FACULTAD DE INGENIERÍA Y CIENCIAS AGROPECUARIAS

IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VOZ APLICADO A DOMÓTICA DISEÑADO COMO MÉTODO DE AYUDA A PERSONAS CON DISCAPACIDAD VISUAL

AUTORES

JUAN FRANCISO CHANGO PERUGACHI

ANDRES SANTIAGO MARQUEZ ALMEIDA

AÑO

2017

FACULTAD DE INGENIERÍA Y CIENCIAS AGROPECUARIAS

IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE VOZ APLICADO A DOMÓTICA DISEÑADO COMO MÉTODO DE AYUDA A PERSONAS CON DISCAPACIDAD VISUAL

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingenieros de Sonido y Acústica

Profesor guía

MSc. Héctor Merino Navarro

Autores

Juan Francisco Chango Perugachi

Andrés Santiago Márquez Almeida

Año

2017

DECLARACIÓN DEL PROFESOR GUÍA

“Declaro haber dirigido este trabajo a través de reuniones periódicas con los estudiantes, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación.”

_____

Héctor Merino Navarro
MASTER UNIVERSITARIO EN POSTPRODUCCIÓN DIGITAL EN LA ESPECIALIDAD AUDIO
CC: 1756785562

DECLARACIÓN DEL PROFESOR CORRECTOR

“Declaro haber revisado este trabajo, dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación”

_____

Miguel Ángel Chávez Avilés
MSC SUSTAINABLE BUILDING ENGINEERING
CC: 1710724848

DECLARACIÓN DE AUTORÍA DE LOS ESTUDIANTES

“Declaramos que este trabajo es original, de nuestra autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que se protegen los derechos de autor vigentes.”

_____	_____
Andrés Santiago Márquez Almeida	Juan Francisco Chango Perugachi
CC: 1717269425	CC: 1726778622

AGRADECIMIENTOS

Agradezco a la vida por bendecirme con una maravillosa familia y rodearme de gente excelente, que sin su ayuda y apoyo no hubiera podido lograr todo esto. Quiero agradecer especialmente a mi amigo Andrés Márquez por su amistad y soporte. Y a Héctor Merino por su incondicional apoyo y sus sabios consejos.

Juan Francisco Chango Perugachi

AGRADECIMIENTOS

Agradezco a mis profesores que me han ayudado a formarme profesionalmente, A Héctor Merino por sus enseñanzas, a mi compañero y amigo Juan Chango por todo su apoyo, su amistad y sus consejos a lo largo de la carrera, inspirándome siempre a seguir adelante.

Andrés Santiago Márquez Almeida

DEDICATORIA

Dedico este trabajo a mis abuelos:
Víctor y Carmen.

Juan Francisco Chango Perugachi

DEDICATORIA

A mi familia, en especial a mi padre por todo su apoyo, soporte y paciencia que me han llevado a donde estoy ahora.

Andrés Santiago Márquez Almeida

RESUMEN

En el siguiente trabajo se presenta la implementación de un sistema de domótica, el cual ha sido diseñado para uso de personas con discapacidad visual, con el fin de brindar asistencia, confort y seguridad dentro de las viviendas de las personas que posean este tipo de discapacidad. Además, se intenta minimizar el costo del sistema para el usuario. El sistema consta de tres partes fundamentales, una aplicación para dispositivos electrónicos móviles, un servicio web de reconocimiento de voz desarrollado en Java, y un ordenador de placa reducida conocido como: Raspberry Pi, que servirá como actuador de los dispositivos en la vivienda del usuario. La aplicación recibe mensajes de voz del usuario, los cuales son enviados a través del protocolo SOAP hacia el servidor web de reconocimiento de voz, donde son procesados, obteniendo de vuelta en el dispositivo móvil un mensaje de texto. El mensaje de texto es enviado nuevamente a través de protocolo TCP/IP al Raspberry Pi para que se ejecuten los comandos solicitados por el usuario. El sistema tiene una buena precisión de reconocimiento en los comandos para los que ha sido adaptado, y posee una velocidad de respuesta bastante buena. Gracias a que el reconocimiento de voz se lo realiza de manera gratuita en el web service, la aplicación no tendrá costo alguno para el cliente y considerando el bajo costo del Raspberry Pi, se logra que el sistema sea económicamente accesible al usuario.

ABSTRACT

The following titling project presents the implementation of a home automation system, which has been designed for the use of people with visual impairment, to provide assistance, comfort and security within the homes of people who have this type of disability. In addition, it tries to minimize the cost of the system for the user. The system consists of three fundamental parts, an application for mobile devices, a voice recognition web service developed in Java, and a reduced-board computer called: Raspberry Pi that will serve as an actuator of the devices in the user's home. The application receives voice messages from the user, which are sent to the speech recognition web service through SOAP protocol, where they are processed, obtaining back in the mobile device a text message, which is sent again through TCP / IP protocol to Raspberry Pi so that it executes the commands requested by the user. The system has good recognition accuracy in the commands for which it has been adapted, and has a good response speed. Thanks to the free voice recognition in the web service, the application will have no cost to the customer and taking into consideration the low cost of Raspberry Pi, it achieves that the system was economically accessible to the user.

ÍNDICE

ÍNDICE DE FIGURAS

1. CAPÍTULO I. INTRODUCCIÓN

1.1 Antecedentes

En los últimos años se ha venido impulsando con fuerza el desarrollo de proyectos relacionados a la integración de tecnologías informáticas y electrónicas para la ayuda a personas con discapacidad visual. Un elemento primordial dentro de estos desarrollos ha sido el uso de tecnología relacionada con el procesamiento de señales de audio. Como resultado de esto se han creado algoritmos, y con ello, dispositivos que son capaces de analizar señales de voz y obtener información de esta, al punto de incluso reconocer mensajes y comandos. De esta manera se logra interconectar la información entre otros dispositivos mediante redes informáticas. Actualmente se han llevado a cabo varias investigaciones (Xiao, J., Ramdath, K., Iosilevish, M., Sigh, D., & Tsakas, A, 2013) con la finalidad de buscar un medio de ayuda para personas con discapacidad visual, muchas de estas investigaciones se basan en la implementación de algoritmos usando el modelo oculto de Markov. Es así que existen un sin número de librerías desarrolladas para distintos lenguajes de programación que permiten adaptar estos algoritmos a cualquier dispositivo.

En algunas ocasiones se han implementado dispositivos electrónicos embebidos en conjunto con la librería API de reconocimiento de voz de Google para generar un sistema el cual recibe la señal de audio y esa señal es procesada por el motor de Google. Este resultado es interpretado y enviado a otros sistemas electrónicos destinados al control de lámparas o motores (Upadhyay & Chavda, 2014).

También se han desarrollado aplicaciones que permiten el control de placas electrónicas y de robots. Los cuales mediante el uso de microcontroladores permiten adaptar circuitos de control de motores, luces, altavoces, sistemas bluetooth, etc. Para ello se han analizado distintas posibilidades de análisis de voz. Entre ellas se menciona a los sistemas autónomos: HMM (Hidden Markov Model) y CMU (Kit de programación de código abierto para el reconocimiento de

voz). Estos sistemas permiten la integración de análisis de voz fácilmente en prototipos electrónicos. El control de elementos mediante comandos de voz ha sido desarrollado en su totalidad con el uso de diferentes lenguajes, pero con frecuencia el lenguaje Python es preferido ya que muestra ser muy flexible y apto para diferentes tipos de aplicaciones (Lu & Hu, 2013).

Otras investigaciones muestran que, con la ayuda de lenguajes de programación como Java, redes inalámbricas y servidores Linux, es posible lograr tareas de reconocimiento de voz logrando porcentajes considerablemente altos de precisión con tiempos de procesamiento menores a 500 ms (Ayres & Nolan, 2006). Inclusive se han integrado sistemas donde la persona con discapacidad visual es guiada por medio de comandos generados por un dispositivo electrónico embebido el cual genera sonido en 3D usando grabaciones binaurales con objetos sonoros en diferentes posiciones. El sistema guía posee herramientas GPS las cuales usan bases de datos de mapas para obtener la ubicación de la persona y de esa manera poder guiarla hasta su destino (Xiao, Ramdath, Iosilevish, Sigh, & Tsakas, 2013).

Finalmente, en casi todas las investigaciones se hace uso de un sistema que procesa la señal de audio, y otro sistema, o en algunos casos el mismo, el cual interactúa con diferentes elementos electrónicos y eléctricos. Es así que el reconocimiento de voz se ha extendido ampliamente en aplicaciones de domótica, es decir, se hace uso de comandos de voz para el control de luces, sistemas de climatización y prácticamente todo dispositivo electrónico y eléctrico puede ser conectado y controlado. Últimamente muchas investigaciones se enfocan en el uso de dispositivos inalámbricos económicos que puedan conectarse directamente a una red y que permiten ser manejados remotamente desde distintos dispositivos como teléfonos inteligentes. Este enfoque se conoce como el IoT (Internet of the Things - Internet de las cosas) (Huang, 2015).

1.2 Alcance

Con el prototipo se busca alcanzar un tiempo máximo de tres segundos entre que la instrucción es dada por el usuario y ésta se ejecute en los ordenadores de placa reducida. Para esto se requiere un rápido procesamiento de las señales de audio para lograr una eficiencia de por lo menos un 90%, una mínima latencia en la transferencia de datos a través de la red inalámbrica y por último una respuesta instantánea por parte de los ordenadores de placa reducida, los cuales ejecutan finalmente los comandos.

Como variable descartada se considera: la puesta a prueba del sistema por parte de personas con discapacidad visual adquirida, las cuales pudieran evaluar subjetivamente la eficiencia del sistema además de la utilidad o aporte que él mismo haría a su vida cotidiana. Esta variable se descarta debido a que no se cuenta con el tiempo ni los recursos necesarios, es decir, se busca evaluar aspectos técnicos de los dispositivos electrónicos que se relacionen con aplicaciones de ayuda para personas invidentes.

1.3 Justificación

En el Ecuador existen 47,996 personas con discapacidad visual registradas en el Consejo Nacional de Igualdad para las Discapacidades CONADIS (2016). Muchas de estas personas no cuentan con los recursos económicos para adquirir dispositivos de ayuda para realizar actividades cotidianas dentro de sus hogares o requieren la ayuda de otra persona. Actividades cotidianas como localizar objetos o la ubicación de lugares pueden resultar tareas sumamente complejas para la persona con discapacidad, especialmente cuando la discapacidad es adquirida y no de nacimiento, ya que el sentido de ubicación toma un largo tiempo en desarrollarlo cuando una persona adquiere esta discapacidad. El prototipo a ser desarrollado tiene la ventaja de ser mucho menos costoso que un sistema de domótica que se comercializa tradicionalmente ya que para su implementación se utiliza ordenadores de placa reducida, cuyo costo es menor en comparación con ordenadores tradicionales. El prototipo, además de realizar las tradicionales tareas de domótica como encendido/apagado de luces, control de climatización, y apertura de puertas, será capaz de localizar objetos y lugares dentro de una casa.

1.4 Objetivos

1.4.1 Objetivo General

Diseñar e implementar un prototipo de un sistema de control para domótica y localización de objetos y lugares mediante un teléfono inteligente, ordenadores de placa reducida y redes inalámbricas, como medio de ayuda para personas con discapacidad visual.

1.4.2 Objetivos Específicos

Desarrollar una aplicación para móvil en el sistema operativo Android adaptada para ser usada por personas con discapacidad visual, que grabe comandos de voz, los cuales activarán o desactivarán las funciones del sistema de domótica.

Implementar un servicio web de reconocimiento de voz que transforme archivos de audio a mensajes de texto de manera remota.

Programar a los ordenadores de placa reducida (Raspberry Pi) para que sean capaces de ejecutar acciones en base a los comandos recibidos desde el teléfono móvil en forma de mensaje de texto a través de redes inalámbricas.

2. CAPÍTULO II. MARCO TEÓRICO

2.1 El Habla

2.1.1 Conceptos básicos del habla

El habla viene a ser un fenómeno complejo poco entendido por las personas, se pensaría que el habla está compuesta de palabras y que cada palabra está compuesta de fonemas, cuando en realidad se trata de algo mucho más complejo. El habla es un proceso dinámico sin partes claramente diferenciadas. Es de ayuda utilizar un editor de sonido para visualizar la forma de onda de palabras grabadas.

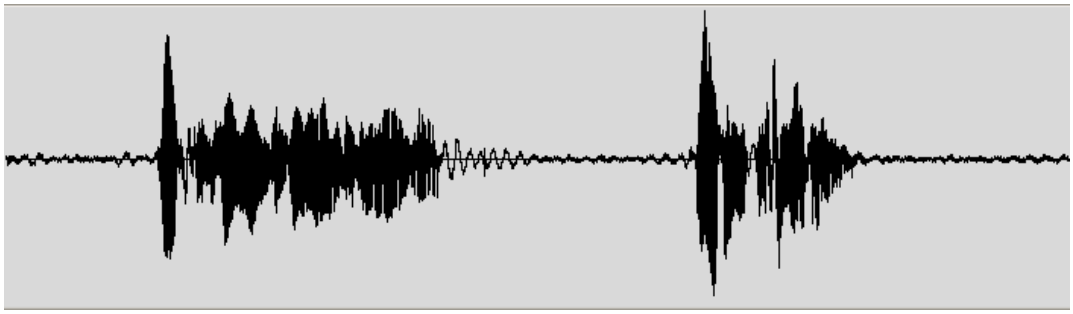


Figura 1. Representación de la forma de onda de palabras grabadas.

Tomado de (CMU Shpinx, 2016)

Las definiciones modernas del habla son en cierto grado probabilísticas, es decir no existen límites entre las unidades o palabras. Aplicaciones del habla como traductores de “Hablado-a-texto” nunca son ciento por ciento correctas. Esta idea suele ser inusual para programadores y desarrolladores ya que ellos están acostumbrados a sistemas deterministas, es decir, sistemas donde el punto de llegada será siempre el mismo con las condiciones iniciales dadas, y el azar no compromete el progreso de los futuros estados del sistema (CMUSphinx, 2016).

2.1.2 Estructura del habla

El habla es una reproducción continua de sonidos donde estados estables se mezclan con estados modificados dinámicamente. En esta secuencia de estados, se puede definir clases de sonidos o fonemas más o menos similares.

Se entiende que las palabras son compuestas solamente por fonemas, pero esto no es correcto. Las propiedades acústicas de una forma de onda correspondiente a un fonema pueden variar de gran manera dependiendo de algunos factores como el contexto del fonema, el hablante, el estilo del habla, entre otros. La llamada “coarticulación” hace que los fonemas se escuchen diferentes de su representación canónica. Ya que las transiciones entre las palabras son más informativas que las regiones estables, los desarrolladores a menudo las llaman “difonemas” (partes entre dos fonemas consecutivos). A veces, los desarrolladores hablan de unidades “subfonéticas”, diferentes estados de un fonema, usualmente tres o más regiones de un fonema pueden ser encontradas. En el caso de tres regiones, la primera parte viene a ser el fonema precedente, la parte de la mitad es estable y la última parte depende del fonema subsecuente. Es por esto que generalmente se utiliza estos tres estados de un fonema para el reconocimiento de voz (Brainbridge, 2004).

En algunas ocasiones los fonemas son considerados en contexto, estos fonemas en contexto son llamados “trifonemas” o incluso “quinfonemas”. Por ejemplo, “A” con el fonema izquierdo M y el fonema derecho L en la palabra MAL, suena distinto con el fonema izquierdo M y el fonema derecho R en la palabra MAR. Nótese que, en esos fonemas diferentes, están unidos con el mismo rango en la forma de onda como simplemente fonemas, estos difieren por el nombre ya que describen sonidos ligeramente distintos.

Para propósitos computacionales es de ayuda detectar las partes de los trifonemas en lugar de los trifonemas como tales. Generalmente se utiliza 4000 distintos detectores de sonidos pequeños que componen los detectores de trifonemas, a este conjunto de detectores se los llama “senones”. Después, los fonemas construyen unidades de “subpalabras”, como sílabas. Algunas veces las sílabas son definidas como “entidades reducidas estables”, para ilustrar, cuando el habla se vuelve más rápida, los fonemas tienden a cambiar, pero las sílabas siguen siendo las mismas. También las sílabas están relacionadas con un modelo de entonación. Las palabras son importantes en el reconocimiento de

voz porque estas restringen la combinación de fonemas significativamente. Si existen alrededor de 40 fonemas y un promedio de 7 fonemas por palabra, debe haber 40^7 palabras. Afortunadamente, incluso una persona con un alto nivel de educación raramente utiliza más de veinte mil palabras en la práctica, lo que hace el reconocimiento de voz más factible. Palabras y otros sonidos no lingüísticos, los cuales se denominan fillers o rellenos como respiraciones, tos, ehm, ah, etc. forman parte del discurso, estos son pedazos de audio separados entre pausas (CMUSphinx, 2016).

2.2 La domótica

Figura 2. Sistema de domótica controlado desde una tableta electrónica. Tomado de (Cedom, 2016)

2.2.1 Concepto

Se conoce como domótica a un sistema cuya finalidad es controlar y automatizar ciertos aspectos de una edificación, estos sistemas recogen información proveniente de entradas o sensores. Esta información es procesada y genera comandos hacia actuadores. La oferta de estos sistemas ha ido aumentando y mejorando su calidad en los últimos años gracias a los avances tecnológicos, cada vez se vuelven más fáciles de instalar y de fácil utilización para el usuario (Junestrand, 2004).

2.2.2 Aportes

- Facilita ahorro energético gestionando de manera inteligente aspectos que consumen energía eléctrica como la iluminación, el sistema de climatización, sistema de calentador de agua, sistema de riego. En algunos sistemas avanzados de domótica se puede monitorear los consumos de energía con el fin de modificar hábitos de consumo y así aumentar el ahorro y eficiencia.

- Aporta seguridad controlando los accesos a la edificación, puede implementarse también un circuito cerrado de cámaras para vigilancia, alarmas tanto de intrusión como en casos de incendio o fugas de gas en caso de ser conectadas a sensores.

- Convierte a la edificación más confortable ya que gestiona actividades domésticas como el encendido o apagado de luces, electrodomésticos, climatización, apertura de puertas, cortinas, persianas, además puede tener el control del suministro de electricidad, agua o gas.

- Gracias a la transmisión de voz y datos se puede tener acceso a la información sobre la vivienda remotamente a través de un computador o teléfono inteligente con acceso a internet. Además de recibir notificaciones en caso de que algo ande mal en la edificación.

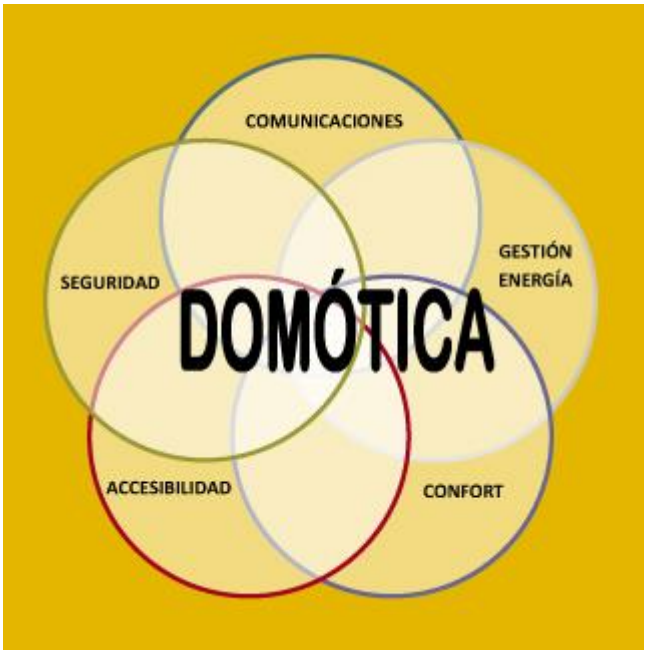


Figura 3. Aportes de la domótica.

Tomado de (Cedom, 2016)

2.3 Comunicación digital

2.3.1 Redes informáticas

Figura 4. Representación de una red informática.

Tomada de (Emaze, 2016)

Una red informática es un conjunto de equipos computacionales y software interconectados entre sí con el fin de transportar datos, compartir recursos entre

ellos siguiendo ciertos reglamentos. En un principio se tenían grandes ordenadores destinados para usos académicos, investigativos y militares los cuales eran controlados desde terminales externos, es decir, mediante el uso de pantallas y teclados los cuales enviaban y recibían información de una forma muy simple. Esa forma en la cual la información era proporcionada a los diferentes dispositivos se conoce como los inicios de las redes.

Así en los años setenta, en plena guerra fría, se requiere de la invención de un medio robusto y seguro para la transmisión de información digital, y así nace ARPANET (Advanced Research Project Agency NETwork). El cual hace uso de las líneas telefónicas para el transporte de información. Esto da paso al desarrollo a futuro de Internet. A mediados de los años setenta ARPANET toma un nuevo rumbo convirtiéndose en lo que hoy se conoce como Transmission Control Protocol/Internet Protocol (TCP/IP), además que deja de ser un secreto militar, lo que posibilita que universidades puedan acceder a su código y sea difundido libremente.

El concepto de red informática toma un rumbo crucial en su desarrollo con la creación de los ordenadores personales a inicios de los años ochenta. Esto marca el inicio de la informática descentralizada ya que cada ordenador tiene la capacidad de manejar la información de forma autónoma y haciendo uso de las redes informáticas para compartir subprocesos, información y recursos, de manera que nace la idea de la Red de Área Local (LAN). En ese entonces se comienza a usar el modem para compartir información entre estos con velocidades 1200 bps hasta los 56 Kbps más tarde lo reemplazaría el ADSL con velocidades mayores (Comer, 1996).

2.3.1.1 Elementos de una red

Existen dos puntos de vista para considerar los elementos que componen una red, estos son desde el software o desde el hardware. Desde el software una red está compuesta por un cliente el cual solicita servicios a una entidad lógica llamada servidor. El servidor podrá ofrecer diferentes servicios o uno solo al

mismo tiempo, entre ellos se menciona: compartir archivos, páginas web, correo electrónico, impresión de archivos, etc (Dordoigne, 2011).

Desde el punto de vista del hardware se tienen elementos tales como:

- La interconexión que se refiere al medio físico por el cual se transmitirá la información.
- Los protocolos de comunicación que se refiere al lenguaje y las reglas que acatarán los dispositivos que quieran comunicarse.

2.3.1.2 Topología de redes

Esto se refiere al mapa físico o lógico que usará una red para comunicar a los diferentes dispositivos. Así se tienen:

- La red personal - Personal Area Network (PAN): Conocida como red doméstica, es una red la cual toma en consideración una separación entre sus ordenadores de no más de una decena de metros.
- La red local - Local Area Network (LAN): una red que se extiende hasta unos centenares de metros.
- La red metropolitana - Metropolitan Area Network (MAN): su función es interconectar varias redes locales.
- La red extendida - Wide Area Network (WAN): Compuesta por varias redes MAN considera un área de interconexión de kilómetros de distancia es la red global que interconecta a países.

2.3.1.3 Usos comunes

- **Servidores de archivos:** Es de las aplicaciones más comunes de las redes informáticas, generalmente se consideran procesos de almacenamiento, transferencia y copia, sincronización y guardado.
- **Base de datos:** Mediante redes informáticas se permite la utilización de datos electrónicos en forma estructurada, su objetivo es permitir el acceso de datos a un esquema definido y de almacenar los datos manipulados.
- **Servicios de impresión:** Permite el control y la administración remota de impresoras y faxes. Estos pueden ser controlados por varios o todos los

dispositivos de una red, lo cual representa es una forma óptima de compartir recursos e información.

- **Servicios de mensajería y trabajo colaborativo:** Las redes permiten agrupar el almacenamiento, la utilización y el envío de datos de manera asíncrona entre varios usuarios. Actualmente las redes agrupan servicios de fax, contestadores de voz, mensajería instantánea, videoconferencias, entre otras.
- **Servicios de aplicaciones:** Mediante servidores especializados las redes permiten que clientes puedan ejecutar aplicaciones en el servidor y obtener resultados remotamente.
- **Servicios de almacenamiento:** Las redes permiten que varias empresas presten servicios de almacenamientos de datos a varios usuarios a lo largo de todo el mundo.

2.3.2 Protocolos de internet

2.3.2.1 Internet

La internet se denomina al conjunto de redes de comunicación, las cuales se interconectan entre sí a través de una familia de protocolos TCP/IP. Internet presta varios servicios, el principal de ellos es la Web o World Wide Web, pero la internet también ofrece otros servicios como el correo electrónico (SMPT), mensajería instantánea, transmisión de archivos (P2P y FTP), voz sobre IP (Telefonía VoIP), entre otros. Internet actualmente permite nuevas formas de interacción entre personas como foros, redes sociales, o servicios de mensajería (IBM Knowledge center, 2016).

2.3.2.2 Protocolo TCP/IP

Un protocolo se define como un conjunto de normas, que en este caso permiten a máquinas y aplicaciones intercambiar información, a través de mensajes y procedimientos. Estas normas deben ser seguidas por cada una de las máquinas que estén involucradas en la comunicación. Con el fin de que el mensaje pueda ser interpretado por el sistema principal de recepción. Los protocolos pueden ser interpretados en términos de capas o niveles como se muestra en la siguiente imagen:

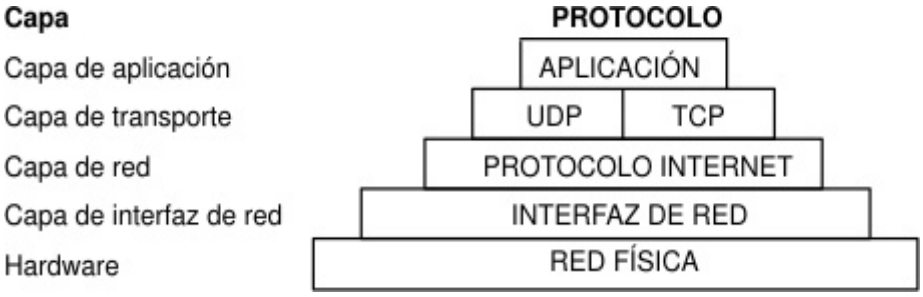


Figura 5. Conjunto de protocolos TCP/IP.

Tomado de (IBM, 2016)

En la figura se muestra gráficamente las capas del TCP/IP. Los protocolos definen el movimiento de la información desde el emisor hasta el receptor. En el inicio, se envían mensajes desde los programas de aplicación hacia uno de los protocolos de la capa de transporte. Los protocolos UDP o TCP reciben datos de la aplicación, los cuales dividen en pequeñas partes que toman el nombre de paquetes, una dirección de destino es añadida y pasan a la capa siguiente. En la capa de red se coloca el paquete en un datagrama de IP, colocando una cabecera y una cola al datagrama, se decide a dónde será enviado el datagrama y pasa a la capa de interfaz de red. En la capa interfaz de red se receptan los datagramas IP y éstos luego son transmitidos como tramas por medio de un hardware de red como por ejemplo redes Ethernet.

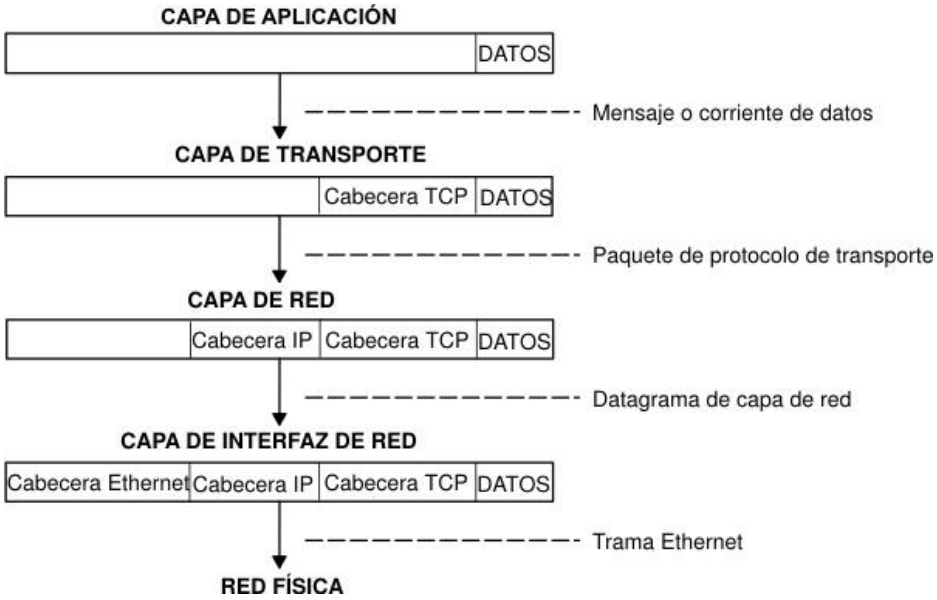


Figura 6. Flujo de información desde el remitente hacia el host, pasando por las capas del protocolo TCP/IP.

Tomado de (IBM, 2016)

Cuando la información retorna lo hace de la siguiente manera, la cabecera Ethernet es retirada por la capa de interfaz y el datagrama es enviado hacia la capa de red. En esta capa la cabecera IP es retirada y el paquete es enviado hacia la capa de transporte. En este punto se quita la cabecera TCP y los datos los datos se envían a la capa de aplicación (Douglas, 2009).

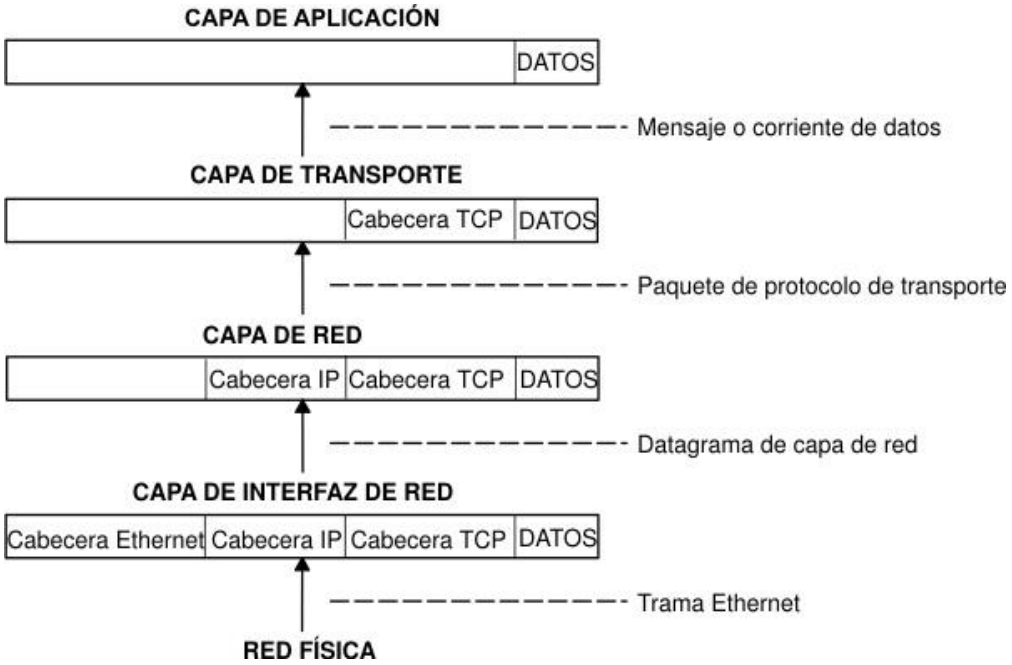


Figura 7. Flujo de información desde el sistema principal hacia el remitente, pasando por las capas del protocolo TCP/IP.

Tomado de (IBM, 2016)

2.3.2.3 Ventajas y desventajas

Ventajas:

- El modelo TCP/IP es diseñado para enrutar
- El grado de fiabilidad es alto
- Adecuado para redes medianas y grandes
- Compatibilidad con herramientas estándar que analizan el funcionamiento de la red
- Capaz de soportar múltiples tecnologías.
- Multiplataforma
- Suministra abstracción de capas

Desventajas:

- Afecta en el diseño de nuevas en base a TCP/IP ya que no distingue entre protocolos, servicios e interfaces.
- Pese a tener menos capas, su configuración es más compleja,

- En redes cuyo volumen de tráfico es bajo se lo considera lento.

2.3.3 Arquitectura Cliente-Servidor

En éste tipo de arquitectura, la computadora de los usuarios es llamada cliente, el cliente demanda información a otro computador que posee ésta información, éstos últimos computadores son denominados servidores. Estos servidores pueden tener dos tipos de conexiones, a una red local como por ejemplo dentro de una empresa o a una red mundial como la internet. Con esta arquitectura todos los usuarios tienen libertad de obtener información en un momento dado, esta información puede provenir de una o varias fuentes internas o externas y procesar esta información a su conveniencia. Dentro de esta arquitectura la información también puede ser intercambiada entre servidores.

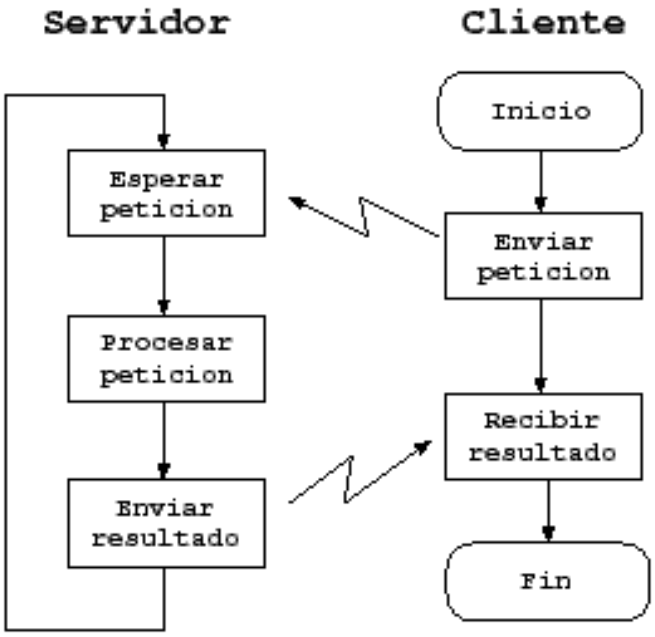


Figura 8. Arquitectura cliente-servidor.

Imagen tomada de (ENTEL, 2009)

Formalmente al cliente se lo define como un programa ejecutable el cual participa en el establecimiento de conexiones. Una petición es enviada al servidor y permanece esperando una respuesta, una vez que sus solicitudes son respondidas termina el trabajo. El servidor puede definirse como un programa el cual brinda un servicio que es obtenido a través de una red, el servidor acepta la

petición del cliente, realiza el servicio y un resultado es devuelto al cliente. Los servidores pueden ejecutar tareas sencillas como por ejemplo devolver una hora o una fecha, o tareas más complejas en las que por ejemplo se requiera hacer operaciones matemáticas o procesar la información antes de devolverla al cliente (Sommerville, 2005).

