection using cepstral features," in *IEEE 10th Conference on Computer, Communication, Control and Power Engineering (TENCON 93)*, no. 0, pp. 321–324 vol.3, 1993.

[40] J. Stegmann and G. Schroder, "Robust voice-activity detection based on the wavelet transform," in *Proceedings of the IEEE Workshop on Speech Coding For Telecommunications*, pp. 99–100, 1997.

[41] R. Tucker, "Voice activity detection using a periodicity measure," *Communications, Speech and Vision, IEEE Proceedings I*, vol. 139, no. 4, pp. 377–380, 1992.

[42] J. Sohn, N. S. Kim, and W. Sung, "A statistical model-based voice activity detection," *Signal Processing Letters, IEEE*, vol. 6, no. 1, pp. 1–3, 1999.

[43] D. Tran, *Fuzzy Approaches to Speech and Speaker Recognition*. University of Canberra, 2000.

[44] K. Jokinen, *Natural Language and Dialogue Interfaces*, ch. 31, pp. 495–506. CRC Press Taylor & Francis Group, 2009.

[45] J. Zhang, W. Ward, B. L. Pellom, X. Yu, and K. Hacioglu, "Improvements in audio processing and language modeling in the cu communicator.," in *INTERSPEECH* (P. Dalsgaard, B. Lindberg, H. Benner, and Z.-H. Tan, eds.), pp. 2209–2212, ISCA, 2001.

[46] A. Schmitt, D. Zaykovskiy, and W. Minker, "Speech Recognition for Mobile Devices," *International Journal of Speech Technology*, vol. 11, no. 2, pp. 63–72, 2009.

[47] Z.-H. Tan and B. Lindberg, "Speech recognition on mobile devices," in *Mobile Multimedia Processing* (X. Jiang, M. Ma, and C. Chen, eds.), vol. 5960 of *Lecture Notes in Computer Science*, pp. 221–237, Springer Berlin Heidelberg, 2010.

[48] B. Logan *et al.*, "Mel frequency cepstral coefficients for music modeling.," in *ISMIR*, 2000.

[49] M. Gales and S. Young, "The application of hidden markov models in speech recognition," in *Foundations and Trends in Signal Processing*, p. 2007.

[50] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–393, 1999.

[51] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. Prentice Hall, 2 ed., May 2008.

[52] K. Jokinen, "Natural interaction in spoken dialogue systems," in *Proceedings of the Workshop on Ontologies and Multilinguality in User Interfaces*, 2003.

[53] R. López-Cózar, A. Rubio, P. García, J. Díaz, and J. López, "Sistema telefónico de información a viajeros," in *Jornadas en Tecnologías del Habla*, pp. 1–4, 2000.

[54] D. G. Barres, *Desarrollo y evaluación de diferentes metodologías para la gestión automática del diálogo*. Tesis doctoral en informática, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2008.

[55] S. Kowalski, K. Pavlovska, and M. Goldstein, "Two case studies in using chatbots for security training," in *World Conference on Information Security Education*, pp. 265–272, 2009.

[56] J. C. Burguillo-Rial, D. A. Rodríguez-Silva, and M. Santos-Pérez, "T-Bot: an intelligent tutoring agent for open e-Learning platforms," in

8th International Conference on Information Technology Based Higher Education and Training (ITHET2007), (Kumamoto City (Japan)), July 2007.

[57] J. Weizenbaum, "ELIZA: computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[58] J. Morie, E. Chance, K. Haynes, and D. Rajpurohit, "Embodied conversational agent avatars in virtual worlds: Making today's immersive environments more responsive to participants," in *Believable Bots: Can Computers Play Like People?*, 2012.

[59] S. Ghose and J. Barua, "Toward the implementation of a topic specific dialogue based natural language chatbot as an undergraduate advisor," in *International Conference on Informatics, Electronics Vision (ICIEV 2013)*, pp. 1–5, 2013.

[60] L. Rabiner, "Applications of voice processing to telecommunications," *Proceedings of the IEEE*, vol. 82, no. 2, pp. 199–228, 1994.

[61] J. P. Van Santen, *Progress in speech synthesis*. Springer, 1997.

