Universitat Autònoma
de Barcelona

Simulador de domótica con interfaz vocal

Memoria del proyecto
de Ingeniería Técnica en
Informática de Sistemas
realizado por
Ander Welton Rodríguez
y dirigido por
Javier Serrano García

Escola Universitària d'Informàtica
Sabadell, Septiembre de 2009

El/la sotasignant, *Javier Serrano García*,

professor/a de l'Escola Universitària d'Informàtica de la UAB,

CERTIFICA:

Que el treball al que correspon la present memòria
ha estat realitzat sota la seva direcció
per en *Ander Welton Rodríguez*

I per a que consti firma la present.
Sabadell, Setembre de 2009

Signat: "*Nom i cognoms director/a*"

Resumen del proyecto

El proyecto del cual se presenta la memoria y que tiene como titulo *"Simulador de domótica con interfaz vocal"* pretende servir de ayuda para adquirir unos conocimientos básicos en el campo del reconocimiento de voz.

Para conseguir este objetivo se ha seleccionado el campo de la domótica. Por domótica entendemos, un conjunto de sistemas capaces de automatizar una vivienda y que aportan distintos servicios de gestión del hogar. Una buena manera de automatizar una vivienda consiste en diseñar un sistema que sea capaz de reconocer una serie de comandos únicamente teniendo como entrada la señal acústica emitida por el usuario.

Introduciendo este sistema, el usuario sería capaz de realizar una serie de tareas sin necesidad de efectuar acción alguna, más que la de pronunciar la orden concreta que se desea ejecutar.

En el caso de este proyecto se ha optado por realizar un entorno de simulación 3D (para poder presentar los resultados del reconocimiento de voz), en el que se recrea una vivienda y el movimiento de un usuario. Los resultados obtenidos de la ejecución de órdenes de voz se pueden apreciar en el propio simulador, viendo en pantalla el resultado que producirían en una casa inteligente con el mismo sistema.

Capítulo 1

Introducción

1.1 Presentación

El uso de los sistemas de reconocimiento de voz, aunque todavía en estudio y en fase de perfeccionamiento esta tendiendo a crecer cada vez más en nuestro entorno. Podemos encontrar aplicaciones que hacen uso de esta tecnología en dispositivos móviles, como la marcación por voz de los teléfonos, o las centralitas telefónicas en las que se efectúan una serie de preguntas de manera automatizada antes de redirigir al usuario al departamento que le dará el soporte adecuado.

Las aplicaciones de estos sistemas de reconocimiento de voz pueden ser muy extensas y a su vez pueden resultar más o menos complicadas en función de las necesidades de reconocimiento. Como se puede suponer no es lo mismo tratar de reconocer una palabra o frase concreta que vaya asociada a una acción preestablecida, o ser capaces de reconocer cualquier palabra del diccionario (para aplicaciones de dictado, por ejemplo).

Con el desarrollo de este proyecto se pretende realizar un sistema que sea capaz de reconocer una serie de órdenes predefinidas y ejecutar una serie de acciones en función de las palabras que se han logrado reconocer, entrando así de manera "básica" en el campo del reconocimiento de voz, de manera que se obtenga una base sobre la que poder partir en un futuro a la hora de diseñar aplicaciones con interfaz vocal.

El proyecto dispone de dos partes diferenciadas. La primera de ellas, y la más importante, se encarga del reconocimiento de voz del usuario. Para lograr reconocer las instrucciones del usuario se emplearán herramientas de reconocimiento del habla de código abierto. La segunda parte del proyecto viene dada por el desarrollo del simulador. Para el desarrollo del simulador se emplearán las librerías gráficas openGL

con el fin de desarrollar un entorno 3D sobre el que sea posible moverse y ver los resultados que se obtienen al procesar las ordenes de voz. El simulador 3D es una manera de mostrar los resultados obtenidos y poder así, apreciar el funcionamiento del sistema reconocedor de voz.

1.2 Motivación

Nos encontramos en un momento en el que parece que las interfases de usuario se han quedado congeladas y que estas no avanzan como lo hace el resto de la tecnología. Al menos, no lo hacen tanto a nivel de interactividad como lo hacen a nivel de visualización y de amigabilidad con el usuario. A día de hoy continuamos utilizando el teclado y el ratón para introducir datos en prácticamente cualquier dispositivo. También hay que decir, como se comentaba en el prólogo de la memoria, que cada día más nos encontramos con interfases de usuario que tratan de utilizar la voz como medio de entrada para tratar con el usuario. Estas interfases las podemos encontrar tanto en dispositivos móviles, como las comentadas anteriormente, como en aplicaciones de escritorio o aplicaciones Web. Un ejemplo de estas últimas es el caso de Google y su "*Google Audio Indexing*", servicio que permite realizar búsquedas sobre el audio contenido en un video. Así pues, el uso de las tecnologías de reconocimiento del habla dotaría a los sistemas que actualmente no los incorporan de un mayor grado de interacción con el usuario, mayor velocidad en su uso y muy posiblemente una reducción drástica en el número de errores en algunas tareas (como la escritura de textos a través de teclado).

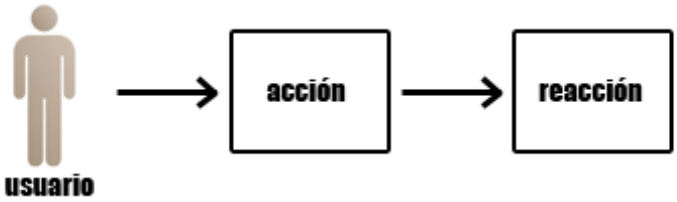
El habla, en entornos que incorporan sistemas domóticos, mejora drásticamente la interfaz con el usuario. Las órdenes vocales son procesadas por una serie de procesadores que se pueden encontrar por toda la superficie de la casa y el ordenador central al que van conectados, se encuentra alejado de la zona donde se utiliza el sistema. La gran cantidad de procesadores que podemos tener distribuidos hace inviable una solución de ratón y teclado para controlar la interfaz, por lo que se sustituyen por interfases vocales.

El hecho de que el reconocimiento de voz sea una tecnología emergente y que probablemente sea de un uso cada vez mayor, ligado al gran número de aplicaciones que esta puede tener y la curiosidad por aprender como se construye una aplicación con una nueva tecnología han hecho que me decantará por este proyecto. De igual forma ha influido el tener que aprender algo nuevo antes de aplicar a un proyecto algo que ya conocía o con lo que ya había trabajado anteriormente.

1.3 Descripción de la situación a tratar

Aunque en nuestro caso se tratará únicamente de un simulador, si consideramos el caso de una vivienda cualquiera, observaremos que todas las tareas se realizan manualmente. Es el propio usuario el que se pone en pie para acercarse hasta el interruptor y accionarlo para así encender la luz.

Podemos considerar que el sistema actual estaría representado por el siguiente gráfico:



Como se ha especificado, introduciendo un sistema de reconocimiento de voz se busca la facilidad de uso para el usuario. En este caso, la facilidad de uso viene dada por una reducción del esfuerzo. Introduciendo un sistema reconocedor de voz, nuestro sistema quedaría representado por el siguiente gráfico:

En este caso, como podemos observar, es el propio sistema reconocedor de voz quien al

reconocer la orden comunicada por el emisor se encargaría de realizar la acción, obteniendo automáticamente la reacción asociada.

La funcionalidad principal de nuestro sistema será la de ser capaz de obtener una entrada en forma de audio, analizar esta entrada y en función del resultado ejecutar la acción esperada. En este caso, el efecto no se verá sobre una vivienda real, sino sobre un simulador. En cualquier caso, montar el sistema sobre una vivienda real no representaría mayor problema que el de montar un sistema de control automático. Dispondríamos de una serie de estaciones comunicadas con el ordenador central encargado de controlar el sistema domótico. Al capturar una entrada sonora en una de estas estaciones, la señal sería enviada al ordenador central, donde posterior decodificación e interpretación se enviarían las ordenes adecuadas a los pequeños procesadores encargados de controlar los elementos del hogar, como las luces, puertas, ventanas, etc.

1.4 Objetivos generales

El objetivo principal del proyecto es el de reconocer una serie de órdenes previamente establecidas. Estas órdenes se deben reconocer a partir de la señal obtenida por un periférico de entrada de audio y utilizando técnicas de reconocimiento del habla de código abierto. Una vez se ha reconocido la orden a partir de la señal acústica de entrada se realizará la acción correspondiente en el simulador. El simulador nos permitirá saber si la orden se ha ejecutado de manera correcta o si por el contrario no ha sido posible reconocerla o no pertenece al conjunto de órdenes del sistema, es decir, nos dará una representación visual de la acción ejecutada.

Así pues, la lista de objetivos concretos que se plantean en este proyecto es:

- Creación de la gramática asociada a nuestro lenguaje.
- Desarrollo del simulador.
- Entrenar los patrones de voz de aquellas órdenes que se pueden reconocer.
- Obtener la señal acústica a través de un dispositivo de entrada. (micrófono)

- Tratar la señal y reconocer la orden.
- Mostrar la acción correspondiente en el simulador.

La descripción de cada uno de estos objetivos se podrá ver con más detalle en el capítulo de análisis de requerimientos, tratado más adelante en esta memoria.

1.5 Planificación Temporal del Proyecto

Para planificar el proyecto y llevarlo a cabo de una manera satisfactoria, este se ha dividido en dos tareas, cada una de ellas con una serie de subtareas asociadas. Cada una de estas tareas tiene una dificultad menor de ser realizada que el conjunto total del proyecto. Las dos tareas principales son la de Reconocimiento de Voz y la de Simulador 3D. Cada una de ellas engloba todas las subtareas que componen el desarrollo final de cada uno de los módulos. Gracias a la especificación de tareas se ha podido realizar una planificación temporal del proyecto, asignando a cada tarea un número de horas. En la siguiente tabla se puede observar la lista de tareas que dan forma al proyecto.

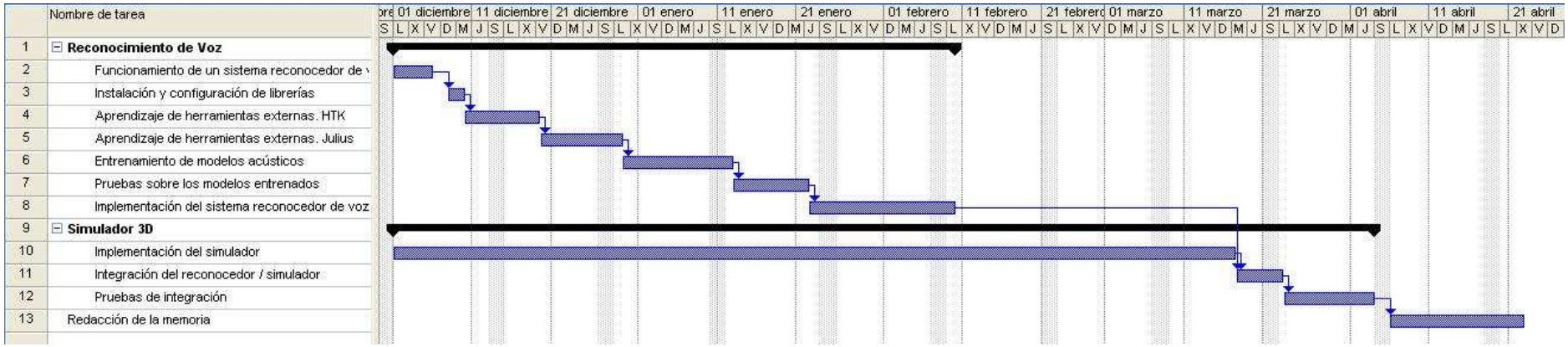
Num. de tarea	Tarea	Carga de horas
1	Como funciona un sistema reconocedor de voz. (Lectura)	10 h
2	Instalación y Configuración de librerías	2 h
3	Aprendizaje de Herramientas externas. HTK.	15 h
4	Aprendizaje de Herramientas externas. Julius	15 h
5	Entrenamiento de los modelos acústicos	20 h
6	Pruebas de reconocimiento sobre los modelos entrenados	10 h
7	Implementación del sistema reconocedor de voz.	30 h
8	Implementación del simulador	50 h
9	Integración del reconocedor/simulador.	15 h
10	Pruebas de Integración	10 h

11	Redacción de la memoria	30 h
Total		207 h

Como se observa en el diagrama de Gantt que se encuentra a continuación, se ha seguido una estrategia de desarrollo lineal para realizar las diferentes tareas asignadas. Debido a la naturaleza del proyecto, que es basado en investigación y desarrollo, se ha optado por esta estrategia para poder seguir un orden coherente durante el desarrollo y no avanzarse en el proyecto con tareas que pueden tener relación con otras marcadas como previas. Para poder llevar a cabo el desarrollo era necesario aprender el funcionamiento de ciertos elementos teóricos y de librerías externas que se han utilizado durante el ciclo de desarrollo. Una vez estos puntos han estado claros, se ha podido pasar a la siguiente tarea. El orden de las tareas, pues, esta establecido para que los conocimientos adquiridos en una, sean utilizados en las tareas posteriores.

El desarrollo del simulador se empezó de manera paralela a la tarea de reconocimiento de voz para poder probar el sistema una vez obtenidos los modelos acústicos. También se inicio al mismo tiempo que el reconocedor de voz puesto que el desarrollo de un mundo virtual se preveía lento y difícil y el hecho de haberlo dejado hasta más tarde podría haber retrasado considerablemente el proyecto.

En el siguiente diagrama se muestra la relación entre tareas. Hay que destacar que la planificación temporal no se ha podido seguir de manera estricta por causas ajenas al proyecto, como exámenes o cuestiones laborales.



Aquí podemos apreciar como para poder empezar con una tarea, previamente se debe haber completado la anterior (Excepto en el caso del simulador, como ya se ha comentado anteriormente). Para la primera tarea, no existen subtareas solapadas, por lo que la estructura que se siguió es de tipo "salto de agua". Para la segunda tarea y sus subtareas asociadas, se siguió el mismo procedimiento, convergiendo al final de la subtarea de implementación del simulador, la tarea de reconocimiento de voz. Las tareas describen todo el proceso de desarrollo del sistema y las podemos relacionar con las distintas secciones que trataremos en la memoria. Las tareas 1-4 se corresponden con el capítulo 2 de la memoria, donde se comentan las tecnologías utilizadas, mientas que las tareas 5-9 forman parte del diseño e implementación del proyecto. Por último, la tarea 10, pertenece a las pruebas del sistema y mejoras derivadas de estas pruebas. Para la última tarea, la redacción de la memoria, hay que destacar, que aunque se encuentra al final del desarrollo del proyecto y esta se ha escrito una vez concluido el proyecto, se han ido tomando notas y apuntes para que su redacción resultará más fácil y coherente con todo lo que se ha ido aprendiendo y desarrollando.

1.6 Estructura de la memoria

Una vez concluida esta pequeña introducción del proyecto, donde se han explicado los puntos principales que lo resumen y la planificación del mismo, se dan paso los diferentes capítulos de la memoria, en los que se pretende profundizar tanto en la planificación y desarrollo del proyecto, como en todos los aspectos teóricos que se encuentran detrás de este.

En el segundo capítulo de la memoria se tratarán con más detalles los aspectos teóricos de las tecnologías utilizadas en el proyecto, los sistemas de reconocimiento de voz y las tecnologías virtuales, concretamente OpenGL.

En el capítulo tercero se tratarán las especificaciones del proyecto, incluyendo el análisis de requerimientos, tanto funcionales como no funcionales, y una descripción en detalle de los objetivos principales del proyecto (enumerados en el punto 1.4 de este mismo capítulo).

El capítulo cuarto esta dedicado al diseño y la implementación del proyecto. En este capítulo se explica como se ha procedido con el desarrollo del proyecto, como se han desarrollado las distintas partes y como estas se relacionan entre sí para obtener la forma y resultado final. También se comentan los distintos problemas que se han debido de afrontar, así como las soluciones aplicadas.

Finalmente, en el quinto y último capítulo se comentarán las conclusiones que se han obtenido fruto del trabajo en la preparación y desarrollo. Cerrarán la memoria la bibliografía y un anexo.

Capítulo 2

Tecnologías utilizadas.

2.1 Sistemas Domóticos

Una vivienda domótica no se diferencia ampliamente de una vivienda normal. En ella podemos encontrar los mismos elementos, como los electrodomésticos o las instalaciones de agua, gas y electricidad. La principal diferencia que se puede encontrar es que las viviendas domóticas incorporan una serie de componentes y sistemas que permiten controlar de forma automatizada todos los equipos e instalaciones que podemos encontrar en una vivienda normal (encendido automático de luces en horas determinadas, gestión automática de aire acondicionado o calefacción y riego automático, entre otras). Para que esto pueda llevarse a cabo, es necesario dotar a la vivienda de una serie de sensores y actuadores, junto con un sistema de control centralizado capaz de procesar la información subministrada por esos componentes.