2.3.3.1 Características

- La única relación que existe entre cliente y servidor es el intercambio de mensajes entre ambos.
- Los clientes tienen un carácter activo ya que son los que realizan las peticiones de los servicios mientras que los servidores son de carácter pasivo pues son los que simplemente receptan las peticiones de los clientes.
- Las tareas entre servidores y clientes en cuanto a recursos de cómputo pueden tener diferentes requerimientos como memoria, velocidad de procesamiento, capacidad y velocidad de disco.
- El ambiente se vuelve heterogéneo ya que el sistema operativo del cliente y el hardware entre cliente y servidor no suelen ser los mismos.
- Es posible establecer una relación entre distintos procesos que pueden ser ejecutados en el mismo computador o en máquinas que se encuentren distribuidas a lo largo de la red.

2.3.3.2 Ventajas y desventajas

Ventajas:

- Gracias al avance tecnológico las plataformas hardware son cada vez mejores y más accesibles económicamente.
- Es posible utilizar componentes de hardware, así como de software de distintos fabricantes, reduciendo costos y aportando mayor flexibilidad en la implantación de estas arquitecturas.
- Permite la integración de información.

Desventajas:

- El diagnóstico de fallas se vuelve complejo ya que las partes de hardware y software que interactúan pueden venir de distintos proveedores, dificultando el mantenimiento de los sistemas.
- Las herramientas que permiten la administración y ajustes en cuanto al desempeño de los sistemas son escasas.

2.4 Software

2.4.1 Java

2.4.1.1 Programación orientada a objetos (POO)

Se conoce como programación orientada a objetos (POO) al tipo de programación que utiliza objetos en las interacciones que realiza. Es basada en técnicas variadas como herencia, abstracción, cohesión, acoplamiento, polimorfismo y encapsulamiento. Su popularidad empezó a inicios de los 90, actualmente existe una amplia variedad de lenguajes de programación capaces de soportar la orientación a objetos. En un inicio la programación tenía una estructura lineal o secuencial, esto significa, pasos consecutivos y estructuras consecutivas con bifurcaciones (Ciberaula, 2016).

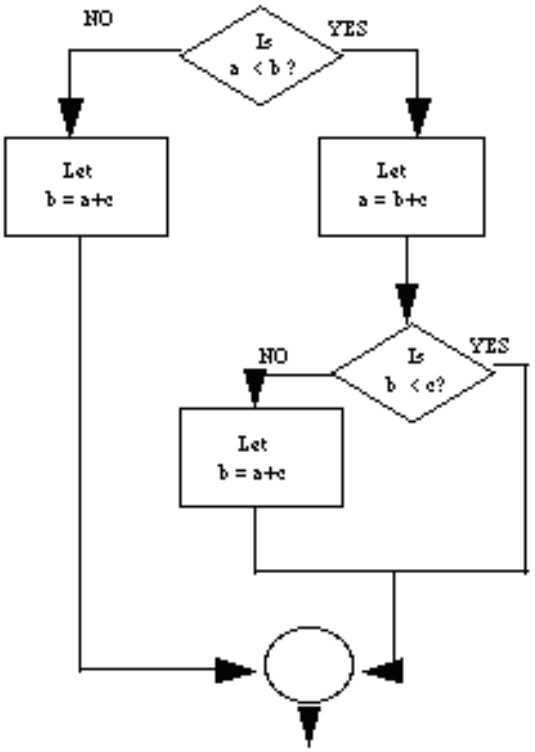


Figura 9. Programación de tipo lineal.

Tomado de (Ciberaula, 2015)

Este tipo de programación en un inicio ofrecía ventajas, pero los problemas aparecen cuando los programas o sistemas se tornan complejos ya que no ofrecen flexibilidad, y el hecho de mantener una cantidad grande de líneas de código en un único bloque, se vuelve una tarea complicada. Ante esta problemática aparecen lenguajes basados en una programación estructurada, los cuales separan las partes complejas del algoritmo en segmentos o módulos que son ejecutados cada vez que se requieran. Con este diseño, sus módulos independientes pueden comunicarse entre sí. Poco a poco este tipo de programación fue reemplazando a la programación lineal.

Desde entonces la programación ha venido evolucionando con tendencia a descomponer el programa, esta descomposición da lugar a la programación orientada a objetos. Ya que se dio una creciente tendencia a la creación de programas más grandes y complejos, los desarrolladores debieron crear un nuevo tipo de programación que permita crear sistemas de escala empresarial con reglas de negocios complejas, para este caso la programación de tipo

estructurada y lineal quedaban obsoletas, con lo que apareció la programación orientada a objetos, la cual divide el programa en pequeñas unidades lógicas de código que reciben el nombre de “objetos” (Valbuena, 2010).

2.4.1.1.1 Objetos

Un objeto se define como una unidad de código compuesto por variables y métodos relacionados. Para comprender mejor este concepto se podría pensar en un objeto en la vida real, por ejemplo, un computador, éste se constituye internamente de varios componentes como la tarjeta madre, el procesador, la unidad de almacenamiento de memoria, tarjetas de audio, tarjeta de video y otros componentes hacen que en conjunto funcione el computador. En el caso de la programación orientada a objetos, todo el programa está constituido por diferentes componentes, en este caso los llamados objetos. Cada uno de estos cumple un rol específico dentro del programa, y todos y cada uno de los componentes tienen comunicación entre ellos.

2.4.1.2 Lenguaje de programación Java

Figura 10. Logotipo de Java.

Tomado de (kbits, 2013)

Java es un lenguaje de programación orientado a objetos, utilizado para desarrollar aplicaciones de red, móviles, embebidas, juegos, contenido basado en web, entre otras. Diseñado con el fin de tener en lo posible muy pocas dependencias de implementación. La intención es que los desarrolladores puedan ejecutar un programa escrito por ellos en cualquier dispositivo, a esto se lo denomina WORA por sus siglas en inglés (“Write Once, Run Anywhere”). Esto significa que el código ejecutado en una plataforma no debe ser recompilado

para que éste corra en otra plataforma. Desde el 2012 Java se ha convertido en uno de los más populares lenguajes de programación, especialmente para aplicaciones cliente-servidor (Oracle, 2015).

2.4.1.3 Máquina virtual de java

Java es compilado e interpretado a la vez, El compilador transforma el código fuente de los archivos de extensión .java a instrucciones que reciben el nombre “bytecodes”, estos son guardados en un archivo con extensión .class. Este conjunto de instrucciones es independiente del sistema operativo del ordenador (Macintosh, Windows, etc). Gracias a esto el programa debe ser compilado una sola vez, pero es interpretado cada que es ejecutado en un ordenador.

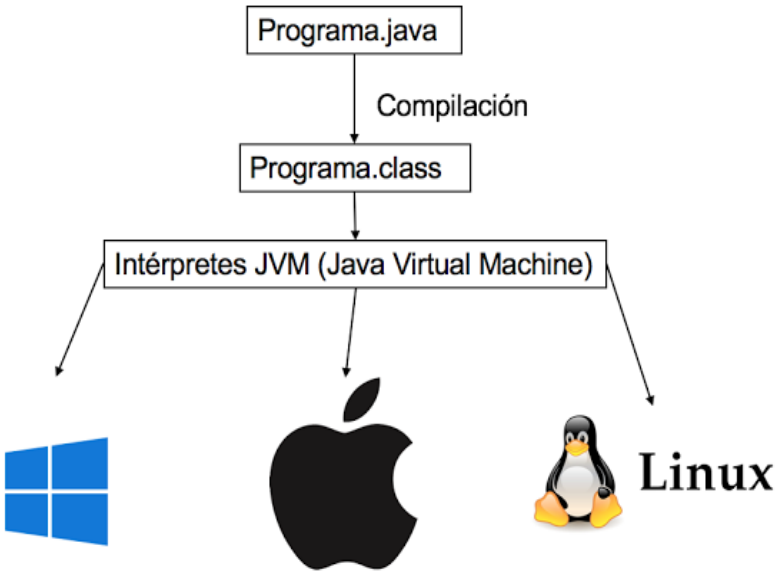


Figura 11. Representación de la máquina virtual de Java.

Cada uno de los intérpretes java es la implementación de la JVM (máquina virtual de java) la cual interpreta los bytecodes. Estos bytecodes hacen posible el concepto de WORA (write once, run anywhere), el cual permite que una vez escrito el programa, éste pueda ser ejecutado en cualquier plataforma que disponga una máquina virtual de java. Por esto, Java va más allá de ser un lenguaje de programación, Java son dos componentes, por un lado el lenguaje

que permite la escritura de programas y por el otro la Máquina virtual de Java que permite ejecutarlos (Franco,2000).

La misión de la máquina virtual de Java es garantizar portabilidad con las aplicaciones de Java. Dentro de las principales tareas que ésta cumple son:

- Reserva un espacio en la memoria para objetos creados.
- Libera memoria que no es utilizada.
- Asignación de variables a registros.
- Vigilancia en el cumplimiento de normas de seguridad en aplicaciones Java.

2.4.1.4 Dispositivos

Java se ha convertido en un estándar para el desarrollo de aplicaciones móviles, juegos y software ya que en el mundo existen más de 9 millones de desarrolladores Java, y un número grande de dispositivos que lo soportan (Oracle, 2016).

- 97% de los computadores empresariales ejecutan Java
- 89% de computadores en Estados Unidos ejecutan Java
- 3000 000 000 de teléfonos móviles ejecutan Java
- 100% de reproductores de discos Blu-Ray incluyen Java
- 5000 000 000 de Java Cards en uso.
- 125 000 000 de dispositivos de TV ejecutan Java

2.4.1.5 Java (Edición ME)

Java Micro Edition o Java ME es un subconjunto de la versión estándar de la plataforma Java la cual permite el desarrollo de software orientado a dispositivos cuyos recursos son restringidos, como el caso de tabletas electrónicas, teléfonos celulares e incluso electrodomésticos. Esta edición de Java es una de las mejores opciones con las que se cuenta para desarrollar videojuegos para teléfonos móviles ya que durante la fase de desarrollo se puede emular la app en un computador.

2.4.2 Android

Figura 12. Logotipo de Android.

Tomado de (Logospike, 2016)

Es un sistema operativo que se basa en el kernel de Linux, diseñado especialmente para dispositivos como teléfonos celulares inteligentes, tabletas electrónicas, relojes inteligentes, es decir, dispositivos móviles y de pantalla táctil. En un inicio, este sistema operativo fue desarrollado por la empresa Android Inc., Propiedad de Google desde el año 2005 (Android, 2016).

2.4.2.1 Características

2.4.2.1.1 Conectividad

Android puede soportar las tecnologías de conectividad que se listan a continuación:

- Bluetooth
- CDMA
- EV-DO
- GPRS
- GSM/EDGE
- HDSPA

- HDSPA+
- HSPA+
- IDEN
- LTE
- NFC
- UMTS
- Wi-Fi
- WiMax

2.4.2.1.2 Soporte de audio

Android soporta los formatos de audio listados a continuación:

- MP3
- MIDI
- Ogg Vorbis
- WAV

2.4.2.1.3 Entorno de desarrollo

Contiene un emulador llamado Android Studio, que incluye herramientas de depuración de memoria y analizador del rendimiento de software. Éste puede ser descargado de la página oficial de desarrolladores de Android.

2.4.2.1.4 Características basadas en voz

Síntesis de voz de Google es un motor que permite a las aplicaciones leer texto que se encuentra en la pantalla del dispositivo en voz alta, también es posible programar una aplicación para guiar a personas con discapacidad visual.

2.4.2.2 Usos y dispositivos

Android actualmente es utilizado principalmente en teléfonos móviles inteligentes, pero también se lo usa en ordenadores personales, tabletas electrónicas, auriculares, plataformas de televisión inteligente, entre otros dispositivos. Siendo éste sistema operativo utilizado en dispositivos cuyo precio va desde menos de $100 dólares americanos hasta dispositivos de más de $1000 dólares. Se estima que 1 500 000 teléfonos móviles son activados diariamente con sistema operativo Android, superando las ventas a dispositivos iOS y Windows Phone juntos (Android, 2016).

2.4.2.3 Aplicaciones móviles

En la actualidad existe un aproximado de 1 000 000 de aplicaciones desarrolladas para Android disponibles en Google Play, la tienda de aplicaciones de Android. Estas aplicaciones son desarrolladas por lo general en lenguaje Java con Android SDK (Software Development Kit), aunque también se dispone de otras herramientas como Google app inventor el cual tiene un entorno visual para programadores novatos o para programadores más avanzados un kit de desarrollo nativo para aplicaciones desarrolladas en lenguaje C o C++. Las aplicaciones son comprimidas en el formato APK con el fin de ser instaladas sin dificultad desde un explorador de archivos en casi todos los dispositivos (Perez, 2016).

2.4.2.4 Uso de lenguaje Java

Pese a que la mayoría de aplicaciones son escritas en Java, en la plataforma no existe una máquina virtual de java (JVM). Los Bytecodes de Java no se ejecutan, estos son primero compilados en un archivo ejecutable “Dalvik” y son ejecutados en la máquina virtual Dalvik, la cual es diseñada exclusivamente para Android y optimizada para dispositivos móviles que utilicen menos batería y tengan limitaciones en aspectos como procesador y memoria. Desde la versión 5.0 de Android es utilizado el Android Runtime ART (Garzón, 2015).

2.4.3 CMU Sphinx

CMU Sphinx o simplemente Sphinx es el nombre con el que se describe a los sistemas de reconocimiento de voz, los cuales vienen siendo desarrollado por la universidad estadounidense de Carnegie Mellon. Consta de un software de reconocimiento de voz, cuya última versión se la denomina Sphinx4 y un modelo entrenador acústico llamado SphinxTrain. Sphinx4 está escrito en su totalidad en lenguaje de programación Java. Al estar Sphinx4 escrito en lenguaje Java, éste puede ser utilizado en una amplia gama de sistemas operativos y hardware (CMUSphinx, 2016).

2.4.3.1 Proceso de reconocimiento

La manera más común de reconocimiento de voz es la siguiente, se toma la forma de onda, se la separa en palabras de acuerdo a los silencios, y finalmente se trata de reconocer qué se está diciendo en cada palabra. Para hacer esto, se requiere tomar todas las posibles combinaciones de palabras y tratar de correlacionarlas con el audio. Se selecciona la mejor combinación posible. Existen criterios importantes en esta selección.

El primer criterio es un concepto de rasgos, mientras el número de parámetros sea mayor, se optimiza el sistema. Los números son calculados del discurso usualmente separando el discurso en cuadros (frames). Después, a cada cuadro de aproximadamente 10 milisegundos de duración se extraen 39 valores que representan el habla. A eso se lo llama feature vector (vector de rasgos). Estos valores son la derivada del espectro.

Segundo, es un concepto de modelo. El modelo describe un objeto matemático que reúne los atributos comunes de las palabras pronunciadas. El modelo es llamado el “Modelo oculto de Markov” o HMM, este modelo pretende describir cualquier tipo de proceso secuencial como el habla. Se ha demostrado que el modelo es sumamente práctico para la decodificación del habla.

Tercero, es un proceso de pareo. Dado que tomaría una gran cantidad de tiempo comparar todos los vectores de rasgos con todos los modelos, la búsqueda es a menudo optimizada con varios atajos. En cualquier punto se mantiene la mejor selección de variantes y se las extiende a medida que pasa el tiempo produciendo la mejor selección de variantes para el siguiente cuadro.

2.4.3.1.1 Modelos

De acuerdo con la estructura del habla, tres modelos son utilizados en el reconocimiento de voz para hacer el proceso de pareo.

Modelo acústico: Modelo que contiene toda la información estadística para diferenciar las características dialécticas, que pueden ser adaptados en cada región que posea un estilo de habla diferente.

Diccionario fonético: Contiene un mapa que va de palabras a fonemas. Este mapeo no es muy eficiente. Por ejemplo, solamente dos o tres variantes de pronunciación son detectadas en él, pero es lo suficientemente práctico la mayor parte del tiempo. El diccionario no es la única variante del mapeo de palabras a fonemas, esto puede ser hecho con una función compleja.

Un modelo de lenguaje: Es utilizado para restringir la búsqueda de palabras. Esto define qué palabras pueden seguir a una palabra previamente reconocida y ayuda significativamente a restringir el proceso de pareo sustrayendo palabras que no pueden ser probables. Los modelos más comunes de lenguaje usados son modelos “n-gram”, estos contienen estadísticas de palabras con secuencia lógica.

Para alcanzar una buena tasa de precisión, el modelo de lenguaje debe ser muy exitoso en la búsqueda de “la siguiente palabra”. Un modelo de lenguaje usualmente restringe el vocabulario considerando las palabras que contiene, esto es un problema con el reconocimiento de nombres propios.

Estos tres modelos son combinados en conjunto para lograr el reconocimiento de voz. Si se desea aplicar este reconocimiento en un idioma diferente, se debe modificar tanto el modelo acústico como el diccionario fonético.

2.4.3.1.2 Optimizaciones

Cuando el reconocimiento de voz es desarrollado, la tarea más compleja es realizar una búsqueda precisa y hacerla en el menor tiempo posible. Pero también existen otro tipo de problemas que pueden darse dado que los modelos descritos anteriormente no son perfectos. Usualmente el sistema es probado en una base de datos de prueba en la que se evalúan características como el

porcentaje de error en el reconocimiento de una palabra, la precisión en el reconocimiento, la velocidad de respuesta, éste tipo de parámetros son evaluados y mejorados para las siguientes versiones de sphinx.

2.4.4 Servicios web

Se denomina servicio web a un grupo de aplicaciones y tecnologías capaces de interoperar en la red, intercambiando datos entre sí con el fin de ofrecer servicios. Un proveedor ofrece servicios, como procedimientos remotos y el usuario los solicita llamando a estos procedimientos a través de la red. Los servicios web aportan mecanismos de comunicación estándar entre diversas aplicaciones, las cuales interactúan entre ellas con el objetivo de brindar información dinámica al usuario (W3C España, 2016).

2.4.4.1 SOAP

El protocolo SOAP (Simple Object Access Protocol) define la comunicación de dos objetos en diferentes procesos a través del intercambio de datos XML (Lenguaje de marcado extensible). Este protocolo ha sido creado por Microsoft e IBM y actualmente se encuentra bajo el auspicio de World Wide Web Consortium. Es actualmente uno de los protocolos que más se utiliza en la web.

2.4.4.2 REST

REST (Representational State Transfer) es el término utilizado para describir a una interfaz entre sistemas, que para la obtención de datos utilice HTTP (Hypertext Transfer Protocol) sin abstracciones adicionales de protocolos como el SOAP descrito anteriormente, los cuales se basan en patrones de intercambio de mensajes.

2.4.4.3 Ventajas de servicios web

Independientemente de las plataformas sobre las que estén instaladas, aportan interoperabilidad entre aplicaciones de software. Además, es fácil acceder al contenido de los servicios y entender el funcionamiento por sus protocolos basados en texto. Finalmente, software y servicios de compañías distribuidas

alrededor del planeta pueden ser fácilmente combinados con el fin de proveer servicios integrados.

2.4.4.4 Inconvenientes de servicios web

Al adoptar un formato basado en texto, el rendimiento se considera bajo comparado con modelos diferentes de computación distribuida como Java Remote Method Invocation, Distributed Component Object Model o CORDBA. En temas de seguridad, al estar apoyado en HTTP, medidas de seguridad que se basan en firewall pueden ser esquivadas.

2.4.4.5 Servicios web de reconocimiento de voz

Actualmente en la internet es posible encontrar servicios web dedicados a transformar la voz en texto, en un inicio este proceso era realizado por un software especializado, el cuál debía estar descargado e instalado en el computador del usuario. Sin embargo, en estos días puede encontrarse varios servicios web dedicados a esto en forma remota, pero con la desventaja que tienen un costo según el tiempo de uso. El audio en servicios web por seguridad es codificado con el sistema Base64, con el uso de caracteres A-Z, a-z y 0-9 (Rouillard, 2016).

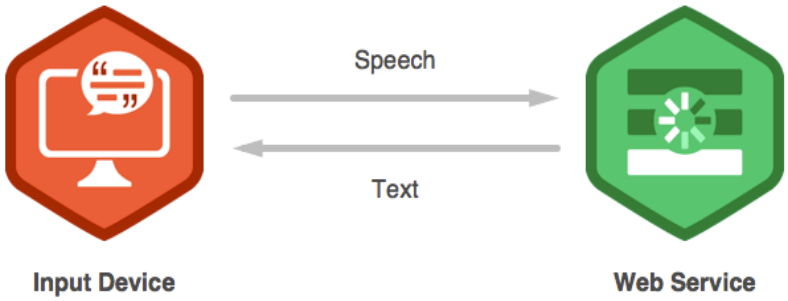


Figura 13. Representación gráfica del servicio web de reconocimiento de voz. Tomado de (TeamTreeHouse, 2016)

2.4.4.5.1 Cloud Speech API de Google

Un ejemplo de uno de los mejores servicios web con los que se cuenta es el CLOUD SPEECH API de Google. Esta API permite a los desarrolladores

convertir audio en texto mediante la aplicación de potentes modelos de redes neuronales en una API fácil de utilizar. En la actualidad reconoce 80 idiomas diferentes y adicional a esto permite el almacenamiento de los archivos de audio y los datos en Google Cloud Storage, el servicio de almacenamiento en la nube de Google. La precisión de la API mejora continuamente a medida que pasa el tiempo a medida que Google perfecciona la tecnología de reconocimiento de voz. El reconocimiento de voz se lo puede realizar en tiempo real con una transmisión de audio o de un archivo de audio ya almacenado. Cuenta además con filtros que eliminan el ruido, con lo que no es necesario que la grabación de audio se realice en ambientes silenciosos. La API soporta cualquier dispositivo como teléfonos móviles inteligentes, computadores, tabletas electrónicas entre otros dispositivos IoT (Internet of things), solamente es necesario que el dispositivo soporte el envío de requerimientos REST o gRCP. El costo del servicio actualmente es gratuito los 60 primeros minutos, desde el minuto 61 en adelante el costo por cada 15 segundos es de $0.006 dólares americanos (Google, 2016).

2.5 Hardware

2.5.1 Sistemas electrónicos embebidos

Los sistemas embebidos o también llamados empotrados se definen como un sistema de tipo electrónico que ha sido diseñado para realizar una serie de funciones específicas, generalmente son parte de un sistema de mayor entidad. El empleo de uno o más procesadores digitales (CPUs) en formato de microprocesadores, procesadores de señales digitales DSPs y microcontroladores es la principal característica de estos sistemas. Estos CPUs permiten “aportar inteligencia al sistema anfitrión al cual ayuda a gobernar y del que forma parte” (Úbeda, 2009). Para el diseño de sistemas embebidos se requiere la colaboración de ingenieros y técnicos con especializaciones en desarrollo tanto de software como de hardware, además en ciertos casos se puede requerir la realimentación de los usuarios de estos dispositivos.

2.5.1.1 Software

En materia de software los requisitos específicos varían de acuerdo a la aplicación. Cabe tener en cuenta que para el diseño de un sistema embebido los recursos tienden a ser limitados en cuanto a cantidad de memoria, capacidad de cálculo. El empleo o no de un sistema operativo determinado y dependerá del sistema a ser desarrollado y es una de las decisiones primordiales a ser tomadas en cuenta en el diseño de un sistema embebido.

2.5.1.2 Hardware

Por lo general, los sistemas embebidos son placas electrónicas alojadas dentro de un sistema mayor (anfitrión o host), al cual colabora en tareas como control de actuadores, procesamiento de información entregada por algún tipo de sensor, procesamiento de audio, etc. El núcleo de estas placas está conformado con al menos un CPU de tipo microprocesador, microcontrolador de 4 hasta 32 bits, DSP de punto fijo o flotante. Adicional a esto, la placa puede ser diseñada con el fin de satisfacer ciertas necesidades específicas de diseño como

- Tamaño.
- Margen de temperatura de trabajo.
- Consumo de energía.
- Robustez mecánica.
- Coste.

2.5.2 Microcontroladores

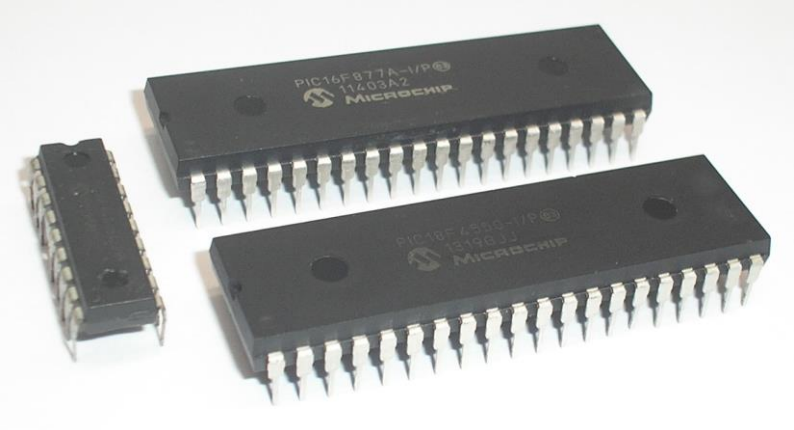


Figura 14. Ejemplo de microcontroladores.

Tomado de (Xbot, 2016)

Un microcontrolador es un CI (circuito integrado) que puede ser programable y es utilizado para el control del funcionamiento de una determinada tarea, posee los mismos componentes que un computador de prestaciones limitadas. Debido a las dimensiones, estos suelen ser incorporados dentro del propio dispositivo al cual gobierna. Por este motivo, se lo denomina como un “controlador incrustado”. Los microcontroladores son dispositivos dedicados, es decir, su memoria contiene un programa cuyo destino es el control de una determinada aplicación. Sus entradas y salidas soportan conexiones de sensores y actuadores. Una vez que el microcontrolador ha sido configurado y programado, solamente funciona para ejecutar la tarea asignada.

2.5.3 Ordenadores de placa reducida

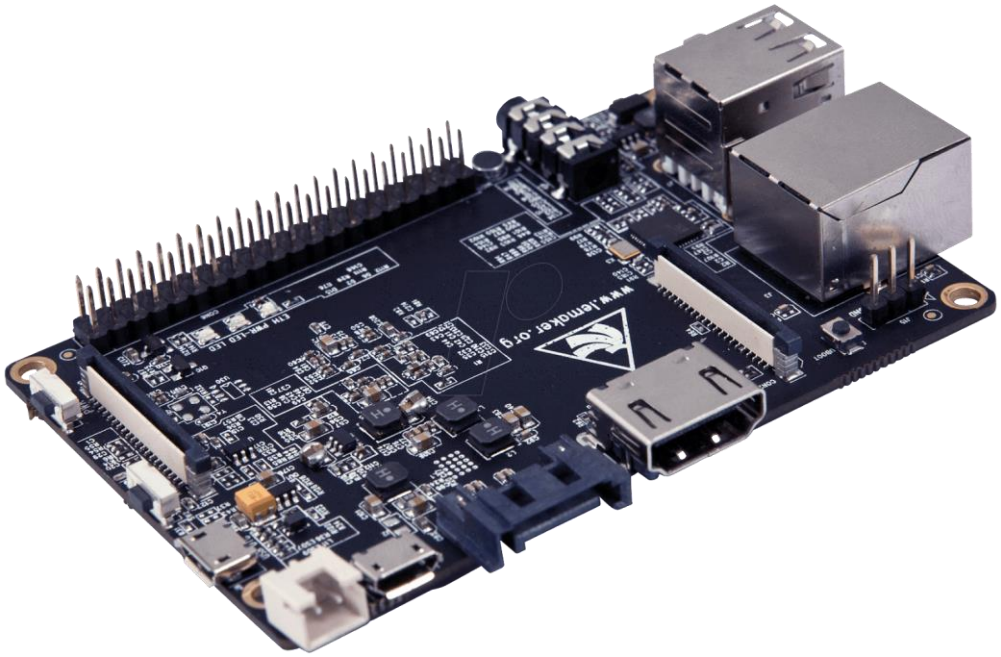


Figura 15. Banana Pro, Ordenador de placa reducida.

Tomado de (CodigoFacilito, 2015)

También llamadas “placa computadora” o “SBCs” por sus siglas en inglés (Single Board Computer), son ordenadores implementados en un solo circuito. El diseño de estos se centra en un único microprocesador, además de la memoria RAM (Random Access Memory), entradas y salidas, y el resto de características de un computador funcional dentro de una sola tarjeta de tamaño reducido. Gracias a los grandes niveles de reducción e integración de conectores y componentes, los SBCs tienen la ventaja de ser de tamaño reducido, livianos, confiables y con un óptimo manejo de la potencia eléctrica.

2.5.4 Internet de las cosas

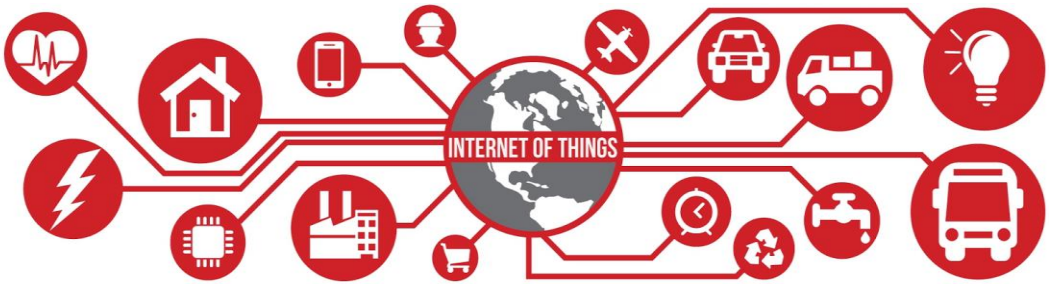


Figura 16. Representación gráfica del internet de las cosas.

Tomado de (HiperTextual, 2014)

IoT (Internet of Things) por sus siglas en inglés se refiere a un concepto el cual explica la interconexión digital a través del internet de objetos cotidianos. Estos objetos físicos se valen de sistemas electrónicos embebidos los cuales permiten además de la conexión a internet, programar eventos en función de tareas que se dicten remotamente. Todos los objetos que se encuentran conectados a la red, cuentan con una dirección IP específica, y a través de ésta puede el objeto ser accedido para recibir órdenes e instrucciones. De igual manera se puede contactar con un servidor con el fin de enviar datos que recoja.

2.5.5 Raspberry Pi

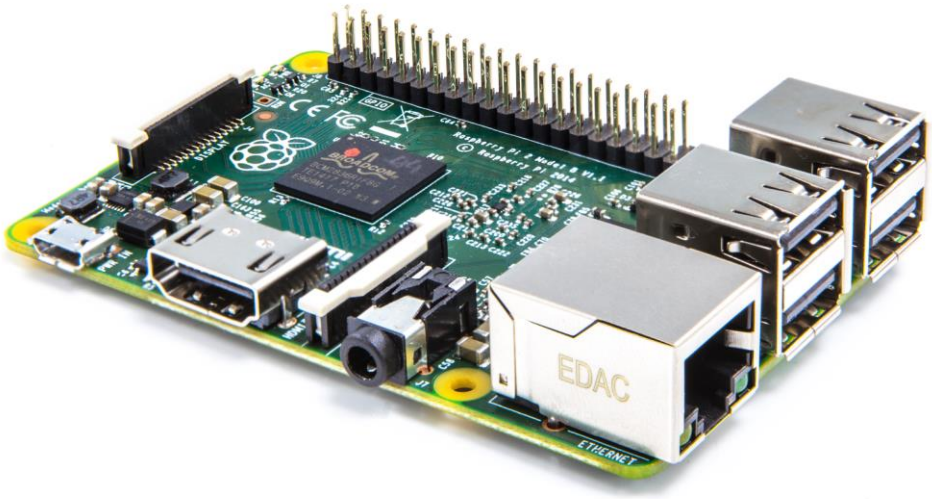


Figura 17. Ordenador de placa reducida Raspberry Pi.

Tomado de (Raspberry Pi, 2015)

Raspberry Pi es un ejemplo de ordenadores de placa reducida de bajo coste desarrollado con el fin de estimular el aprendizaje de las ciencias de la computación en los centros educativos (Cellan-Jones, 2011). Este SBC ha sido desarrollado en Reino Unido por la fundación Raspberry Pi, la cual expresa que éste es un “producto de propiedad registrada, pero de uso libre” (Raspberry Pi Foundation, 2016).

2.5.5.1 Sistemas operativos

El software del Raspberry Pi a diferencia del hardware sí es open source, siendo su sistema operativo oficial RaspBian, el cual es una versión adaptada de Debian. Adicional a éste, el SBC soporta otros sistemas operativos:

- Linux:
    - Android
    - Arch Linux
    - Debian
    - DietPi
    - Firefox OS

    - Gentoo Linux
    - Open webOS
    - PiBang Linux
    - Pidora
    - QtonPi
    - Raspbian
    - Slackware ARM
    - Ubuntu MATE
- Windows:
    - Windows 10
    - Windows CE
- Unix
- FreeBSD
- NetBSD
- AROS
- RISC OS 5
- Plan 9

3. CAPÍTULO III. METODOLOGÍA

3.1 Consideraciones iniciales

La idea principal de este trabajo se basa en implementar el uso de tecnologías como el reconocimiento de voz, redes inalámbricas, dispositivos IoT, entre otras más. Con la finalidad de brindar una ayuda en tareas comunes en el hogar aplicables a la domótica para personas con discapacidad visual. De esta manera para el diseño del sistema se han tomado en cuenta las siguientes consideraciones:

- El cliente, es decir una persona con discapacidad visual, usará una aplicación para dispositivos móviles inteligentes para controlar dispositivos de domótica de su casa mediante el uso de comandos de voz.
- En la casa del cliente se encontrará funcionando un servidor de domótica el cual ejecutará tareas como, por ejemplo: control de iluminación, temperatura, música, etc.
- El dispositivo móvil con la aplicación y el servidor de domótica deben estar conectados en una red informática común, por ejemplo, en la red local de la casa del cliente. Esta red local debe tener acceso a internet.
- El dispositivo móvil escuchará los comandos hablados del cliente, enviará esta información de audio a un servidor remoto el cual interpretará y responderá con información de texto.
- El dispositivo móvil analizará la respuesta del servidor remoto y asociará a un comando específico que se enviará al servidor de domótica.
- El servidor de domótica accionará diversos dispositivos en la casa del cliente dependiendo de cómo se ha configurado inicialmente. Por ejemplo, puede accionar lámparas, activar la música, encender o apagar aparatos eléctricos, cerrar seguros de las puertas de la casa, etc.

En base a lo anterior, los dispositivos necesarios para la correcta implementación del sistema son:

- Una aplicación para dispositivos móviles.
- Un servidor de domótica.
- Un servidor remoto que convierta audio a texto.