[62] H. Hon, A. Acero, X. Huang, J. Liu, and M. Plumpe, "Automatic generation of synthesis units for trainable text-to-speech systems," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 98*, 1998.

[63] J. Allen, M. S. Hunnicutt, D. H. Klatt, R. C. Armstrong, and D. B. Pisoni, *From Text to Speech: The MITalk System*. New York, NY, USA: Cambridge University Press, 1987.

[64] E. Keller, ed., *Fundamentals of Speech Synthesis and Speech Recognition: Basic Concepts, State-of-the-art and Future Challenges*. Chichester, UK: John Wiley and Sons Ltd., 1994.

[65] D. Klatt, "Review of text-to-speech conversation for english," *Journal of the Acoustical Society of America*, vol. 82, 1987.

[66] B. Kröger, "Minimal rules for articulatory speech synthesis," in *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pp. 331–334, 1992.

[67] P. Taylor, A. W. Black, and R. Caley, "The architecture of the festival speech synthesis system," in *The Third ESCA Workshop in Speech Synthesis*, pp. 147–151, 1998.

[68] E. Moulines and F. Charpentier, "Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones," *Speech Communication*, vol. 9, no. 5–6, pp. 453 – 467, 1990. Neuropeech '89.

[69] A. Hunt and A. Black, "Unit selection in a concatenative speech synthesis system using a large speech database," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 96)*, vol. 1, pp. 373–376 vol. 1, 1996.

[70] L. Gauvain, L. F. Lamel, J. L. Gauvain, B. Prouts, C. Bouhier, and R. Boesch, "Generation and synthesis of broadcast messages," in *Proc. ESCA-NATO Workshop on Applications of Speech Technology, Lautrach*, pp. 207–210, 1993.

[71] A. Black, "Perfect synthesis for all of the people all of the time," in *Proceedings of the IEEE Workshop on Speech Synthesis*, pp. 167–170, 2002.

[72] J. Wouters and M. W. Macon, "Unit fusion for concatenative speech synthesis," in *in ICSLP, 2000*, pp. 302–305, 2000.

[73] H. Zen, T. Nose, J. Yamagishi, S. Sako, T. Masuko, A. Black, and K. Tokuda, "The hmm-based speech synthesis system version 2.0," in *Proceedings of the Sixth ISCA Workshop on Speech Synthesis (ISCA SSW6)*, pp. 294–299, 2007.

[74] N. Ersotelos and F. Dong, "Building highly realistic facial modeling and animation: a survey," *The Visual Computer*, vol. 24, no. 1, pp. 13–30, 2008.

[75] M. Mori, "The uncanny valley," *Energy*, vol. 7, no. 4, pp. 33–35, 1970.

[76] K. Liu and J. Ostermann, "Realistic facial animation system for interactive services," in *Interspeech 2008 in the special session: LIPS 2008: Visual Speech Synthesis Challenge*, 2008.

[77] L. Wang, X. Qian, W. Han, and F. K. Soong, "Synthesizing photoreal talking head via trajectory-guided sample selection.," in *INTERSPEECH* (T. Kobayashi, K. Hirose, and S. Nakamura, eds.), pp. 446–449, ISCA, 2010.

[78] Q. Zhang, Z. Liu, B. Guo, D. Terzopoulos, and H.-Y. Shum, "Geometry-driven photorealistic facial expression synthesis.," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 12, no. 1, pp. 48–60, 2006.

[79] F. H. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. Salesin, "Synthesizing realistic facial expressions from photographs.," in *SIGGRAPH*, pp. 75–84, 1998.

[80] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3d faces.," in *SIGGRAPH*, pp. 187–194, 1999.

[81] "CMU Sphinxbase website."
http://sourceforge.net/projects/cmusphinx/sphinxbase/.

[82] M. A. H. Huijbregts, *Segmentation, diarization and speech transcription : surprise data unraveled.* PhD thesis, Enschede, November 2008.

[83] "SHoUT website."
http://shout-toolkit.sourceforge.net/.

[84] J.-M. Valin, "Speex: a free codec for free speech," in *Australian National Linux Conference, Dunedin, New Zealand*, Citeseer, 2006.

[85] "Speex website."
http://www.speex.org/.