Para que la comunicación entre estos dispositivos pueda ser efectiva, es necesario dotar al sistema de una red de intercomunicación, por la que la información captada en los sensores y actuadores, se envía al procesador central. Esta red de comunicación es principalmente la red eléctrica, puesto que permite aplicar sistemas domóticos en hogares que ya poseen una instalación previa, facilitando enormemente la instalación.

Los tres elementos básicos dentro de un hogar inteligente son el sistema de control centralizado, los sensores y los actuadores:

- ***Sistema de control centralizado***: Es el cerebro electrónico del hogar y es el encargado de recoger toda la información suministrada por los sensores. La información recibida

es procesada, generando las órdenes concretas que ejecutarán los actuadores. A día de hoy, únicamente existe un sistema de control, encargado de controlar los distintos tipos de instalaciones, mientras que anteriormente era necesario disponer de un sistema de control para cada "elemento" del hogar (agua, luz, gas…). Esto hace más fácil ampliar los sistemas domóticos del hogar a medida que vayan siendo necesarios. Es posible interactuar con el sistema de control, tanto para monitorizar aspectos de las instalaciones como para programar los sensores y actuadores de la manera deseada. Hasta hace relativamente poco, la interacción con el sistema de control se realizaba a partir de pantalla y teclado. Actualmente, las alternativas son mucho mayores, y se puede interactuar con el sistema de control mediante interfaces vocales, interfaces Web, control a distancia mediante teléfonos inteligentes o incluso envío y recepción de mensajes de móvil desde y hacia el sistema de control.

- ***Sensores***: Estos son los elementos encargados de recoger la información de diferentes parámetros, según el dispositivo que controlan, y enviarla al sistema de control centralizado. Lo normal es que estos sensores no vayan conectados a la red eléctrica, sino que tengan baterías externas, como pilas, para facilitar su conexión dentro del hogar, lejos de tomas de corriente. Algunos ejemplos de sensores son los detectores de gas, sensores de presencia, sensores de luz, etc.

- ***Actuadores***: Son los dispositivos finales que ejecutan las órdenes enviadas por el controlador central en función de los parámetros recogidos. En muchos casos, estos actuadores van integrados en el mismo dispositivo que controlan.

2.2 Reconocimiento de voz

Desde el principio de los tiempos el lenguaje hablado ha sido el modo predilecto utilizado por el ser humano para transmitir información. Como ya sabemos, el lenguaje hablado se utiliza para comunicar información entre dos o más personas. Una de estas personas actúa como hablante o emisor, mientras que las demás actúan como oyentes o receptores. En el uso del lenguaje hablado son importantes tanto la producción del habla

como la percepción que tenemos de la persona que esta hablando (como los gestos o las expresiones faciales).

Las señales del habla están formadas por una serie de patrones que utilizamos como representación del lenguaje hablado. Estos patrones son los fonemas, las silabas y las palabras. La interpretación y la producción de estos patrones vienen dados por la sintaxis y la semántica de la lengua que se habla. Gracias a la sintaxis y la semántica podemos formar la estructura de un lenguaje hablado.

A nosotros, como personas, nos es fácil, después de un proceso de aprendizaje, distinguir estos patrones, entender el contexto sobre el que se esta formulando una oración y analizar lo que el emisor nos quiere dar a entender. En cambio, la dificultad en un ordenador es mucho mayor. El ordenador de hoy en día no posee las habilidades humanas de hablar, razonar, escuchar, entender y aprender, por lo que el proceso de reconocer información obtenida en forma de señales acústicas se hace mucho más complicado. Debemos considerar además, que no solo es necesario obtener una señal acústica de entrada, sino que existen una serie de factores que influyen en el reconocimiento como el acento del hablante o las muletillas (Suponiendo siempre que el emisor emplea la misma lengua en cualquier caso). Incluso, en un sistema reconocedor de voz, afecta el hecho de que cada hablante tiene una anatomía vocal única y que por mucho que nos esforcemos no podemos llegar a pronunciar dos sonidos de la misma forma (Siempre nos encontraremos con variaciones de tono, frecuencia, duración, etc.).

En esta sección se trata de explicar como funciona un sistema reconocedor de voz y que elementos engloba y como así, desde un ordenador, se puede llegar a obtener una hipótesis de lo que dice el emisor.

2.2.1. Interacción Hombre - Máquina

El hecho de que el hombre tenga una predilección por el lenguaje hablado se ve reflejado también en los sistemas hombre-máquina, donde de igual manera se busca la

posibilidad de comunicarse con el sistema de la forma más cómoda y eficaz posible.

Actualmente, los sistemas informáticos comunes están basados en GUIs (*Graphical User Interface*). Estos sistemas representan una serie de objetos en pantalla con los que el usuario puede interactuar (Ventanas, menús, punteros…). La mayoría de sistemas requieren pues de otros dispositivos con los que dar ordenes sobre estos objetos. Estos dispositivos de entrada suelen ser el ratón y el teclado, aunque actualmente y cada vez más existen dispositivos táctiles, que permiten activar estos objetos sin necesidad de un periférico intermedio.

Tanto la predilección que tenemos sobre el lenguaje hablado como el deseo de obtener una mayor flexibilidad en nuestros sistemas nos está llevando a buscar soluciones y cambios sobre los dispositivos y periféricos actuales. Uno de estos cambios es el de dotar a los sistemas de lenguaje hablado. Aún cuando estos sistemas no han evolucionado completamente ya los podemos encontrar en diversas aplicaciones y entornos (Aplicaciones móviles o aplicaciones en el hogar y en la oficina).

Cuando hablamos de sistemas de lenguaje hablado nos referimos a sistemas que sean capaces tanto de realizar reconocimiento de voz como de tener capacidades de síntesis de voz. En el caso de este proyecto se trata únicamente el reconocimiento de voz.

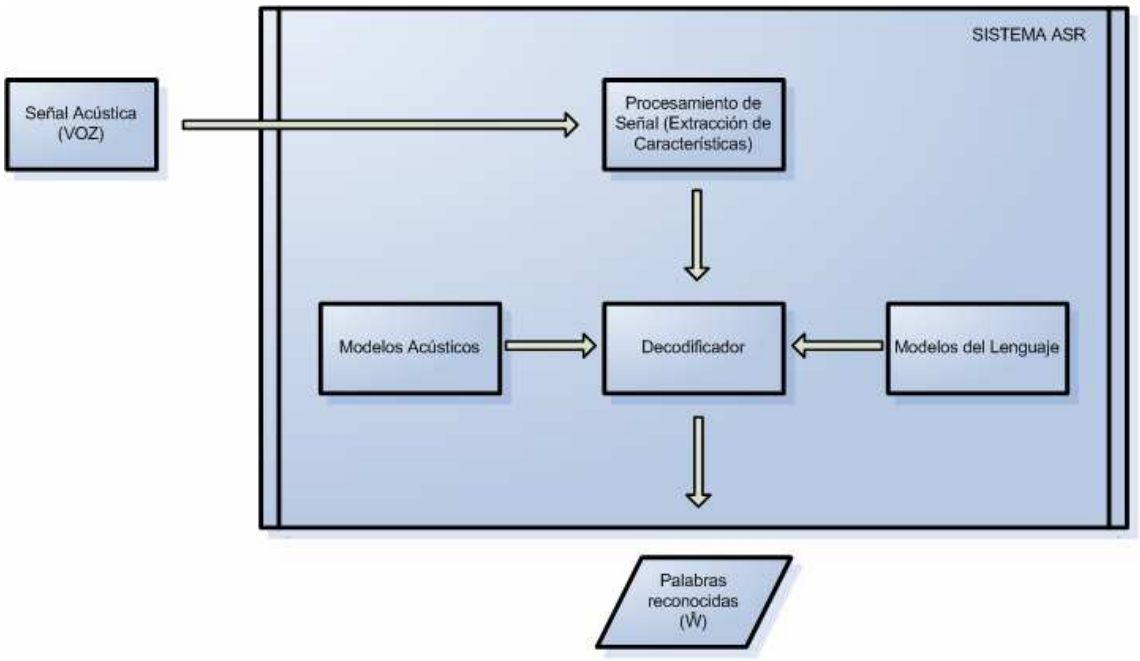
2.2.2. Sistemas ASR

Un sistema ASR (Siglas de *Automatic Speech Recognition*) tiene como principal objetivo abordar el problema del reconocimiento de voz, intentando darle solución con el máximo posible de efectividad. El principal problema que encontramos en estos sistemas es la posibilidad de reconocer el habla para cualquier hablante, con cualquier dominio y en cualquier ambiente (El dominio es el número de palabras que el sistema debe ser capaz de reconocer. Cuanto mayor sea el dominio mayor será la dificultad de implementar el sistema). Esto a día de hoy todavía está lejos de resolverse aunque en los últimos años se ha mejorado considerablemente, llegando a ser factible para dominios

limitados.

El procedimiento que sigue un sistema ASR se iniciaría cuando el hablante emite una secuencia de palabras **W**. La onda del habla generada por el emisor será captada por el componente encargado de procesar la señal del habla y será enviada al decodificador. El decodificador tratará de decodificar la señal acústica de entrada **X** en una palabra o secuencia de palabras $\hat{\mathbf{W}}$. La salida $\hat{\mathbf{W}}$ se espera que sea parecida a la secuencia original de palabras, **W,** pronunciada por el emisor. Esta salida $\hat{\mathbf{W}}$**,** es considerada la hipótesis.

Para llevar a cabo este proceso, un sistema ASR consta de una serie de partes diferenciadas, cada una de las cuales posee una función en la tarea de reconocimiento de voz. En el siguiente esquema se puede apreciar como se relaciona cada uno de los módulos que forman parte del reconocedor. Más adelante se comentará la función de cada uno de estos módulos dentro del sistema ASR.



Como podemos observar en la figura anterior, un sistema ASR consta de un procesador

de señal, un decodificador y los modelos acústicos y de lenguaje.

El procesador de señal es la parte del sistema que se encarga de obtener la señal acústica de entrada. Una vez se ha obtenido, es procesada para extraer los patrones o vectores de características. Los vectores de características son representaciones paramétricas de la señal del habla. En estos está contenida la información más relevante y característica de la onda acústica.

Un modelo acústico es la representación de un sonido basado en la comprensión del comportamiento del sonido, en la disciplina de la acústica. Si consideramos los modelos acústicos en el ámbito del reconocimiento de voz, los podemos ver como elementos que contienen la variabilidad acústica de la lengua. Es gracias a estos modelos y a una señal de entrada que podemos obtener una hipótesis de lo que ha dicho el emisor. En el modelo acústico encontramos los diccionarios de pronunciación. En ellos se especifica la secuencia de sonidos de una palabra y contienen las definiciones de los distintos tipos de sonidos que contiene una lengua. En los diccionarios encontramos las definiciones para las constantes nasales, fricativas, vocales…

El modelo de lenguaje o gramática nos ayuda a obtener una hipótesis de cuales pueden ser las siguientes palabras que el emisor vaya a pronunciar. Esta hipótesis se obtiene en función de las palabras que ya ha pronunciado el emisor. El modelo de lenguaje puede contener gramáticas libres o gramáticas restringidas. Las gramáticas libres necesitan de modelos especiales para ser tratadas e incluyen los lenguajes naturales. Una gramática restringida (como la que vamos a utilizar en este proyecto), tiene un número limitado de palabras o frases que se pueden reconocer. Una gramática restringida se puede representar mediante un autómata finito y únicamente podemos reconocer palabras dentro de los estados del autómata.

En el decodificador del sistema ASR se unen los vectores de características, los modelos acústicos y los modelos del lenguaje para obtener una hipótesis de la señal acústica.

2.2.3. Tipos de Sistemas de Reconocimiento de Voz

El objetivo principal de un sistema de reconocimiento de voz es que estos funcionen de manera que sean capaces de reconocer cualquier voz, de manera espontánea o no, y que el habla sea de manera natural. El principal problema que existe es la amplia variabilidad de la señal acústica, cosa que dificulta enormemente conseguir ese objetivo.

Actualmente, podemos clasificar los sistemas de reconocimiento de voz en función de una serie de restricciones:

- **Dependencia / Independencia del Interlocutor**: Un sistema reconocedor de voz puede ser dependiente o independiente del interlocutor. Los sistemas dependientes del interlocutor están pensados para que el sistema sea capaz de reconocer las órdenes de las voces para las que ha sido entrenado, mientras que en los sistemas independientes, se busca que el sistema reaccione adecuadamente ante cualquier persona que utilice el sistema. En los sistemas dependientes la tasa de porcentaje de aciertos suele ser mayor, debido a que el entrenamiento se ha realizado para una señal acústica en concreto. En los sistemas independientes, en cambio, la tasa de aciertos es menor. El sistema no ha sido entrenado para ninguna señal acústica en concreto, y los vectores de características, son altamente dependientes del locutor.

- **Habla espontánea**: El habla espontánea es prácticamente igual que el habla natural, pero en este tipos de sistemas debemos tener en cuenta que se deben reconocer elementos como las muletillas (“um”, “ah”…), muy característicos del habla no planificada.

- **Palabras aisladas / Palabras conectadas**: También podemos clasificar un sistema ASR en función del tipo de habla que utiliza el locutor. Podemos tener sistemas capaces de reconocer palabras aisladas, en las que es necesario incluir

pausas al principio y al final de cada palabra, que mejoran el reconocimiento y sistemas capaces de reconocer palabras conectadas o habla continua. En el habla continua, a diferencia de las palabras aisladas, las pausas entre palabras no son tan evidentes. Es más difícil saber donde acaba y donde empieza una palabra, cosa que hace más difícil el reconocimiento.

- **Tamaño del Vocabulario**: Como se ha comentado en puntos anteriores, el tamaño del vocabulario también va ligado al porcentaje de éxito que nuestro sistema es capaz de obtener. Existen básicamente dos tipos de vocabularios, los restringidos y los libres. Los vocabularios restringidos normalmente están acotados sobre un cierto número de palabras, mientras que los libres nos permiten reconocer cualquier palabra de la lengua. El principal problema de las gramáticas o vocabularios libres es el tamaño. Cuanto mayor es este, más difícil se hace encontrar equivalencias sobre la señal acústica del emisor, puesto que existen más palabras sobre las que cotejar e incluso palabras con pronunciaciones muy parecidas, y por lo tanto, mayor es el tiempo de respuesta del sistema.

- **Robustez**: Cuando hablamos de robustez de un sistema estamos indicando si este ha sido diseñado para ser utilizado en ambientes muy ruidosos o poco ruidosos. La contaminación acústica es un problema común en los sistemas ASR, puesto que cualquier sonido superpuesto a la señal de entrada puede modificar los resultados obtenidos, haciendo que estos queden invalidados. Muchos sistemas de reconocimiento incorporan filtros para eliminar al máximo las señales de ruido externas, pero estos filtros incrementan el tiempo de respuesta del sistema.

- **Verificación o Identificación de la voz**: Existen sistemas que al margen de realizar el reconocimiento de voz pertinente, buscan también, ser capaces de identificar la identidad del usuario que habla. Estos sistemas son bastante más complejos de implementar y a día de hoy no los podemos encontrar en

aplicaciones de nuestro día a día.

2.2.4. Herramientas de desarrollo: HTK

Puesto que el desarrollo de un sistema reconocedor de voz desde zero es una tarea muy complicada y que requiere una serie de conocimientos matemáticos no impartidos durante la carrera, se ha utilizado una herramienta de código abierto de reconocimiento de voz. La herramienta en cuestión es el conjunto de librerías y aplicaciones HTK.

HTK ha sido desarrollado en el "*Machine Intelligence Laboratory*", en la Universidad de Cambridge. Aunque debido a una serie de cambios en los derechos del software, actualmente, Microsoft, se encuentra en posesión del copyright sobre el código fuente original. Aún estando en propiedad de Microsoft, se anima a contribuir realizando cambios sobre el código fuente original.

El toolkit de HTK permite construir y manipular Modelos Ocultos de Markov (*Hidden Markov Models, HMM*). Los Modelos Ocultos de Harkov son modelos estadísticos que a partir de una serie de patrones observables, se utilizan para reconocer patrones (*Xuedong Huang, Alex Acero & Hsiao-Wuen Hon. "Spoken Language Processing" Prentice Hall PTR (2001), Capítulo 8, página 375*).

HTK se usa principalmente para realizar investigación sobre el reconocimiento del habla, aunque se ha utilizado para realizar investigación sobre aplicaciones basadas en otro tipo de tecnologías. HTK esta formado por una serie de librerías y módulos escritos en C. Entre las herramientas que incorpora se encuentran aplicaciones para realizar algunas de las tareas que se han ido comentando, como el entrenamiento del sistema, o las pruebas y el análisis de resultados. Toda la información sobre HTK, así como ejemplos y código fuente se puede encontrar en su Web: http://htk.eng.cam.ac.uk/.