3.2 Soluciones

Para el desarrollo de la aplicación se consideró que se ejecutará en dispositivos Android, por lo cual será necesario usar el IDE Android Studio y el lenguaje de programación Java. La aplicación se llamará AVVI puesto que es hace referencia a "Asistente virtual controlado por voz para personas con discapacidad visual".
En cuanto a servidor de domótica, se consideró usar el dispositivo “IoT” conocido como Raspberry Pi en su versión 3, la razón de ello es que este dispositivo es de bajo coste. Además, cuenta con salidas GPIO disponibles y abiertas a su programación. Este dispositivo usa sistemas operativos que son distribuciones de Linux, cuenta con un módulo de Internet inalámbrico integrado y además puede ejecutar aplicaciones Java.
Por último, para el desarrollo del servidor web se consideró usar el lenguaje de programación Java en su versión Java EE, un servicio web SOAP que será consumible desde cualquier dispositivo a través de internet y la librería de reconocimiento de voz para Java CMU Sphinx. El servicio web se denominó: "Speech2TextWebService".

3.3 Clasificación servidor-cliente

Como se ha visto hasta el momento el sistema se encuentra conformado por tres elementos fundamentales los cuales se encuentran divididos geográficamente en dos partes que se llamarán: lado de cliente y lado del servicio web (comúnmente se usa el termino: lado del servidor. Sin embargo, se ha considerado más útil el que se muestra, puesto que lo que se brinda es un servicio web).

En el lado del cliente se encuentra el dispositivo Android con la aplicación y el Raspberry Pi, ambos conectados entre sí mediante la red local la cual a su vez debe estar conectada a internet. En el lado del servicio web se encuentra una máquina PC o Linux la cual se encuentra ejecutando el servicio web: "Speech2TextWebService".
Esta clasificación hace que sea más fácil entender el funcionamiento del sistema ya que el lado del cliente será la casa de la persona con discapacidad visual, mientras que el lado del servicio web será otro lugar, en el cual se encuentre

funcionando una máquina con el software adecuado. El túnel por el cual se envía y recibe información entre estos dos lados será Internet.

3.4 Ventajas y desventajas

Al tener al servicio web separado del cliente, se crean grandes ventajas:

- Inicialmente el cliente podrá usar este servicio desde cualquier parte del mundo. No importa el lugar donde se encuentre siempre podrá usar el servicio al enviar datos de audio obtenidos mediante la aplicación móvil. Sin embargo, es necesario que exista una conexión permanente del cliente y el servidor a Internet.
- El servicio web "Speech2TextWebService", al estar separado del cliente permite que se pueda realizar actualizaciones, mejoras o cambios sin afectar al cliente.
- Una máquina se encargará de gestionar la conversión de texto a voz en el lado del servicio web. Por lo cual el rendimiento y la velocidad de la conversión de audio a texto no depende del hardware que use el cliente sino de la capacidad del servicio web.
- Si se implementaría la conversión de audio a texto en el dispositivo Android del cliente, se necesitaría de un dispositivo con una gran potencia de procesamiento lo cual representa en un coste mayor para el cliente.
- Lo cierto es que utilizar un servidor únicamente para la conversión de audio a texto tiene un elevado costo de mantenimiento e implementación. Sin embargo, al ser usado para ayuda a personas con discapacidad visual se espera que este servicio pueda implementarse dentro de una entidad gubernamental.
- Al usar un dispositivo Android y un Raspberry Pi los cuales poseen módulos de Internet inalámbrico integrados, permite que el cliente pueda desplazarse libremente y usar la aplicación sin ningún inconveniente. Así mismo el Raspberry Pi puede colocarse en cualquier lugar en el que se disponga de señal de Internet inalámbrico.
- Un Raspberry es un dispositivo de bajo coste, lo cual hace que sea atractivo para el cliente.

3.5 Diagrama de funcionamiento

Tomando en cuenta todo lo que se ha mencionado hasta el momento, se muestran a continuación dos diagramas que representan el funcionamiento del sistema tanto en el lado del cliente como del servicio web:

Figura 18. Diagrama de funcionamiento del lado del cliente.

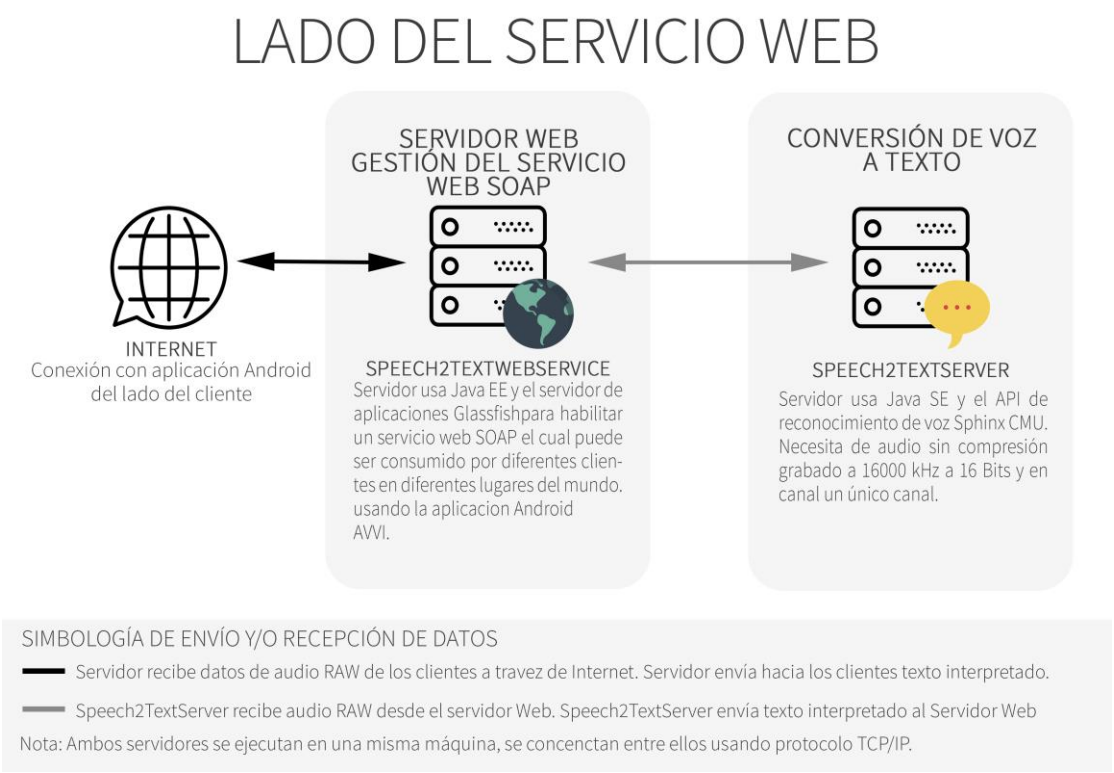


Figura 19. Diagrama de funcionamiento del lado del servidor.

3.6 Desarrollo de librerías propias

Para la implementación del sistema se han desarrollado un conjunto de librerías en lenguaje Java. Estas librerías abstraen gran parte del código lo que hace que sea más entendible y claro el proceso de programación. Todas las librerías generadas usan nombres y nomenclatura en inglés, ya que el código fuente es compartido en un repositorio de la plataforma GitHub y se espera que sea de ayuda para desarrolladores de todo el mundo. Para explicar la programación de cada parte del sistema se usará únicamente fragmentos de código que corresponderá al código de las clases principales ya que de esta manera se vuelve más sencillo y entendible el proceso de funcionamiento. Si el lector desea profundizar en el código de las librerías y los diferentes métodos usados, puede revisar la sección de Resultados, donde se detalla la creación del repositorio y su dirección web o en la parte final del trabajo, en Anexos, donde se encuentra todo el código creado.

3.7 Programación del lado del Cliente

3.7.1 Programación aplicación en Android

Figura 20. Logo de la aplicación para Android: AVVI.

La aplicación AVVI está programada para cualquier Tablet/teléfono inteligente con Android, la versión mínima del sistema operativo debe corresponder al API 10 de desarrollo (Android 2.3 o Gingerbread). La aplicación hace uso de las librerías de audio tanto para grabación, como para reproducción. También hace uso de la librería propia de Android que convierte texto a voz usando el motor TTS predeterminado del dispositivo, la librería de Java para envío y recepción de datos mediante protocolo TCP/IP y una librería externa llamada "ksoap" que permite al dispositivo consumir servicios web.

3.7.1.1 Librerías propias

Para el desarrollo de la aplicación se crearon tres librerías:

- **AudioEngine:** Esta librería se encarga de inicializar el motor de audio del dispositivo, cambiar sus propiedades, reproducir y grabar audio. Esta librería solo trabaja con audio sin compresión que es lo que requiere el servicio web para el posterior procesamiento (ver el Anexo 1).
- **ClientTCPEngine:** Esta librería permite la conexión con el servidor de domótica que se ejecuta en el Raspberry Pi. Permite enviar y recibir datos al servidor de manera correcta y en un formato compatible entre los dispositivos (ver el Anexo 2).

- **CommandEngine:** Esta librería permite transformar el texto recibido desde el servicio web a un texto que representa un comando que podrá interpretar el servidor de domótica, es decir el Raspberry Pi (ver el Anexo 3).

3.7.1.2 Componentes principales

Para el desarrollo de esta aplicación se han considerado 4 componentes principales:
En primer lugar, se tiene al código fuente en el cual se encuentra la lógica de la aplicación, es decir, cómo se harán las cosas cuando se presione un botón, o se presione en la pantalla, etc. Este código fuente es una clase de Java y en el caso de la aplicación AVVI, la clase principal se llama “MainActivity.java“. También se deben considerar a las librerías creadas: "AudioEngine.java", "ClientTCPEngine.java" y "CommandEngine.java". Se tiene así lo siguiente:

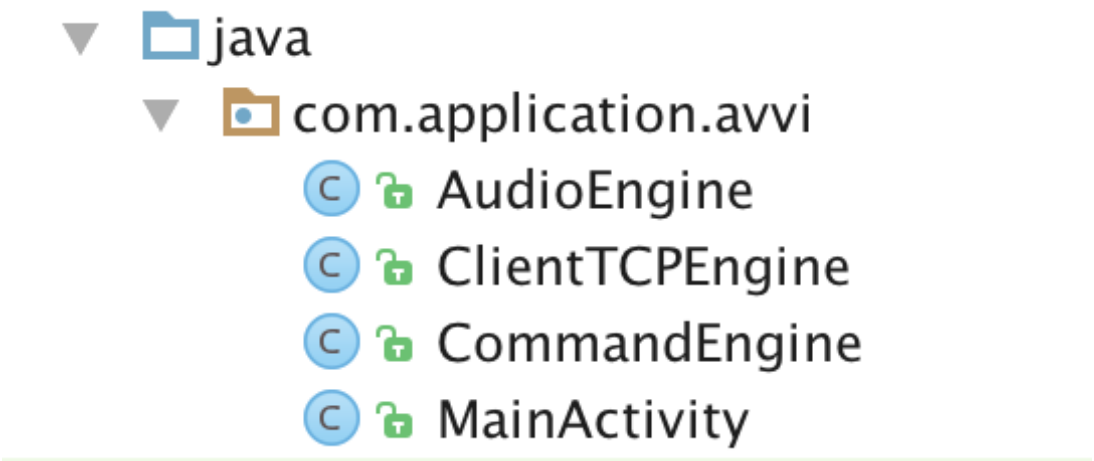


Figura 21. Vista del proyecto Android.

En segundo lugar, se tienen las vistas en las cuales se diseña la interfaz de usuario de la aplicación y que están ligadas al código fuente con la lógica de la aplicación. Estas vistas se desarrollan con el uso de archivos XML con su respectivo lenguaje de etiquetas. En el caso de la aplicación desarrollada se tienen dos vistas: “activity_main.xml” la cual contiene la interfaz de usuario

principal de la aplicación y “activity_loading_screen.xml” la cual muestra la pantalla de bienvenida de la aplicación. De esta manera se tiene lo siguiente:

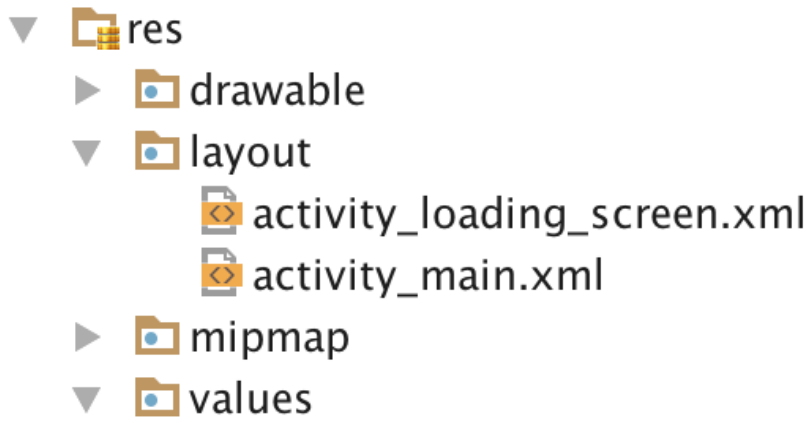


Figura 22. Vistas de la aplicación en la carpeta "layout".

En tercer lugar, se tiene al archivo “AndroidManifest.xml” el cual contiene características esenciales para el funcionamiento de la aplicación en el dispositivo Android. En este archivo se definen las vistas que se usarán y en qué momento serán usadas. Se definen los estilos de la aplicación, los permisos que necesita la aplicación para funcionar, entre otras más. Este archivo se encuentra en la carpeta: “manifests”, tal como se muestra a continuación:

Figura 23. Archivo “AndroidManifest.xml”.

Por último se tiene al archivo “build.grade” en el cual se definen las versiones de Android permitidas para el uso de la aplicación además de las librerías que serán usadas en la aplicación. Dentro de este archivo se pueden implementar las fuentes de librerías externas y sus repositorios externos. Se tendría lo siguiente:

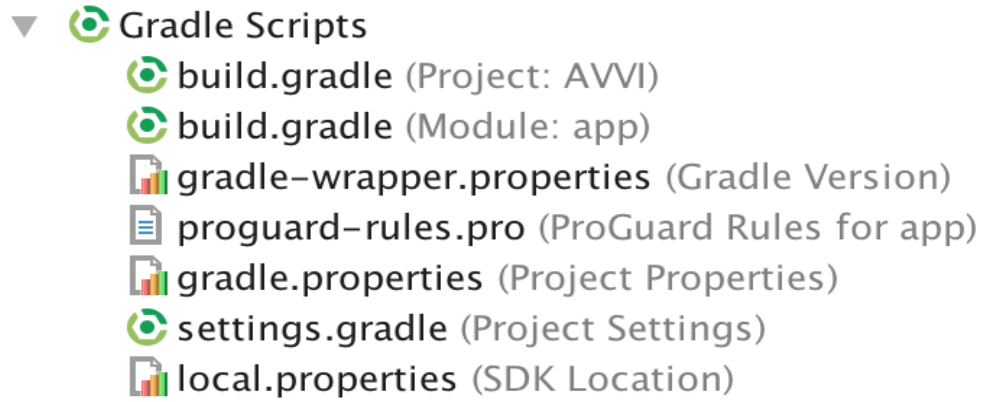


Figura 24. Archivos pertenecientes a la configuración de compilación.

3.7.1.3 Agregando librerías externas

La aplicación AVVI se comunica con un servicio web remoto mediante el protocolo SOAP. Android por defecto no incluye una librería para facilitar el desarrollo de la conexión, sin embargo, existe la librería externa “ksoap". Para ello se debe modificar el archivo “build.grade” y agregar las respectivas dependencias y el repositorio. Por lo cual queda de la siguiente manera:

```
...
repositories {
  maven { url
'https://oss.sonatype.org/content/repositories/ksoap2-android-
releases/' }
}

dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  testCompile 'junit:junit:4.12'
  compile 'com.android.support:appcompat-v7:24.2.1'
  compile 'com.google.code.ksoap2-android:ksoap2-
android:3.6.2'
```

```
}
...
```

También se puede ver que en este archivo se determina la versión mínima y máxima de Android que en teoría deberá ser la versión actual. Esto no es necesario modificar puesto que ya se define inicialmente al crear el proyecto:

```
...
defaultConfig {
  applicationId "com.application.avvi"
  minSdkVersion 10
  targetSdkVersion 24
  versionCode 1
  versionName "1.0"
}
...
```

3.7.1.4 Agregando permisos

La Aplicación AVVI deberá tener acceso a Internet para poder enviar y recibir datos hacia el Raspberry Pi y el servicio Web. Además, la aplicación debe tener los permisos para grabar y reproducir audio en el dispositivo Android. Para ello se modifica el archivo: "AndroidManifest.xml" y se agregan las siguientes líneas:

```
...
<uses-permission
android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.INTERNET" />
...
```

La aplicación tendrá un funcionamiento muy sencillo ya que es destinada para personas con discapacidad visual. De esta manera se diseñó una interfaz de usuario compuesta por un panel que cubre toda la pantalla que generará una acción al ser tocado. Se añadió texto el cual muestra la acción ejecutada, esto con propósitos de depuración. La pantalla de la aplicación principal queda de la siguiente manera (archivo "activity_main.xml", más información en el Anexo 10):

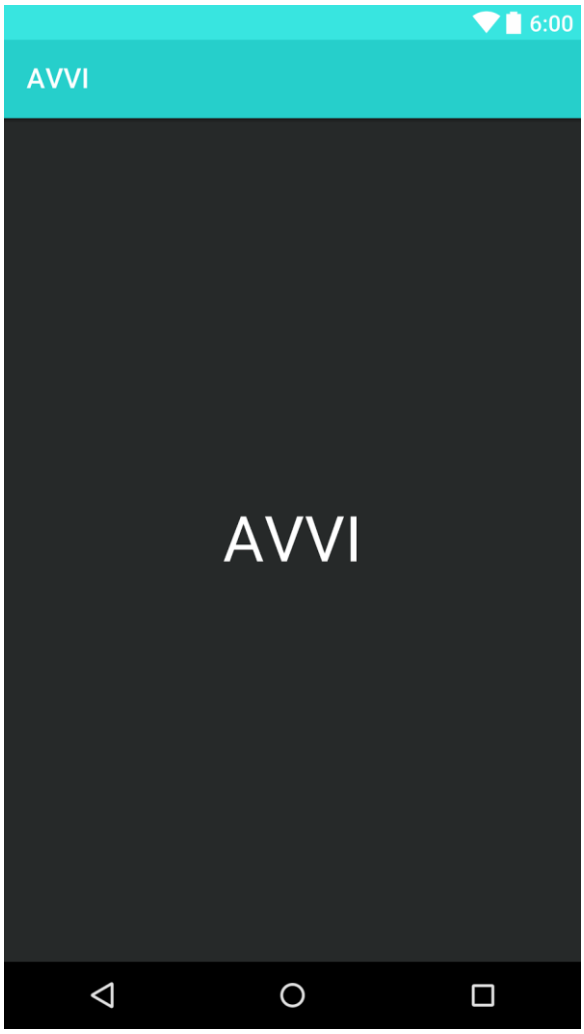


Figura 25. Vista principal de la aplicación.

Al iniciar la aplicación se requiere de un tiempo de espera mientras se cargan los componentes necesarios. De esta manera se creó esta vista y se incluyó información del proyecto. La vista se muestra a continuación:

Figura 26. Vista de la pantalla de bienvenida

En la actividad principal se encuentra todo el funcionamiento de la aplicación y su lógica. A continuación, se explica su funcionamiento utilizando fragmentos de código, si se desea conocer todo el código en conjunto revisar el Anexo 10.

Dentro del proyecto se usan algunas variables globales necesarias para que los distintos métodos puedan usar y cambiar sus valores. Estas variables son:

```
public byte[] byteArrayAudioRecorded;
```

Esta variable contendrá al audio que será grabado por el micrófono del dispositivo mediante el uso de la librería AudioEngine. Más información en el Anexo 1.

```
String byteArraySpeechString64;
```

Esta variable representa al audio grabado en un arreglo de bytes, pero que se ha codificado y convertido en una cadena de caracteres, esto es necesario para enviar esta información hacia el servicio web.

```
String resRasPi;
```

Esta variable contendrá la respuesta del Raspberry Pi, luego de que se envíe un comando para ser procesado.

```
TextToSpeech t1;
```

Esta variable crea una instancia de la clase de Android que permite convertir texto a voz.

```
boolean stateProcess = false;
```

Esta variable que solo puede tomar valores binarios, representa el monitoreo de cada ciclo de procesamiento de la actividad, es decir toma un valor positivo al iniciar la grabación de la voz del cliente y toma un valor negativo una vez que ha ejecutado el comando habado.

```
String rasPiCmd = null;
```

Esta variable representa el comando que ser enviará al Raspberry Pi para ejecutar una determinada acción de domótica.

```
CommandEngine commandEngine;
```

Esta variable representa a la clase CommandEngine.
Existen valores globales los cuales no pueden cambiar en el desarrollo de la aplicación por lo cual se los declara del tipo final. A continuación se muestran:

```
final int timeLoadingApp = 10000;
```

Esta constante representa el tiempo que se mostrará la vista de bienvenida.

```
final String NAMESPACE = "http://com/";
final String
URL="http://192.168.2.68:8080/SpeechWebService_1/SpeechToText?
wsdl";
final String METHOD_NAME = "speech2text";
final String SOAP_ACTION = "http://com/speech2text";
```

Estas constantes son valores de configuración del servicio web SOAP. Son necesarias para la implementación de la librería "ksoap".

```
final String RASPI_IP = "192.168.2.59";
final String RASPI_PORT = "7778";
```

Estas constantes representan a la ubicación en red del servidor de domótica, es decir del Raspberry Pi.

Al iniciar la aplicación se ejecutará el siguiente código que sobre-escribe la función onCreate. Se hará énfasis en las partes más importantes de la programación.
Con la siguiente función se agrega un ícono en la barra superior de la pantalla principal. Y se muestra la vista de bienvenida.

```
getSupportActionBar().setDisplayShowHomeEnabled(true);
getSupportActionBar().setLogo(R.drawable.eyew);
getSupportActionBar().setDisplayUseLogoEnabled(true);
setContentView(R.layout.activity_loading_screen);
```

Luego se inicializa la barra de progreso, una nueva instancia de la clase CommandEngine y se crea una nueva instancia en idioma español de la clase TextToSpeech que se utilizará para convertir texto a habla:

```
commandEngine = new CommandEngine();
final ProgressBar loadingBar =
(ProgressBar)findViewById(R.id.progressBar);

t1 = new TextToSpeech(getApplicationContext(), new
TextToSpeech.OnInitListener() {
  @Override
  public void onInit(int status) {
      if (status == TextToSpeech.SUCCESS) {

          Locale loc = new Locale("es", "", "");
          if (t1.isLanguageAvailable(loc) >=
TextToSpeech.LANG_AVAILABLE) {
              t1.setLanguage(loc);
          }
```

```
        }

    }
});
```

Se procede a crear un nuevo hilo de ejecución el cual crea una animación en la barra de progreso de la pantalla de bienvenida:

```
Thread loading = new Thread(){
    @Override
    public void run() {
        boolean isTerminated = false;
        int progressValue = 0;
        try {

            while (!isTerminated) {
                loadingBar.setProgress(progressValue);
                sleep(timeLoadingApp/10);
                progressValue+=10;

                if (progressValue > 90){
                    isTerminated=true;
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
};loading.start();
```

Por último, se crea otro hilo más de ejecución el cual espera que se complete la animación de la barra de progreso y usa la función “speak()”, para que el dispositivo comunique un mensaje de bienvenida al cliente. Por último, accede a la pantalla principal de la aplicación

```
Handler handler = new Handler();
handler.postDelayed(new Runnable() {
  @Override
  public void run() {
      speak("Hola soy Avi!, toca la pantalla para comenzar.");
      setContentView(R.layout.activity_main);
  }
}, timeLoadingApp);
```

3.7.1.5 Método Speak y Clases Asíncronas

La aplicación hace uso del siguiente método para que el dispositivo hable lo que ingresa como texto:

```
private void speak(String text){
  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
      t1.speak(text, TextToSpeech.QUEUE_FLUSH, null, null);
  }else{
      t1.speak(text, TextToSpeech.QUEUE_FLUSH, null);
  }
}
```

Cuando se requiere que se realicen diferentes tareas al mismo tiempo por la misma aplicación se hace uso de clases asíncronas, las cuales se generan en un nuevo hilo de ejecución y se procesan en segundo plano. Estas clases poseen métodos que deben ser sobre escritos y que se ejecutan de manera consecutiva. Cuando se hace un llamado a esta clase se ejecuta el código que se encuentra

en el método sobrescrito "doInBackground". Al terminar la ejecución en segundo plano, vuelve al hilo principal de la actividad y ejecuta "onPostExecute": Para el envío y recepción de datos hacia el servicio web se necesita de una clase que se ejecute en segundo plano y en un nuevo hilo de ejecución. Al final del proceso se llama a una nueva clase asíncrona la cual se encarga de enviar los comandos hacia el Raspberry Pi. Como se muestra a continuación:

```
public class Speech2TextWebServer extends
AsyncTask<Void,Integer,String> {

  @Override
  protected String doInBackground(Void... voids) {

       stateProcess = true;
       String res = null;
       SoapObject request = new SoapObject(NAMESPACE,
METHOD_NAME);
       request.addProperty("audioData",byteArraySpeechString64)
;
       SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);
       envelope.setOutputSoapObject(request);
       HttpTransportSE ht = new HttpTransportSE(URL);

       try {
           ht.call(SOAP_ACTION, envelope);
           SoapPrimitive response =
(SoapPrimitive)envelope.getResponse();
           res = response.toString();
       }
       catch (Exception e)
```

```
        {
            stateProcess = false;
            Log.i("Error: ",e.getMessage());
            e.printStackTrace();
        }
        return res;
    }

    protected void onPostExecute(String res) {
        commandEngine.createCommandsToSendAndSpeak(res);
        rasPiCmd = commandEngine.getRasPiCmd();
        if (rasPiCmd != null){
            setStringRes(commandEngine.getSpeakCmd());
            new RasPiServer().execute(RASPI_IP,RASPI_PORT);
        }else{
            setStringRes("Inténtalo de nuevo!");
        }
        speak(commandEngine.getSpeakCmd());
        stateProcess = false;
    }
}
```

La siguiente clase pertenece al envío y recepción de datos al RaspberryPi:

```
public class RasPiServer extends
AsyncTask<String,Integer,String>{
    @Override
    protected String doInBackground(String... strings) {
        stateProcess = true;
        String res = null;
```

```
        ClientTCPEngine clientTCPEngine = new
ClientTCPEngine("MAC", strings[0],
Integer.parseInt(strings[1]) );
        try {
            clientTCPEngine.init();
            clientTCPEngine.txCommand(rasPiCmd);
            res = clientTCPEngine.rxCommand();
        } catch (IOException e) {
            stateProcess = false;
            e.printStackTrace();
        }
        return res;
    }

    protected void onPostExecute(String res) {
        resRasPi = res;
        setStringRes(commandEngine.getSpeakCmd());
    }
}
```

3.7.1.6 Métodos relacionados a la UI

A continuación, se muestran los métodos que se relacionan con los elementos de la Interfaz de Usuario.

La aplicación en su pantalla principal comienza su procesamiento cuando la pantalla es tocada por el usuario. Para ello se hace uso del siguiente método:

```
public void onClickScreen(View vista) throws IOException,
InterruptedException {
    t1.stop();
```

```
  AudioEngine audioEngine = new AudioEngine();
  audioEngine.init();
  byteArrayAudioRecorded = audioEngine.recordAudio();
  byteArraySpeechString64                                    =
Base64.encodeToString(byteArrayAudioRecorded,Base64.NO_WRAP);
  Speech2TextWebServer webService = new Speech2TextWebServer();
  webService.execute();
  setStringRes("Procesando...");
  speak("Espera un momento");
}
```

Es importante considerar que, para enviar audio hacia el servicio web, este debe ser codificado en un conjunto de caracteres como se muestra a continuación:

```
...
byteArraySpeechString64 =
Base64.encodeToString(byteArrayAudioRecorded,Base64.NO_WRAP);
...
```

En la vista principal de la aplicación se tiene un recuadro de texto el cual muestra los diferentes comandos que serán activados, para poder enviar u obtener datos de ese recuadro se hace uso del siguiente método:

```
public void setStringRes(String res){
  TextView resSpeechText = (TextView)
findViewById(R.id.resString);
  resSpeechText.setText(res);
}
```

3.7.1.7 Orden de funcionamiento

La aplicación comienza a actuar cuando el cliente toca la pantalla del dispositivo lo cual llama al método "onClickScreen", este método hace uso de la librería "AudioEngine" para grabar la voz desde el micrófono. Estos datos de audio son codificados a una secuencia de caracteres y se envían al servicio web por medio de la clase asíncrona "SpeechToTextWebService". Se obtiene el resultado en texto interpretado por el servicio web e inmediatamente se llama a una nueva clase asíncrona llamada "RasPiServer". Esta se encarga de ejecutar el comando que se ha enviado, usando las librerías "CommandEngine" y "ClientTCPEngine". Al final se cierran todos los hilos de ejecución creados, la aplicación muestra en pantalla el comando de domótica ejecutado y al mismo tiempo reproduce este comando para que el cliente conozca si se realizó la tarea satisfactoriamente o debe repetir la acción.

3.7.2 Programación servidor Domótica

El servidor de domótica se ejecuta en el Raspberry Pi aprovechando sus módulos GPIO, Internet, Bluetooth, etc, los cuales son muy útiles en aplicaciones de domótica. El Raspberry Pi posee una distribución Linux conocida como Raspbian la cual permite programar directamente en el Raspberry Pi. Sin embargo, no es la única forma de realizar un programa, ya que al ser totalmente compatible con la versión de Java ME para sistemas embebidos, este dispositivo puede ser programado remotamente. Es decir, se programa desde un ordenador remoto y se ejecuta el código en el Raspberry Pi, aprovechando la conexión de ambos dispositivos a una red local.

3.7.2.1 Librerías propias

Para la programación del servidor de domótica se crearon dos librerías, las cuales se describen continuación:

- **Librería “CommandEngineServer”:** Esta clase permite interpretar el comando enviado desde el dispositivo Android y luego ejecutar la acción en el Raspberry Pi (ver Anexo 4).

- **Librería "RasPiGPIOEngine":** Esta clase permite el uso del módulo GPIO del Raspberry Pi (ver Anexo 5).

3.7.2.2 Configuración del Raspberry Pi

Lo primero que se debe hacer es fijar una dirección IP para que el dispositivo pueda ser reconocido fácilmente en la red local. Para realizar esto se debe ejecutar el siguiente código desde el Terminal, el cual permitirá editar el archivo de configuración correspondiente a la red:

```
sudo nano /etc/network/interfaces
```

Luego se procede a editar el archivo de configuración, y se obtiene lo siguiente:

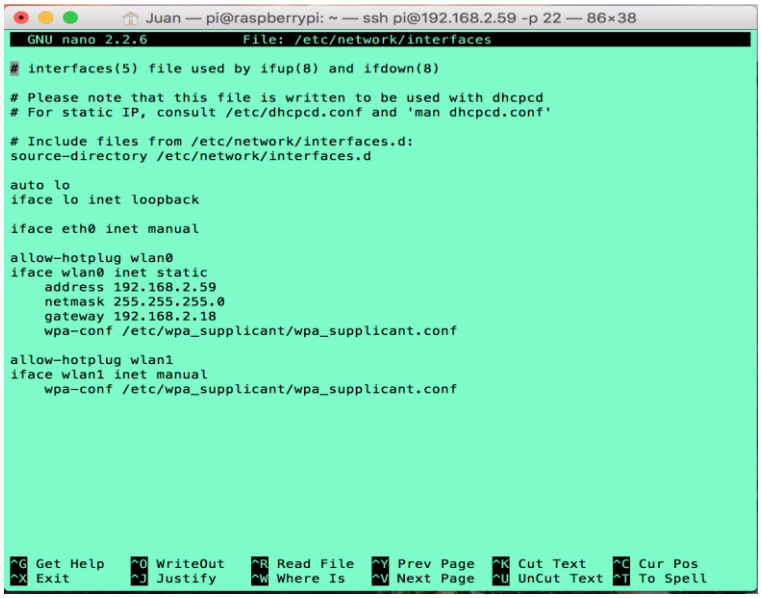


Figura 27. Terminal del Raspberry Pi, configurando dirección IP.

Luego se procede a resetear al dispositivo y una vez que se ha restablecido se comprueba que su dirección IP ha cambiado al usar el siguiente comando:

```
ifconfig
```

Se obtiene lo siguiente:

```
pi@raspberrypi:~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:1a:91:46
          inet6 addr: fe80::f4cb:209a:44ff:98ce/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:200 errors:0 dropped:0 overruns:0 frame:0
          TX packets:200 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:16656 (16.2 KiB)  TX bytes:16656 (16.2 KiB)

wlan0     Link encap:Ethernet  HWaddr b8:27:eb:4f:c4:13
          inet addr:192.168.2.59  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::e182:a4ba:d433:318d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1487 errors:0 dropped:853 overruns:0 frame:0
          TX packets:309 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:208433 (203.5 KiB)  TX bytes:95090 (92.8 KiB)
```

Figura 28. Terminal Raspberry Pi, comando "ifconfig".

Para poder ejecutar programas Java en el Raspberry Pi, es necesario comprobar que tenga instalado el respectivo JDK. Para ello se usa el siguiente comando:

```
java -version
```

Si se obtiene un error se procede a instalar Java usando el siguiente comando:

```
sudo apt-get install oracle-java8-jdk
```

Se comprueba nuevamente:

```
pi@raspberrypi:~ $ java -version
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) Client VM (build 25.65-b01, mixed mode)
pi@raspberrypi:~ $
```

Figura 29. Terminal Raspberry Pi, comando: "java -version".