[86] P. C. Woodland, J. J. Odell, V. Valtchev, and S. J. Young, "Large vocabulary continuous speech recognition using htk," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 94)*, vol. 2, pp. II–125, IEEE, 1994.

[87] "HTK website."
http://htk.eng.cam.ac.uk/.

[88] K.-F. Lee, *Automatic Speech Recognition: The Development of the Sphinx Recognition System*, vol. 62. Springer, 1989.

[89] X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, K.-F. Lee, and R. Rosenfeld, "The sphinx-ii speech recognition system: an overview," *Computer Speech & Language*, vol. 7, no. 2, pp. 137–148, 1993.

[90] "CMU Sphinx 2 website."
http://sourceforge.net/projects/cmusphinx/files/sphinx2/.

[91] M. Seltzer, "Sphinx iii signal processing front end specification," *CMU Speech Group*, 1999.

[92] "CMU Sphinx 3 website."
http://sourceforge.net/projects/cmusphinx/files/sphinx3/.

[93] "CMU SphinxTrain website."
http://sourceforge.net/projects/cmusphinx/files/sphinxtrain/.

[94] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: A flexible open source framework for speech recognition," 2004.

[95] "CMU Sphinx 4 website."
http://sourceforge.net/projects/cmusphinx/files/sphinx4/.

[96] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky, "Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2006)*, vol. 1, pp. I–I, IEEE, 2006.

[97] "Galaxy Communicator website."
http://sourceforge.net/projects/communicator/.

[98] S. Seneff, E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "Galaxy-ii: a reference architecture for conversational system development.," in *ICSLP*, vol. 98, pp. 931–934, 1998.

[99] M. A. Walker, L. Hirschman, and J. S. Aberdeen, "Evaluation for darpa communicator spoken dialogue systems.," in *LREC*, 2000.

[100] "Olympus/Ravenclaw website."
http://wiki.speech.cs.cmu.edu/olympus/index.php/Olympus.

[101] D. Bohus, A. Raux, T. K. Harris, M. Eskenazi, and A. I. Rudnicky, "Olympus: an open-source framework for conversational spoken language interface research," in *Proceedings of the workshop on bridging the gap: Academic and industrial research in dialog technologies*, pp. 32–39, Association for Computational Linguistics, 2007.

[102] D. Bohus and A. I. Rudnicky, "The ravenclaw dialog management framework: Architecture and systems," *Computer Speech & Language*, vol. 23, no. 3, pp. 332–361, 2009.

[103] M. Turunen and J. Hakulinen, "Jaspis$\hat{2}$ – an architecture for supporting distributed spoken dialogues," in *Eurospeech 2003*, pp. 1913–1916, 2003.

[104] "Jaspis website."
http://www.cs.uta.fi/ tko/Tutkimus/hci/spi/Jaspis/.

[105] R. Wallace *et al.*, "Aiml: Artificial intelligence markup language," *DOI= http://www. alicebot. org/TR/2005/WD-aiml*, 2005.

[106] L. Badino, C. Barolo, and S. Quazza, "Language independent phoneme mapping for foreign tts," in *Fifth ISCA Workshop on Speech Synthesis*, 2004.

[107] E. M. Navarro Albaladejo, "Aplicación IVR (Interactive Voice Response) para la consulta telefónica de notas con síntesis texto-voz en tiempo real," 2008.

[108] A. W. Black and K. A. Lenzo, "Flite: a small fast run-time synthesis engine," in *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, 2001.

[109] "eSpeak website."
http://espeak.sourceforge.net/.

[110] T. Dutoit, V. Pagel, N. Pierret, F. Bataille, and O. Van der Vrecken, "The mbrola project: Towards a set of high quality speech synthesizers free of use for non commercial purposes," in *Fourth International Conference on Spoken Language (ICSLP 96)*, vol. 3, pp. 1393–1396, IEEE, 1996.

[111] "The MBROLA Project website."
http://tcts.fpms.ac.be/synthesis/.

[112] A. Black, P. Taylor, R. Caley, R. Clark, K. Richmond, S. King, V. Strom, and H. Zen, "The festival speech synthesis system, version 1.4," *Unpublished document available via http://www.cstr.ed.ac.uk/projects/festival/*, 2001.

[113] "Festival website."
http://www.cstr.ed.ac.uk/projects/festival/.