2.2.5. Herramientas de desarrollo: Julius

HTK nos ofrece un conjunto de librerías y módulos con los que podemos preparar nuestro sistema de reconocimiento de voz, pero en cambio, no nos ofrece una API de desarrollo con la que realizar el reconocimiento de voz a nivel de software. Para solventar este problema hemos utilizado otra utilidad de código abierto, Julius.

Julius es un decodificador para sistemas de reconocimiento de voz basado en HTK, de ahí que haya sido nuestra elección a la hora de buscar un decodificador. Ha sido desarrollado como software de investigación por los japoneses LVSCR. Es capaz de efectuar decodificación en tiempo real en prácticamente cualquier PC actual. La documentación y los códigos fuentes del proyecto se pueden encontrar en su página Web: http://julius.sourceforge.jp/en_index.php

2.3 Tecnologías Virtuales

Al margen del reconocimiento de voz, el proyecto también consta de una parte de simulación, dedicada a la visualización de los resultados de las órdenes ejecutadas. Esta parte del proyecto ha sido realizada con tecnologías virtuales, concretamente, realizando un entorno 3D en el que el usuario se puede mover, intentando transmitir así un poco mejor la sensación que se obtendría utilizando un reconocedor de voz sobre un sistema domótico real.

Las tecnologías virtuales son aquellas tecnologías que intentan dotar a los sistemas de una mayor integración con el usuario y de un mayor nivel de detalle y de realidad. Actualmente, podemos encontrar una serie muy diferente de este tipo de tecnologías, desde las ya utilizadas desde hace años, como los entornos 3D (utilizado en videojuegos o en el cine para incrementar la realidad de las escenas que aparecen en pantalla), hasta las más novedosas como la realidad aumentada.

Puesto que para el desarrollo del proyecto era complicado diseñar un sistema domótico real se ha optado por realizar un simulador, utilizando un motor 3D. Concretamente, el

desarrollo del simulador se ha realizado mediante openGL.

2.3.1. Herramientas de desarrollo: OpenGL

OpenGL es una interfase software para aceleración gráfica por hardware, es decir, es una librería que nos permite comunicarnos con el hardware encargado de la parte gráfica para poder ejecutar comandos de dibujo. De esta manera se consigue cargar al procesador gráfico (*GPU, Graphic Process Unit*) de más trabajo, dejando libre para otras tareas al procesador central del ordenador, la CPU (*Control Process Unit*).

Esta interfase consta de unas 250 órdenes que se utilizan para especificar objetos y operaciones necesarios para producir aplicaciones tridimensionales dotadas de interactividad. OpenGL está diseñado para que sea independiente del hardware y que pueda ser por tanto utilizado en distintos sistemas. OpenGL no contiene primitivas complicadas, sino que ofrece una serie de primitivas básicas con las que combinadas se pueden construir modelos mayores. Las primitivas básicas son puntos, líneas y polígonos.

2.3.1. Herramientas de desarrollo: GLUT

GLUT es un toolkit de utilidades para openGL. Concretamente es un sistema de ventanas independiente para escribir aplicaciones openGL. Permite escribir aplicaciones openGL independientemente del sistema, de manera que estas sean portables. Además ofrece una serie de funciones que permiten implementar tareas comunes a todas las aplicaciones openGL de manera más fácil y cómoda.

GLUT se trata de una librería gratuita para usar junto con openGL aunque no es de código abierto. La documentación y los ejecutables se pueden encontrar en la página Web del proyecto: http://www.opengl.org/resources/libraries/glut/.

Capítulo 3

Análisis de Requerimientos

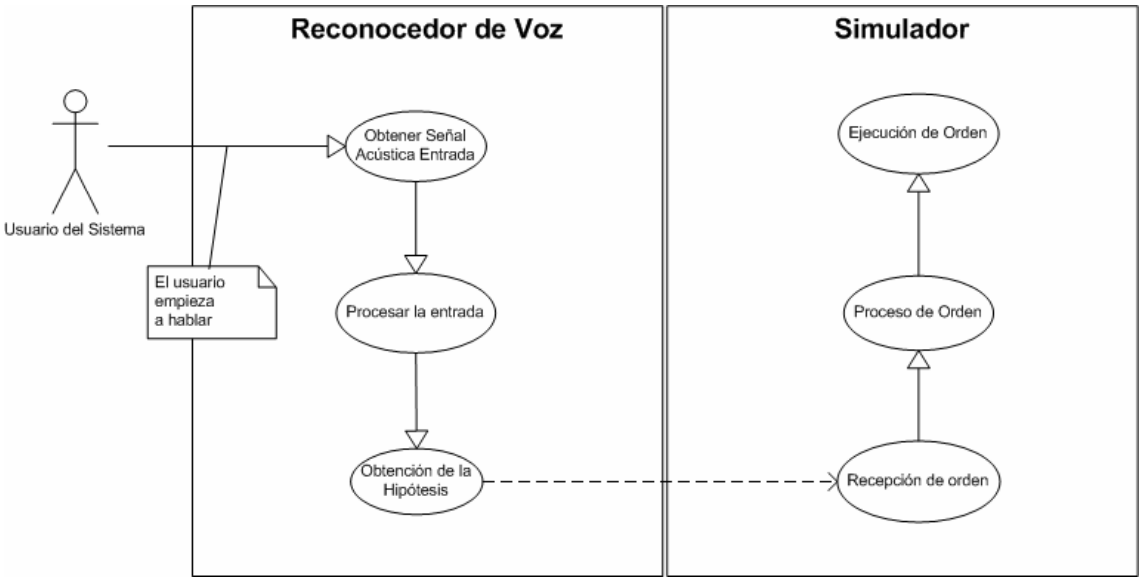
3.1 Análisis de requerimientos

Después de haber visto en que se basa un sistema domótico y como un reconocedor de voz puede ser aplicado en este caso, podemos pasar a discutir los requerimientos del sistema que se desea implementar. Podemos separar las distintas acciones del proyecto en requisitos funcionales y no funcionales. Como sabemos, los requisitos funcionales se basan en el comportamiento interno del software (que debe hacer), mientras que los no funcionales no se basan en los comportamientos específicos, sino en la operación que lleva a cabo el sistema (como lo debe hacer).

En este caso, al tratarse de un sistema con pocos puntos a ejecutar, nos encontramos con un análisis de requerimientos bastante acotado. Como sabemos, se necesita un sistema que tenga la posibilidad de ser controlado por la voz de un usuario. Este sistema se corresponderá con un simulador domótico.

3.1.1. Requerimientos funcionales

Los requerimientos funcionales nos indican que es lo que hace el software. La tarea principal de este sistema es obtener una señal acústica de entrada, procesarla y enviarla al simulador para su posterior ejecución. El proceso general de nuestro sistema se puede resumir con el siguiente diagrama de caso de uso. Este diagrama muestra todo el proceso anterior:



A partir del caso de uso anterior podemos realizar un análisis de los puntos que debe incluir nuestro proyecto y que son esenciales para el funcionamiento del mismo de la manera deseada.

El primer punto a cuestionarse es que debe hacer nuestro sistema. Nuestro sistema debe ser capaz de obtener una serie de señales acústicas desde un dispositivo externo. Una vez obtenidas estas señales debe ser capaz de procesarlas, contra una gramática preestablecida y entrenada, para encontrar una hipótesis de su contenido y por último, siempre y cuando se consiga encontrar una hipótesis adecuada, y tras el paso de mensajes entre los dos módulos, realizar la ejecución de lo que el usuario ha solicitado sobre un simulador virtual. Los puntos que podemos extraer como requerimientos funcionales y que nos indican que es lo que hace nuestro software (los casos de uso) son:

- Crear una gramática con el conjunto de comandos aceptados.
- Entrenamiento de los patrones de voz.
- Obtener una señal acústica de entrada.
- Obtener una hipótesis final sobre la señal de entrada a partir de los patrones de voz.

- Generar un paso de mensajes entre el reconocedor y el simulador.

- Ejecutar la hipótesis final sobre el simulador

Gramática Aceptada:

Como en todo lenguaje deberemos crear y especificar la gramática que el sistema es capaz de entender. De las combinaciones de las palabras de esta gramática se obtendrá todo el conjunto de órdenes que pueden ser entendidas por el reconocedor de voz. Este conjunto de órdenes es lo que formará el lenguaje de nuestro sistema, definiendo el conjunto de reglas y principios que lo gobiernan. Como es de suponer, la gramática es un aspecto de la aplicación que varía según el tipo de aplicación por lo que nuestra gramática se debe poder aplicar al campo de la domótica.

Entrenamiento:

El poseer una gramática para ser utilizada en funciones de reconocimiento, requiere que el sistema se vea entrenado con la gramática en cuestión. Realizar un entrenamiento del sistema permite obtener un porcentaje de acierto más elevado durante la ejecución. El entrenamiento es la fase donde el sistema aprende a partir de un número elevado de grabaciones el conjunto de órdenes que se debe reconocer. Cuanto más grande sea el número de muestras, mayor será la posibilidad de que nuestro sistema sea capaz de reconocer las órdenes que le llegan del usuario. Este entrenamiento sirve para poder reconocer variaciones sobre la pronunciación de una misma orden, ya que por mucho que nos lo parezca y/o lo intentemos, nunca pronunciamos una palabra de la misma forma. Entre otros factores influyen la situación y el estado de ánimo, o el ruido ambiente de la zona donde esta funcionando el sistema.

Información de entrada:

Obtener la señal acústica de entrada desde una fuente externa es un requerimiento clave del sistema. Es necesario que el sistema obtenga la señal de entrada en vivo, de manera que el usuario pueda interactuar directamente con el sistema. Esta señal acústica se deberá obtener en vivo a través de un periférico conectado al PC. Debemos destacar que

no será válido que el sistema funcione ejecutando acciones a partir de grabaciones efectuadas anteriormente. Un sistema domótico debe poder funcionar en vivo, de manera que aunque también pueda aceptar grabaciones, esto no debe ser considerado un requerimiento funcional.

Obtención de la hipótesis:

Tras obtener la señal de entrada el sistema debe compara los patrones obtenidos con los patrones entrenados anteriormente para ser utilizados en el sistema. Esto permitirá obtener una hipótesis final sobre las palabras pronunciadas por el usuario. Para llevar a cabo la obtención de la hipótesis se debe realizar una decodificación de la señal de entrada. Una vez tratada esta señal, existen tres resultados posibles. El primer resultado es que no se haya podido reconocer la orden (el sistema entra en el % de palabras que es incapaz de saber si forman parte de la gramática o no). La segunda posibilidad es que el sistema reconozca la orden y la tercera posibilidad es que la orden reconocida sea una orden errónea (es decir, que no se corresponda con la entrada).

Paso de mensajes:

En el supuesto de que la orden sea reconocida, el reconocedor de voz se debe encargar de enviar un mensaje hacia el simulador. Este mensaje contendrá la orden que el simulador debe ejecutar. Una vez recibido el mensaje, su objetivo será el de procesar el mensaje recibido, obteniendo la orden que se ha solicitado y en función de la orden solicitada, ejecutar la acción pertinente en el simulador.

Simulador:

Este proceso se debe repetir de manera constante hasta que el usuario finalice la ejecución tanto del reconocedor, como del simulador. El reconocedor de voz debe estar permanentemente a la escucha y detectar cada vez que se produzcan variaciones sobre la fuente de entrada. El simulador en cambio, debe estar a la escucha del canal de entrada ligado al reconocedor y procesar los mensajes recibidos tan rápido como estos lleguen.

3.1.2. Requerimientos no funcionales

Como requerimientos no funcionales, tenemos el tiempo de procesamiento de las señales acústicas. Como es de suponer, no podemos permitir que el sistema tarde demasiado en reconocer las ordenes y posteriormente ejecutarlas, puesto que lo que se busca es una respuesta rápida a una manera más cómoda de ejecutar una serie de comandos. Esto hace que uno de los puntos clave sea el desarrollo y el entrenamiento de la gramática. Este proceso se debe efectuar de la manera más acertada posible, intentando obtener unos tiempos de respuesta bajos y coherentes con el tipo de aplicación. El funcionamiento interno en lo que se refiere a tiempos, también debe tenerse en cuenta en el paso de mensajes entre el reconocedor de voz y el simulador, puesto que un tiempo de envío demasiado grande causaría los mismos problemas que un tiempo de proceso de señales demasiado grande. Por eso, igual que se debe buscar la manera idónea de entrenar la gramática, se debe buscar la mejor manera de implementar un paso de mensajes entre cliente – servidor.

Otros requerimientos no funcionales que nos afectan, es el deber de usar el sistema HTK para el entrenamiento de la gramática. HTK es la herramienta que actualmente se utiliza en el departamento de Telecomunicaciones e Ingeniería de Sistemas y que ha sido propuesta para ser utilizada en el proyecto.

Capítulo 4

Diseño e Implementación

4.1 Introducción

En este capítulo se documenta el proceso de diseño e implementación del proyecto, comentando cada una de las distintas partes que lo forman y como estas se relacionan entre ellas. Después de esta visión general de cómo se ha realizado la implementación del proyecto encontramos cuatro puntos diferenciados que forman el resto del capítulo. Los puntos se corresponden a los dos principales elementos que forman el sistema, el reconocedor de voz y el simulador, otro punto dedicado a la comunicación entre ambos elementos y por último, un punto en el que se comentan las pruebas realizadas sobre el sistema.

El desarrollo del proyecto se ha dividido en tres módulos. El objetivo principal, sobre el que se basaba todo el proyecto era el reconocedor de voz, por lo que este tenía que ser el pilar principal. Si no se conseguía el objetivo de reconocer las órdenes no tenía ningún sentido proceder con las partes restantes del proyecto. En el siguiente punto se describe todo el proceso que se ha llevado a cabo para conseguir reconocer las distintas señales acústicas, desde la generación de la gramática hasta la implementación del reconocedor de voz.

Una vez conseguido el primer objetivo era necesario centrarse en la parte visual de la aplicación, sobre la que se podría ver el resultado de las órdenes acústicas interpretadas. En el punto 4.3 se comenta como y porque se ha optado por una solución 3D y como se ha llevado a cabo el proceso de implementación.

Finalmente era necesario comunicar los dos elementos de manera que se pudiera

producir una respuesta a las órdenes de entradas dadas por el usuario. El proceso de comunicación entre el reconocedor de voz y el simulador se explica con más detalle en el punto 4.4 de este mismo capítulo.

4.2 Sistema Reconocedor de Voz

Como se ha venido comentando hasta ahora, el reconocedor de voz es el elemento más importante del sistema. Es el componente que da nombre al proyecto y por lo tanto, sin un reconocedor de voz en condiciones no podemos plantearnos un proyecto de este tipo. Este era por tanto el punto principal a resolver y sobre el que giraría el resto del sistema.

Para realizar el entrenamiento se han utilizado las librerías de código libre y los módulos disponibles en HTK, por lo que ha sido necesario dedicar una serie de horas para aprender el funcionamiento de HTK. Esto ya se ha tenido en cuenta en el cómputo de horas de las tareas, aunque no se ha establecido como objetivo ya que es indispensable para llegar a conseguir el objetivo final del entrenamiento. Con esta herramienta es posible realizar el entrenamiento del sistema de manera más cómoda y además ofrece una serie de herramientas con las que probar los resultados obtenidos sobre el reconocedor de voz.

4.2.1. Construcción de la Gramática

Dentro del reconocedor de voz, la gramática es el punto principal que debemos resolver. La gramática nos sirve para especificarle al sistema reconocedor de voz que es lo que queremos reconocer. Una gramática debe estar bien definida de manera que sea posible reconocer únicamente las órdenes que nos interesan.

En nuestro caso, estamos hablando de un sistema domótico con reconocimiento de voz por lo que la gramática que emplearemos debe basarse en órdenes domóticas. Así pues, el primer punto es obtener la lista de órdenes o frases con las que queremos entrenar el sistema y de la que obtendremos la gramática. En nuestro caso, son:

```
- ABRE o CIERRA LA VENTANA DEL COMEDOR
- ABRE o CIERRA LA VENTANA DE LA HABITACIÓN
- ABRE o CIERRA LA VENTANA DE LA COCINA
- ABRE o CIERRA LA VENTANA DEL ESTUDIO
- ABRE o CIERRA LA VENTANA DE LA SALA DE ESTAR
- ABRE o CIERRA LA VENTANA DEL SALÓN
- ABRE o CIERRA LA VENTANA DEL LAVABO
- ABRE o CIERRA LA PUERTA DEL COMEDOR
- ABRE o CIERRA LA PUERTA DE LA HABITACIÓN
- ABRE o CIERRA LA PUERTA DE LA COCINA
- ABRE o CIERRA LA PUERTA DEL ESTUDIO
- ABRE o CIERRA LA PUERTA DE LA SALA DE ESTAR
- ABRE o CIERRA LA PUERTA DEL SALÓN
- ABRE o CIERRA LA PUERTA DEL LAVABO
- ENCIENDE o APAGA LA LUZ
- ENCIENDE o APAGA LAS LUCES
- ENCIENDE o APAGA LA CALEFACCIÓN
- ENCIENDE o APAGA EL AIRE ACONDICIONADO
- FIJA LA TEMPERATURA A VEINTE
- FIJA LA TEMPERATURA A VEINTIUNO
- FIJA LA TEMPERATURA A VEINTIDOS
- FIJA LA TEMPERATURA A VEINTITRES
- FIJA LA TEMPERATURA A VEINTICUATRO
- FIJA LA TEMPERATURA A VEINTICINCO
```

A partir de esta serie de órdenes podemos generar nuestra gramática, de manera que sea entendida por las herramientas de HTK. Las gramáticas para HTK se definen de manera que entre dos etiquetas de inicio y de fin se definan todas las distintas opciones de órdenes a reconocer, separadas por "|". Cada una de estas órdenes puede estar a su vez formada por un conjunto de variables y/o opciones. A continuación se muestra la gramática que se ha definido para nuestro sistema, donde se podrá comprender mejor como se han de estructurar las gramáticas para HTK.