3.7.2.3 Configuración de NetBeans

El Raspberry Pi está listo para ejecutar programas Java. Pero para poder ser programado es necesario configurar el IDE Netbeans que se ejecuta en un ordenador remoto. Para ello, en el ordenador remoto, se agrega un nuevo perfil de ejecución desde las propiedades del proyecto como se muestra a continuación:

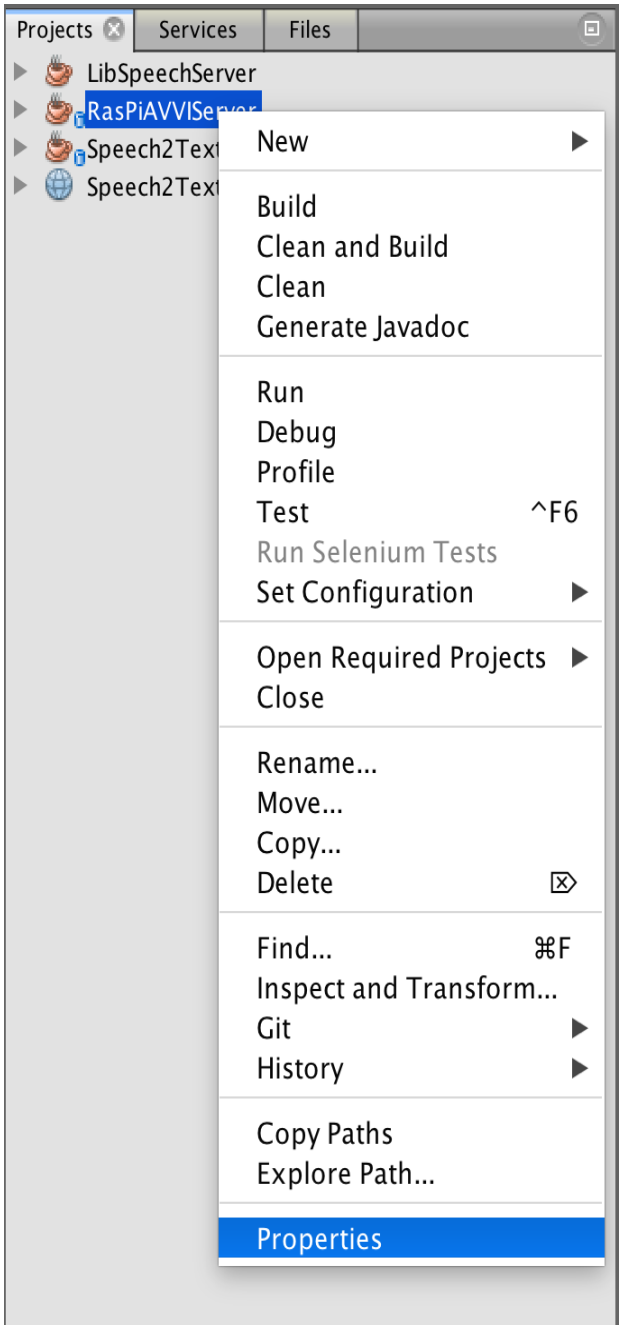


Figura 30. Propiedades del proyecto NetBeans.

Se accede a la opción "run" y en la ventana se selecciona "Manage Platforms...":

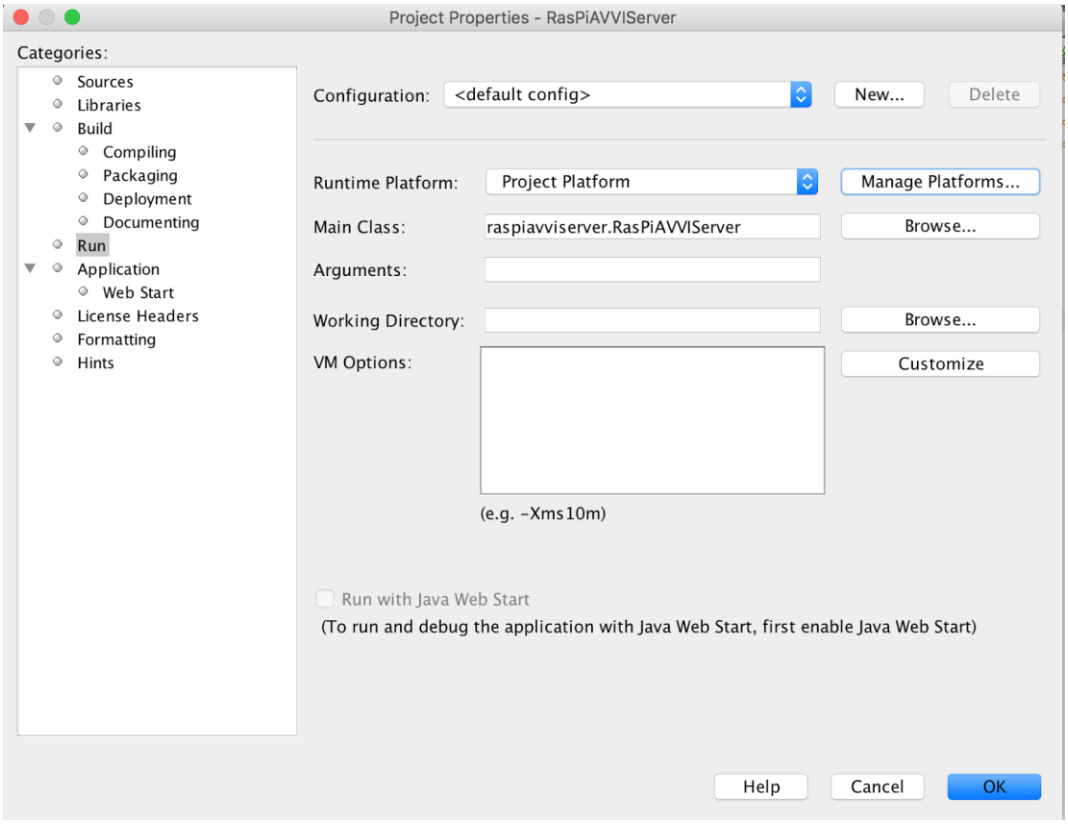


Figura 31. Propiedades del proyecto, opción "run".

Se crea un nuevo perfil usando "Java Remote Platform Edition", se agregan los datos del Raspberry Pi:

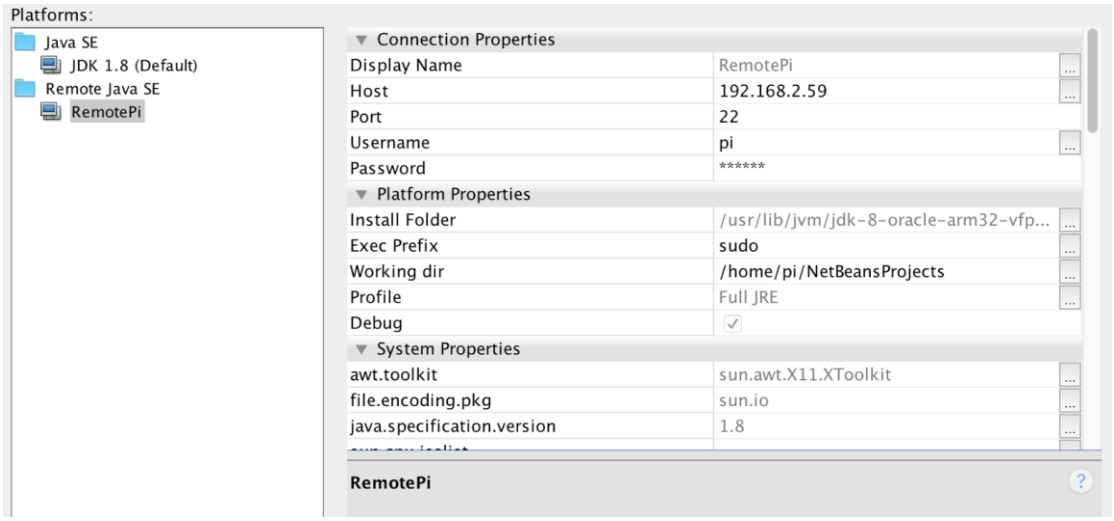


Figura 32. Administrador de plataformas de NetBeans.

Se comprueba si son correctos los datos introducidos haciendo click en "Test Platform":

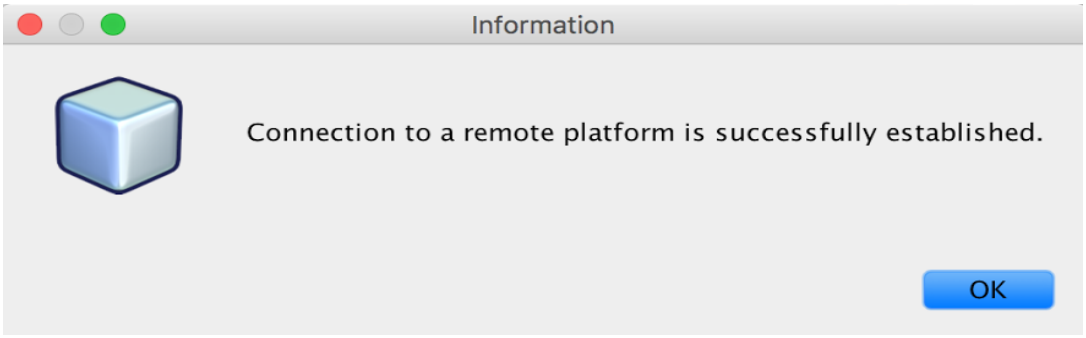


Figura 33. Cuadro de diálogo de conexión exitosa con el Raspberry Pi.

Una vez que se ha realizado esta comprobación, se procede a elegir el nuevo perfil tal como se muestra a continuación:

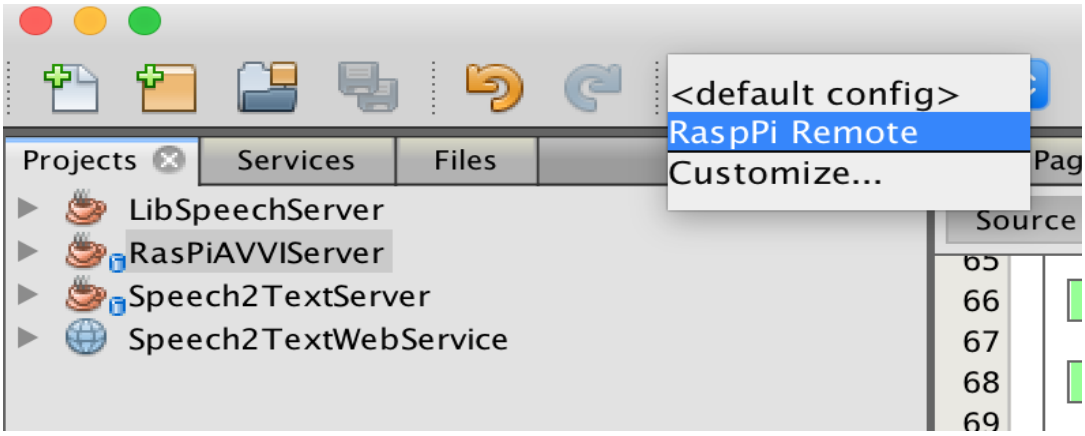


Figura 34. Selección de nuevo perfil de ejecución.

Por último, para programar el servidor de domótica es necesario agregar la librería que permite usar el módulo GPIO en el Raspberry Pi, la librería se llama: “pi4j-core”. Una vez que ha sido añadida se obtiene lo siguiente:

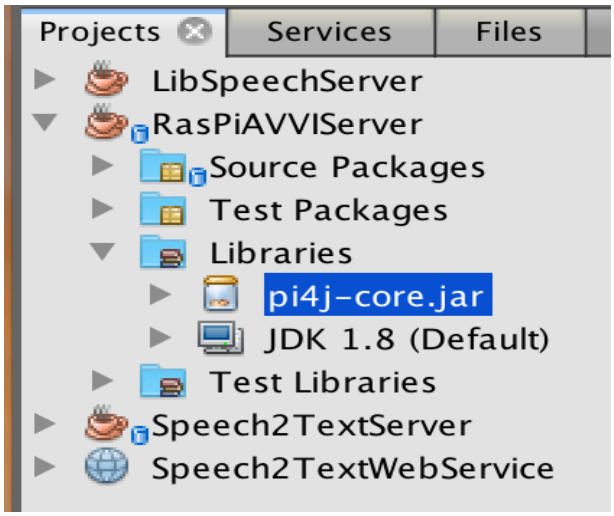


Figura 35. Librería "pi4j-core.jar" añadida al proyecto.

3.7.2.4 Desarrollo de la clase principal

La clase principal se encarga de recibir las peticiones de los clientes, es decir de los dispositivos Android hacia el Raspberry Pi. Cada vez que hay una petición se crea un nuevo hilo de ejecución, de esta manera se evitan problemas de

concurrencia. A continuación, se mostrarán los fragmentos de código más importantes de la clase principal, para conocer todo el código ver el Anexo 11. Inicialmente se crean tres variables las cuales serán usadas durante toda la ejecución del programa como se muestra a continuación:

```
...
Socket id;
static CommandEngineServer commandEngine;
String command = null;
...
```

En primer lugar, se tiene a la variable “id” la cual será usada para la creación de una conexión con los clientes a través del protocolo TCP/IP. Luego está la variable “commandEngine” la cual controla la ejecución de las tareas de domótica y por último se tiene una cadena de caracteres denominada “command” la cual almacena el comando recibido desde el dispositivo Android.

3.7.2.5 Método "main" de la clase

El código que se encuentra en este método es el que se ejecuta una vez que se inicia el programa. En este método se agregó información del servidor para que sea mostrado en la consola:

```
...
System.out.println("***************       Servidor       RasPi
AVVI  ***************");
       System.out.println("***************   Reconocimiento  de
voz  ***************");
       System.out.println("**********************  UDLA  ****
********************");
       System.out.println("***************      ING     SONIDO
ACUSTICA  ***************");
```

```
        System.out.println("******* Creadores: Juan Chango -
Andrés Márquez *******");
        System.out.println("");
        System.out.println("Inicializando Servidor Raspberry
Pi...");
...
```

Para crear un servidor y una conexión TCP/IP se hace uso de la librería: “java.net”. Inicialmente se crea un objeto del tipo “ServerSocket”, el cual necesita de parámetro de entrada el puerto en el cual el servidor escuchará a los clientes. Cuando un cliente es reconocido se crea un objeto del tipo “Socket” y se crea un hilo de ejecución únicamente para el cliente. Sin importar que exista un hilo de ejecución en proceso, si otro cliente se conecta al servidor se creará otro hilo de ejecución. El siguiente fragmento de código muestra lo que se ha explicado:

```
...
ServerSocket ss = new ServerSocket(7778);

        commandEngine = new CommandEngineServer();
        commandEngine.init();
        while (true) {
            System.out.println("Escuchando en puerto:7778...");
            Socket s = ss.accept();
            System.out.println("Cliente encontrado");
            RasPiAVVIServer t = new RasPiAVVIServer(s);
            t.start();
        }
...
```

Durante la ejecución de cada hilo, el servidor espera un comando desde el cliente usando un objeto del tipo “BufferedReader” y luego de ello el servidor envía el

estado del comando, es decir si fue ejecutado o no, esto lo hace mediante un objeto del tipo “PrintWriter” el método “runCommand” de la librería “CommandEngine” (para más información ver Anexo 4), tal como se muestra a continuación:

```
...
try {
            PrintWriter out = new
PrintWriter(id.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new
InputStreamReader(id.getInputStream()));

            System.out.print("Cliente conectado: ");
            command = in.readLine();
            System.out.println(command);
            String res = commandEngine.runCommand(command);
            out.println(res);
        } catch (IOException | InterruptedException ex) {
            Logger.getLogger(RasPiAVVIServer.class.getName()).l
og(Level.SEVERE, null, ex);
        }
...
```

3.7.2.6 Servidor en ejecución

Al iniciar el servidor este lanza un mensaje de bienvenida e inmediatamente comienza a escuchar a los clientes en el puerto 7778. A continuación, se muestra lo que se imprime en la consola:

```
Connecting to 192.168.2.59:22
cmd : cd '/home/pi/NetBeansProjects/RasPiAVVIServer'; '
**********************  Servidor RasPi AVVI  **********************
**********************  Reconocimiento de voz  **********************
**************************  UDLA  **************************
**********************  ING SONIDO ACUSTICA  **********************
******* Creadores: Juan Chango – Andrés Márquez *******
Inicializando Servidor Raspberry Pi...
Escuchando en puerto:7778...
```

Figura 36. Consola de NetBeans, mensaje de bienvenida del servidor de domótica.

Cuando un cliente se ha conectado lanza un mensaje que indica que se ha creado un nuevo hilo de ejecución, pero al mismo tiempo el servidor vuelve a escuchar a otros clientes en el hilo principal de ejecución.

3.8 Programación del lado del Servicio Web

El lado del servidor está compuesto por dos elementos esenciales, un servidor web y un servidor que convierte voz a texto. Ambos servidores se encuentran interconectados entre sí por medio del protocolo TCP/IP y su vez tienen conexión a Internet. Estos servidores son programas informáticos que pueden ejecutarse en diferentes máquinas o en una sola. Es decir, ambos servidores pueden correr sin problema alguno en un ordenador que permita el uso de Java EE. Pero si se quisiera, también se podrían utilizar dos computadoras independientes para cada servidor conectadas entre sí mediante una red local.

De manera general, se puede decir que el servidor web es como una puerta de acceso hacia el servidor "Speech2TextServer". Por ello cualquier cliente que quiera usar el servicio deberá pasar por esa puerta la cual provee un control de seguridad. Así mismo el servidor web permite brindar el servicio a todo el mundo, ya que provee un fichero estandarizado WSDL el cual describe el servicio web y puede ser usado con cualquier lenguaje de programación o plataforma. Estos dos servidores en conjunto representan un servicio web, el cual puede ser consumido desde cualquier parte del mundo y desde cualquier tipo de cliente.

En el caso de este trabajo, el servicio será consumido desde dispositivos Android con la aplicación AVVI.

3.8.1 Programación del Servidor "Speech2TextServer"

Este servidor se encarga de convertir información de voz a texto. La programación del servidor se realizó considerando que el envío y recepción de datos será mediante protocolo TCP/IP. Por ello se desarrollaron un conjunto de librerías las cuales permiten el transporte ordenado de información con el servidor web.

Para poder convertir voz a texto se usó el API para Java de Sphinx CMU, el cual está integrado en el servidor "Speech2TextServer", este API posee una base de datos con diferentes modelos acústicos para cada idioma. Estos modelos se basan en el entrenamiento de una gran cantidad de palabras y los diferentes fonemas que conforman el habla. El modelo más completo es el Inglés puesto que tiene un mayor entrenamiento y un gran diccionario. El modelo del idioma español es un modelo entrenado con acentos de personas de España y además ha sido entrenado con un número menor de palabras en su diccionario, lo cual lo hace menos preciso al usar con hablantes ecuatorianos. Sin embargo Sphinx CMU permite entrenar un nuevo modelo acústico o adaptar el existente.

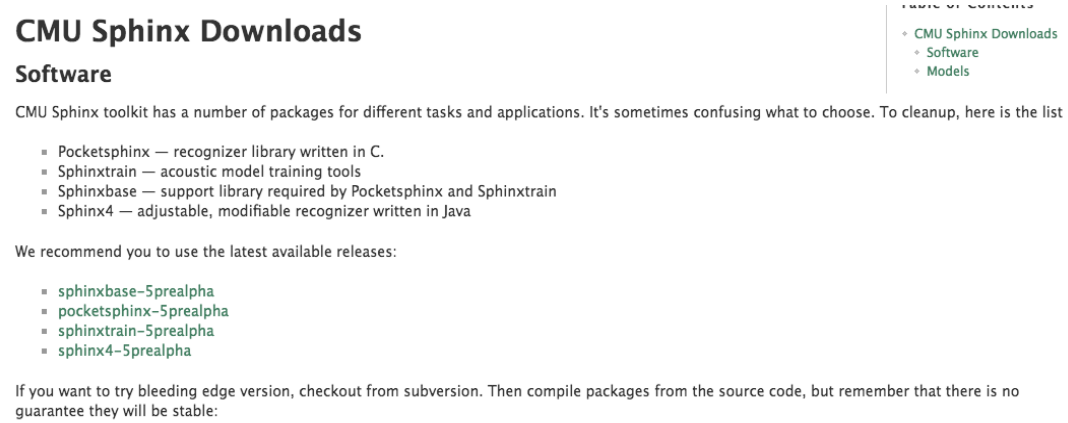
La primera alternativa es ideal si se requiere de un modelo con una alta precisión y una gran cantidad de palabras en su diccionario, sin embargo el tiempo y el esfuerzo que conlleva crear todo eso es enorme y no es viable para el presente proyecto. Por otro lado, la segunda alternativa es adaptar al modelo acústico español lo cual incrementa la precisión del convertidor usando el diccionario ya existente, el proceso lleva menos tiempo que el anterior y los resultados son bastante buenos.

3.8.1.1 Adaptación del modelo acústico

Sphinx CMU permite adaptar el modelo acústico de una manera fácil y rápida usando un conjunto de comandos y herramientas incluidas en su API. Esta adaptación se realizó remotamente en un ordenador con sistema operativo

Ubuntu Server, una distribución popular de Linux. Se consideró al sistema operativo Linux ya que presenta mayores facilidades para la realización de la adaptación.

Primero es necesario obtener el paquete de herramientas de entrenamiento de Sphinx CMU: "Sphinxtrain", estos se los pueden encontrar en el siguietne link: http://cmusphinx.sourceforge.net/wiki/download.



CMU Sphinx Downloads

Software

- CMU Sphinx Downloads
  - Software
  - Models

CMU Sphinx toolkit has a number of packages for different tasks and applications. It's sometimes confusing what to choose. To cleanup, here is the list

- Pocketsphinx — recognizer library written in C.
- Sphinxtrain — acoustic model training tools
- Sphinxbase — support library required by Pocketsphinx and Sphinxtrain
- Sphinx4 — adjustable, modifiable recognizer written in Java

We recommend you to use the latest available releases:

- sphinxbase-5prealpha
- pocketsphinx-5prealpha
- sphinxtrain-5prealpha
- sphinx4-5prealpha

If you want to try bleeding edge version, checkout from subversion. Then compile packages from the source code, but remember that there is no guarantee they will be stable:

Figura 37. Sitio web de descarga del API Sphinx CMU.

Tomado de (CMU Sphinx, 2016)

En la carpeta descargada con el nombre: "sphinxtrain-5prealpha" se encuentran los siguientes archivos:

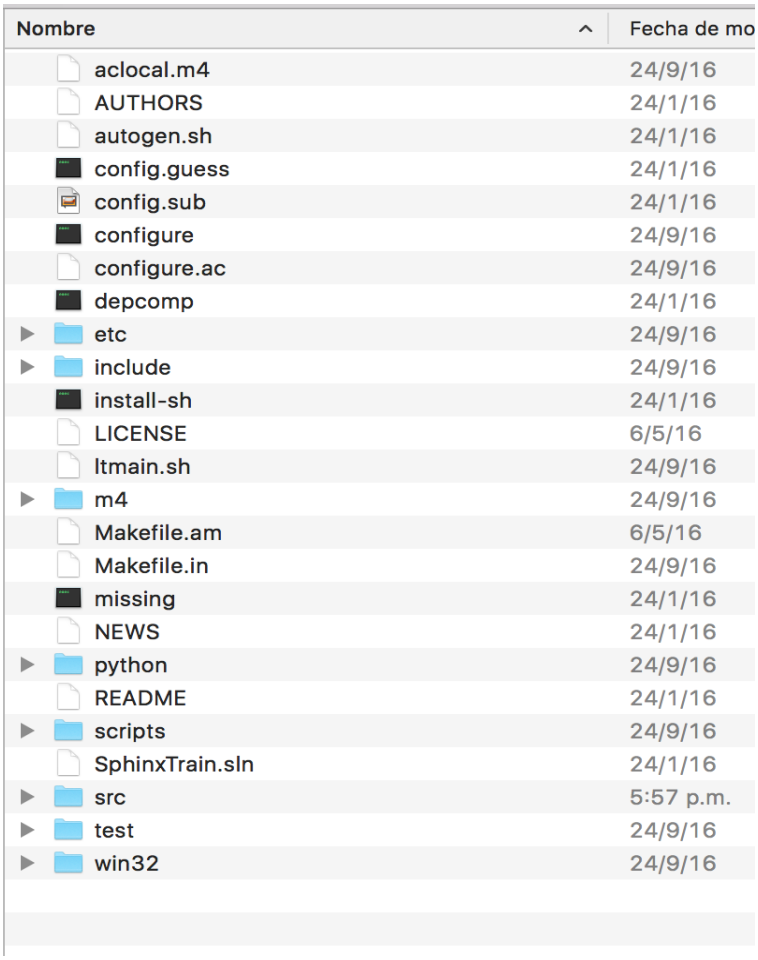


Figura 38. Archivos descargados del API CMU Sphinx.

Tomado de (CMU Sphinx, 2016)

Se debe abrir un terminal desde el directorio de la carpeta mostrada e instalar los archivos en la máquina Linux usando los siguientes comandos:

```
./autogen.sh
./configure
make
make install
```

La instalación generó nuevos archivos en el sistema necesarios para la adaptación del modelo acústico. A continuación, se debe copiar los archivos generados por la instalación con los nombres: "bw", "map_adapt" y

"mk_2sendump" que se encuentran en la dirección: "/usr/local/libexec/sphinxtrain" a una nueva carpeta con el nombre: "CarpetaDeTrabajo", la cual debe ser creada en el directorio principal de Linux.

Luego se procede a instalar en la máquina Linux las herramientas necesarias usando los siguientes códigos en el terminal:

```
sudo apt-get install sphinxbase-utils
sudo apt-get install pocketsphinx
sudo apt-get install sphinxtrain
```

A continuación, se procede a descargar el modelo acústico del idioma español desde el mismo link presentado anteriormente, se accede a la siguiente página:

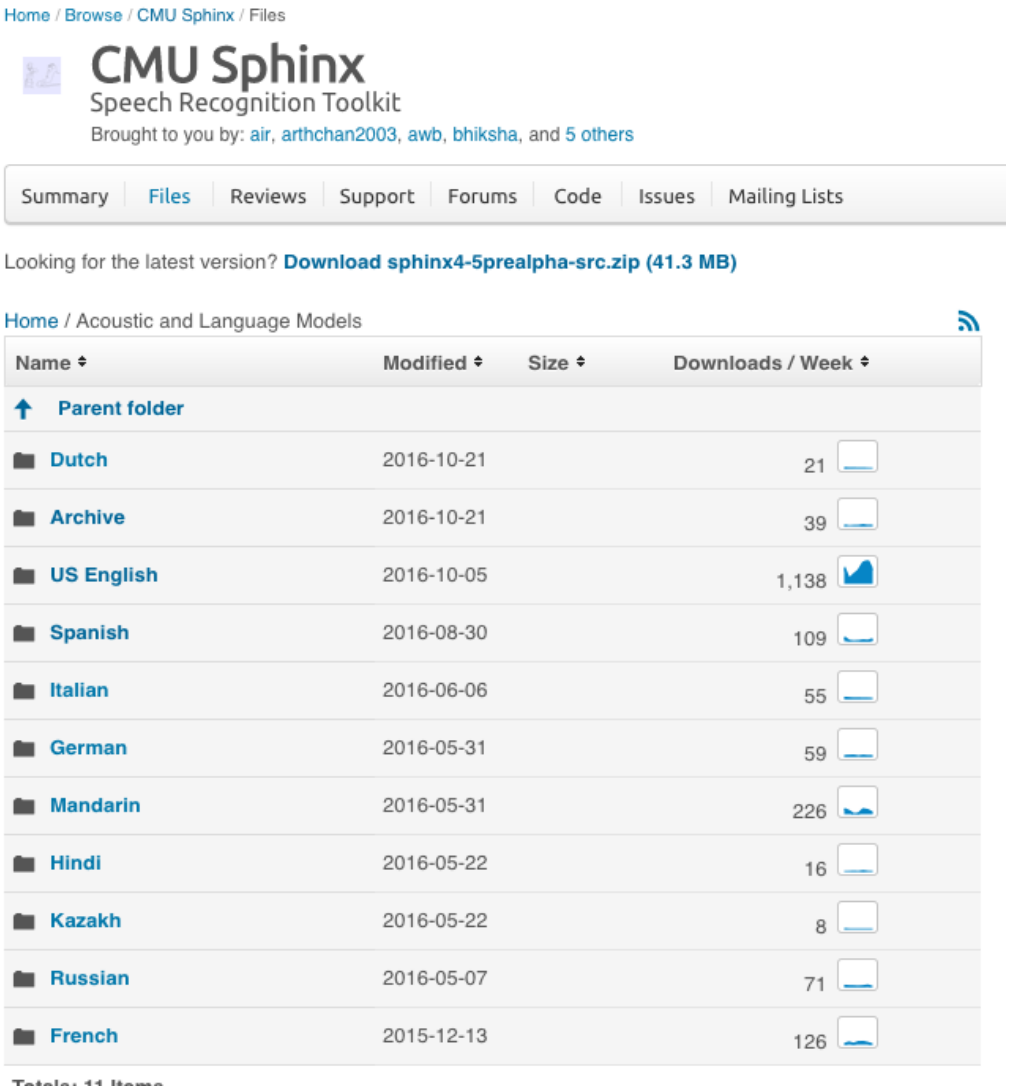


Figura 39. Repositorio de modelos acústicos de CMU Sphinx.

Tomado de (CMU Sphinx, 2016)

Una vez completa la descarga, la carpeta principal contiene los archivos mostrados a continuación:

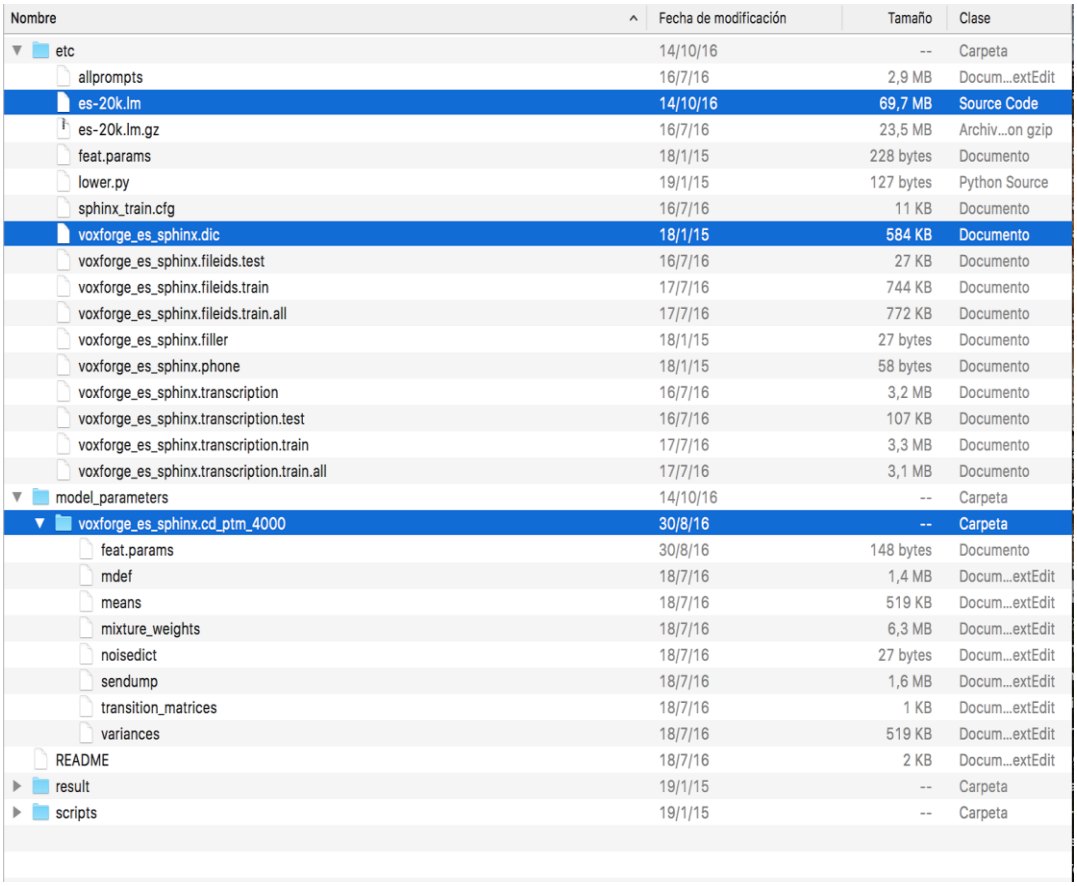


Figura 40. Carpeta descargada con el modelo acústico español.

Los dos archivos seleccionados y la carpeta, son necesarios para el funcionamiento del API de Sphinx CMU. A continuación, se describe sus características:

- Los archivos dentro de la carpeta de nombre: "voxforge_es_sphinx.cd_ptm_40000", representan al modelo acústico el cual contiene las propiedades de cada "senone" para el idioma español.
- El archivo: "voxforge_es_sphinx.dic" corresponde con el diccionario, en el cual se encuentran todas las palabras disponibles para la transcripción y sus fonemas.
- El archivo "es-20k.lm" corresponde al modelo de lenguaje el cual contiene información de probabilidad que asocia una palabra con otra, ya que en el proceso del habla siempre una palabra tiene relación con otra para dar un mensaje con sentido.

Estos archivos son copiados a la carpeta creada anteriormente en el ordenador Linux con el nombre: "CarpetaDeTrabajo". Para trabajar con mayor facilidad con los archivos copiados, se cambia el nombre de los archivos de la siguiente manera:

1. El nombre de la carpeta "voxforge_es_sphinx.cd_ptm_40000" por el nuevo nombre: "esp-ec".
2. El nombre del archivo: "voxforge_es_sphinx.dic" por el nuevo nombre: "es.dict".

Y por último se deben descargar los archivos que contiene información del procesamiento de audio que se usará para generar el nuevo modelo acústico adaptado, los archivos se llaman: "arctic20.fileids" y "arctic20.transcription".

Se los encuentra en el siguiente link:
http://cmusphinx.sourceforge.net/wiki/tutorialadapt
Estos archivos deben copiarse en la carpeta: "CarpetaDeTrabajo".

3.8.1.2 Creación de los archivos de audio

El proceso de adaptación requiere crear varios archivos de audio los cuales deben contener un conjunto de frases. Cada archivo de audio debe ser creado en formato wav, 16.000 kHz de frecuencia de muestreo, 16 bits por muestra, monofónico y codificado en PCM little/big-endian.m Cada archivo de audio debe tener un nombre el cual debe estar listado en el archivo: "arctic20.fileids" y su transcripción deberá estar escrito en el archivo "arctic20.transcription". Para ello se han creado 36 archivos de audio como se muestra a continuación:

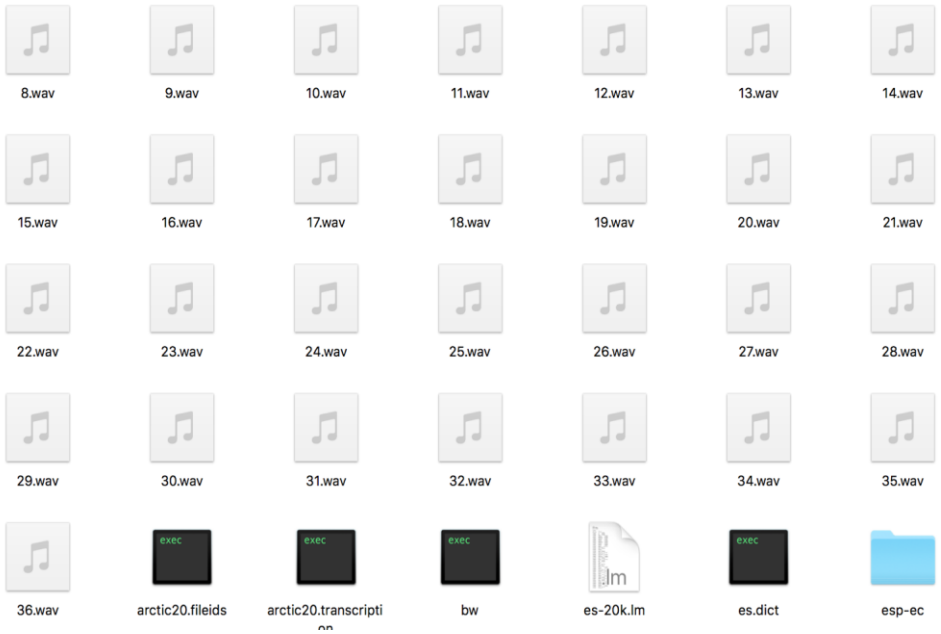


Figura 41. Archivos de audio creados para la adaptación del modelo acústico.