[114] A. W. Black and K. A. Lenzo, "Flite: a small fast run-time synthesis engine," in *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, 2001.

[115] "Flite website."
http://www.festvox.org/flite/.

[116] N. Gebhardt *et al.*, "Irrlicht engine," 2010.

[117] "Irrlicht website."
http://irrlicht.sourceforge.net/.

[118] F. Kerger, *Ogre 3D 1.7 Beginner's Guide: Create Real Time 3D Applications Using Ogre 3D from Scratch.* Packt Publishing, 2010.

[119] "Ogre3D website."
http://www.ogre3d.org/.

[120] K. Vertanen, "Baseline wsj acoustic models for htk and sphinx: Training recipes and recognition experiments," *Cavendish Laboratory, University of Cambridge*, 2006.

[121] "PyAIML website."
http://pyaiml.sourceforge.net/.

[122] "VoxForge website."
http://www.voxforge.org/.

[123] K. Vertanen and P. O. Kristensson, "Parakeet: A continuous speech recognition system for mobile touch-screen devices," in *Proceedings of the 14th international conference on Intelligent user interfaces*, pp. 237–246, ACM, 2009.

[124] "PocketSphinx optimizations for embedded devices website."
http://cmusphinx.sourceforge.net/wiki/pocketsphinxhandhelds.

[125] K. Demuynck, J. Duchateau, and D. Van Compernolle, "Reduced semi-continuous models for large vocabulary continuous speech recognition in dutch," in *Fourth International Conference on Spoken Language (ICSLP 96)*, vol. 4, pp. 2289–2292, IEEE, 1996.

[126] "Pocketsphinx optimizations for embedded devices."
http://cmusphinx.sourceforge.net/wiki/pocketsphinxhandhelds.

[127] "Proyecto Galaia website."
http://papa.det.uvigo.es/~galaia/ES/.

[128] H. Jiang, "Confidence measures for speech recognition: A survey," *Speech Communication*, vol. 45, no. 4, pp. 455–470, 2005.

[129] I. R. Lane, T. Kawahara, T. Matsui, and S. Nakamura, "Dialogue speech recognition by combining hierarchical topic classification and language model switching," *IEICE Transactions on Information and Systems*, vol. E88-D, pp. 446–454, March 2005.

[130] B. Ballinger, C. Allauzen, A. Gruenstein, and J. Schalkwyk, "On-demand language model interpolation for mobile speech input," in *Proceedings of Interspeech* (T. Kobayashi, K. Hirose, and S. Nakamura, eds.), pp. 1812–1815, ISCA, 2010.

[131] M. Ravishankar, *Efficient Algorithms for Speech Recognition*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1996. Available as tech report CMU-CS-96-143.

[132] V. N. Vapnik, *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[133] A. W. Black and K. A. Lenzo, "Limited domain synthesis," tech. rep., DTIC Document, 2000.

[134] A. I. Rudnicky, C. Bennett, A. W. Black, A. Chotomongcol, K. Lenzo, A. Oh, and R. Singh, "Task and domain specific modelling in the carnegie mellon communicator system," tech. rep., DTIC Document, 2000.

[135] P. Price, W. M. Fisher, J. Bernstein, and D. S. Pallett, "The darpa 1000-word resource management database for continuous speech recognition," in *International Conference on Acoustics, Speech, and Signal Processing (ICASSP 88)*, pp. 651–654, IEEE, 1988.

[136] B.-J. Hsu and J. Glass, "Iterative language model estimation: efficient data structure & algorithms," in *Proceedings of Interspeech*, vol. 8, pp. 1–4, 2008.

[137] "MITLM website." http://code.google.com/p/mitlm/.

[138] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[139] "LibSVM website."
http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[140] P. Ponmuthuramalingam and T. Devi, "Effective dimension reduction techniques for text documents," *International Journal of Computer Science and Network Security*, vol. 10, no. 7, pp. 101–109, 2010.

[141] J. Vilares, F. M. Barcala, and M. A. Alonso, "Using syntatic Dependency-Pairs conflation to improve retrieval performance in spanish," in *Computational Linguistics and Intelligent Text Processing* (A. Gelbukh, ed.), vol. 2276 of *Lecture Notes in Computer Science*, pp. 381–390, Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[142] L. Padró, M. Collado, S. Reese, M. Lloberes, I. Castellón, *et al.*, "Freeling 2.1: Five years of open-source language processing tools," 2012.