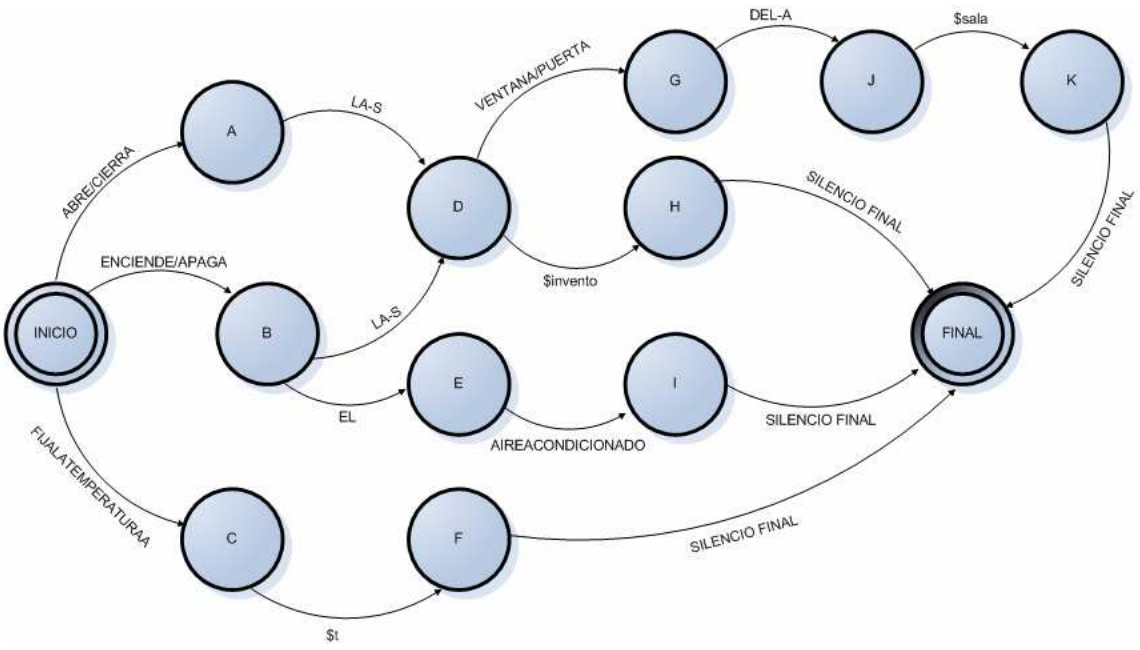
```
$sala = COMEDOR | HABITACIÓN | COCINA | ESTUDIO | SALADEESTAR | SALON | LAVABO
$t = VEINTE | VEINTIUNO | VEINTIDOS | VEINTITRES | VEINTICUATRO | VEINTICINCO
$ invento = LUZ | LUCES | CALEFACCION

(INICIO (((ABRE | CIERRA) (LA-S) (VENTANA | PUERTA) (DEL-A) $sala) |
((ENCIENDE | APAGA) (LA-S) $invento) |
```

```
((ENCIENDE | APAGA) (EL) (AIREACONDICIONADO)) |
((FIJALATEMPERATURAA) $t)) FINAL)
```

Las variables precedidas del símbolo "$" indican partes de la frase que pueden ser substituidas por cualquiera de los valores que puede tomar esa variable. Esto hace más fácil construir la gramática, de manera que no hay necesidad de realizar muchas repeticiones para todas las órdenes que se pueden reconocer. El símbolo "|" indica alternativa, por lo tanto, donde encontremos ese símbolo solo se podrá poner un único valor sobre los que hay disponibles. Las etiquetas de INICIO y de FINAL que se incluyen en la gramática nos servirán para atribuirles el valor de silencio y así poder saber cuando comienza una nueva orden a la hora de decodificar. Esta definición de gramática nos da el conjunto de órdenes enumerado anteriormente.

Podemos traducir la gramática generada a un Autómata finito no determinista que es capaz de generar nuestro lenguaje. (Las palabras del lenguaje aceptadas, serán aquellas que acaben en un estado final.



Aún así, esta representación de la gramática está más dirigida al usuario que al propio HTK. HTK necesita un formato interno. Este formato es una red de palabras que usa un

nivel más bajo de notación llamado *HTK Standard Lattice Format* (*SLF*), en el que cada palabra y cada transición de palabra a palabra está incluida específicamente. En este punto es donde entran las herramientas HTK. Una vez definida la gramática a nivel "usuario", podemos utilizar una de las herramientas para convertir la gramática a formato *SLF*. Para obtener la gramática en formato *SLF*, ejecutamos:

```
HParse domoticaGrammar domoticaWdnet
```

DomoticaGrammar es el fichero que contiene nuestra gramática, mientras que ***domoticaWdnet*** es el fichero de salida que contiene la gramática a nivel de red de palabras. Una parte del fichero ***domoticaWdnet*** se puede ver en el Anexo de esta memoria.

4.2.2. Diccionario

Una vez definida la gramática, el siguiente paso es definir el diccionario asociado a la gramática. Recordemos, que uno de los elementos utilizados a la hora de reconocer la voz eran las gramáticas y los diccionarios. En el diccionario se especifican las transcripciones de las distintas palabras que forman parte de la gramática, ordenadas alfabéticamente. Las transcripciones se pueden realizar a nivel de palabras o a nivel de fonemas, obteniendo mejores resultados si se realiza a nivel de fonemas ya que se obtienen mayores variaciones de un mismo sonido. En el caso de querer ampliar la gramática, y por consiguiente, el diccionario, es más fácil realizar el proceso si se han utilizado modelos de palabras. Nuestro reconocedor de voz se basa en una aplicación en la que los modelos acústicos modelan fonemas y no palabras.

Para realizar las transcripciones de las palabras a fonemas se pueden utilizar símbolos arbitrarios o se pueden utilizar alfabetos predefinidos. En este caso se ha optado por utilizar un alfabeto ya predefinido, concretamente el alfabeto SAMPA. El alfabeto SAMPA (*Speech Assessment Methods Phonetic Alphabet*) es un alfabeto fonético legible por ordenador mediante caracteres ASCII de 7 bits que está basado en el Alfabeto Fonético Internacional o IPA. El alfabeto SAMPA para español se puede

encontrar en el Anexo de esta memoria (punto 2).

Así pues, una vez se tiene claro que símbolos se van a utilizar para cada uno de los fonemas presentes en nuestra gramática, podemos pasar a definir el diccionario. Así, si en el diccionario establecemos las distintas palabras que forman la gramática, con sus trascripciones pertinentes, tenemos:

```
ABRE                a B r e sp
AIREACONDICIONADO   a i r e a k o n d i T i o n a d o sp
APAGA               a p a g a sp
CALEFACCION         k a l e f a k T i o n sp
CIERRA              T j e rr a sp
COCINA              k o T i n a sp
COMEDOR             k o m e d o r sp
DEL-A               d e l sp
DEL-A               d e l a sp
EL                  e l sp
ENCIENDE            e n T j e n d e sp
ESTUDIO             e s t u d i o sp
FIJALATEMPERATURA   f i x a l a t e m p e r a t u r a sp
FINAL []            sil
HABITACION          a b i t a T i o n sp
INICIO []           sil
LA-S                l a sp
LA-S                l a s sp
LAVABO              l a b a B o sp
LUCES               l u T e s sp
LUZ                 l u T sp
PUERTA              p u e r t a sp
SALADEESTAR         s a l a d e e s t a r sp
SALON               s a l o n sp
VEINTE              b e j n t e sp
VEINTICINCO         b e j n t i T i n k o sp
VEINTICUATRO        b e j n t i k u a t r o sp
VEINTIDOS           b e j n t i d o s sp
VEINTITRES          b e j n t i t r e s sp
VEINTIUNO           b e j n t i u n o sp
VENTANA             b e n t a n a sp
```

Como podemos ver, cada palabra posee la trascripción en fonemas según el alfabeto

SAMPA. Al final de cada palabra se ha añadido el símbolo *sp* (por *short pause*) que utilizaremos en pasos posteriores. Básicamente nos servirá para detectar las pequeñas pausas entre palabras. Además, se han añadido las dos etiquetas antes comentadas de INICIO y FINAL. Estas etiquetas nos servirán para indicar las pausas que habrá entre el final de una frase y el principio de la siguiente, y que serán espacios de silencio más grandes que los definidos por *sp*.

4.2.3. Grabación de los comandos de voz.

Una vez preparados el diccionario y la gramática, ya podemos pasar a realizar las grabaciones de los comandos. Estas grabaciones serán las muestras con las que realizaremos el entrenamiento del sistema. Hay que destacar que cuanto mayor sea el número de muestras, mayores serán los porcentajes de acierto que obtengamos posteriormente. En este caso, las grabaciones se han realizado sobre un único emisor, es decir, que el reconocimiento será sumamente efectivo sobre la persona que ha grabado las muestras, mientras que si se emplea con personas para las que no se ha realizado un entrenamiento, los porcentajes de acierto pueden ser bastante bajos. Si quisiéramos realizar un sistema para más de una persona deberíamos obtener muestras de cada una de ellas y realizar los pasos que se detallan de aquí en adelante para cada una de las diferentes muestras de voz. Para este caso, el número de grabaciones realizadas ha sido de 158 (Inicialmente eran 150, pero se han añadido grabaciones nuevas para mejorar el resultado final de los modelos).

HTK proporciona una serie de herramientas para realizar las grabaciones de audio que se utilizaran en el entrenamiento. Esta herramienta es HSLab. Se pueden utilizar otras herramientas de captura de audio, pero debemos tener en cuenta que las muestras se deben grabar en formato mono a 16 Mhz (16000 Khz.).

Para saber que ordenes son las que debemos grabar ejecutaremos la herramienta HSGen de HTK, que nos genera una lista aleatoria de comandos obtenidos a partir de la gramática. Uno de los parámetros que recibirá será el número deseado de muestras (en

nuestro caso 158).

```
HSGen -l -n 158 domoticaWdnet domoticaDict > domoticaPrompts
```

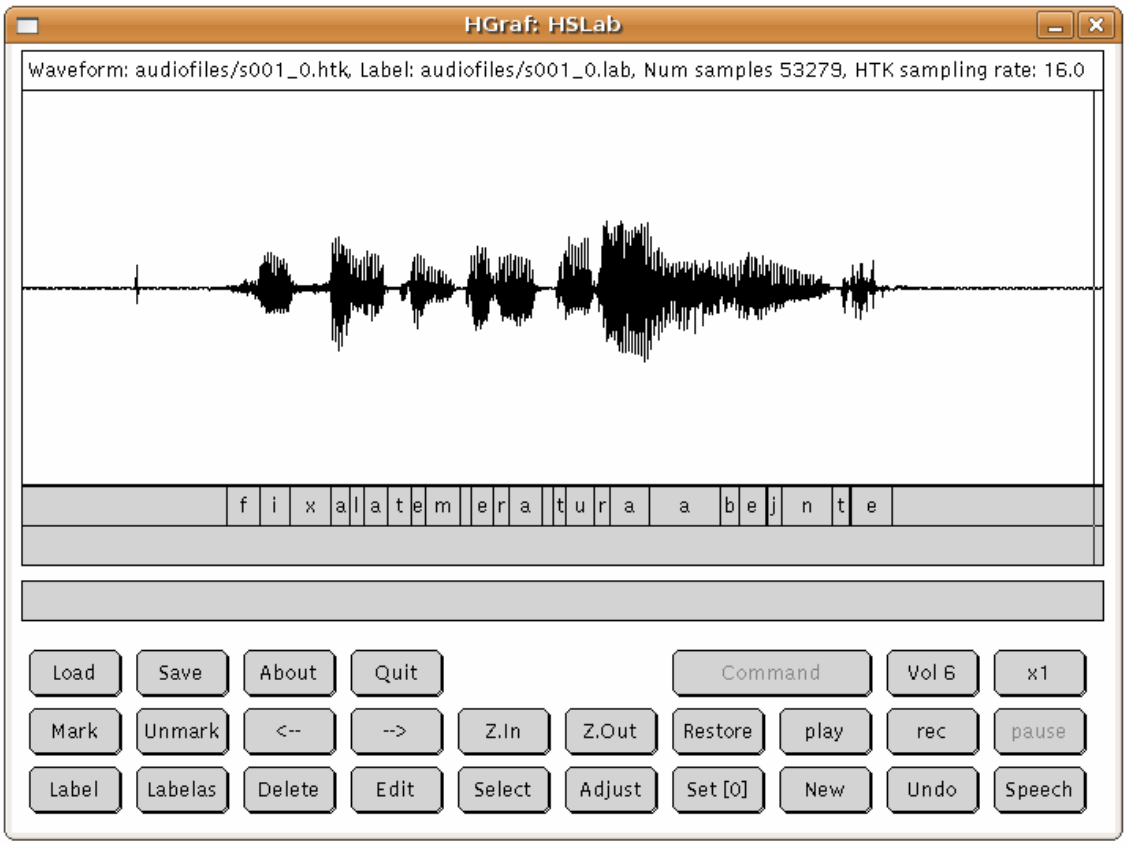
El fichero domoticaPrompts se corresponderá con un fichero de texto plano en el que tendremos un conjunto de 150 órdenes aleatorias y que se corresponderá con el contenido que debe tener cada uno de los 150 ficheros de audio. La estructura del fichero es la siguiente:

```
1. FIJALATEMPERATURAA VEINTE
2. APAGA LA-S LUCES
3. ABRE LA-S PUERTA DEL-A HABITACION
…
158. APAGA LA-S LUCES
```

En este momento, seguiremos el contenido del fichero y realizaremos todas y cada una de las grabaciones utilizando HSLab o cualquier otro programa de captura de audio.

4.2.4. Creación de los modelos acústicos.

El primer punto al crear los modelos acústicos es realizar el etiquetado de los fonemas en las diferentes grabaciones de audio que se llevaron a cabo en el punto anterior. El etiquetado de los archivos de audio se puede realizar con HSLab. El proceso de etiquetado consiste en buscar y marcar las posiciones de inicio y de fin de los fonemas que aparecen en la palabra o frase. HSLab nos guardará un fichero por cada archivo de audio etiquetado que deberemos guardar para utilizar posteriormente. Los ficheros generados tienen extensión *.lab* e indican los intervalos (inicio y final) de tiempo de cada fonema etiquetado. La siguiente imagen muestra el proceso de etiquetado utilizando HSLab. En ella, se puede apreciar que debajo de la señal acústica, aparecen una serie de etiquetas para cada fonema.



Una parte del contenido del fichero *.lab* generado para la imagen anterior se corresponde a: (se puede ver el fichero completo en el Anexo de esta memoria)

```
6349375 7361250 f
7361250 8272500 i
8272500 9537500 x
9542500 10150000 a
10150000 10576250 l
10583750 11306875 a
. . .
```

A la hora de etiquetar, podemos crear tantos archivos *.lab* como deseemos, pero debemos tener en cuenta que todos los fonemas deben estar etiquetados y que deben estarlo como mínimo tres veces. Cuantas más etiquetas de un mismo fonema tengamos, mayor será el porcentaje de éxito que podremos obtener a la hora de reconocer ese fonema, puesto que tendremos más variaciones del mismo y los vectores de características serán generados a partir de más datos de entrada.