Los nombres de los archivos se encuentran listados en el archivo "arctic20.fileids", el cual se muestra a continuación:

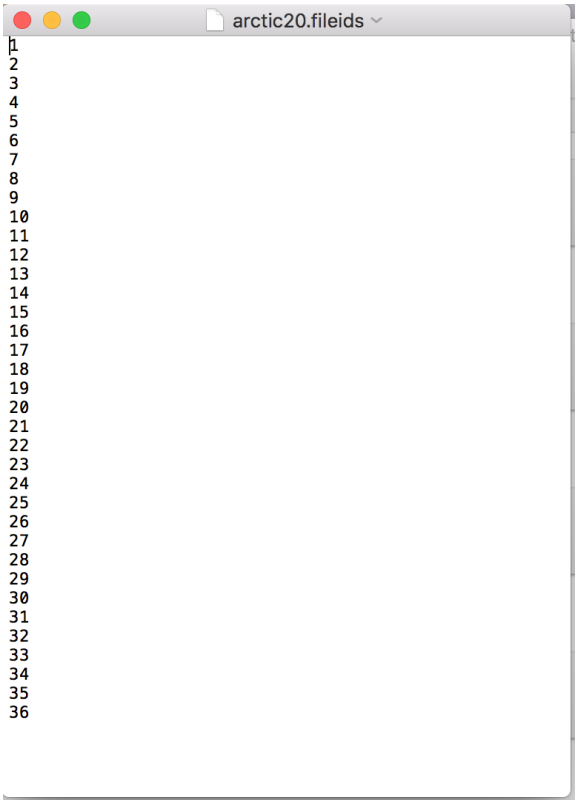


Figura 42. Archivo "arctic20.fileids" abierto en un editor de texto.

Y el archivo "arctic20.transcription" contiene un conjunto de comandos los cuales serán usados en la aplicación Android para el control de domótica. El archivo queda de la siguiente manera:

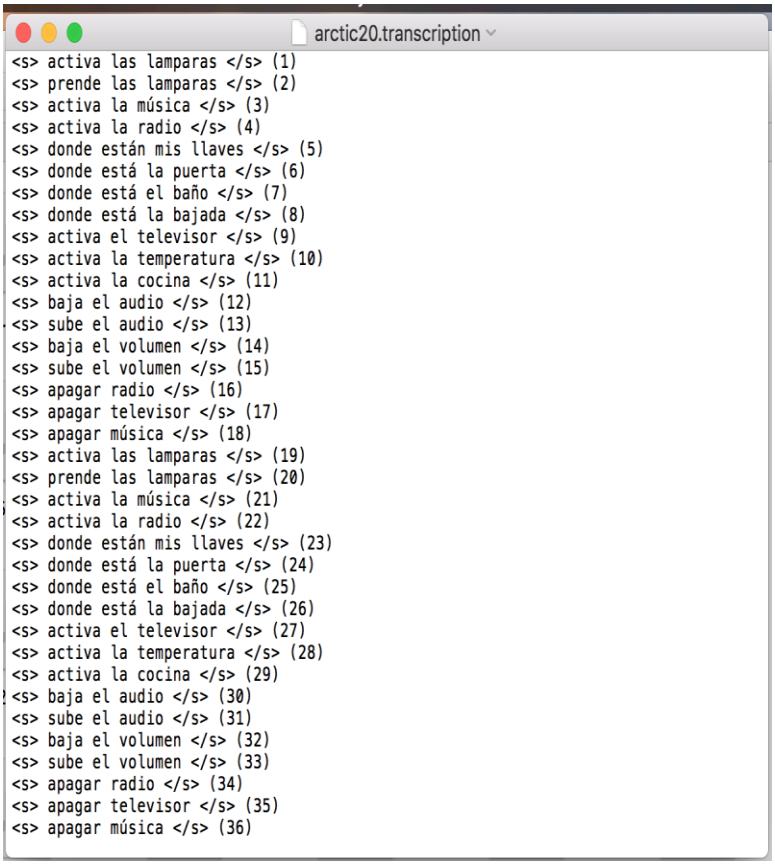


Figura 43. Archivo "arctic20.transcription" abierto desde un editor de texto.

Se puede ver con la imagen anterior que los comandos se repiten ya que fueron grabados usando dos voces diferentes. Con este entrenamiento no solo se adapta el modelo acústico con los comandos usados, sino que de manera global cada fonema será adaptado. Por último, se debe copiar en la misma ubicación a la carpeta: "esp-ec" y renombrarla como "esp-ec-adapt", la cual, al finalizar el proceso, contendrá el nuevo modelo acústico adaptado.

3.8.1.3 Adaptación usando Linux

Una vez que se han generado estos archivos ya se puede comenzar a adaptar el modelo acústico. Para ello se debe tener en el directorio principal de la máquina Linux o en cualquier directorio conocido a la carpeta: "CarpetaDeTrabajo" con todos los archivos que se han creado hasta el momento. Desde la ubicación de la carpeta mencionada se debe abrir el Terminal y ejecutar los siguientes comandos:

Inicialmente se deben generar los archivos con las nuevas características acústicas del modelo. Para ello se usa el siguiente código:

```
sphinx_fe -argfile esp-ec/feat.params \
       -samprate 16000 -c arctic20.fileids \
      -di . -do . -ei wav -eo mfc -mswav yes

pocketsphinx_mdef_convert -text esp-ec/mdef esp-ec/mdef.txt
```

El siguiente paso consiste en recolectar datos estadísticos de los datos de adaptación, para ello se usa el comando "bw" perteneciente al conjunto de herramientas de "SphinxTrain". Se usa el siguiente código:

```
./bw \
-hmmdir esp-ec \
-moddeffn esp-ec/mdef.txt \
-ts2cbfn .ptm. \
-feat 1s_c_d_dd \
-svspec 0-12/13-25/26-38 \
-cmn current \
-agc none \
-dictfn es.dict \
-ctlfn arctic20.fileids \
-lsnfn arctic20.transcription \
-accumdir .
```

Por último, se genera el nuevo modelo acústico usando la función "map_adapt". Esta función actualiza cada parámetro del modelo, es decir a cada archivo que se encuentra dentro de la carpeta creada al inicio llamada: "esp-ec-adapt". Para ello se usa el siguiente código:

```
./map_adapt \
   -moddeffn esp-ec/mdef.txt \
   -ts2cbfn .ptm. \
   -meanfn esp-ec/means \
   -varfn esp-ec/variances \
   -mixwfn esp-ec/mixture_weights \
   -tmatfn esp-ec/transition_matrices \
   -accumdir . \
   -mapmeanfn esp-ec-adapt/means \
   -mapvarfn esp-ec-adapt/variances \
   -mapmixwfn esp-ec-adapt/mixture_weights \
   -maptmatfn esp-ec-adapt/transition_matrices

./mk_s2sendump \
   -pocketsphinx yes \
   -moddeffn esp-ec-adapt/mdef.txt \
   -mixwfn esp-ec-adapt/mixture_weights \
   -sendumpfn esp-ec-adapt/sendump
```

3.8.1.4 Creación de la librería "LibSpeechServer"

El proceso de adaptación del modelo acústico generó nuevos archivos los cuales se encuentran en la carpeta: “esp-ec-adapt”. Esta carpeta junto con los archivos: “es-20k.lm” y “es.dict” deben agregarse al servidor “Speech2TextServer”, ya que son necesarios para la conversión de voz a texto. La manera más sencilla de agregar estos archivos a cualquier proyecto de Java es compilarlos como una librería con la extensión “.jar”. para ello se crea un nuevo proyecto en NetBeans del tipo “Java Class Library” y de nombre: “LibSpeechServer”, como se muestra a continuación:

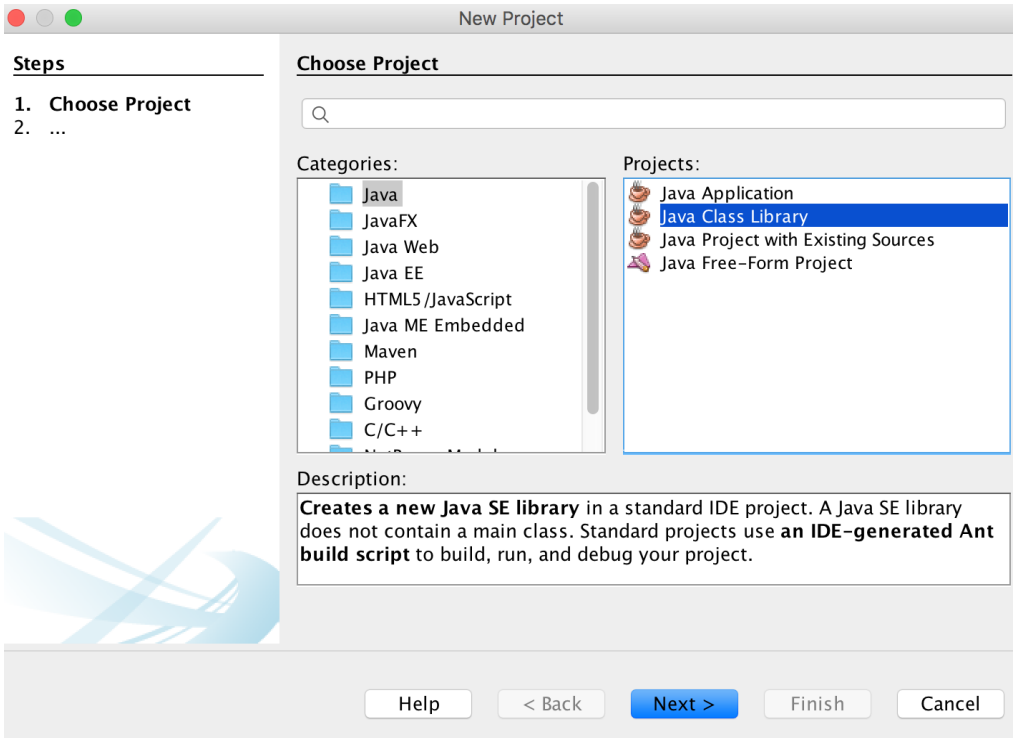


Figura 44. Creación de un nuevo proyecto de Java en NetBeans.

Luego se crea un paquete principal con el nombre: “modelserver”. En este paquete se copian los tres archivos y se compila. A continuación, se muestra el árbol de archivos del proyecto:

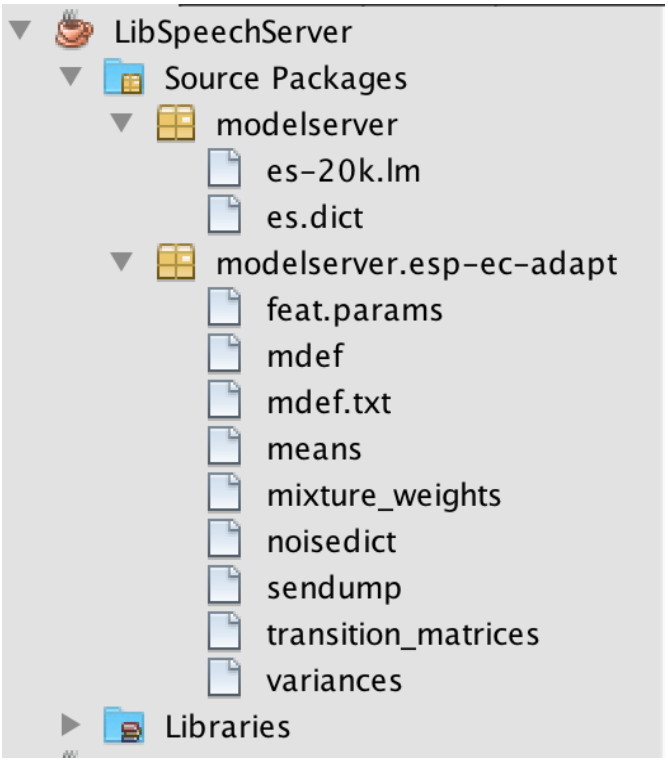


Figura 45. Librería "LibSpeechServer" y sus archivos.

Una vez compilada la librería, se genera un archivo de nombre: “LibSpeechServer.jar”, este archivo se encuentra en la carpeta principal del proyecto, para lo cual se accede al directorio. Ahí se puede encontrar a la carpeta de nombre “dist”. Se accede a esta y dentro de ella se encuentra el archivo como se muestra a continuación:

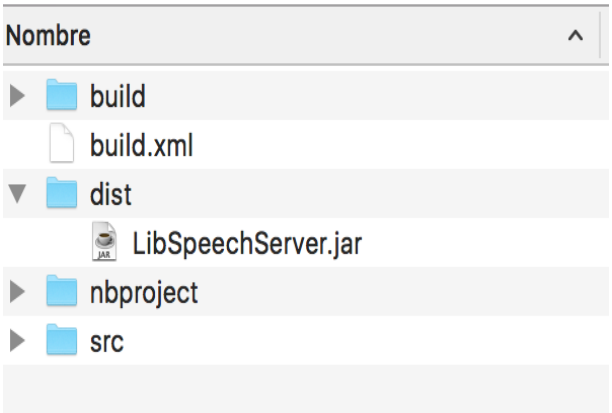


Figura 46. Librería "LibSpeechServer" compilada en un archivo de Java.

3.8.1.5 Programación de la clase “Speech2TextServer”

Este servidor se encargará de convertir voz a texto usando el API de CMU Sphinx y la librería que fue creada anteriormente: “LibSpeechServer.jar” la cual contiene el modelo acústico en español adaptado. El envío y recepción de datos se lo hará usando protocolo TCP/IP. Para el presente trabajo se ha considerado la creación de un solo hilo de ejecución para la conversión de texto a audio. Sin embargo, se ha considerado a futuro mejorar este inconveniente creando más hilos simultáneamente. El servidor no tendrá problemas de concurrencia, a pesar de trabajar con un solo hilo de ejecución ya que el servidor web se encarga de brindar el servicio de conversión de voz a texto y por ende de gestionar las peticiones de los clientes.

En primer lugar, se deben agregar dos librerías esenciales, la primera corresponde al API de CMU Sphinx. Esta librería se llama "sphinx4-core-5prealpha-20160628.232526-10.jar". Para obtener esta librería se debe acceder a la plataforma web: "Nexus Repository Manager". Se debe buscar a la librería con el nombre: "sphinx4-core" y descargarla.
En segundo lugar, se debe agregar la librería creada anteriormente: "LibSpeechServer.jar". El proyecto queda de la siguiente manera:

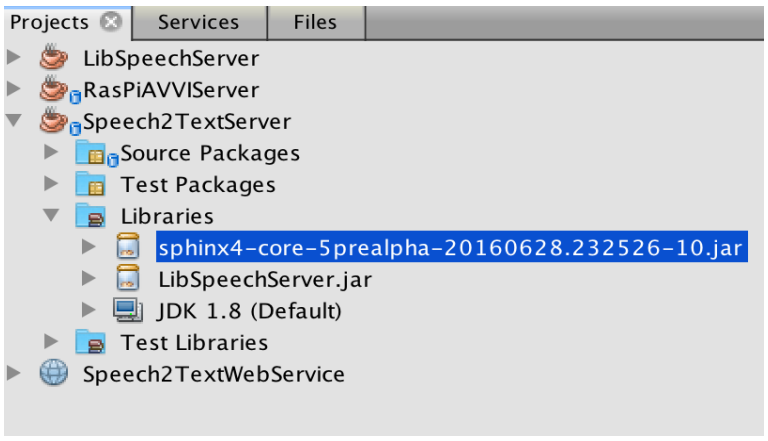


Figura 47. API de CMU Sphinx agregado como librería en el proyecto.

3.8.1.6 Librerías propias

A continuación, se detalla brevemente a las librerías creadas:

- **Librería "SpeechEngine":** Esta clase contiene los métodos necesarios para configurar, inicializar, y usar al convertidor de voz a texto de CMU Sphinx. Además, contiene las herramientas necesarias para manipular la información de audio que será ingresada al servidor (ver Anexo 6).
- **Librería "TCPIPTools":** Esta clase contiene los métodos necesarios para recibir y transmitir información desde el servidor hacia los clientes usando protocolo TCP/IP (ver Anexo 7).

3.8.1.7 Configuración de motor de conversión

Para el uso del convertidor de voz a texto con el API de CMU Sphinx, es totalmente necesario crear un archivo de configuración con el nombre: "config.xml" con las recomendaciones que brindan los desarrolladores de CMU Sphinx. Este archivo formato XML contiene un conjunto de componentes los cuales están relacionados al funcionamiento del algoritmo de conversión de voz a texto. El archivo debe estar ubicado en el paquete principal del proyecto. De manera que queda así:

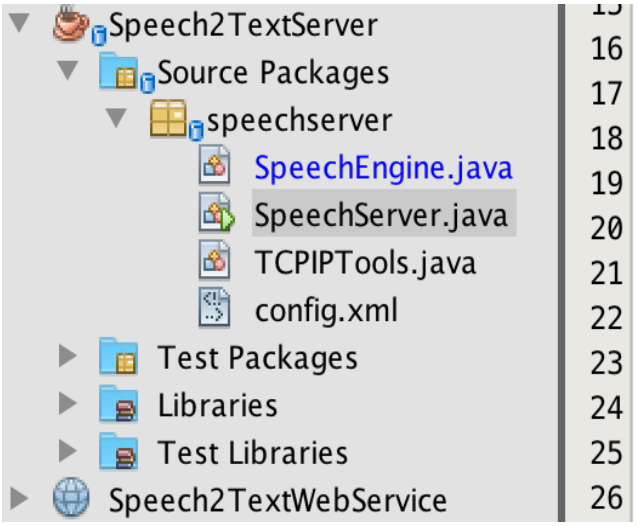


Figura 48. Proyecto "Speech2TextServer" y sus archivos.

Este archivo contiene muchos parámetros que tienen influencia directa sobre el funcionamiento del convertidor. Sin embargo, hay algunos parámetros que se deben configurar cuidadosamente que se explican a continuación:

En un principio se dijo que el API de CMU Sphinx necesita de un modelo acústico de un idioma específico para funcionar correctamente. Este modelo acústico ya se agregó al proyecto en la librería: "LibSpeechServer". Sin embargo, se debe indicar la ubicación de la librería en el archivo de configuración. Para ello se debe abrir el archivo y buscar el componente llamado: "dictionary". En este se debe agregar la dirección del diccionario en la propiedad "dictionaryPath". Se debe agregar la dirección del archivo: "noisedict" que se encuentra en la carpeta del modelo acústico, de tal forma que el componente queda de la siguiente manera:

```
<component name="dictionary"
  type="edu.cmu.sphinx.linguist.dictionary.TextDictionary">
  <property name="dictionaryPath"
            value="resource:/modelserver/es.dict"/>
  <property name="fillerPath"
            value="resource:/modelserver/esp-ec-adapt/noisedict"/>
  <property name="unitManager" value="unitManager"/>
</component>
```

Figura 49. Fragmento de código correspondiente al diccionario.

Además, se debe agregar la dirección del modelo de lenguaje, que corresponde al archivo "es-20k.lm", en el componente llamado: "simpleNGramModel", de tal manera que el componente queda de la siguiente manera:

```
<component name="simpleNGramModel"
  type="edu.cmu.sphinx.linguist.language.ngram.SimpleNGramModel">
  <property name="location"
            value="resource:/modelserver/es-20k.lm"/>
  <property name="dictionary" value="dictionary"/>
  <property name="maxDepth" value="3"/>
  <property name="unigramWeight" value=".7"/>
</component>
```

Figura 50. Fragmento de código correspondiente a la gramática.

Por último, se debe agregar la dirección de la carpeta del modelo acústico en el componente llamado: "acousticModelLoader". De manera que queda de la siguiente manera:

```
<component name="acousticModelLoader"
  type="edu.cmu.sphinx.linguist.acoustic.tiedstate.Sphinx3Loader">
  <property name="unitManager" value="unitManager"/>
  <property name="location"
            value="resource:/modelserver/esp-ec-adapt"/>
  <property name="topGaussiansNum" value="4"/>
</component>
```

Figura 51. Fragmento de código correspondiente al modelo acústico.

3.8.1.8 Desarrollo de la clase principal

Una vez que se ha configurado todo correctamente, se procedió a escribir la clase principal del servidor. Esta clase es muy parecida a la clase principal del servidor de domótica en el Raspberry Pi. Ya que ambas clases se basan en el envío y recepción de información por protocolo TCP/IP, además que trabajan con la creación de hilos de ejecución para los pedidos de cada cliente.

Durante todo el ciclo de vida del servidor, se requiere que varios métodos hagan uso de variables que no necesiten ser inicializadas cada vez que se ejecuta el proceso. De manera que el rendimiento y eficiencia de procesamiento del

servidor sea máximo. Para ello se han definido como globales a las siguientes variables:

```
...
public Socket id;
public static SpeechEngine speechEngine;
public static TCPIPTools netTools;
public AudioFormat format = new AudioFormat(16000.0f, 16, 1,
true, false);
...
```

Inicialmente se tiene a la variable "id", la cual será usada para crear la conexión con el cliente. Luego se tiene a la variable "speechEngine" la cual será una instancia de la librería propia creada para procesar la información de audio recibida. Luego se tiene a la variable "netTools" la cual será una instancia de la librería propia necesaria para la comunicación con el cliente usando el protocolo TCP/IP. Por último, se tiene a la variable format, que es una instancia de la clase "AudioFormat". Esta variable será usada para dar el formato correcto de audio a la información enviada desde los clientes. El formato especificado define a un archivo de audio comprimido, con 16.000Hz de frecuencia de muestreo, 16 bits por muestra, monofónico y codificación PCM en little-endian.

3.8.1.9 Método "main" de la clase

Este método será el que se ejecute una vez el servidor sea activado. Inicialmente en este método se agrega información del servidor que será mostrado en la pantalla de la consola. El código se muestra a continuación:

```
System.out.println("***************                    Servidor
Speech2Text  ***************");
```

```
        System.out.println("***************   Reconocimiento  de
voz  ***************");
        System.out.println("***********************  UDLA  ****
********************");
        System.out.println("****************      ING     SONIDO
ACUSTICA  ****************");
        System.out.println("*******  Creadores:  Juan  Chango  -
Andrés Márquez *******");
        System.out.println("");
        System.out.println("Inicializando Sphinx...");
```

Luego se procede a crear una nueva instancia de la clase "SpeechEngine" y "TCPIPTools", e inmediatamente se inicializa al convertidor de voz a texto con la función "init()". Este proceso conlleva una cantidad considerable de tiempo puesto que se están cargando en memoria todos los archivos necesarios para la conversión de voz a texto. A continuación, se muestra el código:

```
...
speechEngine = new SpeechEngine();
netTools = new TCPIPTools();
speechEngine.init();
...
```

Una vez que se ha inicializado el convertidor, se crea un objeto del tipo "ServerSocket" y de esta manera comienza el servidor a escuchar las peticiones del cliente en el puerto: 7777. En este caso el único cliente del servidor será el servidor web, ya que los clientes conectados desde los dispositivos Android son gestionados desde el servidor mencionado. Una vez que un cliente se ha conectado, se crea un nuevo hilo de ejecución junto con una instancia del convertidor de voz a texto. Luego el servidor vuelve a escuchar en espera de un nuevo cliente. A continuación, se muestra el código:

```
...
ServerSocket ss = new ServerSocket(7777);

        while (true) {
            System.out.println("Escuchando en puerto:7777...");
            Socket s = ss.accept();
            System.out.println("Cliente encontrado");
            SpeechServer t = new SpeechServer(s);
            t.start();
        }
...
```

Cada cliente conectado al servidor entrega información de audio en un arreglo de bytes. Esto se lo hace usando los métodos de la librería propia: "TCPIPTools", como se muestra a continuación:

```
byte data[] = netTools.rxBytesFromClient(id);
```

Esta información debe ser convertida en un formato de audio conocido para el sistema, esto se lo hace usando la variable global "format" y un método de la librería propia: "SpeechEngine". A continuación, se muestra el código descrito:

```
InputStream stream =
speechEngine.formatWaveByteInputTCPIP(data, format);
```

En este punto ya se puede convertir la información de audio formateada en texto, para ello se utiliza el siguiente código:

```
String result = null;
```

```
result = speechEngine.speech2str(stream);
```

Por último, se envía de vuelta el resultado de la conversión al cliente usando el siguiente código:

```
netTools.txHypothesis(id, result);
```

Cuando se ejecuta al servidor, se obtiene una pantalla de bienvenida. A continuación, el servidor indica el momento en el que ya se han cargado todos los componentes necesarios y comienza a escuchar en el puerto: 7777. El mensaje en la consola es el siguiente:

```
run:
**********  Servidor Speech2Text  **********
**********  Reconocimiento de voz  **********
**********  UDLA  **********
**********  ING SONIDO ACUSTICA  **********
******* Creadores: Juan Chango – Andrés Márquez *******

Inicializando Sphinx...
Escuchando en puerto:7777...
```

Figura 52. Mensaje desplegado en la consola del sistema al ejecutar el servidor: "Speech2TextServer".

Una vez que se conecta un cliente, se mostrará un mensaje en la consola de alerta y después de un momento se mostrará el resultado de la conversión. Dentro de la librería "SpeechEngine"existe un método que permite grabar todas las consultas de los clientes, esto podría ser útil para realizar análisis estadísticos o adaptar nuevamente al sistema (para más información ver Anexo 6).

Por último, hay que recordar que este servidor se ejecutará en una máquina en la cual se ejecutará al mismo tiempo el servidor web. De hecho, el único cliente que se conectará al servidor "Speech2TextServer" será el servicio web, lo cual es beneficioso ya que así se evitan problemas de concurrencia. Ambos

servidores se conectan internamente a través del protocolo TCP/IP. Es una forma de trabajo conjunto muy similar a la que se desarrolla entre un servidor web y una base de datos. Al tener separados ambos servidores pueden trabajar independientemente en cada uno, en caso de mantenimiento o mejoras.

3.8.2 Programación del Servidor Web

El servidor Web tendrá la función de administrar y gestionar el servicio de conversión de voz a texto para los clientes.

El servidor web pondrá a disposición de los clientes Android un servicio web SOAP y será descrito mediante un fichero XML (ver el Anexo 14). Una vez que un dispositivo Android comience a consumir el servicio, enviará información de audio codificada hacia el servidor. El servidor web recibe esta información y la decodifica. Luego envía hacia el servidor "Speech2TextServer", este servidor convierte el audio en texto y envía el resultado al servidor web. El servidor web envía nuevamente hacia el dispositivo Android bajo el protocolo SOAP.

El servidor web fue programado usando Java EE y Glassfish. Glassfish es un servidor de aplicaciones de software libre el cual permite ejecutar servicios web basados en Java EE y además gracias a herramientas incluidas en el IDE NetBeans, se puede programar y ejecutar cualquier aplicación y el servidor desde el IDE en una misma máquina Linux, Pc o Mac. NetBeans incluye un conjunto de herramientas para poder crear fácilmente servicios web SOAP, el procedimiento es bastante intuitivo y una vez creado el servicio lo único que queda es usar las librerías propias.

3.8.2.1 Librerías propias

Para el desarrollo del servidor web se creó la siguiente librería:

- **ClientSpeechEngine:** Esta clase permite el intercambio de datos entre el servicio web y el servidor de conversión de voz a texto. Permite enviar y recibir datos en forma correcta y en un formato compatible a través del protocolo TCP/IP (más información ver Anexo 8).

3.8.2.2 Glassfish server

Lo primero que debe hacerse es crear el servidor Glassfish, para ello se debe tener instalado la última versión del IDE NetBeans junto con los componentes necesarios para usar Java EE. Primeramente, se accede a "Services" y se selecciona "Add Server":

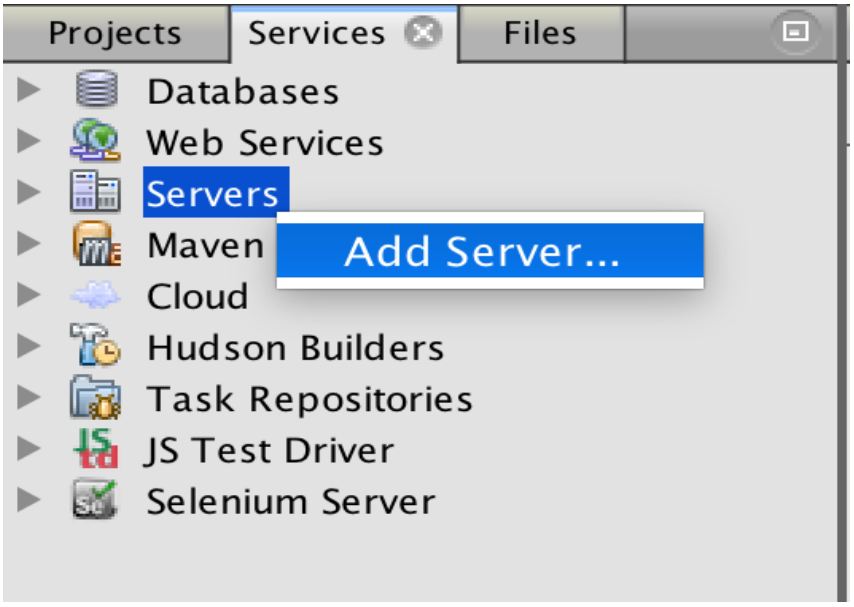


Figura 53. Agregando un nuevo servidor en NeatBeans.

A continuación, se muestra una ventana con las diferentes opciones de servidores y se debe seleccionar "GlassFish Server":

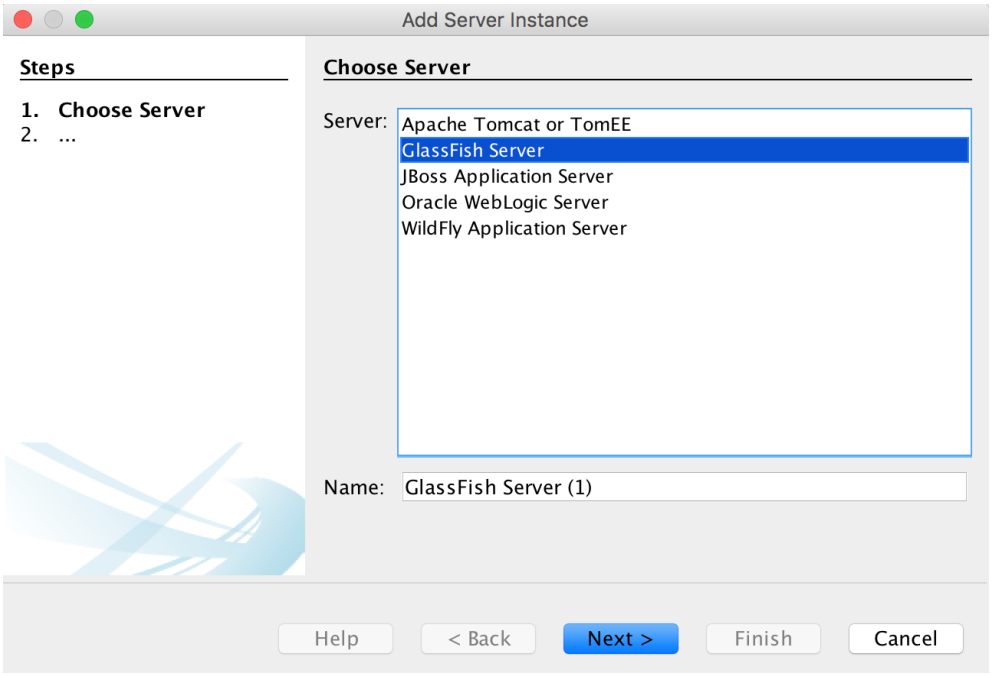


Figura 54. Ventana de selección de diferentes servidores.

Se selecciona la ubicación del servidor, para este trabajo será una ubicación local, sin embargo, es posible desplegar aplicaciones remotamente:

Figura 55. Especificando la ubicación del Servidor Glassfish.

Por último, se agrega información adicional sobre el servidor, como el dominio, la dirección, el puerto, etc. Como se muestra a continuación:

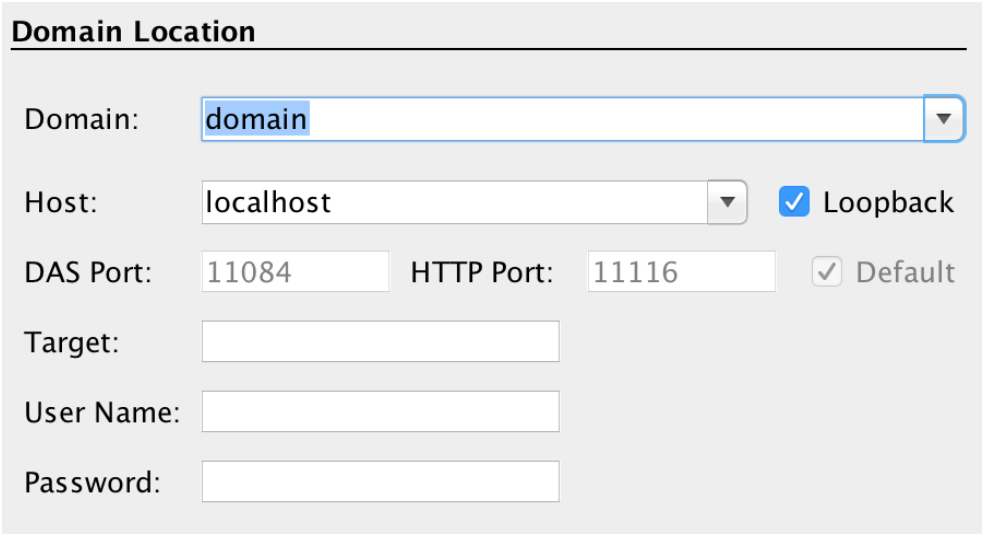


Figura 56. Configurando dirección IP del servidor Glassfish.