[143] "FreeLing website."
http://nlp.lsi.upc.edu/freeling/.

[144] J. Fulton, "Introduction to the zope object database," in *Proceedings of the 8th International Python Conference*, 2000.

[145] "ZODB website."
http://www.zodb.org/en/latest/.

[146] "Persistency," in *Web Component Development with Zope 3*, pp. 83–94, Springer, 2007.

[147] D. Comer, "Ubiquitous b-tree," *ACM Computing Surveys (CSUR)*, vol. 11, no. 2, pp. 121–137, 1979.

[148] R. Hess, *The essential Blender: guide to 3D creation with the open source suite Blender*. No Starch Press, 2007.

[149] "IHMAN website."
http://www.ihman.com/.

[150] L. Li, Y. Liu, and H. Zhang, "A survey of computer facial animation techniques," in *International Conference on Computer Science and Electronics Engineering (ICCSEE 2012)*, vol. 3, pp. 434–438, IEEE, 2012.

[151] T. Ezzat and T. Poggio, "Visual speech synthesis by morphing visemes," *International Journal of Computer Vision*, vol. 38, no. 1, pp. 45–57, 2000.

[152] Z. Deng and J. Noh, "Computer facial animation: A survey," in *Data-Driven 3D Facial Animation*, pp. 1–28, Springer, 2007.

[153] T. Bradley, "Android dominates market share, but apple makes all the money," 2013.

[154] "Android SDK website."
http://developer.android.com/sdk/index.html.

[155] "Android NDK website."
http://developer.android.com/tools/sdk/ndk/index.html.

[156] "OpenSL ES website."
http://www.khronos.org/opensles/.

[157] Y.-H. Lee, P. Chandrian, and B. Li, "Efficient java native interface for android based mobile devices," in *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2011)*, pp. 1202–1209, IEEE, 2011.

[158] "SL4A website."
http://code.google.com/p/android-scripting/.

[159] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*, pp. 69–72, Association for Computational Linguistics, 2006.

[160] "Python NLTK website."
http://nltk.org/.

[161] Z. Yang, "PowerTutor A Power Monitor for Android-Based Mobile Platforms," *EECS, University of Michigan*, vol. 2, 2012.

[162] M. Santos-Pérez, E. González-Parada, and J. M. Cano-García, "Avatar: An open source architecture for embodied conversational agents in smart environments," in *Ambient Assisted Living*, vol. 6693 of *Lecture Notes in Computer Science*, pp. 109–115, Springer Berlin Heidelberg, 2011.

[163] M. Santos-Pérez, E. González-Parada, and J. M. Cano-García, "Efficient use of voice activity detector and automatic speech recognition in embedded platforms for natural language interaction," in *Highlights in Practical Applications of Agents and Multiagent Systems*, vol. 89 of *Advances in Intelligent and Soft Computing*, pp. 233–242, Springer Berlin Heidelberg, 2011.

[164] M. Santos-Pérez, E. González-Parada, and J. M. Cano-García, "Topic-dependent language model switching for embedded automatic speech recognition," in *Ambient Intelligence - Software and Applications*, vol. 153 of *Advances in Intelligent and Soft Computing*, pp. 235–242, Springer Berlin Heidelberg, 2012.

[165] M. Santos-Pérez, E. González-Parada, and J. M. Cano-García, "Embedded conversational engine for natural language interaction in spanish," in *Proceedings of the Paralinguistic Information and its Integration in Spoken Dialogue Systems Workshop*, pp. 365–374, Springer New York, 2011.

[166] M. Santos-Pérez, E. González-Parada, and J. Cano-García, "ECA-based control interface on Android for home automation system," in *IEEE International Conference on Consumer Electronics (ICCE 2013)*, pp. 70–71, 2013.

[167] M. Santos-Pérez, E. González-Parada, and J. M. Cano-García, "Mobile embodied conversational agent for task specific applications," *IEEE*

Transactions on Consumer Electronics, vol. 59, no. 3, pp. 610–614, 2013.