Una vez etiquetados todos los fonemas crearemos un fichero prototipo para cada uno de los modelos o fonemas. El contenido del fichero se corresponderá con el prototipo de cada uno de los modelos acústicos y todos los modelos y por lo tanto todos los ficheros tendrán la misma estructura. El único modelo que no incluiremos ya que le daremos un tratamiento especial, es el del fonema *sp* que añadimos al final de cada palabra en nuestro diccionario. Un ejemplo del fichero prototipo para el modelo acústico "a", sería:

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "a"
 <BeginHMM>
    <NumStates> 5
        <State> 2
            <Mean> 39
                0.0 0.0 0.0 ...
            <Variance> 39
                1.0 1.0 1.0 ...
        <State> 3
            <Mean> 39
                0.0 0.0 0.0 ...
            <Variance> 39
                1.0 1.0 1.0 ...
        <State> 4
            <Mean> 39
                0.0 0.0 0.0 ...
            <Variance> 39
                1.0 1.0 1.0 ...
        <TransP> 5
             0.0 1.0 0.0 0.0 0.0
             0.0 0.6 0.4 0.0 0.0
             0.0 0.0 0.6 0.4 0.0
             0.0 0.0 0.0 0.7 0.3
             0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

Estos modelos se componen de 5 estados y como podemos ver se asignan los valores a los vectores (de medida 39) de las medias y las varianzas, para cada uno de los estados, excepto para el primero y el último, ya que estos dos estados se omiten. Inicialmente las

medias toman valor 0 y las varianzas valor 1. Finalmente se define una matriz de 5x5 que define las transiciones entre estados. Deberemos definir este archivo prototipo para cada uno de los modelos acústicos (fonemas) que hemos etiquetado en nuestra gramática y deberemos cambiar el nombre (definido en la opción "~h") para cada uno de ellos.

HInit es otra de las herramientas del sistema HTK que permite realizar una inicialización de los modelos acústicos. Con HInit utilizaremos los archivos de prototipo que acabamos de crear, pero antes debemos obtener los vectores de características para los archivos de audio grabados anteriormente. Para la inicialización de los modelos acústicos no vamos a obtener los vectores de características de todos los ficheros de audio, sino únicamente de aquellos en los que hayamos etiquetado alguno de sus fonemas.

Para obtener los vectores de características necesitamos dos ficheros más. El primero nos defina las rutas de nuestros ficheros de audio y las rutas donde guardaremos los ficheros de los vectores de características. El segundo fichero es el de configuración, donde se especifican los parámetros necesarios para llevar a cabo la codificación de los ficheros de audio a vectores de características:

El primer fichero es básicamente una lista de directorios (se corresponde al fichero ***codetr.scp*** de la orden ejecutada más abajo):

```
./audiofiles/s0003_0.htk ./train/s0003_0.mfc
./audiofiles/s0004_0.htk ./train/s0004_0.mfc
./audiofiles/s0007_0.htk ./train/s0007_0.mfc
./audiofiles/s0025_0.htk ./train/s0025_0.mfc
...
```

El fichero de configuración utilizado se corresponde con (se corresponde con el fichero ***config*** de la orden ejecutada más abajo):

```
#Parámetros de codificación
TARGETKIND = MFCC_0_D_A
```

```
TARGETRATE= 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F

#Si las grabaciones son WAV
#SOURCEKIND = WAVEFORM
#SOURCEFORMAT = WAV
#ZMEANSOURCE = TRUE
```

Si se ha utilizado otro software diferente al provisto por HTK para realizar las grabaciones se deben descomentar las últimas tres líneas, que permiten reconocer otros tipos de formato WAV.

Con la siguiente orden obtuvimos los vectores de características para la gramática:

```
HCopy -T 1 -C ../config -S codetr.scp
```

Con esto habremos obtenido la lista de vectores de características para los ficheros de la gramática que hemos utilizado para etiquetar los fonemas. Una vez tenemos los vectores de características ya podemos inicializar los modelos acústicos mediante HInit, que es lo que buscábamos desde que etiquetamos los fonemas.

```
HInit -T 1 -S train.scp -M proto/hmms/X -l X -L lab proto/protos/X
```

El fichero ***train.scp*** pasado como argumento contiene las rutas de los vectores de características creados anteriormente. La ruta definida por –M indica el directorio de trabajo donde se dejarán los ficheros de prototipo ya recalculados para cada modelo (recordemos que se le habían dado unos valores por defecto). –L indica la ruta donde se encuentran los archivos de etiquetado (los archivos *.lab* descritos anteriormente). X se corresponde a cada uno de los diferentes modelos acústicos, por lo que deberemos

sustituir X por cada uno de los fonemas que tengamos y ejecutar tantas veces el comando anterior como número de fonemas. (El modelo *sp* sigue sin incluirse en este punto).

4.2.5. Entrenamiento de los modelos acústicos

Es en este punto, una vez tenemos los vectores de características y los prototipos de los modelos, cuando podemos realizar el entrenamiento del sistema. El entrenamiento del sistema lo realizaremos con la ayuda de otra de las herramientas de las que nos provee HTK, en este caso HERest. El proceso de entrenamiento consiste en ir realizando una serie de reestimaciones sobre los ficheros de prototipos creados anteriormente. Cuando creamos los prototipos de los modelos, tuvimos que realizar el proceso para cada modelo por separado, en cambio, para realizar el entrenamiento, podemos realizar el proceso unificando todos los modelos en un único fichero.

Para empezar el entrenamiento, primero, crearemos un fichero llamado ***hmacs*** que contendrá todas las especificaciones de los prototipos de los modelos:

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "a"
...
~h "b"
...
~h "B"
...
```

A continuación deberemos realizar las transcripciones de los archivos de audio que hemos grabado previamente. Esta trascripción se realizará a nivel de palabras y a nivel de fonemas. Para obtener las transcripciones automáticamente, utilizaremos una utilidad de HTK que permite obtener un fichero de tipo MLF (*Maste Label File*), que permite juntar todas las transcripciones en un único fichero, en vez de tener cada trascripción en un fichero diferente. Esta utilidad es ***prompts2mlf***. Las trascripciones que debemos

generar deben ser sobre cada una de las grabaciones de audio (Recordemos que tenemos la lista completa en el fichero ***domoticaPrompts***).

```
perl prompts2mlf words.mlf domoticaPrompts
```

El fichero ***words.mlf*** es el fichero de salida, que contiene todas las transcripciones del fichero de entrada ***domoticaPrompts***. A continuación podemos ver una parte de este fichero generado:

```
#!MLF!#
"*/s001_0.lab"
FIJALATEMPERATURAA
VEINTE
.
"*/s002_0.lab"
ENCIENDE
LA-S
LUCES
.
"*/s003_0.lab"
FIJALATEMPERATURAA
VEINTITRES
.
"*/s004_0.lab"
APAGA
LA-S
CALEFACCION
.
"*/s005_0.lab"
ABRE
LA-S
VENTANA
DEL-A
COCINA
.
"*/s006_0.lab"
FIJALATEMPERATURAA
VEINTIUNO
…
```

Como podemos observar, en el fichero tenemos la trascripción de cada uno de los

archivos de audio a nivel de palabras aunque para pasos posteriores, necesitaremos también la trascripción a nivel de fonemas. Las trascripciones a nivel de fonemas las obtendremos con la herramienta HLEd. En este caso crearemos dos ficheros MLF; uno de ellos incluirá el fonema "*sp*" en sus trascripciones, mientras que el otro no lo incluirá. Para realizar este paso, necesitaremos el fichero generado anteriormente, ***words.mlf***, así como dos ficheros de configuración que nos permitirán definir que en un caso queremos el fonema "*sp*" y en otro de los casos no. Los ficheros de configuración los llamaremos ***mkphones0.led*** y ***mkphones1.led***. El contenido de estos ficheros de configuración se corresponde con:

```
#mkphones0.led
EX
IS sil sil
DE sp

#mkphones1.led
EX
IS sil sil
```

Como podemos ver, los ficheros anteriores contienen una serie de instrucciones. La instrucción EX indicará a HLEd que las palabras se deben expandir a nivel de fonemas. La instrucción IS permite añadir un modelo al principio y al final de cada frase, en este caso, añadimos el modelo "*sil*", puesto que nuestras órdenes irán precedidas de silencios para ser interpretadas. La instrucción DE elimina todas las repeticiones de un fonema. Para el primer fichero de configuración eliminaremos todas las ocurrencias del modelo "*sp*", mientras que para el segundo fichero, no realizaremos modificación alguna en este aspecto, de ahí que el primer fichero contenga la orden DE. Una vez disponemos de los ficheros de configuración y del fichero ***words.mlf*** que contiene las trascripciones a nivel de palabras, podemos obtener la trascripción final a nivel de fonemas. Ejecutamos:

```
HLEd –l '*' –d domoticaDict –i phones0.mlf mkphones0.led words.mlf
HLEd –l '*' –d domoticaDict –I phones1.mlf mkphones1.led words.mlf
```

Para obtener las trascripciones hemos pasado como parámetro nuestro diccionario, donde tenemos definida la trascripción de cada palabra a nivel de fonemas. El resultado

se ha escrito en los ficheros ***phones0.mlf*** y ***phones1.mlf*** respectivamente. El contenido de estos ficheros es como sigue:

```
#!MLF!#
"*/s001_0.lab"
sil
f
i
x
a
l
a
t
e
m
p
e
r
a
t
u
r
a
sp
b
e
j
n
t
e
sp
sil
.
"*/s002_0.lab"
sil
e
n
T
j
e
n
d
e
```

```
sp
l
a
sp
l
u
T
e
s
sp
sil
.
. . .
```

El ejemplo de fichero anterior se corresponde con el fichero ***phones1.mlf*** y podemos ver como en este se tiene en cuenta el modelo "*sp*". Si observáramos el fichero ***phones0.mlf***, veríamos como el modelo "*sp*" no aparece en ninguna de las trascripciones.

Ahora debemos obtener los vectores de características de todas las grabaciones. Anteriormente, habíamos obtenido los vectores de características únicamente de aquellas grabaciones que habíamos etiquetado, pero para el entrenamiento es necesario obtener los vectores de todas y cada una de ellas. Para ello, seguiremos el mismo proceso realizado anteriormente. Crearemos un fichero ***codetr.scp*** que contendrá las rutas de las grabaciones y la salida donde se almacenarán sus vectores de características, igual que realizamos anteriormente, pero esta vez para todas ellas. Igualmente, rescataremos el fichero de configuración ya que nos vuelve a ser necesario. Para generar los vectores de características, ejecutamos:

```
HCopy –T 1 –C config –S codetr.scp
```

Finalmente, crearemos un fichero ***train.scp*** que contendrá las rutas de los vectores de características que acabamos de generar y dos ficheros llamados ***monophones0*** y ***monophones1***. Estos ficheros contendrán la lista de todos los modelos ordenados alfabéticamente. Como anteriormente, en el fichero ***monophones0*** eliminaremos el modelo "*sp*", mientras que en ***monophones1*** lo incluiremos.

```
a
b
B
d
e
f
. . .
```

En este punto podemos realizar las primeras aproximaciones para nuestro entrenamiento. Como hemos comentado utilizaremos la herramienta HERest, y realizaremos 6 aproximaciones de nuestros modelos, esto es, ejecutar 6 veces la siguiente orden, cambiando los directorios de origen y destino.

```
HERest –C config –S train.scp –I phones0.mlf –H hmm0/hmacs –M hmm1 monophones0
```

El directorio de origen es el definido en hmm0, que contiene el fichero creado anteriormente con el conjunto de prototipos (***hmacs***). El directorio de salida es el especificado por hmm1 y es donde se generará el nuevo fichero ***hmacs*** recalculado. Para realizar nuestra primera aproximación, cambiaremos hasta 6 veces estos directorios (de hmm0 a hmm6), obteniendo cada vez un fichero ***hmacs*** recalculado.

4.2.6. Inserción de los modelos "sp" y "sil", y reentrenamiento

El entrenamiento llevado a cabo en el punto anterior ha sido una primera aproximación sin incluir el modelo de pausa corta entre palabras ("*sp*"), ni el modelo de silencio a principio y final de orden ("sil"). El siguiente punto es insertar estos modelos en el sistema para que puedan ser reconocidos, el sistema sea más robusto y el reconocimiento de las órdenes sea más efectivo.

Para añadir el modelo "sp", añadiremos al ultimo fichero ***hmacs*** recalculado, un nuevo modelo "*sp*" de 3 estados, en el que el estado central será una copia del estado central del modelo sil. Añadiremos:

```
~h "sp"
<BEGINHMM>
```

```
    <NUMSTATES> 3
        <STATE> 2
        <MEAN> 39
            -1.425806e+01 -1.514490e+00 -2.348586e+00 5.470936e-01 ...
        <VARIANCE> 39
            6.063219e+00 2.295287e+00 3.860351e+00 5.096084e+00 ...
        <GCONST> 6.210821e+01
        <TRANSP> 3
            0.0 1.0 0.0
            0.0 0.7 0.3
            0.0 0.0 0.0
<ENDHMM>
```

A continuación utilizaremos la herramienta HHEd para añadir las transacciones extra necesarias para ligar el modelo sp al estado central del modelo sil. HHEd funciona de manera similar a HLEd, aplicando un conjunto de comandos a un script para modificar un conjunto de vectores de características. En este caso, el fichero de configuración que contiene las ordenes, ***sil.hed***, tendrá la siguiente estructura.

```
AT 2 4 0.2 {sil.transP}
AT 4 2 0.2 {sil.transP}
AT 1 3 0.3 {sp.transP}
TI silst {sil.state[3], sp.state[2]}
```

El comando AT añade transiciones entre la matriz de transiciones dada (2, 4) mientras que el comando TI crea un estado ligado llamado silst. Para finalmente integrar el nuevo modelo y las nuevas transiciones en nuestro conjunto del modelo general, ejecutamos:

```
HHEd -H hmm7/hmacs -M hmm8 sil.hed monophones1
```

Para generar los parámetros de nuestros modelos, esta vez ya hemos utilizado la lista de fonemas que incluye el modelo "*sp*". Ahora realinearemos el sistema dos veces más con HERest, igual que lo hemos hecho anteriormente, pero esta vez utilizaremos la lista de fonemas y la lista de transcripciones incluyendo "*sp*".

```
HERest –C config –S train.scp –I phones1.mlf –H hmm9/hmacs –M hmm10
monophones1
```

Ahora podemos realinear los datos de entrenamiento para incluir las palabras que tienen

más de una pronunciación, como LA-S o DEL-A. Realizaremos el realineamiento con la herramienta HVite utilizando los modelos obtenidos hasta ahora, así crearemos nuevas trascripciones para las palabras con más de una pronunciación.

```
HVite -l '*' -o SWT -b INICIO -b FINAL -C config -a -H hmm10/hmacs –I
aligned.mlf -m -y lab -I words.mlf -S train.scp domoticaDict monophones1
```

Con esto se ha obtenido una nueva trascripción a nivel de fonemas. Los resultados se han escrito en el fichero ***aligned.mlf***. Con este paso conseguimos que el reconocedor tenga en cuenta todas las diferentes pronunciaciones de las palabras del diccionario. En el siguiente ejemplo del fichero ***aligned.mlf*** podemos ver como se tienen en cuenta las diferentes variaciones, en las palabras con más de una pronunciación, en las transcripciones. Ahora tenemos LAS o LA en función de la oración que se está transcribiendo.

```
"*/s002_0.lab"
sil
e
n
T
j
e
n
d
e
sp
l
a
s
sp
l
u
T
e
s
sp
sil
.
. . .
"*/s008_0.lab"
```

```
sil
e
n
T
j
e
n
d
e
sp
l
a
sp
l
u
T
sp
sil
.
. . .
```

Una vez hecho esto, re-estimaremos los parámetros de los modelos dos veces más con HERest (igual que anteriormente pero modificando de nuevo los directorios de entrada y de salida.)

4.2.7. Modelos de triphones con estados ligados

Llegados a este punto, nuestro sistema podría considerarse entrenado. Hasta ahora hemos creado modelos acústicos que se corresponden con fonemas. Este tipo de modelos son independientes del contexto, puesto que no tienen en cuenta que fonema se ha sucedido o cuál puede venir posteriormente. Para mejorar la respuesta y eficacia de nuestro reconocedor, aplicaremos modelos de triphones a nuestros modelos. Los modelos de triphones, son conjuntos de tres fonemas. Para crear nuestros triphones, empezaremos convirtiendo las trascripciones de fonemas a trascripciones de triphones. Como en otros casos, vamos a necesitar un fichero de configuración para indicar que queremos convertir y que no. El fichero de configuración ***mktri.led*** contendrá las líneas:

```
WB sp
WB sil
TC
```

El comando WB define los símbolos *sp* y *sil*, como símbolos de unión entre palabras, mientras que TC convierte todos los símbolos, excepto los de unión entre palabras, a triphones. Una vez tengamos listo el fichero de configuración podremos obtener las trascripciones a nivel de triphones. Ejecutamos:

```
HLEd –n triphones1 –l ‘*’ –i wintri.mlf mktri.led aligned.mlf
```

La salida de esta orden nos da dos ficheros, el primero, ***triphones1***, contiene la lista de todos los triphones generados, mientras que ***wintri.mlf*** contiene las trascripciones a nivel de triphones. El contenido de wintri.mlf es ahora:

```
#!MLF!#
"*/s001_0.lab"
sil
f+i
f-i+x
i-x+a
x-a+l
a-l+a
l-a+t
a-t+e
t-e+m
e-m+p
m-p+e
p-e+r
e-r+a
r-a+t
a-t+u
t-u+r
u-r+a
r-a
sp
b+e
b-e+j
e-j+n
j-n+t
```

```
n-t+e
t-e
sp
sil
.
. . .
```

Mientras que ***triphones1*** contiene una lista de todos los diferentes triphones aparecidos en ***wintri.mlf***. Lo siguiente que debemos hacer es integrar los resultados obtenidos con nuestro modelo acústico. Algunos pasos atrás ya habíamos integrado los cambios de añadir los modelos de silencio, mediante HHEd. Esta vez necesitamos el fichero ***mktri.hed***, que se puede generar con otra de las utilidades de HTK (el script perl ***maketrihed***). El contenido de este fichero es:

```
CL triphones1
TI T_a {(*-a+*,a+*,*-a).transP}
TI T_b {(*-b+*,b+*,*-b).transP}
TI T_B {(*-B+*,B+*,*-B).transP}
TI T_d {(*-d+*,d+*,*-d).transP}
TI T_e {(*-e+*,e+*,*-e).transP}
...
```

Las órdenes que se especifican dentro de ***mktri.hed*** son CL, que es un comando de clonación y la orden TI que liga todas las matrices de transición en cada set de triphones.