Finalmente, ya se tiene al servidor creado y se lo puede iniciar:

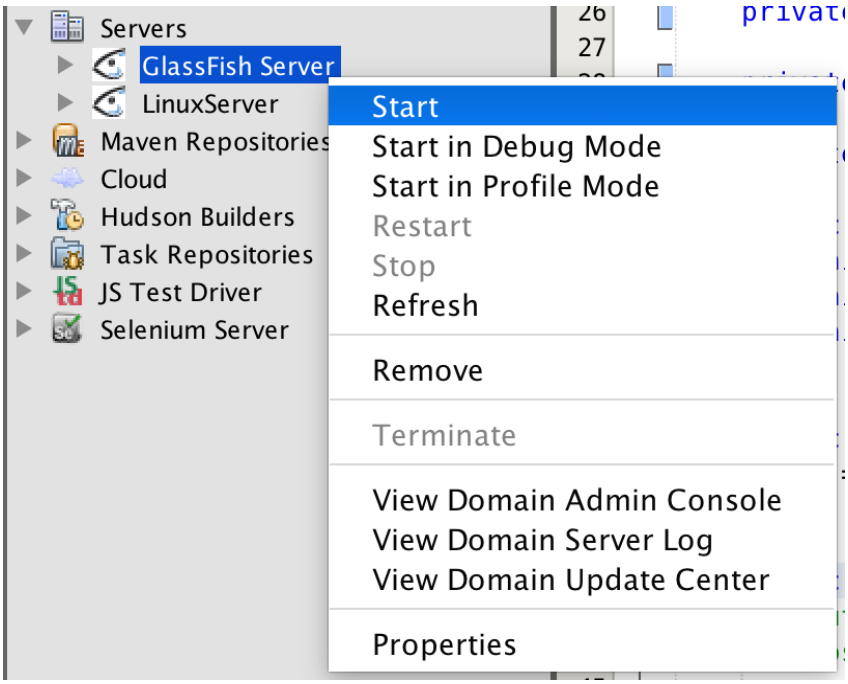


Figura 57. Inicializando al servidor Glassfish.

3.8.2.3 Servicio web SOAP

Una vez creado el servidor Glassfish, se procede a crear al servicio web sobre una aplicación web de Java EE. Para ello se crea un nuevo proyecto y se selecciona "Web Application":

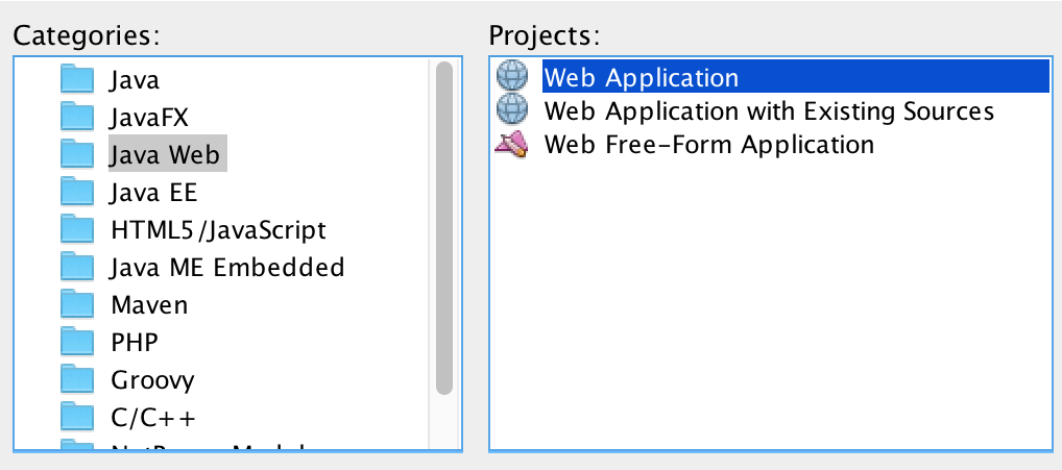


Figura 58. Creación de un nuevo proyecto web en Java.

Una vez creado el proyecto, dentro de este se crea un nuevo servicio web de la siguiente manera:

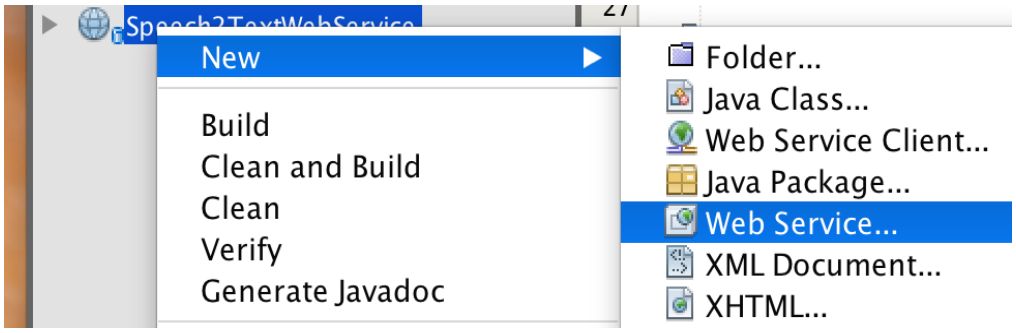


Figura 59. Creación de un nuevo servicio web en Java.

Se le pone un nombre, se le asigna un paquete en el proyecto y se finaliza. Como se muestra a continuación:

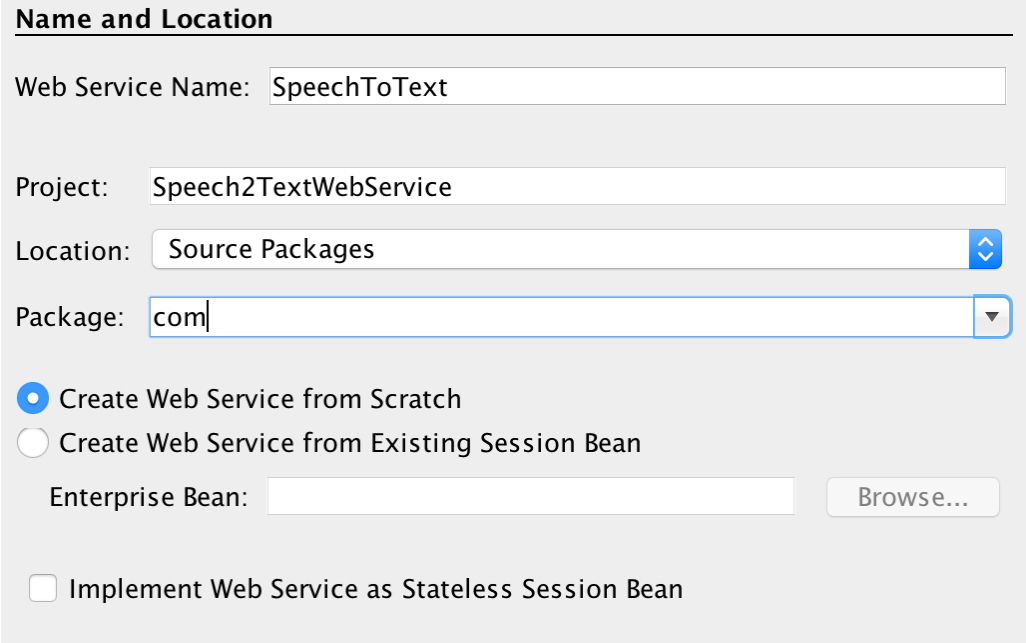


Figura 60. Configuración del servicio web.

Dentro del proyecto se puede ver que se ha creado el servicio:

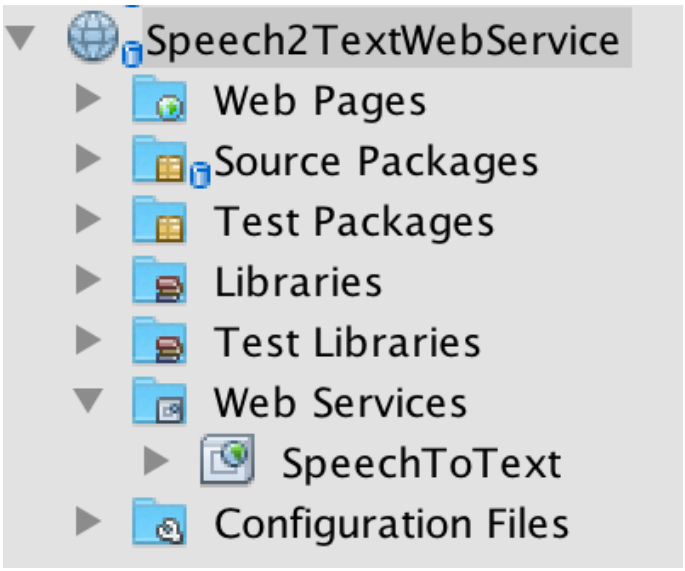


Figura 61. Carpetas del proyecto junto con el nuevo servicio web.

Antes de iniciar la programación del servicio web se debe agregar la librería propia: "ClientSpeechEngine", en el paquete "com" del proyecto:

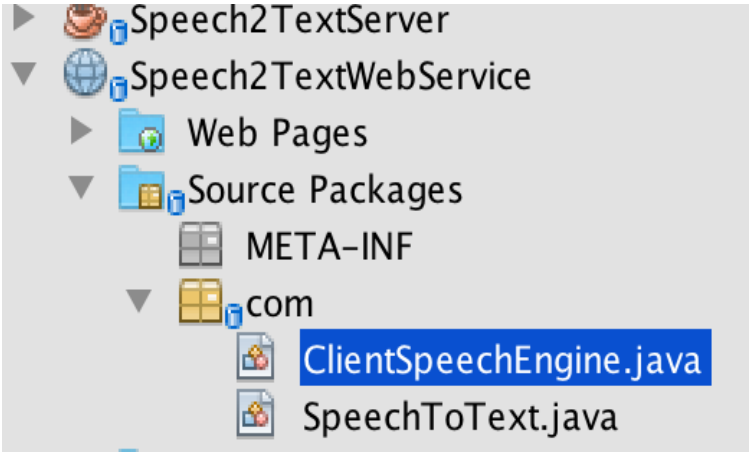


Figura 62. Agregando librerias al servidor web.

Se abre el servicio creado y se agrega una operación de nombre "speech2text", esta operación del servicio web tendrá de parámetros de entrada y de salida una variable del tipo "String", como se muestra a continuación:

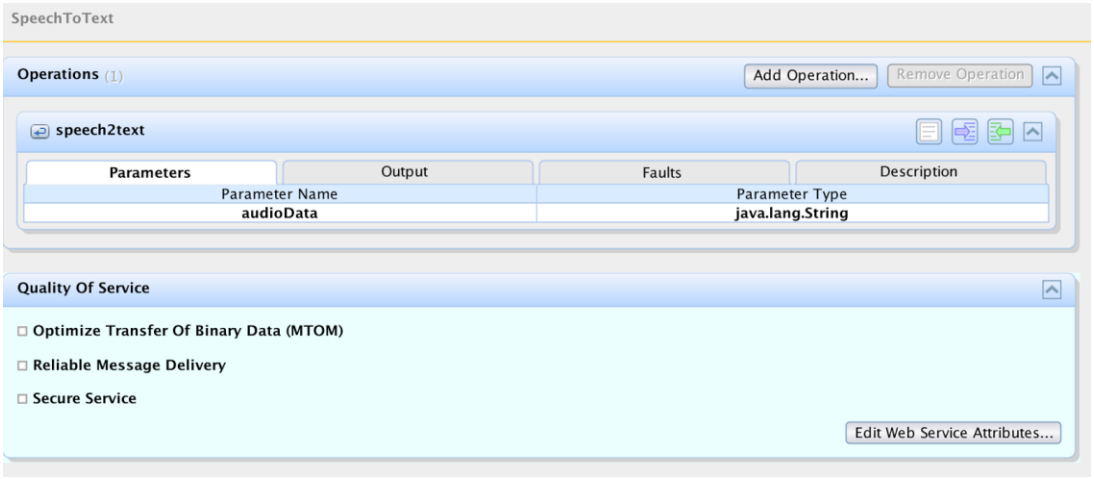


Figura 63. Configuración de las operaciones del servicio web.

La operación "speech2text", es la encargada de recibir las peticiones de los clientes Android y enviar la información de audio recibida hacia el servidor "Speech2TextServer". De igual manera esta operación se encarga de recibir el resultado de la conversión y de enviar este resultado hacia el cliente Android. Se debe notar que los datos de entrada y de salida que llegan a la operación del servicio web es del tipo "String", es decir es texto. Por ello anteriormente se ha

mencionado que para lograr esto la información de audio es codificada en Base64 en el cliente Android y decodificada en el servidor web.

3.8.2.4 Conexión con servidor "Speech2TextServer"

El servicio web ya contiene la operación la cual sirve básicamente para transferir la información entre el cliente y el servidor: "Speech2TextServer". Para ello hace uso de la librería: "ClientSpeechEngine", y además utiliza la librería "java.util.Base64" para decodificar la información de audio. A continuación, se describe el funcionamiento de la operación de servicio web (para ver todo el código, ver el Anexo 12).

El servicio web está representado por una clase de java con el mismo nombre: "SpeechToText.java". Cuando el cliente empieza a consumir el servicio web, esta llama a un método web el cual contiene la operación que el cliente está usando. El único parámetro de entrada del servicio web es el audio que se almacena en una variable de nombre: "audioDataString64". Este contiene audio codificado en base64, esta información se decodifica y se almacena en una nueva variable del tipo "byte[]", como se muestra a continuación:

```
...
byte[] audioData =
Base64.getDecoder().decode(audioDataString64);
...
```

Así ya es posible enviar el audio almacenado al servidor para la conversión de voz a texto, para ello se emplea el siguiente código:

```
...
ClientSpeechEngine clientSpeechEngine = new
ClientSpeechEngine("MAC", "localhost", 7777);
clientSpeechEngine.init();
clientSpeechEngine.txBytesToServer(audioData);
...
```

Por último, se recibe la respuesta del servidor y se envía hacia el cliente al retornar como un valor del método web, como se muestra a continuación:

```
...
String res = clientSpeechEngine.rxHypothesis();
System.out.println(res);
return res;
...
```

Como se puede ver, la implementación del servicio web es relativamente sencilla, sin embargo, su aporte al sistema es enorme si se consideran las ventajas de un protocolo estandarizado como es el SOAP.

4. CAPÍTULO IV. ANÁLISIS DE RESULTADOS

4.1 Pruebas

Inicialización del sistema:

Inicialmente, se activa el programa en el Raspberry Pi, que corresponde al servidor de domótica. El Raspberry lanza el siguiente mensaje:

```
profile-rp-calibrate-passwd:
Connecting to 192.168.0.59:22
cmd : cd '/home/pi/NetBeansProjects/RasPiAVVIServer'; '
**********  Servidor RasPi AVVI  **********
**********  Reconocimiento de voz  **********
**********  UDLA  **********
**********  ING SONIDO ACUSTICA  **********
******* Creadores: Juan Chango - Andrés Márquez *******
Inicializando Servidor Raspberry Pi...
Escuchando en puerto:7778...
```

Figura 64. Mensaje de bienvenida del servidor de domótica.

Luego se inicializa al servidor web en el servidor de aplicaciones Glassfish y se muestra el siguiente mensaje:

```
Información:   visiting unvisited references
Información:   visiting unvisited references
Información:   visiting unvisited references
Información:   Webservice Endpoint deployed SpeechToText
 listening at address at http://MacBook-Pro-de-Juan.local:8080/Speech2TextWebService/SpeechToText.
Información:   Loading application [Speech2TextWebService] at [/Speech2TextWebService]
Información:   Speech2TextWebService was successfully deployed in 736 milliseconds.
```

Figura 65. Mensaje de bienvenida del servidor web.

Por último, para inicializar las pruebas se ejecuta el servidor "Speech2TextServer". El cual lanza el siguiente mensaje de bienvenida:

```
run:
**********************  Servidor Speech2Text  **********************
**********************  Reconocimiento de voz  **********************
**************************  UDLA  **************************
**********************  ING SONIDO ACUSTICA  **********************
******* Creadores: Juan Chango – Andrés Márquez *******

Inicializando Sphinx...
Escuchando en puerto:7777...
```

Figura 66. Mensaje de bienvenida del servidor: "Speech2TextServer".

4.2 Pruebas en la aplicación

Se inicia la aplicación instalada en el dispositivo Android, la aplicación se muestra de la siguiente manera:

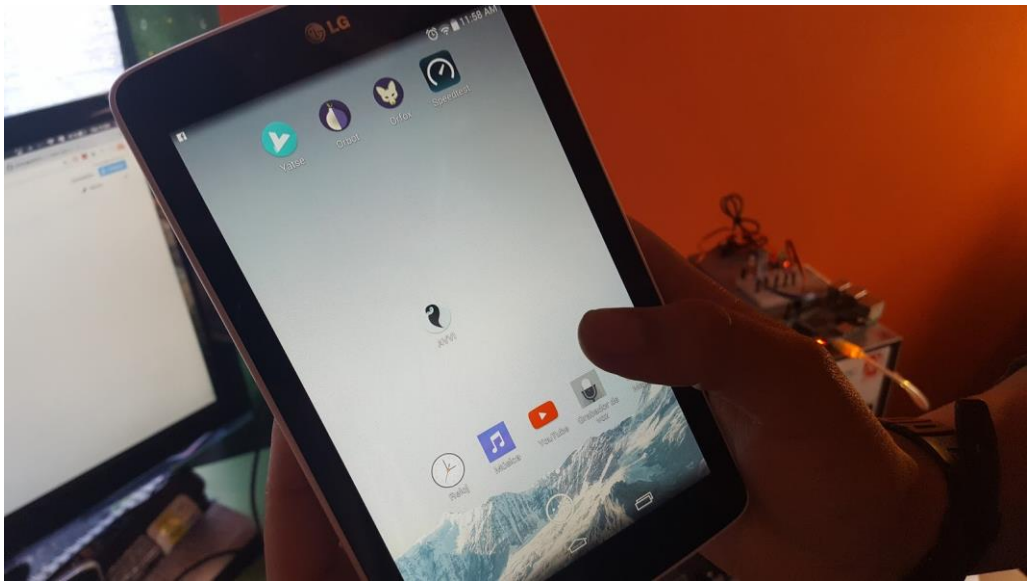


Figura 67. Pantalla principal de: dispositivo Android y logo de la app. AVVI.

Una vez abierta se muestra la pantalla de bienvenida, la cual se muestra a continuación:

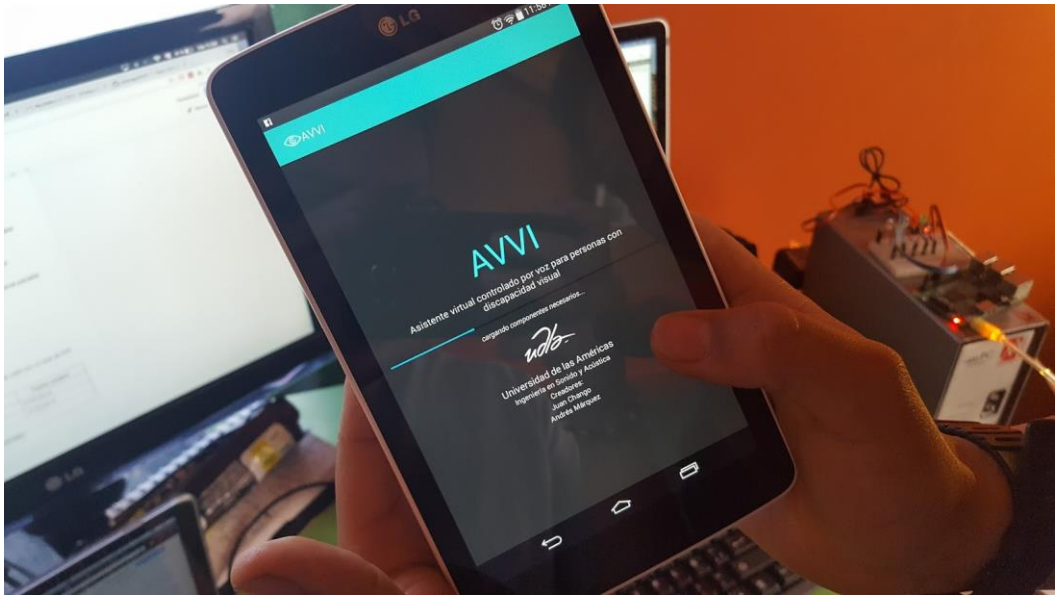


Figura 68. Pantalla de bienvenida de la aplicación AVVI.

Cuando la aplicación se ha iniciado correctamente se muestra la siguiente pantalla:

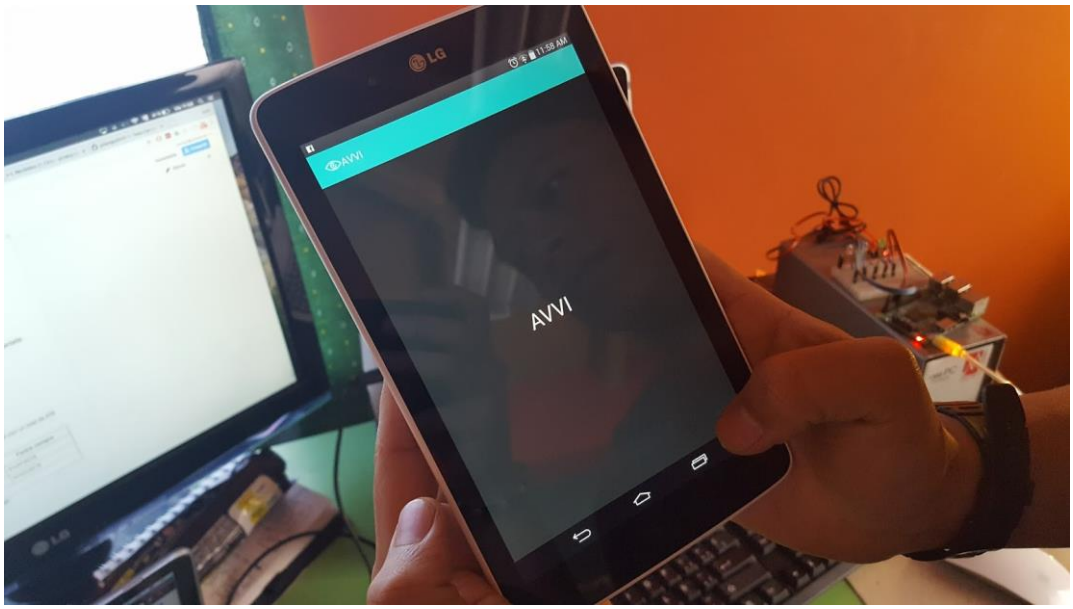


Figura 69. Pantalla principal de la aplicación AVVI.

Una vez que se ha tocado la pantalla, la aplicación inicia el procesamiento tal como se muestra a continuación:

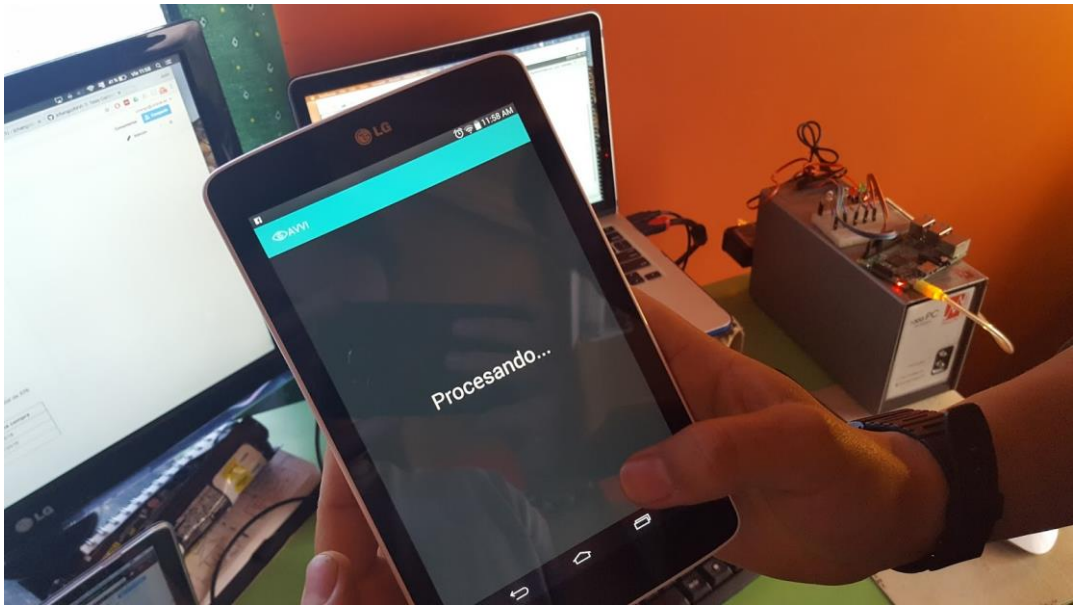


Figura 70. Aplicación AVVI procesando información.

Una vez que el comando es reconocido, este se muestra en la pantalla y lo reproduce para que el cliente conozca si se activó o no:

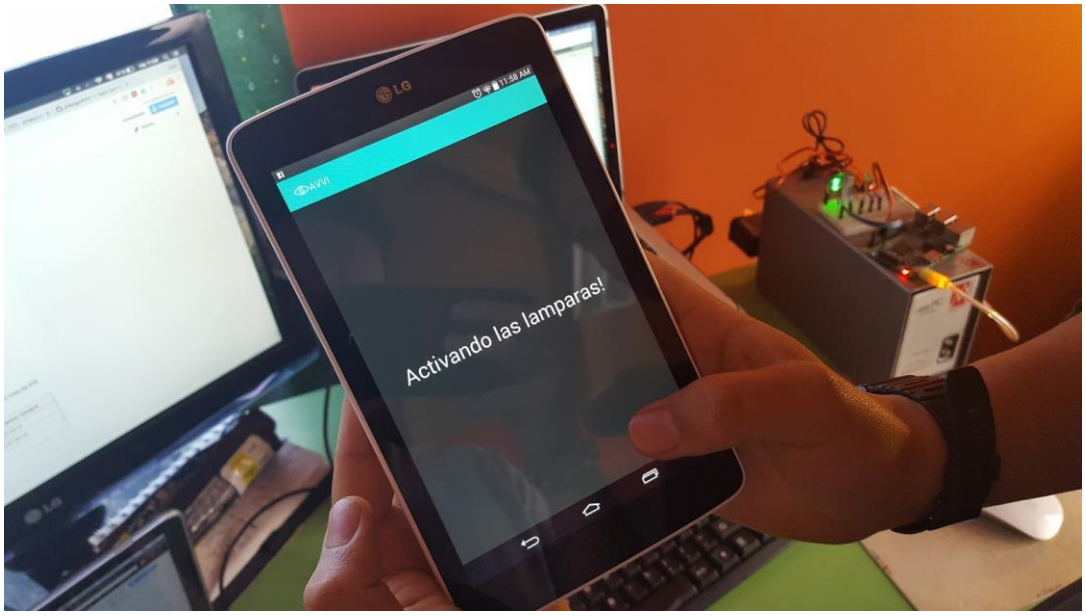


Figura 71. Comando mostrado en la pantalla principal de la aplicación AVVI.

El Raspberry se encuentra conectado inalámbricamente a la red local y recibe datos desde el dispositivo Android. En el Raspberry Pi se usaron focos led, los

cuales simulan los diferentes actuadores. Cuando el Raspberry Pi recibe el comando, enciende el foco como se muestra a continuación:

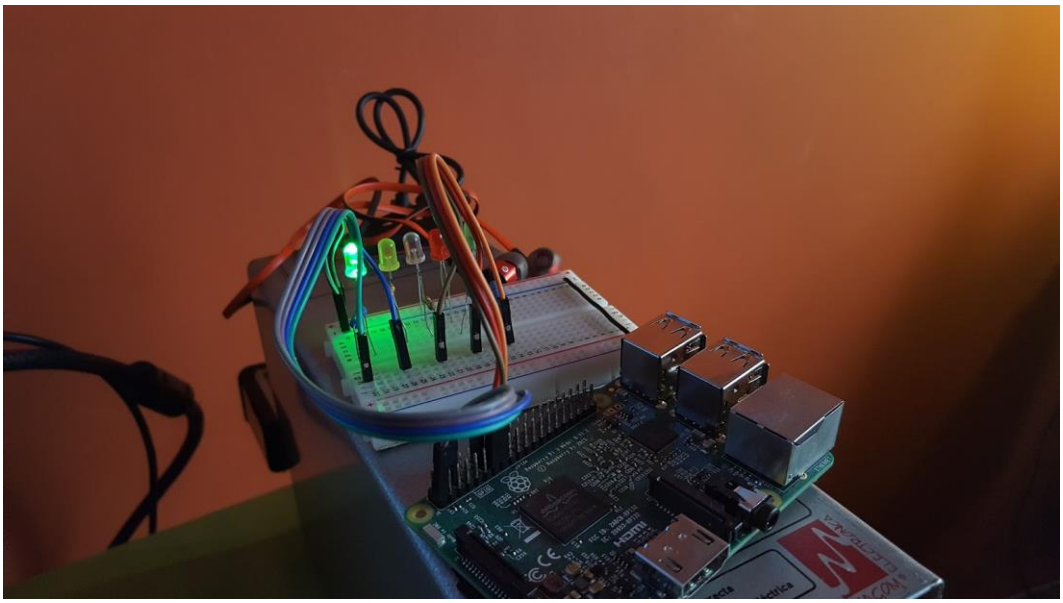


Figura 72. Raspberry Pi simulando un dispositivo de domótica.

Si el comando no se reconoce la aplicación muestra un mensaje de texto y reproduce un mensaje hablado de error, como se muestra a continuación:

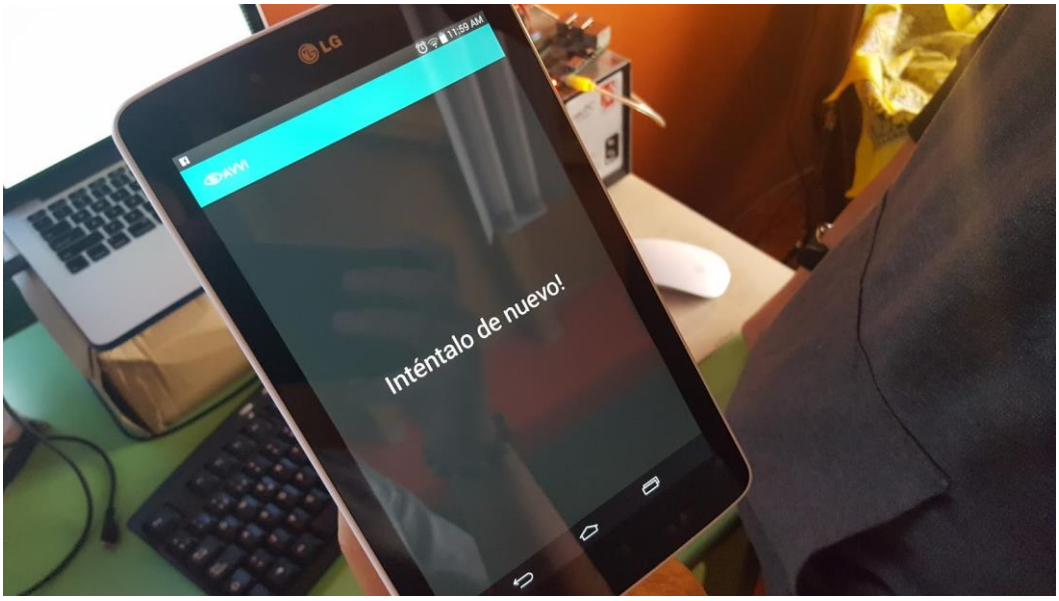


Figura 73. Mensaje de error mostrado en la pantalla principal de la app. AVVI.

4.3 Ventajas para personas con discapacidad visual

Dentro de las ventajas para las personas con discapacidad visual se tiene principalmente el confort, ya que evita el desplazamiento de la persona dentro de la vivienda para la realización de tareas cotidianas como encendido de luces, o apertura remota de puertas, a través de comandos de voz.

El procesamiento del algoritmo de reconocimiento se realiza remotamente, por lo cual la persona con discapacidad visual no debe disponer de un equipo con grandes capacidades de procesamiento, esto representa un ahorro de dinero sin comprometer el funcionamiento del convertidor de voz a texto.

El servicio de reconocimiento de voz es de uso libre, lo que implica que el usuario no debe cancelar ningún valor económico por este servicio.

El costo del hardware para el cliente es relativamente accesible ya que es controlado a través de un teléfono inteligente y lo único que debe adquirir el usuario es un ordenador de placa reducida Raspberry Pi.

4.4 Repositorio en GitHub

El proyecto finalizado se encuentra ubicado en un repositorio de GitHub. Este repositorio lleva por nombre: "AVVI" y pertenece al usuario: "marquezchango". Este repositorio contiene todos los archivos necesarios para el funcionamiento de todo el sistema. Por el momento solo se incluirá información con un link de acceso a la tesis, en la cual se puede conocer el funcionamiento detallado de cada parte. A continuación se muestra el repositorio creado:

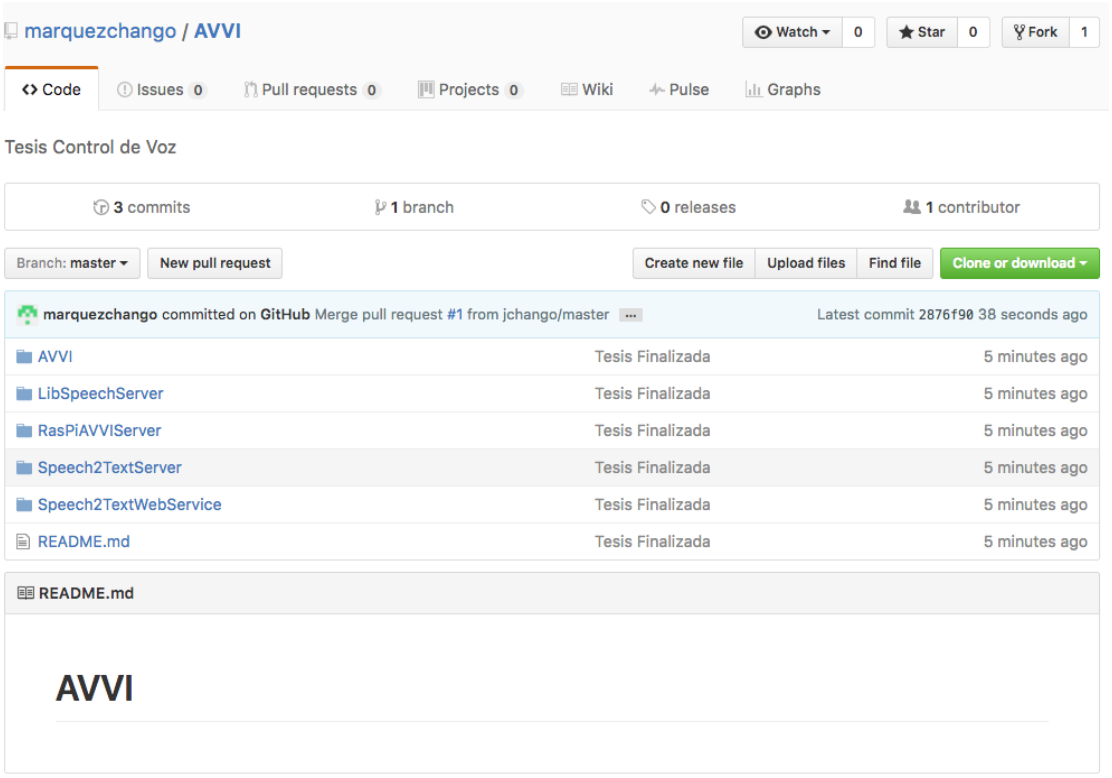


Figura 74. Captura de pantalla del repositorio del proyecto.

5. CAPÍTULO V. ANÁLISIS ECONÓMICO

5.1 Inversión

El proyecto tuvo una inversión de alrededor de: 1.110,00 USD con un total de 576 horas de trabajo.

Tabla1.

Tabla de materiales.

Producto	Valor total	Fecha compra
Raspberry Pi	$45,00	01/07/2016
Cables y otros materiales	$ 5,00	01/07/2016
Total	$50,00	

Tabla 2.

Tabla de Gastos.

Gasto	Valor
Programación	$350,00
Investigación	$500,00
Internet	$90,00
Transporte	$30,00
Electricidad	$90,00
Total	$1.060,00

Tabla 3.

Horas de trabajo

Actividad	Horas
Investigación	48
Diseño e Implementación	336
Montaje y Evaluación	48
Redacción	144
Total	576

5.2 Estimación del costo del sistema para el cliente

El sistema será de libre acceso y el código estará disponible para todo el público, por lo cual no existe un costo por la programación del código. La aplicación se espera que esté disponible en la plataforma "Google Play" y no tenga costo, es decir que cualquier persona con un dispositivo Android pueda usarla. El cliente, es decir la persona con discapacidad visual, únicamente tiene que adquirir un Raspberry Pi por el costo de venta que en el Ecuador se puede conseguir por aproximadamente $80 USD.

El mayor costo del sistema radica en la implementación del servidor de conversión de voz a texto. No se tiene un valor aproximado de esta implementación, ya que no fue necesaria para la implementación de este proyecto porque una computadora puede simular un servidor. Sin embargo, se espera que este servidor pueda ser costeado por una institución gubernamental la cual vele por los derechos de las personas discapacitadas.

6. CAPÍTULO VI. PROYECCIONES

A futuro, se espera que, con la ayuda de la comunidad de programadores y estudiantes de carreras relacionadas, el proyecto logre lo siguiente:

- Realizar más adaptaciones del modelo acústico, mediante el procesamiento de varios archivos de audio con diferentes dialectos ecuatorianos.
- Entrenar un modelo acústico de licencia libre para el Ecuador.
- Mejorar los protocolos de comunicación, y en el mejor de los casos implementar protocolos estandarizados.
- Crear un hardware especializado y de fácil uso para el control de los elementos de domótica.
- Probar el dispositivo con varios usuarios con discapacidad visual y evaluar su facilidad de uso.
- Implementar el servidor en una máquina especializada para correr servidores web.

7. CONCLUSIONES Y RECOMENDACIONES

7.1 Conclusiones

Para concluir, uno de los principales objetivos de este proyecto fue alcanzado con éxito. La reducción de costos para el usuario pudo ser posible gracias a la utilización de los ordenadores de placa reducida Raspberry Pi, estos ordenadores se caracterizan por su bajo costo y es la parte fundamental del lado del cliente para la implementación del sistema de domótica además del servicio de reconocimiento de voz que en este caso será gratuito.

Se concluye que la creación del servicio web, permite que cualquier persona con discapacidad visual pueda usarlo. Además, abarata el costo que representa la implementación del sistema en la casa del cliente. Esto sin reducir la velocidad de respuesta y rendimiento del convertidor de voz a texto. Además, permite que se puedan realizar mejoras, adaptar mejor al modelo acústico, integrar aprendizaje automático, entre otras más mejoras en el lado del servidor, todo esto sin afectar a los clientes.

La adaptación del modelo acústico del API de CMU Sphinx mejora significativamente el rendimiento y la precisión del sistema. Se pudo comprobar que una vez entrenado el sistema, el reconocimiento de los fonemas se adapta de manera general para todo el vocabulario, sin embargo, en palabras que no fueron entrenadas se pierde rendimiento, incluso con palabras entrenadas, pero dentro de distintas frases, el rendimiento disminuye.

No existen muchos servicios web libres destinados al procesamiento de audio, sin embargo, esto es una gran ventaja del proyecto realizado, ya que la comunidad de desarrolladores puede consumir el servicio web únicamente conociendo el descriptor WSDL del servicio. Esto sin importar el lenguaje de programación ni la plataforma. Cabe destacar que el código fuente de este trabajo está disponible para cualquier persona a través de la plataforma de desarrollo colaborativo GitHub.

Se concluye que el proceso de adaptación del modelo acústico del API Sphinx mejora sustancialmente el desempeño de la conversión de audio a texto, sin embargo, cuando no se tenga un resultado adecuado, podrá hacer uso de aprendizaje automático el cual adapta los resultados de la conversión a comandos conocidos.

Cabe destacar que para enviar audio a través del protocolo SOAP este debe ser convertido de un arreglo bytes a una cadena de caracteres codificada en base64. Java y Android poseen las librerías necesarias para la codificación y decodificación lo cual facilita enormemente el trabajo de programación. También se puede enviar audio como un arreglo de bytes sin embargo se debe desarrollar un fragmento de código encargado de codificar audio en una variable primitiva de la librería “ksoap”, lo cual complica el trabajo de programación, ya que estos tipos primitivos no son compatibles en la librería estándar de Java.

Es muy importante considerar el tipo de audio que se necesita para que el sistema trabaje tanto del lado del servidor como del cliente, ya que ambos deben ser los mismos. El API de reconocimiento de voz Sphinx CMU, solo acepta audio en el formato de 16.000 u 8.000 Hz en frecuencia de muestreo, codificado en PCM en big-endian o little endian y con un número de bits por muestra de 16. En caso de ser configurado de manera incorrecta cualquier parámetro referente al formato de audio el sistema fallará. Especialmente se debe tomar en cuenta al formato de almacenamiento que puede ser little-endian ó big-endian.

El envío de datos hacia el servidor "Speech2Text" podría ser únicamente usando protocolo TCP/IP sin ningún protocolo de transporte, sin embargo, el servidor tendría vulnerabilidad de seguridad frente a ataques informáticos y si en un futuro otros desarrolladores desearían usar el servidor, estos deberían tener una librería única para el servidor. Por esta razón es importante el uso de un servicio web SOAP, ya que permite que cualquier desarrollador use el servicio sin la necesidad de conocer su código fuente sino únicamente su descriptor WSDL,

además que un servicio web permite agregar controles de seguridad sin tener que escribir código extra en el servidor.

Se debe señalar que si muchos clientes consumieran el servicio web se requiere, en teoría, que existan tantos hilos de ejecución del servidor como clientes. Sin embargo, el servidor Sphinx consume una considerable cantidad de procesamiento y memoria del ordenador, de manera que crear nuevos hilos de ejecución del servidor requiere de un ordenador con muy buenas características de rendimiento. No resulta práctico crear muchos hilos de ejecución simultáneos. Para solucionar este inconveniente, Java permite gestionar cada hilo de acuerdo al trabajo que se encuentra realizando, de tal manera que se puede optimizar el rendimiento con una cantidad reducida de hilos para todos los clientes.

Cabe destacar que la domótica no es simplemente una tecnología diseñada para el aumentar el confort de las personas como usualmente se piensa, además del confort el sistema contribuye en otros aspectos como el ahorro del consumo de energía eléctrica ya que puede brindar automatización a través de sensores a sistemas de iluminación, riego y calentamiento de agua. Por otro lado, en temas de seguridad brinda también un gran aporte al controlar el acceso de las viviendas, pudiendo implementarse monitoreo de cámaras de vigilancia, además de alarmas de intrusión, fuego o fugas de gas.

Los problemas de concurrencia en el servidor “Speech2Text” pueden solucionarse con la creación de varios hilos de ejecución, sin embargo, esto hace que sea necesario una maquina con gran capacidad de procesamiento para manejar todos los pedidos de los clientes.
El entrenamiento de un modelo acústico conlleva una inversión de tiempo muy grande, puesto que se debe crear un diccionario y cada palabra debe ser entrenada en varias fases. Si se dispondría del tiempo necesario y los recursos, se podría generar un modelo acústico enfocado al dialecto ecuatoriano, sin embargo, adaptar un modelo acústico existente resulta más práctico para el trabajo desarrollado.

Pese al desarrollo tecnológico existente actualmente, se puede deducir que un sistema de reconocimiento de voz difícilmente puede llegar a ser 100% preciso debido a que estos sistemas siempre se basan en sistemas estadísticos y probabilísticos, a diferencia de sistemas deterministas donde unos datos de partida llevarán siempre a los mismos resultados. En el caso presentado en esta tesis, son pocos los comandos de voz que llegaran a ser utilizados, por lo que con un buen entrenamiento del modelo acústico se podrá llegar a tener una precisión bastante elevada.

Al usar un servicio web remoto para el convertir voz a texto, se reduce el costo de implementación para la persona no vidente al tiempo que genera un mayor rendimiento de conversión. Además, que el servicio web puede seguir mejorando y teniendo más funcionalidades sin que los clientes sean afectados. Sin embargo, cabe considerar que todo esto es posible, siempre y cuando el servidor pueda ser costeado por una entidad gubernamental.

Cabe destacar que la creación del servicio web SOAP fue relativamente sencilla, ya que las herramientas de programación como las que son incorporadas por el IDE NetBeans escriben la mayor parte del código necesario.

Se concluye que Java contiene librerías que facilitan el proceso de intercambio de datos entre dispositivos usando el protocolo TCP/IP. El proceso es transparente para el programador, sin embargo, es importante considerar los tipos de datos y el orden de recepción y envío.

El lenguaje de programación Java se usó en todo el desarrollo del presente trabajo, lo cual muestra una de las grandes ventajas de Java, puesto que como el código se ejecuta en una máquina virtual es compatible en casi cualquier dispositivo. De hecho, los programas realizados en este trabajo pueden usarse en cualquier sistema operativo y cualquier dispositivo, desde servidores hasta teléfonos inteligentes.

El Raspberry Pi puede ser configurado remotamente ya que permite ejecutar un servidor SSH. Además de esto, permite que el dispositivo pueda ser programado desde plataformas como NetBeans, el cual fue usado para el desarrollo de este proyecto.

Las clases asíncronas en Android representan una gran ayuda para la ejecución de código en diferentes hilos de ejecución lo cual es necesario cuando se trabajan con procesos los cuales requieren que se ejecuten en segundo plano. En audio esto es principalmente beneficioso, cuando se requiere de la reproducción de la conversión de texto a habla por parte de la aplicación Android.

7.2 Recomendaciones

Uso de audio comprimido:

Cuando se genera la transmisión de audio desde el cliente Android hacia el servicio web, se usa audio sin compresión. Aunque se envía información relativamente liviana, se podría mejorar la velocidad de transmisión al usar audio con compresión, ya que el peso sería significativamente menor. Por ello se recomienda usar un algoritmo de compresión de preferencia sin pérdidas.

Mejora del modelo acústico:

Se recomienda adaptar al modelo acústico con más frases y comandos, así como con voces de diferentes personas de diferentes géneros y edades. Un buen proceso de adaptación a futuro puede mejorar significativamente el rendimiento del sistema, sobre todo si se quisiera poner en marcha el servicio web a nivel regional.

Base de datos para aplicación AVVI:

Se recomienda actualizar la base de datos de comandos para la aplicación y que puedan ser configurados por el usuario. De esa manera se podría asignar cada comando con una salida del Raspberry Pi, si se tienen más de estos dispositivos se podrían seguir configurando con más comandos.

Detector automático de grabación:

Cuando el sistema comienza la grabación del comando de voz, el usuario tiene aproximadamente 4 segundos para decir el comando. Por ello, se recomienda la implementación de un algoritmo que detecte automáticamente el momento en que la persona deje de hablar y proceda a realizar el respectivo procesamiento.

Usar gestor de dependencias Maven en librerías externas:
Se recomienda usar el gestor de dependencias Maven para el control y gestión de librerías externas, esto ayudaría a que el proyecto pueda ser utilizado por otros programadores sin tener que preocuparse por descargar las librerías y sus dependencias, ya que el repositorio de Maven contiene todas las librerías usadas actualmente.

Concurrencia servidor Speech2TextServer:
Se recomienda crear varios hilos de ejecución los cuales puedan atender a las peticiones de los clientes simultáneamente, esto requiere que el ordenador en el que se esté ejecutando el servidor "Speech2TextServer" tenga un procesador de varios núcleos y una cantidad considerable de memoria RAM. En la práctica es imposible tener una cantidad elevada de núcleos, sin embargo, Java permite gestionar los hilos de ejecución de manera ordenada, de tal forma que con un número reducido de hilos de ejecución se puede atender los pedidos de los clientes.

Seguridad del servicio web:
Se recomienda establecer parámetros de seguridad como encriptamiento de datos, el uso de conexiones privadas y el uso de identificación para el cliente. Esto ayudaría a que tanto el servidor como el cliente consuma el servicio web de forma segura. De hecho, NetBeans provee un conjunto de herramientas que de una manera sencilla permiten hacer uso de seguridad en servicios web SOAP.

Aprendizaje automático con ayuda por voz:
Se recomienda que el servidor utilice aprendizaje automático, lo cual puede hacerse usando interacción con el usuario por medio de comandos por voz. Es decir, cuando un comando no pueda ser reconocido, el sistema pueda encontrar

el comando más probable y aprenderlo. Así el sistema relaciona un comando con otro de tal manera que mejore el reconocimiento y cada vez se adapte a la voz de diferentes personas. Además, se puede usar las librerías creadas que permiten grabar la información de audio que recibe el servidor con la finalidad que el servidor realice una adaptación constante al modelo acústico.

Extensión de funcionamiento a diferentes dispositivos:

La aplicación está diseñada para un modelo específico de dispositivo Android, es decir en el dispositivo que se usó para depurar la aplicación. Sin embargo, otros dispositivos tienen diferentes motores de audio, tamaños de pantalla, motores TTS, etc. Por ello se recomienda extender la funcionalidad a diferentes dispositivos de manera que pueda ser usado en una amplia gama de tablets o smartphones.

Control y gestión de los servidores por medio de terminal

Una vez que los servidores de domótica o de conversión de voz a texto comienzan a ejecutarse, no pueden ser configurados sino hasta que son detenidos. Es decir, solo pueden ser configurados antes de ser ejecutados. Por ello, se recomienda agregar un algoritmo que permite configurarlos mediante el terminal durante su ejecución. Esto sería principalmente útil si en un futuro se usan en un servidor público y se requiere actualizar el sistema o corregir errores en la marcha.

Descubrir dispositivos automáticamente en la red y agregar más servidores de domótica

La aplicación ha sido configurada para usar únicamente un Raspberry Pi conocido, es decir lo ubica en la red local mediante una dirección IP fija. Sin embargo, esto no es práctico ya que no todas las redes poseen un mismo tipo de direcciones IP. Por lo cual, se recomienda que el sistema pueda encontrar automáticamente a dispositivos Raspberry Pi. Así no importaría la red en la cual se instale al dispositivo ya que el dispositivo esperaría que se le asigne una

dirección IP desde el servidor DHCP, y la aplicación se encargaría de encontrarlo.

Creación de localización de objetos

Como una mejora a futuro del sistema se plantea que este sea capaz de localizar objetos fáciles de perder dentro del domicilio como por ejemplo un control remoto. A éste objeto se le puede adherir un dispositivo de tamaño reducido el cual sea capaz de emitir un sonido una vez que reciba la orden por parte del sistema.

Crear una librería general para la programación de todo el sistema.

El sistema posee varias librerías separadas unas de otras, ya que cada librería ha sido diseñada para cada parte del sistema. Sin embargo, cuando un programador desee modificar el sistema en caso de querer mejorarlo o por experimentación, tendrá que manejar todas las librerías separadas lo cual haría que el proyecto sea difícil de estudiar y modificar. Por ello, se recomienda crear una librería general para todo el sistema, y esta librería a su vez incorpore al gestor de dependencias Maven. De esta manera si una persona desea replicar cualquier parte del proyecto, solo bastaría que use la librería general que contiene todas las clases y métodos usados. Además, esto permitiría en un futuro poder subir la librería a un repositorio como Maven, lo cual mejoraría la capacidad del proyecto de crecer con aportes de terceros.

REFERENCIAS

Android. (2016). Getting Started | Android Developers. Recuperado el 3 de septiembre de 2016, de: https://developer.android.com/training/index.html

Arquitectura cliente-servidor. (2009). Recuperado el 15 de agosto de 2016 de: http://maite29.upc.es/labt2/practiques/html/esp/practica4/c_tcp003.html

Ayres, T., & Nolan, B. (2006). Voice activated command and control with speech recognition over WiFi. *Science of Computer Programming*, *59*(1), 109–126.

Bainbridge, W. S. (2004). *Berkshire Encyclopedia of Human-computer Interaction*. Berkshire Publishing Group. Recuperado el 14 de agosto de 2016 de: https://books.google.com.ec/books?id=568u_k1R4lUC

Buyya, R., & Dastjerdi, A. V. (2016). *Internet of Things: Principles and Paradigms*. Elsevier Science. Recuperado el 14 de agosto de 2016 de: https://books.google.com.ec/books?id=_k11CwAAQBAJ

Campa, A. (2010). Ventajas y Beneficios de los Servicios Web. Recuperado el 16 de septiembre de 2016 de: https://es.scribd.com/doc/44894535/Ventajas-y-Beneficios-de-los-Servicios-Web

Campo-Carlos, C., Rubio, G., & Me, J. (2016). Java ME. Recuperado el 13 de noviembre de 2016 de: http://ocw.uc3m.es/ingenieria-telematica/aplicaciones-moviles/material-de-clase-2/javame

CEDOM - Asociación Española de Domótica e Inmótica. (2016). ¿Qué es Domótica? Recuperado el 11 de noviembre de 2016 de:

http://www.cedom.es/sobre-domotica/que-es-domotica

Cellan-Jones, R. (2011). A 15 pound computer to inspire young programmers. Recuperado el 14 de noviembre de 2016 de: http://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html

Ciberaula. (2016). Programacion Orientada a Objetos. Recuperado el 13 de octubre de 2016 de: http://www.ciberaula.com/articulo/tecnologia_orientada_objetos

CMUSphinx. (2016). Basic concepts of speech [CMUSphinx Wiki]. Recuperado el 20 de octubre de 2016 de: http://cmusphinx.sourceforge.net/wiki/tutorialconcepts

Comer, D. E. D. E., Equihua, M., Equihua, S. M., & Huitema, C. H. (1996). *Redes globales de información con Internet y TCP/IP: Principios básicos, protocolos y arquitectura*. Prentice-Hall Hispanoamericana,.

Dennis, A. K. (2013). *Raspberry Pi Home Automation with Arduino*. Packt Publishing. Recuperado el 20 de septiembre de 2016 de: https://books.google.es/books?id=AMdyVnM9CroC

DÍAZ BARRÓN, P. (2012). *Aplicación gráfica para dispositivos iOS para creación/edición de patches del lenguaje de programación orientado a flujo denominado“ Pure Data.”* Recuperado el 9 de diciembre de 2016 de: https://riunet.upv.es/handle/10251/18226?show=full

Dordoigne, J. (2011). *REDES INFORMATICAS. NOCIONES FUNDAMENTALES*. Ediciones ENI. Recuperado el 3 de septiembre de 2016 de: https://books.google.com.ec/books?id=25Lmhq58C38C

EcuRed. (2016). Arquitectura Cliente Servidor. Recuperado el 6 de septiembre de 2016 de: https://www.ecured.cu/Arquitectura_Cliente_Servidor

Franco, A. (2000). La Máquina Virtual Java. Recuperado el 9 de septiembre de 2016 de: http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/introduccion/virtual.htm

Garzón, J. (2015). Android Lollipop: novedades. Actualización a Android 5.1 y Android 5.0 Lollipop - CNET en Español. Recuperado el 12 de octubre de 2016 de: https://www.cnet.com/es/analisis/google-android-5-0-lollipop/

Google. (2016). Síntesis de voz de Google - Aplicaciones de Android en Google Play. Recuperado el 2 de diciembre de 2016 de: https://play.google.com/store/apps/details?id=com.google.android.tts&hl=es

Google. (2016). Speech API - Speech Recognition - Google Cloud Platform. Recuperado el 7 de septiembre de 2016 de: https://cloud.google.com/speech/

Goyal, S. J. and A. V. and L. (2014). Raspberry Pi based interactive home automation system through E-mail. *Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on*, 277–280. https://doi.org/10.1109/ICROIT.2014.6798330

Holtmann, M. (2008). Audio streaming over Bluetooth. *Linux Symposium*, 193.

Huang, Y. (2015). *Contributions to the Resilience Management in the Internet of Things*. ETSIS_Telecomunicacion. Recuperado el 9 de diciembre de 2016 de: http://oa.upm.es/37209/1/YUANJIANG_HUANG.pdf

IBM - Knowledge Center. (2016). Protocolos TCP/IP. Recuperado el 16 de noviembre de 2016 de: https://www.ibm.com/support/knowledgecenter/es/ssw_aix_72/com.ibm.aix.networkcomm/tcpip_protocols.htm

Jiang, H., Han, Z., Scucces, P., Robidoux, S., & Sun, Y. (2000). Voice-activated environmental control system for persons with disabilities. In *Bioengineering Conference, 2000. Proceedings of the IEEE 26th Annual Northeast* (pp. 167–168).

Kurniawan, A. (2014). *Getting Started with Java ME Embedded 8 and Raspberry Pi:* PE Press. Recuperado el 15 de octubre de 2016 de: https://books.google.com.ec/books?id=M1CuAwAAQBAJ

Franco, A. (2001). *La Máquina Virtual Java.* Recuperado el 12 de septiembre de 2016 de: http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/introduccion/virtual.htm

LangPop.com. (2009). *Programming Language Popularity*. Recuperado el 1 de octubre de 2016 de: http://web.archive.org/web/20150829085248/http://langpop.com/

Mitra, S. K. (2007). *Procesamiento de señales digitales: un enfoque basado en computadora*. McGraw-Hill-Interamericana. Recuperado el 1 de noviembre de 2016 de: https://books.google.com.ec/books?id=9V3kAQAACAAJ

Oracle. (2016). Conozca más sobre la tecnología Java. Recuperado el 2 de septiembre de 2016 de: https://www.java.com/es/about/

Oracle. (2016). Java Platform, Micro Edition (Java ME). Recuperado el 4 de septiembre de 2016 de: http://www.oracle.com/technetwork/java/embedded/javame/index.html

Pérez, A. E. (2013). *Diseño de aplicación móvil para la comunicación inalámbrica de señales audiovisuales*. Universitat Politècnica de Catalunya. Recuperado el 28 de septiembre de 2016 de: http://upcommons.upc.edu/bitstream/handle/2099.1/21895/alberto.esteban.perez_90828.pdf?sequence=1

Pérez, D. (2015). Google Play supera a AppStore en cantidad de aplicaciones. Recuperado el 4 de octubre de 2016 de: https://www.elandroidelibre.com/2015/01/google-play-supera-la-appstore-en-cantidad-de-aplicaciones-y-desarrolladores.html

Prasanna, G., & Ramadass, N. (2014). Low Cost Home Automation Using Offline Speech Recognition. Recuperado el 9 de diciembre de 2016 de http://www.ijsps.com/uploadfile/2014/0806/20140806030353475.pdf

Raspberry Pi Foundation. (2016). Raspberry Pi FAQs - Frequently Asked Questions. Recuperado el 27 de septiembre de 2016 de: https://www.raspberrypi.org/help/faqs/#buyingWhere

Robalino Puente, L. D. (2007). *Diseño e implementación de un control remoto controlado por órdenes de voz para aplicaciones de control en una vivienda*. Recuperado el 9 de diciembre de 2016 de http://bibdigital.epn.edu.ec/handle/15000/4167

Rouillard, J. (2006). Web services and speech-based applications around VoiceXML. Recuperado el 9 de diciembre de 2016 de https://www.researchgate.net/publication/42804092_Web_services_and_speech-based_applications_around_VoiceXML

Sherlin.xBot.es. (2016). ¿Qué es un microcontrolador? Recuperado el 12 de octubre de 2016 de: http://sherlin.xbot.es/microcontroladores/introduccion-a-los-microcontroladores/que-es-un-microcontrolador

Shiyam Raghul, M., Surendhar, K., & Suresh, N. (2016). Raspberry-Pi Based Assistive Device for Deaf, Dumb and Blind People. Recuperado el 9 de diciembre de 2016 de http://www.academia.edu/25126910/Raspberry-Pi_Based_Assistive_Device_for_Deaf_Dumb_and_Blind_People

Super Info. (2014). MODELO TCP/IP VENTAJAS Y DESVENTAJAS. Recuperado el 6 de septiembre de 2016 de: http://superinformacionweb.blogspot.com/2014/03/modelo-tcpip-ventajas-y-desventajas.html

Te lo dije ... (2010). Servicios web (1): SOAP no, gracias. Recuperado el 7 de octubre de 2016 de: https://eamodeorubio.wordpress.com/2010/07/19/servicios-web-1-soap-no-gracias/

Te lo dije ... (2010). Servicios web (2): ¿Qué es REST? Recuperado el 7 de octubre de 2016 de: https://eamodeorubio.wordpress.com/2010/07/26/servicios-web-2-¿que-es-rest/

TechNorms. (2016). Top 10 Of The Best Free Web Services For Text-To-Speech Conversion. Recuperado el 9 de noviembre de 2016 de: http://www.technorms.com/980/top-10-web-based-services-for-text-to-speech-conversion

Treehouse Blog. (2014). Accepting Speech Input in HTML5 Forms. Recuperado

el 11 de noviembre de 2016 de: http://blog.teamtreehouse.com/accepting-speech-input-html5-forms

Úbeda, B. (2009). Apuntes de: Sistemas embebidos. Recuperado el 4 de diciembre de 2016 de: http://ocw.um.es/ingenierias/sistemas-embebidos/material-de-clase-1/ssee-t01.pdf

Universidad de Granada. (2016). Internet TCP/IP: Transmisión de datos y redes de ordenadores. Recuperado el 9 de diciembre de 2016 de http://www.lawebdelprogramador.com/pdf/1503-Internet-TCP-IP-Transmision-de-datos-y-redes-de-ordenadores.html

Upadhyay, M. S. K., & Chavda, M. V. N. (2014). 'INTELLIGENT SYSTEM BASED ON SPEECH RECOGNITION WITH CAPABILITY OF SELFLEARNING'. *International Journal For Technological Research In Engineering ISSN (Online)*, 2347–4718.

W3C España. (2016). Guía Breve de Servicios Web. Recuperado el 8 de diciembre de 2016 de: http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

Valbuena, S. J., & Rodríguez, L. A. H. (2010). *Programación Orientada a Objetos*. Ediciones Elizcom. Recuperado el 6 de septiembre de 2016 de: https://books.google.com.ec/books?id=sWaoGgTfu-QC

Xiao, J., Ramdath, K., Iosilevish, M., Sigh, D., & Tsakas, A. (2013). A low cost outdoor assistive navigation system for blind people. In *Industrial Electronics and Applications (ICIEA), 2013 8th IEEE Conference on* (pp. 828–833).

ANEXOS

ANEXO 1

LIBRERÍA AUDIOENGINE

Clase: AudioEngine.java

Detalles:
La clase AudioEngine permite usar el motor de audio del dispositivo Android. Permite inicializar y configurar sus propiedades para la reproducción y grabación de audio.

Librerías usadas:
android.media
android.util.Log
java.io
java.nio

Campos:

`String SAMPLE_RATE = 16000`

Constructor:

`public AudioEngine()` Crea una nueva instancia de la clase Audio Engine.	
Argumentos	Sin argumentos de entrada.

Métodos:

`public void init()` Inicializa el motor de audio del dispositivo, con valores por defecto: Frecuencia de muestreo: 16.000 Hz. Número de bits por muestra: 16 bits. Tipo de Codificación: PCM. Formato de almacenamiento: little-endian.	
Argumentos	Sin argumentos de entrada.
Salida	No devuelve nada.

`public void setSAMPLE_RATE(int SAMPLE_RATE)` Asigna un valor de frecuencia de muestreo, valor por defecto 16000.	
Argumentos	**SAMPLE_RATE:** Un número entero con el valor de la frecuencia de muestreo, valores típicos: 8.000, 16.000, 22.050, 44.100 y 48.000.
Salida	No devuelve nada.

`public byte[] recordAudio()` Captura el audio del micrófono del dispositivo por tres segundos aproximadamente.	
Argumentos	Sin argumentos de entrada.
Salida	Devuelve el audio sin compresión capturado en un arreglo de tipo byte.

`public void playAudio(byte[] audioDataByteArray)` Reproduce audio en el dispositivo.	
Argumentos	**audioDataByteArray:** Información de audio en un arreglo de tipo byte.
Salida	No devuelve nada.

ANEXO 2

LIBRERÍA CLIENTTCPENGINE

Clase: ClientTCPEngine.java

Detalles:
Esta clase permite la conexión con el servidor de domótica que se ejecuta en el Raspberry Pi. Permite enviar y recibir datos al servidor de manera correcta y en un formato compatible entre los dispositivos.

Librerías usadas:
java.io
java.net

Campos:

No existen campos disponibles.

Constructor:

`public ClientTCPEngine(String clientName, String ipAddress, int port)` Crea una nueva instancia de la clase ClientTCPEngine, considerando el nombre del cliente, la dirección IP y el puerto del servidor.	
Argumentos	**clientName:** nombre del cliente, es decir del dispositivo Android que quiere conectarse al servidor. **ipAddress:** Dirección IP del Servidor de domótica, es decir del Raspberry Pi. **port:** Puerto del servidor.

Métodos:

`public void init()` Crea una conexión con el servidor de domótica.	
Argumentos	Sin argumentos de entrada.
Salida	No devuelve nada.

`public void txCommand(String command)` Envía al servidor de domótica el comando especificado.	
Argumentos	**command:** Comando en formato "String" que será enviado al servidor, este comando corresponde a una cadena de números que representan una acción que se ejecutará en el Raspberry Pi, por ejemplo: "10601" que significa comando del tipo 1 poner en el canal 06 el estado 01. Si el canal es digital solo se usará 00 o 01, si el canal es analógico se usarán valores del 00 al 99.
Salida	No devuelve nada.

`public String rxCommand()` Recibe la respuesta del servidor que se genera cuando se envía un comando.	
Argumentos	Sin argumentos de entrada.
Salida	Devuelve una respuesta en formato "String", la respuesta puede ser "null" en caso de que el servidor no entienda el comando enviado.

ANEXO 3

LIBRERÍA COMMANDENGINE

Clase: CommandEngine.java

Detalles:
Esta clase permite transformar el texto recibido desde el servicio web a un texto que representa un comando que podrá interpretar el servidor de domótica, es decir el Raspberry Pi.

Librerías usadas:
Ninguna librería.

Campos:

```
String rasPiCmd = null
String speakCmd = "Lo siento, no pude reconocer ese comando,
inténtalo nuevamente!"
```

Constructor:

`public CommandEngine()` Crea una nueva instancia de la clase CommandEngine.	
Argumentos	Sin argumentos de entrada.

Métodos:

`public void createCommandsToSendAndSpeak(String speechText)` Crea un comando entendible por el Raspberry Pi considerando una base de datos almacenada en el dispositivo Android en la que busca el comando de voz interpretado.	
Argumentos	**speechText:** Texto que representa al comando de voz interpretado por el servicio web, por ejemplo: "Activa las lámparas".
Salida	No devuelve nada, el resultado se guarda en el campo rasPiCmd.

`public String getRasPiCmd()` Devuelve el valor del campo rasPiCmd. Valor por defecto: "null".	
Argumentos	Sin argumentos de entrada.
Salida	Campo rasPiCmd.

`public void setRasPiCmd(String rasPiCmd)` Cambia el valor del campo rasPiCmd. Valor por defecto: "null".	
Argumentos	**rasPiCmd:** Nuevo valor.
Salida	No devuelve nada.

`public String getSpeakCmd()` Devuelve el valor del campo speakCmd. Valor por defecto: "Lo siento, no pude reconocer ese comando, inténtalo nuevamente!".	
Argumentos	Sin argumentos de entrada.
Salida	Campo speakCmd.

`public void setSpeakCmd(String speakCmd)` Cambia el valor del campo speakCmd. Valor por defecto: "Lo siento, no pude reconocer ese comando, inténtalo nuevamente!".	
Argumentos	**speakCmd:** Nuevo valor.
Salida	No devuelve nada.

ANEXO 4

LIBRERÍA COMMANDENGINESERVER

Clase: CommandEngineServer.java

Detalles:
Esta clase permite interpretar el comando enviado desde el dispositivo Android y luego ejecutar la acción en el Raspberry Pi.

Librerías usadas:
Ninguna librería.

Campos:

No existen campos disponibles.

Constructor:

`public CommandEngineServer()` Crea una nueva instancia de la clase CommandEngineServer.	
Argumentos	Sin argumentos de entrada.

Métodos:

`public String runCommand(String command)` Interpreta y ejecuta en el Raspberry Pi el comando enviado por el dispositivo Android.	
Argumentos	**command:** Comando recibido desde el dispositivo Android, debe ser del tipo String.
Salida	Devuelve una cadena de caracteres que muestra el estado de la ejecución. Si la ejecución tuvo éxito devolverá: “Ready”, caso contrario devuelve: "null".

ANEXO 5

LIBRERÍA CLASE RASPIGPIOENGINE

Clase: RasPiGPIOEngine.java

Detalles:
Esta clase permite el uso del módulo GPIO del Raspberry Pi, esto es posible ya que hace uso de la librería "pi4j". Para que esta clase funcione correctamente, la respectiva librería debe estar agregada al proyecto.

Librerías usadas:
com.pi4j

Campos:

No existen campos disponibles.

Constructor:

`public RasPiGPIOEngine()` Crea una nueva instancia de la clase RasPiGPIOEngine. Argumentos	
Argumentos	Sin argumentos de entrada.

Métodos:

`public void init()` Inicializa el módulo GPIO y crea un conjunto de variables representando las diferentes entradas y salidas que pueden ser usadas del Raspberry Pi (para más información ver Anexo 9).	
Argumentos	Sin argumentos de entrada.
Salida	No devuelve nada.