```
HHEd –B –H hmm12/hmacs –M hmm13 mktri.hed monophones1
```

Con esto, hemos generado modelos de triphones a partir de las trascripciones de fonemas, y hemos integrado estos cambios a nuestro modelo. Todas las modificaciones sobre ficheros han sido realizadas en ficheros planos de texto, el problema es que cada vez los ficheros de reestimación son más pesados, cosa que puede afectar al rendimiento del reconocedor. En esta ultima ejecución hemos especificado la opción –B para que el resultado se almacene el formato binario, para mejorar así la velocidad de reconocimiento durante las ejecuciones. En este punto podemos re-estimar nuestro modelo dos veces más como ya hemos ido realizando hasta ahora, y prácticamente tendremos nuestro modelo acústico completamente entrenado. El último punto a

considerar es que debemos crear un modelo acústico que comparta estados (estados ligados), ya que hasta ahora, como hemos hecho en el paso anterior, únicamente comparten matrices de transición.

En los pasos anteriores, cuando hemos realizado las estimaciones de los modelos, en muchas de las varianzas habremos obtenidos resultados erróneos ya que no habrían suficientes datos asociados con muchos de los estados de los modelos. Por eso, el último paso es el de ligar estados con conjuntos de triphones, para que así compartan información y las estimaciones de parámetros (como la varianza) puedan ser más robustas. Uno de los métodos que se pueden utilizar es el basado en árboles de decisión. Este método se basa en realizar preguntas sobre los contextos izquierdos y derechos de cada triphon. Los árboles de decisión tratan de encontrar los contextos con una mayor diferencia acústica y para los que por tanto, se deben distinguir sus estados de otros triphones con acústicas más parecidas. Para poder ejecutar este proceso basado en árboles de decisión, necesitamos crear un último fichero de configuración, llamado ***tree.hed***. Este fichero puede ser generado, parcialmente, de manera automática con la utilidad ***mkclsscript***, aunque una parte del mismo deba ser escrita manualmente. El resultado del fichero debe ser:

```
RO 100.0 stats

TR 0

QS "L_a" {a-*"}
QS "R_a" {*+a"}
QS "L_b" {b-*"}
. . .
QS "L_x" {x-*"}
QS "R_x" {*+x"}

TR 2

TB 350.0 "ST_a_2_" {("a","*-a+*","a+*","*-a").state[2]}
TB 350.0 "ST_b_2_" {("b","*-b+*","b+*","*-b").state[2]}
TB 350.0 "ST_B_2_" {("B","*-B+*","B+*","*-B").state[2]}
TB 350.0 "ST_d_2_" {("d","*-d+*","d+*","*-d").state[2]}
TB 350.0 "ST_e_2_" {("e","*-e+*","e+*","*-e").state[2]}
```

```
. . .
TB 350.0 "ST_s_4_" {("s","*-s+*","s+*","*-s").state[4]}
TB 350.0 "ST_sil_4_" {("sil","*-sil+*","sil+*","*-sil").state[4]}
TB 350.0 "ST_sp_4_" {("sp","*-sp+*","sp+*","*-sp").state[4]}
TB 350.0 "ST_t_4_" {("t","*-t+*","t+*","*-t").state[4]}
TB 350.0 "ST_T_4_" {("T","*-T+*","T+*","*-T").state[4]}
TB 350.0 "ST_u_4_" {("u","*-u+*","u+*","*-u").state[4]}
TB 350.0 "ST_x_4_" {("x","*-x+*","x+*","*-x").state[4]}

TR 1

CO "tiedlist"
ST "trees"
```

Este fichero contiene instrucciones sobre que contextos examinar para que compartan estados. Los comandos TB son los encargados de tomar decisiones sobre los grupos de estados que se van a formar, y son los generados automáticamente con la utilidad mencionada anteriormente. Los comandos QS, son las preguntas sobre el contexto izquierdo y derecho, y deben ser escritas por el usuario. Todos los comandos QS siguen la misma estructura. Definen cada uno de los modelos tanto por izquierda (L) como por derecha (R)

```
Pregunta de contexto por la izquierda: QS "L_a" {a-*"}
Pregunta de contexto por la derecha: QS "R_a" {*+a"}
```

Las últimas dos líneas del fichero, definen el fichero de salida con todos los triphones con estados ligados, ***tiedlist***, y un fichero para cálculos de uso interno, ***trees***. Una vez definido el fichero ***tree.hed***, ejecutaremos la instrucción que nos generará los modelos de triphones con matrices de transición y estados compartidos.

```
HHEd -B -H hmm15/hmacs -M hmm16 tree.hed triphones1 > log
```

El resultado de esta instrucción se guardará en ***tiedlist***, que contiene la lista de todos los modelos acústicos resultantes y de los modelos de triphones con estados y matrices de transición ligados. Ahora podemos realizar dos reestimaciones más con HERest obteniendo nuestros modelos acústicos, y finalizando el entrenamiento del sistema. El último fichero ***hmacs*** re-estimado, contiene nuestros modelos acústicos correctamente

entrenados y es el fichero que procesaremos junto con las señales de entrada del micrófono para reconocer las órdenes pronunciadas por el usuario. Debemos tener en cuenta, que para que el fichero de modelos pueda ser entendido por Julius, debemos ejecutar la última reestimación en modo no binario (quitando la opción –B).

4.2.8. Programación del reconocedor de voz

Una vez obtenidos los modelos acústicos ya podemos pasar a desarrollar la aplicación que nos realizará el reconocimiento de voz. Para conseguir decodificar la señal de entrada realizaremos una aplicación utilizando la API de Julius. Julius nos permite obtener datos de entrada tanto directamente por micrófono, como a partir de grabaciones en formato WAV, simplemente cambiando parámetros de configuración. Cabe decir, que en nuestro caso, utilizaremos la primera opción, ya que al tratarse de un controlador de domótica, las acciones se deben procesar y realizar en el mismo momento en que el usuario solicita la orden. Esta API incorpora un conjunto de funciones que permite realizar reconocimientos de voz sobre modelos acústicos desarrollados y entrenados con HTK.

La estructura de la aplicación, sin entrar en la comunicación con el simulador, debía ser sencilla. Iniciar un dispositivo de entrada por el que obtener las señales acústicas (distinguiendo el tipo de reconocimiento que queremos efectuar en caso de ser necesario) y una vez obtenida la señal de entrada, procesarla contra los modelos acústicos y mostrar la información de salida necesaria. Para comparar los modelos acústicos de nuestro sistema con los modelos de la señal de entrada es necesario disponer de tanto la gramática como el diccionario en un formato específico que pueda ser reconocido por Julius. El primer paso, pues, es realizar esta conversión, de la gramática en formato HTK a la gramática en formato Julius. Para ello necesitamos los archivos originales del diccionario y de la gramática (**domoticaWdnet** y **domoticaDict**) y los renombraremos de la siguiente manera:

```
domoticaWdnet : domotica.slf
domoticaDict : domotica.htkdic
```

En el caso de ***domoticaDict***, deberemos eliminar los fonemas *sp* de cada trascripción de palabra para que Julius no lo tome como un fonema más. Si no eliminamos *sp*, obtendremos errores al iniciar el simulador. Una vez hecho esto, editaremos la utilidad SLF2DFA (Script que convierte de gramática SLF a DFA) para que contenga las líneas:

```
determinize_bin=/home/bin/julius-4.0/gramtools/dfa_determinize/dfa_determinize
minimize_bin=/home/bin/julius-4.0/gramtools/dfa_minimize/dfa_minimize
```

Y ejecutaremos la orden:

```
./../slf2dfa-1.0/SLF2DFA.sh domotica
```

Este comando busca los ficheros slf y htkdic renombrados anteriormente y realiza las conversiones a la gramática en formato Julius. Una vez ejecutado el comando obtendremos tres ficheros: ***domotica.dfa***, ***domotica.dict*** y ***domotica.term***. Estos tres ficheros, junto con los obtenidos previamente (***hmacs*** y ***tiedlist***), son los que nos permitirán comparar los modelos acústicos con las señales de entrada.

Al utilizar la API de Julius, debemos construir un fichero de configuración con una serie de parámetros necesarios para efectuar las comparaciones de modelos (Especificar el fichero de la gramática, el del diccionario, los modelos acústicos, etc.). A continuación se describe el fichero de configuración utilizado y sus parámetros.

```
#Fichero de configuración para reconocimiento de voz
#-dfa y –v especifican los ficheros de la gramática y del diccionario
-dfa domotica.dfa
-v domotica.dict
#Fichero que contiene los modelos acústicos en formato HTK
-h hmacs
#Si se ha utilizado un modelo de triphones, se debe especificar el fichero que
#contiene las referencias de los triphones. Se utiliza para mapear los
#fonemas.
-hlist tiedlist
#Penalti de inserción entre palabras para la primera y segunda pasada
```

```
-penalty1 5.0
-penalty2 20.0
#Selecciona el método para aproximar triphones entre palabras al principio y
#al final de una palabra
-iwcd1 max
#Utiliza algoritmo gausiano. Safe indica que se utiliza de manera estándar.
-gprune safe
-b2 200
-sb 200.0
#Habilita el uso de pausas cortas entre palabras
-iwsp
-iwsppenalty -70.0
#Frecuencia de los ficheros de entrada
-smpFreq 16000
#Tipo de entrada por la que se obtendrán los datos. En este caso micrófono
-input mic
#Si la entrada es por fichero, reconoce los ficheros indicados en el archivo
#filelist
-filelist filelist
```

El software reconocedor de voz se ha programado con el lenguaje de programación C. En este caso estaba más restringido puesto que la librería de Julius se encuentra escrita con este mismo lenguaje. Como comentábamos al principio de este punto el reconocedor debía ser capaz de leer el fichero de configuración de Julius, abrir una comunicación con el micrófono para obtener las señales de entrada y procesar los datos obtenidos para dar resultados.

Para leer el fichero de configuración de Julius bastaba con utilizar una de las funciones de la API. La función ***j_config_load_file_new*** abre y lee un fichero de configuración especificado, y guarda la información en una estructura de tipo Jconf. Los datos contenidos en esta estructura se pueden usar luego para pasar datos entre las distintas funciones del software. Una vez cargado el fichero, se crea una instancia de reconocedor de voz a partir de la estructura de configuración. La instancia del reconocedor se crea con la función ***j_create_instance_from_jconf***. Cuando ambas ejecuciones han ido bien, hemos conseguido cargar la información de nuestro reconocedor de voz. Ahora, sobre la instancia de tipo Recog que se ha creado, podemos asociar tanto el dispositivo de entrada, como una serie de callbacks, que se ejecutan a

medida que se efectúan distintas acciones sobre el reconocedor (cuando este empieza a reconocer datos, cuando acaba, etc.)

Para asociar el micrófono al reconocedor, utilizamos la función ***j_adin_init***. Esta función se encarga de abrir un canal hacia el dispositivo de entrada especificado en la configuración, ya sea un micrófono, un fichero de texto, etc. Ahora solo basta definir que tipo de procesamiento vamos a efectuar. Para ello podemos distinguir entre las distintas entradas:

```
if (jconf->input.speech_input == SP_RAWFILE)
{
…
}
else if (jconf->input.speech_input == SP_MIC)
{
…
}
else if (jconf->input.speech_input == SP_MFCFILE)
{
…
}
```

En función de la entrada, el proceso de comparación se debe efectuar de una u otra manera, por eso debemos distinguir que tipo de entrada se ha configurado. *SP_RAWFILE* permite realizar reconocimiento sobre ficheros WAV, *SP_MIC* permite realizar reconocimiento de voz en vivo a través de un micrófono y *SP_MFCFILE* permite realizar el reconocimiento a partir de ficheros de vectores de características (Los ficheros MFC creados mediante HTK). En nuestro caso, puesto que queremos capturar el audio en vivo y procesarlo, deberemos crear un stream de entrada (***j_open_stream***) y sobre este stream, llamar a nuestra instancia de reconocimiento a través de la función ***j_recognize_stream***. Si no se corta el flujo del programa por causa del algún error y hemos definido correctamente los callbacks, podemos obtener la información del reconocimiento justo cuando se acaba de procesar la orden pronunciada. A partir de la instancia de reconocimiento (Recog), podemos acceder al objeto RecogProcess que contiene toda la información del proceso, como la secuencia de palabras reconocida o los errores de reconocimiento que se hayan podido suceder.

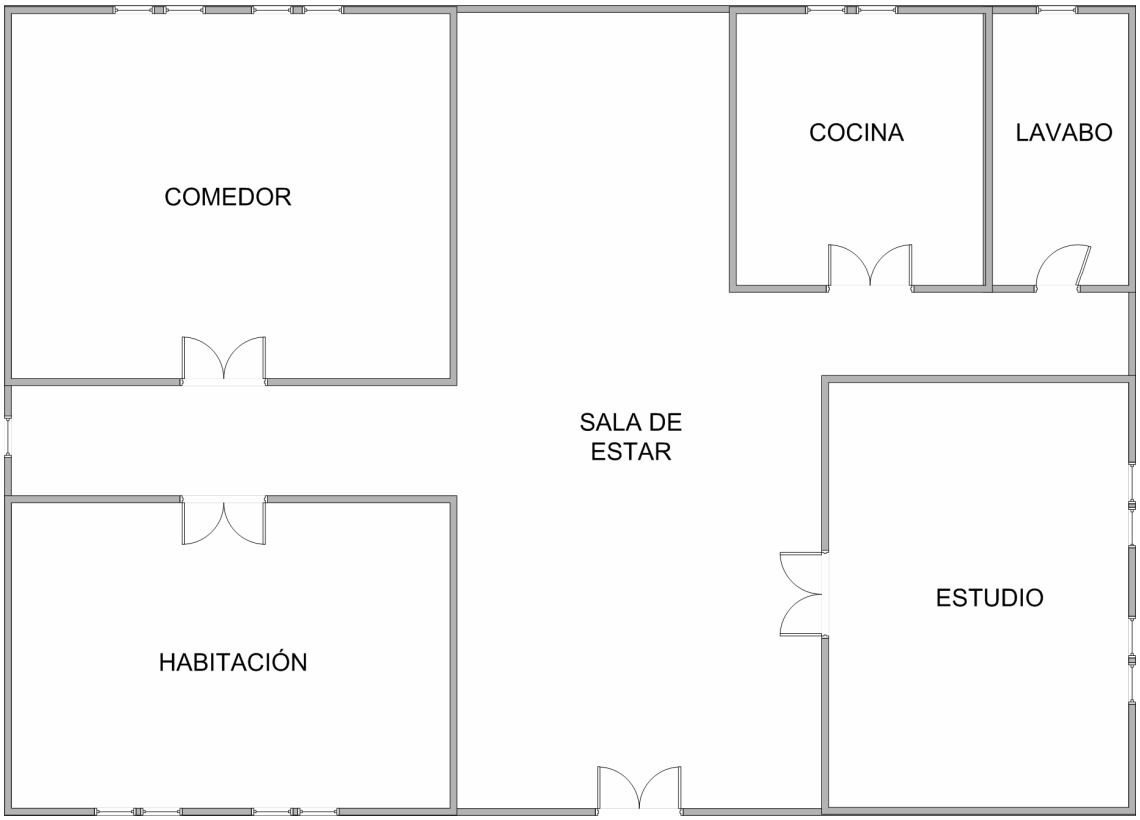
4.3 Simulador

El simulador es el otro elemento principal del proyecto y es donde se pueden ver los resultados de las órdenes enviadas por el usuario. El simulador ha sido desarrollado utilizando openGL y GLUT sobre Windows y Visual C++. OpenGL es uno de los sistemas no propietarios y *open source* utilizados para desarrollar mundos, objetos o escenas virtuales, y aunque son necesarias bastantes horas de dedicación para entender muchos de sus puntos, es fácil encontrar información y ejemplos sencillos. La elección de plataforma y lenguaje han venido condicionados por la necesidad de incluir una librería de conversión de modelos 3ds a openGL. La librería en cuestión necesitaba incluir referencias a librerías de Windows, por lo que a diferencia del reconocedor de voz, el simulador ha sido implementado finalmente para este sistema operativo. La separación de sistemas operativos permite ver con más claridad la separación existente entre el reconocedor y los actuadores (en este caso los actuadores se corresponden con el simulador 3D), como pasa en un sistema domótico real.

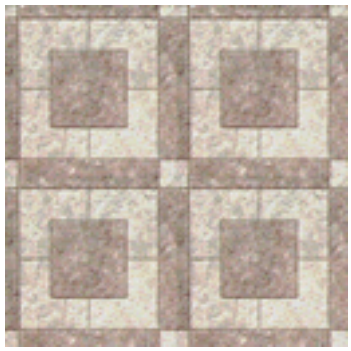
El objetivo de este simulador era el de desarrollar un entorno virtual que representara una casa sobre la que el usuario, mediante el uso del teclado, se pudiera desplazar, y a partir del reconocedor de voz observar las acciones ejecutadas en forma de animaciones o modificaciones sobre un panel de control.

El primer paso consistía en dibujar "los planos" de la casa que queríamos representar, de manera que todas las sentencias que nuestro reconocedor de voz es capaz de reconocer pudiesen verse reflejadas. Para poder reflejar correctamente cada una de las sentencias, debíamos como mínimo, añadir una sala por cada una de las diferentes estancias nombradas en la gramática. De esta forma obtuvimos una casa estándar, de carácter pequeño. Una vez dibujados los planos de la casa se debían representar en la escena 3D, de manera que pudiésemos ver las paredes que delimitan cada una de las habitaciones. La representación de la casa dentro de la escena 3D se llevó a cabo añadiendo a la escena cuadrados utilizados para dibujar las paredes. Lo mismo se hizo

para el techo y el suelo, y se utilizaron triángulos para reproducir el tejado exterior. A continuación se muestra un plano donde se puede apreciar la planta de la casa y la distribución que tiene.



El siguiente paso, era dotar a las paredes de la casa de algo de vida, utilizando texturas. Las texturas son imágenes que se cargan en el entorno 3d y que se aplican sobre los objetos para dotarlos de cierta sensación de realidad. Se aplicaron texturas para las paredes, las puertas y el suelo de la casa, por lo que respecta al interior, y una textura de hierba para el suelo del exterior de la casa. Las texturas se encuentran también dentro del directorio texturas, que encontramos en el mismo directorio que el ejecutable del simulador. Las texturas se cargan al iniciar la aplicación para poder ser utilizadas durante el renderizado de la escena. Un ejemplo de texturas utilizadas es:




Una vez concluida la estructura de la casa se utilizó una librería para cargar objetos 3ds y así poder dotar a la escena de un mayor grado de realidad. Utilizando esta librería se cargaron diferentes muebles y complementos en las distintas estancias. Los objetos 3ds utilizados se encuentran en la carpeta objetos, dentro del mismo directorio en el que se encuentra el ejecutable principal del simulador, junto con los ficheros de texturas que utilizan los mismos objetos. Los objetos 3ds se cargan en tiempo de ejecución al arrancar el programa. Una vez cargados se dibujan y renderizan en pantalla.

Teniendo la escena que queríamos representar llegaba la hora de aplicar las animaciones que responderían a las ordenes de voz. Como se ha visto anteriormente hay órdenes de apertura y cierre de puertas y ventanas y otras que controlan elementos como las luces, la calefacción o la temperatura. Para las puertas y ventanas se dejaron espacios sobre los que dibujarlas. Al ejecutar una orden de voz relacionada con la apertura o el cierre de puertas y ventanas se activan las variables necesarias y se puede observar como se abren o se cierran las puertas según sea el caso. Para conseguir este efecto era necesario utilizar una variable que controlara el ángulo de la puerta o la ventana incrementado o decrementando su valor según fuese necesario y otra variable para comprobar si una puerta ya se encuentra abierta o cerrada. Para el resto de órdenes, como no se encontró una manera clara de representar los cambios se incorporó un panel de control sobre la escena. Este panel muestra el estado de los elementos como el aire acondicionado, la temperatura o el estado de las luces. A continuación se puede ver una imagen del simulador 3D, donde se aprecia el panel de control y una de las puertas de las salas abriéndose, así como el resultado de aplicar objetos y texturas a la escena.

Como añadido al panel de control, se introdujo en la parte inferior izquierda de la pantalla del simulador, una sección donde se puede ver la última orden enviada desde el reconocedor de voz. El paso de mensajes desde el reconocedor de voz hasta el simulador se comenta con más detalle en el siguiente punto de este mismo capítulo.

A continuación se comenta la estructura que tiene la implementación del simulador y como se han separado los distintos ficheros de código según su función.

La estructura básica del simulador es bastante sencilla, aunque el código es bastante extenso debido a la necesidad de dibujar la escena tridimensional. Como cualquier aplicación en C, el punto de arranque se encuentra en la función ***main***, dentro del fichero ***domotica3d.cpp***. Dentro de esta función se realizan todas las llamadas que permiten tanto iniciar la escena como interactuar ella (moverse a través del entorno tridimensional, por ejemplo). La librería GLUT nos ofrece una serie de mejoras sobre

las librerías openGL y define una serie de funciones que nos permiten iniciar el sistema de ventanas o los controles de teclado, de manera más sencilla, independientemente del sistema operativo sobre el que se trabaje.

Así pues, dentro de la función principal de nuestra aplicación encontramos una serie de llamadas a funciones GLUT:

- **glutInit(&argc, argv)**: Inicializa la librería glut. Esto nos permite utilizar las funciones definidas en la librería Glut, dentro de nuestro aplicación.
- ***glutInitWindowSize(1024, 768)***: Inicializa el tamaño inicial de la ventana de la aplicación.
- ***glutInitWindowPosition(100, 100)***: Inicializa la posición inicial de la ventana de la aplicación.
- ***glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUR_RGBA)***: Inicializa el modo de visualización, es decir, como y con que propiedades se renderiza la escena. En este caso, concretamente, iniciamos la visualización con buffer de profundidad, doble buffer para dibujar y modo de ventana RGBA.
- ***glutCreateWindow("")***: Inicializa una ventana de aplicación. Esta ventana contendrá la escena que se renderizará más tarde.
- ***glutSetWindowTitle("…")***: Establece el titulo de la ventana.
- ***CargarModelos3DS()***: Esta función no forma parte de la librería glut, sino que es una función propia. Se encarga de iniciar todos los modelos 3ds de los objetos que no son primitivas simples (cuadrados o triangulos) y que aparecen en la escena.
- ***CargarTexturasEscena()***: Esta función no forma parte de la librería glut, sino que es una función propia. Se encarga de realizar la carga de todas las imágenes que se utilizarán como texturas.
- ***DefineLights()***: Igual que la anterior, no forma parte de la librería Glut. Inicializa las luces de la escena.
- ***glutReshapeFunc(reshapeEvent)***: Asocia una función de tipo callback para los

eventos de cambio de tamaño. Cuando se produzca uno de estos eventos, será gestionado por la función reshapeEvent.

- ***glutDisplayFunc(displayEvent)***: Asocia una función de tipo callback para dibujar la escena. Esta función se llama a cada vuelta para repintar la escena. Contiene todas las llamadas openGL de dibujo.
- ***glutSpecialFunc(specialKeyEvent)***: Asocia una función de tipo callback para gestionar la pulsación de teclas especiales del teclado (F1-F12, teclas de dirección, insert, end, etc.)
- ***glutKeyboardFunc(normalKeyEvent)***: Igual que la función anterior, pero permite gestionar todo el conjunto de teclas no incluido en las teclas especiales.
- ***glutMainLoop()***: Inicializa el bucle principal de la aplicación en el que se procesan todos los eventos anteriores cada vez que se recibe alguno. La aplicación openGL se queda en este bucle hasta que salimos de ella.

A partir de estas funciones se puede acceder a todo el código que genera la escena, aunque existen una serie de "librerías" que gestionan algunos aspectos en concreto. Uno de estos aspectos es la carga de modelos 3DS que se ha comentado en uno de los puntos anteriores. Crear un modelo concreto en openGL, como por ejemplo un sofa, requiere una dedicación de muchas horas (de ahí que existan editores externos de 3D que permiten realizar estos objetos de manera más sencilla), por esto se ha utilizado una librería que permite leer ficheros 3ds para más tarde añadirlos a la escena. La función que se encarga de leer estos ficheros es la función ***Load3DSModel***. Esta función se apoya sobre las librerias ***3ds.h*** y ***3DSobj.h***. La primera, permite realizar el paso de un fichero 3ds a un objeto openGL, mientras que la segunda es la referencia en sí del objeto openGL.

Igual que para la gestión de objetos 3d tenemos las librerias y funciones comentadas anteriormente, también se ha separado un modulo para la carga de las texturas que van a utilizarse dentro de la escena (y de las que van a utilizar los objetos). Estas funciones están definidas en la librería ***texturas.h***. Básicamente consta de dos funciones, una en la que se definene todas las texturas a cargar (***CargarTexturaEscena***) y otra que se utiliza

para cargar las texturas tanto de la escena como de los objetos (***CrearTextura***).

Las comunicaciones se definen en la librería ***comms.h***. En esta librería se encuentra las funciones de inicio del socket para el simulador y las funciones de recepción de mensajes. Al recibir un mensaje se modifican las variables necesarias para que al redibujarse la escena se puedan apreciar los cambios. Estas variables se encuentran declaradas como extern, por lo que pueden ser utilizadas por otra librería que incluya a ***comms.h***.

Por último, se ha comentado que la función desde la que se dibuja toda la escena es la función de callback ***displayEvent***, pero esta función se apoya sobre la librería ***render.h***, que contiene todas las funciones encargadas de dibujar realmente todos los elementos en pantalla.

4.4 Comunicación Reconocedor de voz - Simulador

Tanto el reconocedor de voz como el simulador son aplicaciones totalmente separadas, por lo que era necesario buscar una manera de que estas se comunicaran entre si para pasarse la información necesaria. Los sistemas domóticos están dotados de sensores, que envían información al ordenador central, que actúa en consecuencia de la información recibida. Normalmente, los tipos de transmisión utilizados en sistemas domóticos, son protocolos sobre línea de corriente (Como el protocolo X10). Estos, son protocolos de comunicaciones para el control remoto de dispositivos eléctricos que utilizan la línea eléctrica para transmitir señales de control. En este caso era necesario enviar información entre el reconocedor de voz y el simulador, de manera que una vez reconocida la orden el simulador supiese que debía ejecutar, por lo que había que buscar otro tipo de comunicación más aplicable al entorno informático sobre el que tratan ambas aplicaciones.

Finalmente me decidí por sockets, puesto que es una manera sencilla de conectar aplicaciones que necesitan enviarse información y la estructura de cliente – servidor que montan era perfecta para la relación existente entre el reconocedor de voz y el simulador. En este caso el reconocedor de voz actúa como cliente y se conecta al simulador, que actúa como servidor. Además, el paso de mensajes entre ambos clientes a través de sockets, se realiza de manera eficaz, por lo que uno de los puntos tratados en el análisis de requerimientos, sobre velocidad en el paso de mensajes, queda resuelto.

El reconocedor de voz corre bajo Linux, por lo que utilizar sockets tan solo requería importar la librería del sistema dedicada a este fin, ***sys/socket.h***, y realizar todas las inicializaciones necesarias para conectar con el servidor. En el simulador, en cambio, desarrollado bajo Windows en Visual C++, no existe la librería estándar de c para sockets. En el simulador por tanto, se utiliza la librería propia de Windows, ***winsock***.

Para realizar la comunicación entre ambos entornos únicamente era necesario, desde el cliente, abrir una conexión al servidor y que el servidor se encontrase esperando una petición de conexión desde el propio cliente. Una vez la comunicación esta establecida, el cliente envía los mensajes de las órdenes reconocidas de manera que el servidor las reciba y las pueda interpretar para obrar en consecuencia.

Comunicación cliente > servidor:

```
…
//Crear el socket cliente
if ((clientsock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
…
```

```
//Obtener referencia al servidor
if ((server = gethostbyname("192.168.1.2")) == NULL)
…
//Conectar el cliente al servidor
If (connect(clientsock, &serverAddress, sizeof(serverAddress)) < 0)
…
```

Inicio del servidor para esperar peticiones de conexión:

```
…
//Crear socket del servidor
socketServer = ::socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
…
//Iniciar la escucha en el Puerto definido por el socket
if (::listen(socketServer, 1) == SOCKET_ERROR)
…
//Aceptar petición de conexión (cuando se produzca)
socketClient = ::accept(socketServer, NULL, NULL);
…
```

Los mensajes que envía el cliente hacia el servidor consisten en la orden reconocida incluida entre dos tags de inicio y de fin, de esta manera, el servidor puede saber que orden se ha reconocido y ejecutar la acción correspondiente.

```
<msg>ENCIENDEELAIREACONDICIONADO</msg>
<msg>FIJALATEMPERATURAAVEINTE</msg>
```

4.5 Test

Una vez realizado el entrenamiento de la gramática, era necesario realizar las pruebas pertinentes para comprobar la respuesta de la misma. Así pues, antes de empezar con el desarrollo del reconocedor se realizaron pruebas sobre la gramática utilizando una de las herramientas de HTK, en este caso, HVITE. Esta utilidad recibe como argumentos todos aquellos ficheros que más tarde serán necesarios para el reconocimiento de la voz a partir del API de Julius, como el diccionario (***domoticaDict***) o la red de palabras (***domoticaWdnet***). Se puede ejecutar tanto para reconocer la voz a partir de ficheros pregrabados, o para reconocer la voz en vivo. En nuestro caso, las pruebas se han

realizado en vivo, que es como el reconocedor va a funcionar una vez implementado. Al hablar por el micrófono se obtiene el resultado en pantalla que indica la hipótesis final obtenida. Una muestra de la salida por pantalla es:

```
READY[1]>
INICIO ENCIENDE LA-S LUZ FINAL  ==  [404 frames] -73.4434 [Ac=-29671.1 LM=0.0]
(Act=272.8)

READY[2]>
INICIO ENCIENDE LA-S LUCES FINAL  ==  [170 frames] -79.8758 [Ac=-13578.9
LM=0.0] (Act=228.4)

READY[3]>
INICIO ABRE LA-S PUERTA DEL-A SALON FINAL  ==  [339 frames] -72.3764 [Ac=-
24535.6 LM=0.0] (Act=266.6)

READY[4]>
INICIO ABRE LA-S PUERTA DEL-A LAVABO FINAL  ==  [285 frames] -73.5581 [Ac=-
20964.1 LM=0.0] (Act=259.3)

READY[5]>
INICIO CIERRA LA-S PUERTA DEL-A LAVABO FINAL  ==  [223 frames] -75.3025 [Ac=-
16792.5 LM=0.0] (Act=246.6)

READY[6]>
INICIO ABRE LA-S VENTANA DEL-A LAVABO FINAL  ==  [213 frames] -79.4076 [Ac=-
16913.8 LM=0.0] (Act=243.9)

. . .
```

Las pruebas se realizaron sobre todas las frases de nuestra gramática, obteniendo un único fallo en una de las oraciones de temperatura. Al repetir la oración el resultado fue satisfactorio, por lo que algún sonido externo (ruido ambiente) o una mala captación de la señal por parte del micrófono (debido a la posición del hablante) pudieron producir el resultado erróneo.

Una vez realizadas las pruebas en vivo, utilizamos la herramienta *HResults* de HTK para obtener la tasa de acierto de la gramática. *HResults* funciona a partir de una serie de grabaciones de audio con oraciones de la gramática. Los ficheros son procesados obteniendo las hipótesis finales igual que para las pruebas en vivo. Una vez se han

obtenido las hipótesis finales escribe un fichero de resultado con el porcentaje de aciertos. Para realizar esta prueba se grabaron 10 nuevas oraciones, puesto que no tendría sentido utilizar las empleadas para el entrenamiento del sistema, y se ejecutó:

```
HVite -H hmm18/hmacs -C config -S test.scp -l '*' -i
results/domoticaResultado.mlf -w domoticaWdnet -p 0.0 -s 5.0 domoticaDict
tiedlist
```

En ***domoticaResultado.mlf*** se escribe el resultado con las hipótesis finales obtenidas para cada una de las grabaciones. Un trozo de este fichero es:

```
#!MLF!#
"*/test1_0.rec"
7200000 20400000 FIJALATEMPERATURAA -10931.764648
20400000 26700000 VEINTE -5249.232422
.
"*/test2_0.rec"
11400000 14900000 ABRE -2873.425537
14900000 16000000 LA-S -870.685974
16000000 20400000 PUERTA -3805.590088
20400000 21700000 DEL-A -1167.404297
21700000 28000000 COMEDOR -5709.369629
.
. . .
```

Ahora se puede ejecutar *HResults* para obtener el informe sobre la gramática.

```
HResults -I testref.mlf tiedlist results/domoticaResultado.mlf >
results/domoticaInforme.mlf
```

El fichero ***testref.mlf*** contiene las trascripciones originales de las grabaciones, de manera que se puedan comparar con las hipótesis obtenidas en ***domoticaResultado.mlf*** anteriormente. En ***domoticaInforme.mlf*** se graba el informe. A continuación podemos ver como el 100% de las frases se ha traducido correctamente por lo que el funcionamiento de la gramática es muy bueno.

```
====================== HTK Results Analysis =======================
  Date: Fri Sep  4 16:18:28 2009
```

```
  Ref : testref.mlf
  Rec : results/domoticaResultado.mlf
----------------------- Overall Results --------------------------
SENT: %Correct=100.00 [H=10, S=0, N=10]
WORD: %Corr=100.00, Acc=100.00 [H=33, D=0, S=0, I=0, N=33]
================================================================
```

Por ultimo, una vez comprobado el funcionamiento de la gramática, las pruebas de integración entre el reconocedor de voz y el simulador se realizaron iniciando ambas aplicaciones y ejecutando diferentes ordenes, viendo como se producían los cambios en el simulador que indicaban que la orden se había reconocido correctamente.

Capítulo 5

Conclusiones

Una vez llegados al final del desarrollo, debemos considerar si los objetivos que se buscaban al plantear el proyecto han sido correctamente solucionados y si se ha conseguido un "producto" satisfactorio, así como dar una serie de valoraciones críticas sobre lo que finalmente ha conseguido aportar este proyecto.

Para empezar, hay que destacar que todos los objetivos descritos anteriormente durante la memoria, se han podido llevar a cabo, obteniendo los resultados esperados. La creación de la gramática, el entrenamiento del sistema, el reconocimiento de las órdenes, el paso de información al simulador y la ejecución final son todos los objetivos que se han conseguido plasmar en el proyecto de manera satisfactoria, obteniendo un sistema que realiza su objetivo principal, ser capaz de reconocer las órdenes dadas por un usuario. Uno de los puntos clave, la creación de la gramática y el entrenamiento de los patrones de voz, ha supuesto un gran esfuerzo a la hora de ser implementado, debido en parte a la necesidad de aprender un entorno y una serie de utilidades nuevas hasta ahora. Igualmente hay que decir, que este esfuerzo ha sido ampliamente recompensado con los resultados obtenidos, encontrándome ahora capaz de desarrollar aplicaciones totalmente distintas que se basen en el reconocimiento de voz.

Lo que buscaba en este proyecto era aprender una serie de tecnologías nuevas, tanto a nivel de interacción con el usuario, como de programación, y creo que esto lo he podido conseguir mezclando las distintas tecnologías utilizadas. Aunque si bien es cierto que gracias a asignaturas cursadas durante el último curso, ya tenía nociones de openGL, el desarrollo del simulador me ha permitido profundizar más en algunos temas que no se comentaron en la asignatura, como la integración de una escena, movimientos o animaciones, etc.

Finalmente, considero que este proyecto me ha aportado tanto una serie de conocimientos informáticos nuevos, como la habilidad de recoger y buscar la información necesaria para solucionar los problemas que iban apareciendo a lo largo del desarrollo.

5.1 Mejoras futuras

Una vez concluido el desarrollo del proyecto, podemos considerar una serie de mejoras aplicables en un futuro, que podrían mejorar tanto la tarea principal del sistema (el reconocimiento de voz) como la interacción del usuario con el simulador. Estas mejoras no son imprescindibles para el funcionamiento del proyecto presentado aunque son puntos interesantes a tener en cuenta, por eso no se han llevado a cabo durante el desarrollo (principalmente por falta de tiempo). Algunas de estas mejoras podrían ser la ampliación de la gramática y por lo tanto de las órdenes que es posible ejecutar, velocidad del simulador o crear un estado en el que vayan todas las palabras que no forman parte de la gramática para obtener así mejores resultados sobre el reconocimiento.

Actualmente, si nos fijamos en el reconocedor de voz, cabe la posibilidad de que el usuario pronuncie alguna orden no incluida en la gramática. Dependiendo de la orden pronunciada obtendremos un resultado de búsqueda fallida, con lo que el simulador no ejecutará acción alguna. Puede pasar, pero, que en algunos casos, la hipótesis devuelta por el reconocedor de voz, sobre lo que ha dicho el usuario, sea una orden de nuestra gramática aunque realmente no sea lo que ha dicho el emisor. Esto puede llevar a errores en la ejecución de órdenes, dando resultado a la ejecución de acciones que no se deseaban ejecutar. Modificar la gramática y los modelos acústicos para que palabras que no corresponden a la gramática no sean entendidas por el reconocedor, sería un paso importante en la mejora del sistema.

Ampliar la gramática dotaría al sistema de un conjunto de órdenes más amplio. Este punto quizá no sea vital sobre el simulador, donde se deberían desarrollar las acciones

pertinentes, pero daría más libertad en caso de aplicar el sistema de reconocimiento de voz a un sistema domótico real. Como la gramática ha sido entrenada con separaciones por fonemas, realizar una ampliación de la gramática requerirá entrenar las nuevas oraciones y además, obtener vectores de características para los nuevos fonemas.

Otro punto a mejorar, esta vez en el caso del simulador, se corresponde con la velocidad de renderización del mismo. Al añadir diversos objetos sobre la escena, que son cargados y dibujados todos al mismo tiempo, la tasa de frames por segundo de la aplicación se ve ampliamente reducida. Una mejora sería la de aplicar algoritmos de ocultación sobre los objetos de la escena, para así mostrar únicamente los objetos visibles por la cámara. Esto dotaría al sistema de una movilidad más rápida dentro de la escena y de unas animaciones más suaves (como al abrir ventanas o puertas).

Bibliografía

- Shreiner, Dave; Woo, Mason; Neider, Jackie; Davis, Tom: *OpenGL Programming Guide (fifth edition).* Addison-Wesley, 2006.

- Huang, Xuedong; Acero, Alex; Hon, Hsiao-Wuen: *Spoken Language Processing*. Prentice Hall PTR, 2001.

- *The HTK Book (for HTK Version 3.4)*. 2006

- *Multipurpose Large Vocabulary Continuous Speech Recognition Engine*. *Julius rev 3.2*. 2001 - 2001

Bibliografía on-line:

- *HTK, Hidden Markov Model Toolkit*. Página oficial que incluye toda la documentación del proyecto, así como los ficheros binarios. http://htk.eng.cam.ac.uk

- *Open-Source Large Vocabulary CSR Engine Julius*. Página oficial con la documentación del proyecto, los binarios y ejemplos sobre la API. http://julius.sourceforge.jp/en_index.php

- *SAMPA Computer Readable Phonetic Alphabet*. Página oficial del diccionario informático fonético. http://www.phon.ucl.ac.uk/home/sampa/index.html

- *OpenGL. The Industry's Foundation for High Performance Graphics*. Página oficial de la librería openGL. Incluye documentación y ejemplos de desarrollo en opengl. http://www.opengl.org

- *Glut. The OpenGL Utility Toolkit.* Página oficial de la librería auxiliar para openGL, Glut. Ofrece documentación de la librería, códigos de ejemplo y la propia librería. http://www.opengl.org/resources/libraries/glut

Anexo

En el siguiente anexo se muestran algunas partes de ficheros o tablas que no se han añadido en la memoria, pero a las que se hace referencia.

1. Fichero domoticaWdnet

```
VERSION=1.0
N=38   L=56
I=0    W=!NULL
I=1    W=!NULL
I=2    W=INICIO
I=3    W=ABRE
I=4    W=!NULL
I=5    W=CIERRA
I=6    W=LA-S
I=7    W=VENTANA
. . .
J=0     S=37   E=1
J=1     S=0    E=2
J=2     S=2    E=3
J=3     S=3    E=4
J=4     S=5    E=4
. . .
```

2. Diccionario SAMPA

Consonantes		
Símbolo	***Palabra***	***Trascripción***
Plosivas		
P	padre	**p**adre
b	vino	**b**ino
t	tomo	**t**omo

d	donde	**d**onde
k	casa	**k**asa
g	gata	**g**ata
Africadas		
tS	mucho	mu**tS**o
jj	hielo	**jj**elo
Fricativas		
f	fácil	**f**aTil
B	cabra	ka**B**ra
T	cinco	**T**inko
D	nada	na**D**a
S	sala	**S**ala
x	mujer	mu**x**er
G	luego	lwe**G**o
Nasales		
m	mismo	**m**ismo
n	nunca	**n**unca
J	año	a**J**o
Líquidas		
l	lejos	**l**exos
L	caballo	kaba**L**o
r	puro	pu**r**o
rr	torre	to**rr**e
Semivocales		
j	pie	p**j**e
w	muy	m**w**i
Vocales		
a	valle	b**a**Le
e	pero	p**e**ro
i	pico	p**i**ko
o	toro	t**o**ro
u	duro	d**u**ro

3. Fichero Lab

```
6349375 7361250 f
7361250 8272500 i
8272500 9537500 x
9542500 10150000 a
```

```
10150000 10576250 l
10583750 11306875 a
11311875 12049375 t
12049375 12468750 e
12474375 13537500 m
13542500 13875000 p
13881875 14554375 e
14555000 15046875 r
15055625 16084375 a
16406250 16790625 t
16790625 17666875 u
17674375 18148750 r
18148750 19355625 a
19363750 21545625 a
21553750 22143750 b
22152500 22993125 e
23001875 23440625 j
23449375 25014375 n
25021250 25566875 t
25575625 26863750 e
```

Manual de Instalación

Este anexo describe el proceso de instalación, tanto de las aplicaciones como de las librerías que utilizan, y el proceso de ejecución del sistema una vez finalizada la instalación. El documento está dividido en dos puntos, el primero cubre los aspectos de la instalación, mientras que el segundo cubre los aspectos de la ejecución.

Instalación

El sistema requiere una instalación de algunas librerías necesarias para que las aplicaciones se ejecuten correctamente. La instalación de las librerías se puede dividir en dos puntos, la instalación para el simulador y para el reconocedor de voz. A continuación se describe el proceso a llevar a cabo:

Configuración del Simulador

El simulador ya lleva todo lo necesario para que funcione sin necesidad de instalar ninguna librería. Es necesario, pero, asegurarse que las dlls glut32.dll, glu32.dll y opengl32.dll están incluidas en el directorio de la aplicación. Sin estas dlls la aplicación no inicializará.

Si tenemos varios proyectos que utilicen glut y opengl, podemos copiar las librerías a directorios de sistema, de manera que no sea necesario arrastrarlas para cada aplicación que las utilice. Podemos mover glut32.dll al directorio C:/Windows/system y glu32.dll al directorio C:/Windows/system32. Lo mismo sucede con el fichero opengl32.dll, que lo podemos situar también en el directorio system32.

En una instalación Windows, las librerías opengl se deberían encontrar instaladas tras finalizar la instalación del sistema operativo.

Por lo que respeta al simulador, el directorio que contiene todos los ficheros, puede ser copiado en cualquier directorio del sistema y ejecutado desde cualquier punto del sistema.

Configuración del reconocedor de voz

Para poder ejecutar correctamente el simulador, es necesario instalar las librerías de Julius. Para instalar Julius empezaremos creando dentro del directorio de usuario (Recordar que estamos en Linux), un directorio llamado bin. Una vez creado este directorio, descargaremos Julius y slf2dfa (en caso que queramos utilizar gramáticas nuevas). Ambos paquetes se pueden descargar desde:

- http://sourceforge.jp/projects/julius/downloads/28551/julius-4.0.tar.gz
- http://sourceforge.jp/projects/julius/downloads/23332/slf2dfa-1.0.tar.gz

Extraeremos los contenidos de cada uno de los paquetes dentro del directorio bin, de manera que nos queden dos directorios más, uno llamado julius-4.0 y otro llamado slf2dfa-1.0. Para instalar Julius, entraremos dentro del directorio correspondiente y ejecutaremos:

```
/home/bin/julius-4.0$> ./configure

/home/bin/julius-4.0$> make

/home/bin/julius-4.0$> make install
```

Esto ha dejado Julius completamente compilado, instalado y listo para ejecutarse. Ahora, al crear nuestro propio reconocedor ya podemos lindar a la librería

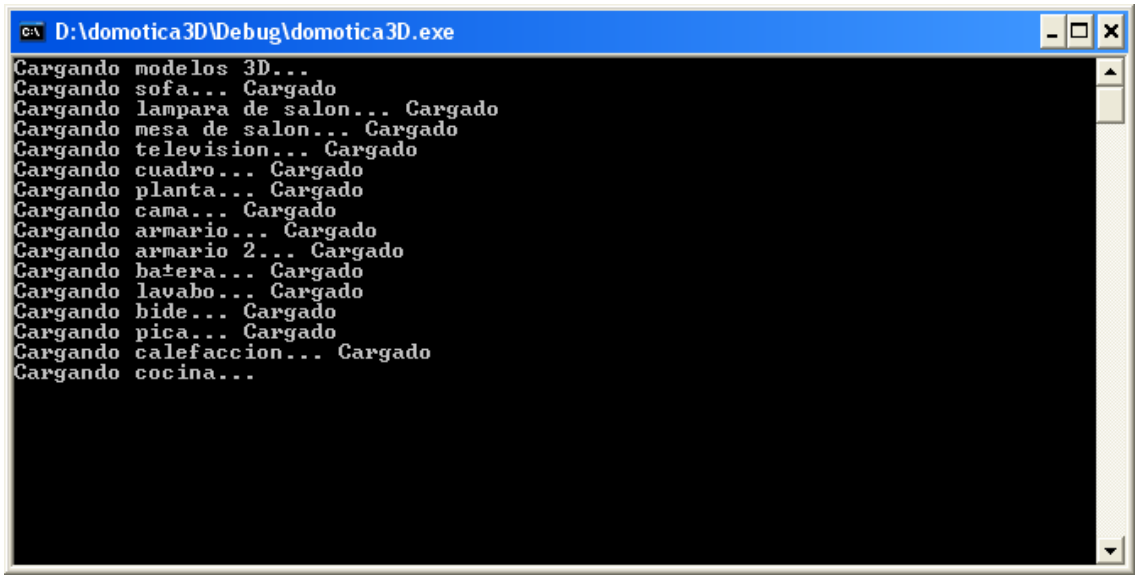
julius/julius.h. La otra aplicación, es un conjunto de utilidades, y tal y como las descomprimimos ya quedan instaladas.

El directorio de la aplicación rv, junto con todos sus ficheros, se puede guardar en cualquier directorio del sistema. Puede funcionar desde cualquier punto del sistema.

Ejecución

Una vez se han instalado todas las librerías necesarias, y ambas aplicaciones, ya podemos ejecutarlas y ver que realmente funcionan. Antes de nada hay que recordar que al tener una estructura cliente – servidor, es necesario iniciar las aplicaciones en un orden concreto, ya que sino, la aplicación cliente no será capaz de encontrar el host servidor.

En este caso, la aplicación cliente es el reconocedor de voz, y la aplicación servidor es el simulador. La primera aplicación a arrancar, es pues, el servidor. Para iniciarla realizaremos doble clic sobre el icono del ejecutable ***domotica3D.exe***. Al iniciarse, veremos una pantalla msdos en la que podremos ver como se van cargando los diferentes modelos antes de mostrar la escena final.



Una vez ejecutado el simulador, ya podemos ejecutar el reconocedor de voz. Para ello nos iremos a Linux, y en la línea de comandos ejecutaremos:

```
./rv domotica.jconf host_destino
```

La aplicación es rv y domotica.jconf y host_destino son dos parámetros necesarios para arrancar el reconocedor de voz. El primero es el fichero de configuración que indica de donde se debe coger el audio (entre otros parámetros). Este fichero se puede encontrar en el CD-ROM, junto al ejecutable rv. El segundo parámetro es el nombre de máquina donde se encuentra el simulador, por lo tanto, si el simulador se ha arrancado en una máquina con IP 192.168.1.3, en línea de comandos substituiremos host_destino por la IP anterior. Prácticamente después de haber arrancado el reconocedor de voz podremos ver en pantalla que se conecta al simulador y que arranca el dispositivo de entrada (micrófono). En este punto ya podemos iniciar la ejecución de órdenes.

Firmado: **Ander Welton Rodríguez**

Sabadell, 14 de Septiembre de 2009