`public void setHighPin(int pinNumber)` Pone un uno lógico en el pin especificado.	
Argumentos	**pinNumber:** Número de pin en el Raspberry Pi.
Salida	No devuelve nada.

`public void setLowPin(int pinNumber)` Pone un cero lógico en el pin especificado.

Argumentos	**pinNumber:** Número de pin en el Raspberry Pi.
Salida	No devuelve nada.

```
public void togglePin(int pinNumber)
```
Cambia el estado actual del pin especificado al estado contrario. Solo válido para salidas digitales.

Argumentos	**pinNumber:** Número de pin en el Raspberry Pi.
Salida	No devuelve nada.

```
public void setPin(int pinNumber, int state)
```
Pone el valor del argumento “state” en el pin especificado. Válido para salidas analógicas y digitales.

Argumentos	**pinNumber:** Número de pin en el Raspberry Pi. **state:** Número entero que representa el valor en el pin, para salidas analógicas los valores disponibles van desde 0 al 99. Para salidas digitales, los valores van de 0 a 1.
Salida	No devuelve nada.

ANEXO 6

LIBRERÍA SPEECHENGINE

Clase: SpeechEngine.java

Detalles:
Esta clase permite usar el convertidor de voz a texto para ello usa el API de CMU Sphinx de manera continua. Esta clase brinda los métodos necesarios para inicializar y detener el convertidor, dar formato a la información de audio que se usará en la conversión y un método para guardar en el disco duro la información de audio en formato wav.

Librerías usadas:
java.io
java.net
edu.cmu.sphinx
javax.sound.sampled

Campos:

No existen campos disponibles.

Constructor:

`public SpeechEngine()` Crea una nueva instancia de la clase SpeechEngine y se inicializan variables globales de configuración.	
Argumentos	Sin argumentos de entrada.

Métodos:

`public void init()` Inicializa el motor de conversión, el sistema carga todos los recursos necesarios en la memoria, por ejemplo: el modelo acústico, el modelo de lenguaje, el diccionario, etc.	
Argumentos	Sin argumentos de entrada.
Salida	No devuelve nada.

`public void stop()` Detiene al motor de conversión de voz a texto.	
Argumentos	Sin argumentos de entrada.

Salida	No devuelve nada.

`public String speech2str (InputStream stream)` Convierte voz a texto, el formato de audio usado debe tener el siguiente formato: Frecuencia de muestreo: 16000 Hz. Número de bits por muestra: 16 bits. Tipo de Codificación: PCM. Formato de almacenamiento: little-endian. Además debe ser almacenado en una variable del tipo: "InputStream".	
Argumentos	**stream:** Información de audio almacenado adecuadamente en una variable formato InputStream.
Salida	Devuelve el resultado de la conversión en una cadena de caracteres.

`public InputStream formatWaveByteInputTCPIP(byte[] data, AudioFormat format)` Convierte el arreglo de bytes recibido mediante protocolo TCP/IP en una variable de tipo: "InputStream", con formato de audio especificado en el argumento: "format".	
Argumentos	**data:** Arreglo de bytes que almacena información de audio. **format:** Formato de audio para la conversión entre variables.
Salida	Devuelve la información de audio en una variable del tipo: "InputStream".

`public void saveWaveFileFromByteArray(byte[] data, AudioFormat format,String fileAddress)` Guarda la información de audio almacenada en un arreglo de bytes en un archivo sin compresión tipo wav. Se debe especificar el formato y la dirección del archivo en el disco duro.	
Argumentos	**data:** Arreglo de bytes que almacena información de audio. **format:** Formato de audio para la conversión entre variables. **fileAddress:** Nombre del archivo.
Salida	No devuelve nada.

ANEXO 7

LIBRERÍA TCPIPTOOLS

Clase: ClientTCPIPTools.java

Detalles:
Esta clase permite la conexión con el servidor web u otro cliente que contenga la información de audio de voz para ser convertida a texto. Permite enviar y recibir datos desde el cliente manera correcta y en un formato compatible a través del protocolo TCP/IP.

Librerías usadas:
java.io
java.net

Campos:

No existen campos disponibles.

Constructor:

`public TCPIPTools()` Crea una nueva instancia de la clase TCPIPTools.	
Argumentos	Sin argumentos de entrada.

Métodos:

`public byte[] rxBytesFromClient(Socket id)` Recibe datos desde el cliente a través del protocolo TCP/IP.	
Argumentos	**id:** Objeto del tipo Socket con información del cliente conectado al servidor.
Salida	Devuelve un arreglo de bytes con la información de audio recibida desde el servidor.

`public void txHypothesis(Socket id, String hypothesis)` Envía datos desde el cliente a través del protocolo TCP/IP, el cliente recibirá una cadena de caracteres con la conversión de voz a texto o un mensaje de error.	
Argumentos	**id:** Objeto del tipo Socket con información del cliente conectado al servidor. **hypothesis:**

	Información de la conversión de voz a texto que se envía al cliente.
Salida	No devuelve nada.

ANEXO 8

LIBRERÍA CLIENTSPEECHENGINE

Clase: ClientSpeechEngine.java

Detalles:

Esta clase permite el intercambio de datos entre el servicio web y el servidor de conversión de voz a texto. Permite enviar y recibir datos en forma correcta y en un formato compatible a través del protocolo TCP/IP.

Librerías usadas:

java.io

java.net

Campos:

No existen campos disponibles.

Constructor:

`public ClientSpeechEngine(String clientName, String ipAddress, int port)` Crea una nueva instancia de la clase ClientSpeechEngine. Es necesario especificar el nombre del cliente, la dirección IP del servidor y el puerto.	
Argumentos	**clientName:** Nombre del cliente que se conecta al servidor, puede ser nulo. **ipAddress:** Dirección IP del servidor. **port:** Puerto activo del servidor.

Métodos:

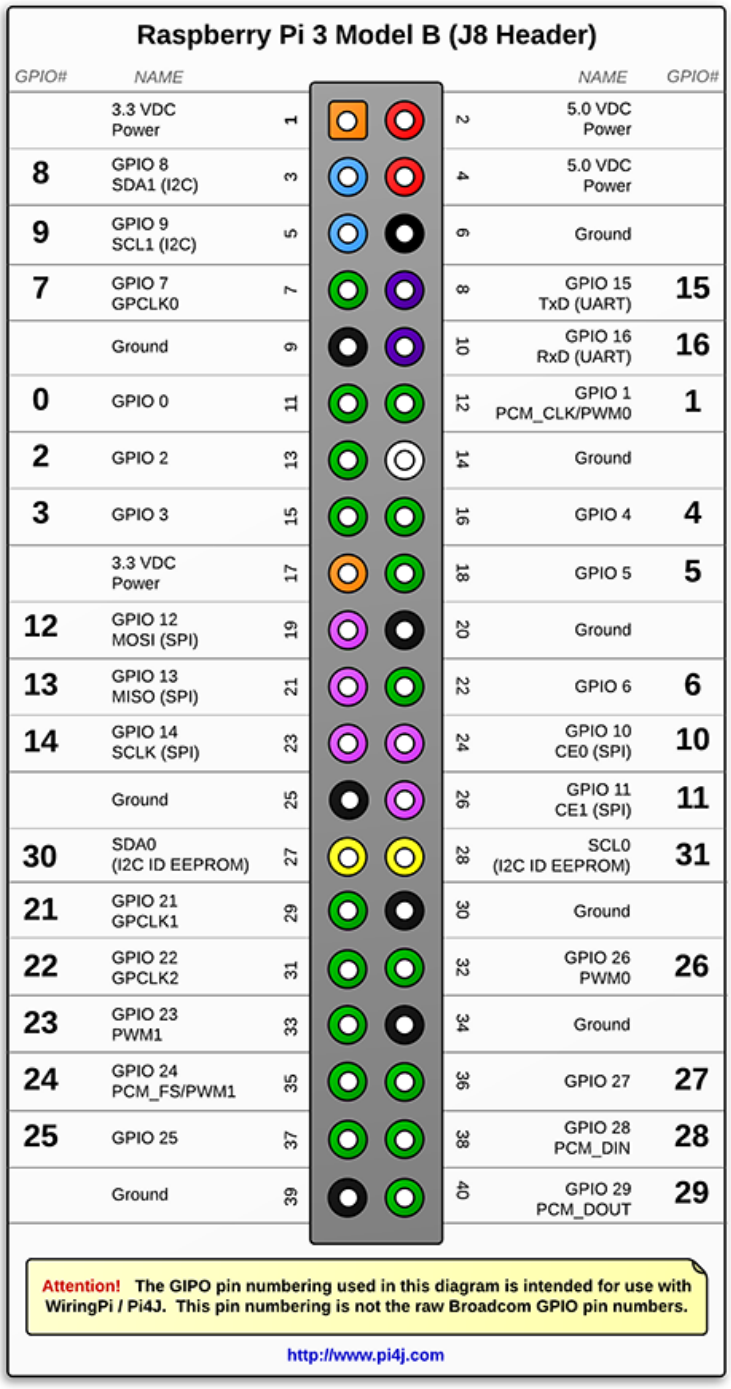
`public void init()` Inicializa la conexión con el servidor.	
Argumentos	Sin argumentos de entrada.
Salida	No devuelve nada.

`public void txBytesToServer(byte[] data)` El servidor web recibe desde un dispositivo Android información de audio. Mediante este método, esta información de audio es enviada hacia el servidor que convierte voz a texto.	
Argumentos	**data:** Información de audio almacenada en un arreglo de datos.
Salida	No devuelve nada.

`public String rxHypothesis()` El servidor web recibe la conversión de voz a texto.	
Argumentos	Sin argumentos de entrada.
Salida	Devuelve la conversión de voz a texto almacenada en una cadena de caracteres.

ANEXO 9

DIAGRAMA DE PINES DE UN RASPBERRY PI 3 MODELO B



Tomado de: Pi4j

ANEXO 10

CÓDIGO FUENTE APLICACIÓN ANDROID AVVI

1. Clase MainActivity.java:

```
package com.application.avvi;

import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
import android.util.Log;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.nio.ByteBuffer;


public class AudioEngine {

  private static final String LOG_TAG = null;
  private int SAMPLE_RATE = 16000;

  private AudioRecord audioRecorder;
  private ByteArrayOutputStream out;
  private int factorBuffer;
  private ByteBuffer aB;
  private int numBytesRead;
  private int bytesRead;
  private int audioRecordBufferSize;

  private AudioTrack audioPlayer;
  private byte[] bytesBuffer;
  int audioTrackBufferSize;


  public AudioEngine() {
  }

  public void init(){
      setAudioRecordBufferSize();
      setAudioRecord();
      configInitParamsAudioRecord();
```

```
        setAudioTrackBufferSize();
        setAudioTrack();
        configInitParamsAudioTrack();
    }



    public void setSAMPLE_RATE(int SAMPLE_RATE) {
        this.SAMPLE_RATE = SAMPLE_RATE;
    }

    private void setAudioRecordBufferSize (){

        audioRecordBufferSize =
AudioRecord.getMinBufferSize(SAMPLE_RATE,
                AudioFormat.CHANNEL_IN_MONO,
                AudioFormat.ENCODING_PCM_16BIT);

        if (audioRecordBufferSize == AudioRecord.ERROR ||
audioRecordBufferSize == AudioRecord.ERROR_BAD_VALUE) {
            audioRecordBufferSize = SAMPLE_RATE * 2;
        }
    }

    private void setAudioTrackBufferSize() {
        audioTrackBufferSize =
AudioTrack.getMinBufferSize(SAMPLE_RATE,
AudioFormat.CHANNEL_OUT_MONO,
                AudioFormat.ENCODING_PCM_16BIT);
        if (audioTrackBufferSize == AudioTrack.ERROR ||
audioTrackBufferSize == AudioTrack.ERROR_BAD_VALUE) {
            audioTrackBufferSize = audioRecordBufferSize;
        }
    }

    private void setAudioRecord(){

        audioRecorder = new
AudioRecord(MediaRecorder.AudioSource.MIC,
                SAMPLE_RATE,
                AudioFormat.CHANNEL_IN_MONO,
                AudioFormat.ENCODING_PCM_16BIT,
                audioRecordBufferSize);

        if (audioRecorder.getState() !=
AudioRecord.STATE_INITIALIZED) {
            Log.e(LOG_TAG, "Audio Record can't initialize!");
```

```
        }
    }

    private void setAudioTrack(){
        audioPlayer = new AudioTrack(
                AudioManager.STREAM_MUSIC,
                SAMPLE_RATE,
                AudioFormat.CHANNEL_OUT_MONO,
                AudioFormat.ENCODING_PCM_16BIT,
                audioTrackBufferSize,
                AudioTrack.MODE_STREAM);
    }


    private void configInitParamsAudioRecord(){
        out = new ByteArrayOutputStream();
        factorBuffer = (int)
((1280.0/audioRecordBufferSize)*73.0);
        aB=
ByteBuffer.allocateDirect(audioRecordBufferSize*factorBuffer);

    }


    private void initParamsRecording(){
        numBytesRead = 0;
        bytesRead = 0;
    }

    private void configInitParamsAudioTrack(){
        bytesBuffer = new byte[audioRecordBufferSize];

    }


    public byte[] recordAudio(){

        initParamsRecording();

        audioRecorder.startRecording();

        while (bytesRead < audioRecordBufferSize*factorBuffer) {
            numBytesRead =
audioRecorder.read(aB,audioRecordBufferSize);
            out.write(aB.array(), 0, numBytesRead);
            bytesRead = bytesRead + numBytesRead;
```

```
        }
        audioRecorder.stop();
        audioRecorder.release();

        return out.toByteArray();
    }

    public void playAudio(byte[] audioDataByteArray){

        ByteArrayInputStream in = new
ByteArrayInputStream(audioDataByteArray);

        audioPlayer.play();

        int bytesReadAudioPlayer;

        while((bytesReadAudioPlayer =
in.read(bytesBuffer,0,bytesBuffer.length)) != -1){
            audioPlayer.write(bytesBuffer,0,
bytesReadAudioPlayer);
        }

        audioPlayer.release();

    }


}
```

2. Clase ClientTCPEngine.java:

```
package com.application.avvi;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;


public class ClientTCPEngine {

    String clientName;
```

```
  String ipAddress;
  int port;

  OutputStream out;
  DataOutputStream dos;

  BufferedReader in;

  Socket s;

  public ClientTCPEngine(String clientName, String ipAddress,
int port) {
      this.clientName = clientName;
      this.ipAddress = ipAddress;
      this.port = port;
  }

  public void init() throws IOException {
      s = new Socket(ipAddress, port);
  }

   public void txCommand(String command) throws IOException {
      PrintWriter out = new PrintWriter(s.getOutputStream(),
true);
      out.println(command);
      out.flush();

  }

  public String rxCommand() throws IOException {
      BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
      return in.readLine();

  }


}
```

3. Clase CommandEngine.java:

```
package com.application.avvi;

public class CommandEngine {
```

```
  String rasPiCmd = null;
  String speakCmd = "Lo siento, no pude reconocer ese comando,
inténtalo nuevamente!";

  public CommandEngine() {
  }

  public void createCommandsToSendAndSpeak(String speechText)
{
      rasPiCmd = null;
      speakCmd = "Lo siento, no pude reconocer ese comando,
inténtalo nuevamente!";

      switch (speechText) {
          case "activa las lámparas" :
              rasPiCmd = "10101";
              speakCmd = "Activando las lamparas!";
              break;
          case "activa la música":
              rasPiCmd = "10201";
              speakCmd = "Activando la música!";
              break;
          case "activa la radio":
              rasPiCmd = "10301";
              speakCmd = "Activando la radio!";
              break;

          case "iba las lámparas" :
              rasPiCmd = "10101";
              speakCmd = "Activando las lamparas!";
              break;
          case "iba la música":
              rasPiCmd = "10201";
              speakCmd = "Activando la música!";
              break;
          case "iba la radio":
              rasPiCmd = "10301";
              speakCmd = "Activando la radio!";
              break;

          case "sube el audio":
              rasPiCmd = "10401";
              speakCmd = "Subiendo el volumen de la música!";
              break;
          case "dónde está la puerta":
              rasPiCmd = "10501";
```

```
                speakCmd = "La Puerta está donde escuches el
beep!";
                break;

            case "des activa las lámparas":
                rasPiCmd = "10100";
                speakCmd = "Desactivando las lamparas!";
                break;
            case "des activa la música":
                rasPiCmd = "10200";
                speakCmd = "Desactivando la música!";
                break;
            case "des activa la radio":
                rasPiCmd = "10300";
                speakCmd = "Desactivando la radio!";
                break;
            case "desactivar las lámparas":
                rasPiCmd = "10100";
                speakCmd = "Desactivando las lamparas!";
                break;
            case "desactivar la música":
                rasPiCmd = "10200";
                speakCmd = "Desactivando la música!";
                break;
            case "desactivar la radio":
                rasPiCmd = "10300";
                speakCmd = "Desactivando la radio!";
                break;

            case "baja el audio":
                rasPiCmd = "10400";
                speakCmd = "Bajando el volumen de la música!";
                break;
            case "dónde están mis llaves":
                rasPiCmd = "10500";
                speakCmd = "Tus llaves están donde escuches el
beep!";
                break;



        }

    }

    public String getRasPiCmd() {
        return rasPiCmd;
```

```
  }

  public void setRasPiCmd(String rasPiCmd) {
      this.rasPiCmd = rasPiCmd;
  }

  public String getSpeakCmd() {
      return speakCmd;
  }

  public void setSpeakCmd(String speakCmd) {
      this.speakCmd = speakCmd;
  }
}
```

4. Archivo AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.application.avvi">

  <uses-permission
android:name="android.permission.RECORD_AUDIO" />
  <uses-permission android:name="android.permission.INTERNET"
/>

  <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
      <activity android:name=".MainActivity"
          android:screenOrientation="userPortrait">
          <intent-filter>
              <action
android:name="android.intent.action.MAIN" />

              <category
android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
      </activity>
  </application>
```

```
</manifest>
```

5. Archivo build.gradle:

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 24
  buildToolsVersion "25.0.0"

  defaultConfig {
      applicationId "com.application.avvi"
      minSdkVersion 10
      targetSdkVersion 24
      versionCode 1
      versionName "1.0"
  }
  buildTypes {
      release {
          minifyEnabled false
          proguardFiles getDefaultProguardFile('proguard-
android.txt'), 'proguard-rules.pro'
      }
  }
}

repositories {
  maven { url
'https://oss.sonatype.org/content/repositories/ksoap2-android-
releases/' }
}

dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  testCompile 'junit:junit:4.12'
  compile 'com.android.support:appcompat-v7:24.2.1'
  compile 'com.google.code.ksoap2-android:ksoap2-
android:3.6.2'
}
```

6. Archivo activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.application.avvi.MainActivity"
  android:id="@+id/screenId"
  android:background="@color/fondo2"
  android:clickable="true"
  android:onClick="onClickScreen">

  <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="AVVI"
      android:id="@+id/resString"
      android:textColor="@color/letrasInicioMd"
      android:textAlignment="center"
      android:textSize="45dp"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />
</RelativeLayout>
```

7. Archivo activity_loading_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:background="@color/fondo2"
tools:context="com.application.avvi.MainActivity">

<ImageView
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:id="@+id/imageView3"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="@string/launchname"
android:id="@+id/textView"
android:textSize="65dp"
android:textColor="@color/letrasInicio"
android:textIsSelectable="false"
android:layout_above="@+id/textView2"
android:layout_centerHorizontal="true" />

<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceMedium"
android:text="@string/asistente_controlado_por_voz_para_person
as_invidentes"
android:id="@+id/textView2"
android:textColor="@color/letrasInicioMd"
android:textIsSelectable="false"
android:textAlignment="center"
android:padding="8dp"
android:textSize="18dp"
android:layout_above="@+id/progressBar"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true" />

<ProgressBar
style="?android:attr/progressBarStyleHorizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:id="@+id/progressBar"
android:max="100"
android:progress="0"
android:progressTint="@color/colorAccent"
android:layout_centerVertical="true"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:indeterminate="false" />

<TextView
```

```
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceSmall"
android:text="@string/cargando_componentes_necesarios"
android:textColor="@color/letrasInicioMd"
android:textAlignment="center"
android:textStyle="italic"
android:id="@+id/stateLoadingText"
android:layout_below="@+id/progressBar"
android:layout_centerHorizontal="true"
android:padding="5dp" />

<ImageView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/imageView"
android:src="@drawable/udlalogo"
android:layout_alignLeft="@+id/textView"
android:layout_alignStart="@+id/textView"
android:layout_below="@+id/stateLoadingText"
android:layout_alignRight="@+id/textView"
android:layout_alignEnd="@+id/textView" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceSmall"
android:text="Universidad de las Américas"
android:textSize="19dp"
android:id="@+id/textView3"
android:textColor="@color/letrasInicioMd"
android:layout_below="@+id/imageView"
android:layout_centerHorizontal="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceSmall"
android:text="Ingeniería en Sonido y Acústica"
android:id="@+id/textView4"
android:textColor="@color/letrasInicioMd"
android:layout_below="@+id/textView3"
android:layout_centerHorizontal="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:textAppearance="?android:attr/textAppearanceSmall"
android:text="Creadores:"
android:id="@+id/textView5"
android:textColor="@color/letrasInicioMd"
android:layout_below="@+id/textView4"
android:layout_centerHorizontal="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceSmall"
android:text="Juan Chango"
android:id="@+id/textView6"
android:textColor="@color/letrasInicioMd"
android:layout_below="@+id/textView5"
android:layout_centerHorizontal="true" />

<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceSmall"
android:text="Andrés Márquez"
android:id="@+id/textView7"
android:textColor="@color/letrasInicioMd"
android:layout_below="@+id/textView6"
android:layout_centerHorizontal="true" />

</RelativeLayout>
```

ANEXO 11

CÓDIGO FUENTE SERVIDOR DE DOMÓTICA EN EL RASPBERRY PI

1. Clase Principal RasPiAVVIServer.java:

```
package raspiavviserver;

import java.io.BufferedReader;
import java.io.IOException;

import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;


public class RasPiAVVIServer extends Thread{

   Socket id;
   static CommandEngineServer commandEngine;
   String command = null;


   public RasPiAVVIServer(Socket s) {
       id = s;

   }

   @Override
   public void run() {


       try {
           PrintWriter out = new
PrintWriter(id.getOutputStream(), true);
           BufferedReader in = new BufferedReader(new
InputStreamReader(id.getInputStream()));

           System.out.print("Cliente conectado: ");
           command = in.readLine();
           System.out.println(command);
           String res = commandEngine.runCommand(command);
           out.println(res);
```

```
        } catch (IOException | InterruptedException ex) {
            Logger.getLogger(RasPiAVVIServer.class.getName()).l
og(Level.SEVERE, null, ex);
        }


    }

    public static void main(String[] args) throws IOException,
InterruptedException {
        System.out.println("***************  Servidor RasPi
AVVI  ***************");
        System.out.println("***************  Reconocimiento de
voz  ***************");
        System.out.println("**********************  UDLA  ****
********************");
        System.out.println("***************  ING SONIDO
ACUSTICA  ***************");
        System.out.println("******* Creadores: Juan Chango -
Andrés Márquez *******");
        System.out.println("");
        System.out.println("Inicializando Servidor Raspberry
Pi...");

        ServerSocket ss = new ServerSocket(7778);

        commandEngine = new CommandEngineServer();
        commandEngine.init();


        while (true) {
            System.out.println("Escuchando en puerto:7778...");
            Socket s = ss.accept();
            System.out.println("Cliente encontrado");
            RasPiAVVIServer t = new RasPiAVVIServer(s);
            t.start();
        }

    }

}
```

2. Clase CommandEngine.java:

```
package raspiavviserver;
```

```
public class CommandEngineServer {

   static RasPiGPIOEngine gpio;

   public CommandEngineServer() {
   }

   public void init() {
        gpio = new RasPiGPIOEngine();
        gpio.init();

   }


   public String runCommand(String command) throws
InterruptedException{
        String state = null;

        int code = Integer.parseInt(command);
        int typeCmd = code/10000;
        int chCmd = (code= code-typeCmd*10000)/100;
        int stateCmd = (code= code-chCmd*100)/1;

        //System.out.println(code);
        //System.out.println(typeCmd);
        //System.out.println(chCmd);
        //System.out.println(stateCmd);

        switch (typeCmd) {
            case 1:
                gpio.setPin(chCmd, stateCmd);
                state = "Ready";
                break;

        }
        return state;
   }

}
```

3. Clase RasPiGPIOEngine.java:

```
package raspiavviserver;

import com.pi4j.io.gpio.GpioController;
```

```
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.*;

public class RasPiGPIOEngine {

   private GpioController gpio;
   private GpioPinDigitalOutput pin1;
   private GpioPinDigitalOutput pin2;
   private GpioPinDigitalOutput pin3;
   private GpioPinDigitalOutput pin4;
   private GpioPinDigitalOutput pin5;


   public RasPiGPIOEngine() {

   }


   public void init() {
       gpio = GpioFactory.getInstance();

       pin1 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_01,
PinState.LOW);
       //pin1.setShutdownOptions(true, PinState.LOW);

       pin2 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_02,
PinState.LOW);
       //pin2.setShutdownOptions(true, PinState.LOW);

       pin3 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_03,
PinState.LOW);
       //pin3.setShutdownOptions(true, PinState.LOW);

       pin4 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_04,
PinState.LOW);
       //pin4.setShutdownOptions(true, PinState.LOW);

       pin5 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_05,
PinState.LOW);
       //pin5.setShutdownOptions(true, PinState.LOW);
   }
```

```
   public void setHighPin(int pinNumber) throws
InterruptedException {

       switch (pinNumber) {
           case 1:
               pin1.high();
               break;
           case 2:
               pin2.high();
               break;
           case 3:
               pin3.high();
               break;
           case 4:
               pin4.high();
               break;
           case 5:
               pin5.high();
               break;
       }


   }

   public void setLowPin(int pinNumber) throws
InterruptedException {

       switch (pinNumber) {
           case 1:
               pin1.low();
               break;
           case 2:
               pin2.low();
               break;
           case 3:
               pin3.low();
               break;
           case 4:
               pin4.low();
               break;
           case 5:
               pin5.low();
               break;
       }

   }
```

```
   public void togglePin(int pinNumber) throws
InterruptedException {

       switch (pinNumber) {
           case 1:
               pin1.toggle();
               break;
           case 2:
               pin2.toggle();
               break;
           case 3:
               pin3.toggle();
               break;
           case 4:
               pin4.toggle();
               break;
           case 5:
               pin5.toggle();
               break;
       }

   }


    public void setPin(int pinNumber, int state) throws
InterruptedException {

        if (state > 0){
            setHighPin(pinNumber);
        }else{
            setLowPin(pinNumber);
        }

    }

}
```

ANEXO 12

CÓDIGO FUENTE SERVIDOR CONVERTIDOR DE VOZ A TEXTO

1. Clase Principal SpeechServer.java:

```
package speechserver;

import java.net.*;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.sound.sampled.AudioFormat;

public class SpeechServer extends Thread {

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    public Socket id;
    public static SpeechEngine speechEngine;
    public static TCPIPTools netTools;
    public AudioFormat format = new AudioFormat(16000.0f, 16,
1, true, false);


    public SpeechServer(Socket s) {
        id = s;

    }

    @Override
    public void run() {

        try {
            byte data[] = netTools.rxBytesFromClient(id);

            InputStream stream =
speechEngine.formatWaveByteInputTCPIP(data, format);

            System.out.println("--->Reconociendo");

            String result = null;
            result = speechEngine.speech2str(stream);

            netTools.txHypothesis(id, result);

            System.out.println("--->Resultado:" + result);
```

```
            System.out.println("--->FIN");

            //speechEngine.saveWaveFileFromByteArray(data,
format,
"/Users/Juan/Desktop/AudiosCMU/audioSpeechServer.wav");


        } catch (IOException ex) {
            Logger.getLogger(SpeechServer.class.getName()).log(
Level.SEVERE, null, ex);
        }

    }

    public static void main(String[] args) throws IOException,
InterruptedException {
        // TODO code application logic here

        System.out.println("***************  Servidor
Speech2Text  ***************");
        System.out.println("***************  Reconocimiento de
voz  ***************");
        System.out.println("**********************  UDLA  ****
********************");
        System.out.println("***************  ING SONIDO
ACUSTICA  ***************");
        System.out.println("******* Creadores: Juan Chango -
Andrés Márquez *******");
        System.out.println("");
        System.out.println("Inicializando Sphinx...");

        speechEngine = new SpeechEngine();
        netTools = new TCPIPTools();
        speechEngine.init();

        ServerSocket ss = new ServerSocket(7777);

        while (true) {
            System.out.println("Escuchando en puerto:7777...");
            Socket s = ss.accept();
            System.out.println("Cliente encontrado");
            SpeechServer t = new SpeechServer(s);
            t.start();
        }

        //ss.close();
```

```
    }

}
```

2. Clase SpeechEngine.java:

```
package speechserver;

import edu.cmu.sphinx.frontend.util.StreamDataSource;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;

public class SpeechEngine {

    private URL configURL ; //=
RxCMU.class.getResource("config.xml");

    private ConfigurationManager cm ;//= new
ConfigurationManager(configURL);
    private Recognizer recognizer ;//= (Recognizer)
cm.lookup("recognizer");

    /* allocate the resource necessary for the recognizer */
    private AudioFileFormat.Type fileType =
AudioFileFormat.Type.WAVE;

    private StreamDataSource dataSource ;//= (StreamDataSource)
cm.lookup("DataSource");

    private File wavFile;

    private Result result;

    public SpeechEngine() {
```

```
        configURL =
speechserver.SpeechServer.class.getResource("config.xml");
        cm = new ConfigurationManager(configURL);
        recognizer = (Recognizer) cm.lookup("recognizer");
        dataSource = (StreamDataSource)
cm.lookup("DataSource");

    }


    public void init() {

        recognizer.allocate();

    }

    public void stop() {

        recognizer.deallocate();

    }


    public String speech2str (InputStream stream){
        dataSource.setInputStream(stream);
        String resultText = null;

        while ((result = recognizer.recognize()) != null) {

            resultText = result.getBestResultNoFiller();
            //System.out.println(resultText);
        }

        return resultText;
    }


    public InputStream formatWaveByteInputTCPIP(byte[] data,
AudioFormat format) throws IOException{
        int len = data.length;
        InputStream stream = new ByteArrayInputStream(data);
        AudioInputStream ais = new AudioInputStream(stream,
format, len/ format.getFrameSize());
        ByteArrayOutputStream out1 = new
ByteArrayOutputStream(len);
        AudioSystem.write(ais, fileType, out1);
        data= out1.toByteArray();
```

```
        stream = new ByteArrayInputStream(data);
        return stream;
    }


    public void saveWaveFileFromByteArray(byte[] data,
AudioFormat format,String fileAddress) throws IOException{
        wavFile = new File(fileAddress);
        int len = data.length;
        InputStream stream = new ByteArrayInputStream(data);
        AudioInputStream ais = new AudioInputStream(stream,
format, len/ format.getFrameSize());
        AudioSystem.write(ais, fileType, wavFile);
    }
}
```

3. Clase TCPIPTools.java:

```
package speechserver;

import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.net.Socket;

public class TCPIPTools {

    private byte data[];
    private int len;
    private InputStream in;
    private DataInputStream dis;

    private PrintWriter out;

    public TCPIPTools() {
    }

    public byte[] rxBytesFromClient(Socket id) throws
IOException {
        in = id.getInputStream();
        dis = new DataInputStream(in);
        len = dis.readInt();
        data = new byte[len];
        dis.readFully(data);
        return data;
    }
```

```
   public void txHypothesis(Socket id, String hypothesis)
throws IOException {
       out = new PrintWriter(id.getOutputStream(), true);
       out.println(hypothesis);
   }
}
```

4. Archivo config.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<config>

 <property name="logLevel" value="WARNING"/>

 <property name="absoluteBeamWidth"  value="20000"/>
 <property name="relativeBeamWidth"  value="1e-60"/>
 <property name="absoluteWordBeamWidth"  value="200"/>
 <property name="relativeWordBeamWidth"  value="1e-40"/>

 <property name="wordInsertionProbability" value="0.1"/>
 <property name="silenceInsertionProbability" value="0.1"/>
 <property name="fillerInsertionProbability" value="1e-2"/>

 <property name="phoneticLookaheadWindow" value="5"/>
 <property name="phoneticLookaheadWeight" value="6"/>
 <property name="acousticLookaheadWeight" value="1.7"/>
 <property name="phoneticBeam" value="1e-12"/>

 <property name="oogProbability" value="1e-30"/>
 <property name="oogLoopProbability" value="1e-10"/>

 <property name="languageWeight" value="8.0"/>

 <component name="recognizer"
type="edu.cmu.sphinx.recognizer.Recognizer">
   <property name="decoder" value="decoder"/>
   <propertylist name="monitors">
     <item>speedTracker</item>
     <item>memoryTracker</item>
   </propertylist>
 </component>

 <component name="decoder"
type="edu.cmu.sphinx.decoder.Decoder">
```

```
   <property name="searchManager"
value="wordPruningLookaheadSearchManager"/>
 </component>

 <component name="alignerSearchManager"
   type="edu.cmu.sphinx.decoder.search.SimpleBreadthFirstSearc
hManager">
   <property name="linguist" value="alignerLinguist"/>
   <property name="pruner" value="trivialPruner"/>
   <property name="scorer" value="trivialScorer"/>
   <property name="activeListFactory" value="activeList"/>
 </component>

 <component name="allphoneSearchManager"
   type="edu.cmu.sphinx.decoder.search.SimpleBreadthFirstSearc
hManager">
   <property name="linguist" value="allphoneLinguist"/>
   <property name="pruner" value="trivialPruner"/>
   <property name="scorer" value="trivialScorer"/>
   <property name="activeListFactory" value="activeList"/>
 </component>

 <component name="simpleSearchManager"
   type="edu.cmu.sphinx.decoder.search.SimpleBreadthFirstSearc
hManager">
   <property name="linguist" value="flatLinguist"/>
   <property name="pruner" value="trivialPruner"/>
   <property name="scorer" value="trivialScorer"/>
   <property name="activeListFactory" value="activeList"/>
 </component>

 <component name="wordPruningSearchManager"
   type="edu.cmu.sphinx.decoder.search.WordPruningBreadthFirst
SearchManager">
   <property name="linguist" value="lexTreeLinguist"/>
   <property name="pruner" value="trivialPruner"/>
   <property name="scorer" value="trivialScorer"/>
   <property name="activeListManager"
value="activeListManager"/>

   <property name="growSkipInterval" value="0"/>

   <property name="buildWordLattice" value="true"/>
   <property name="keepAllTokens" value="true"/>

   <property name="acousticLookaheadFrames"
value="${acousticLookaheadWeight}"/>
```

```
   <property name="relativeBeamWidth"
value="${relativeBeamWidth}"/>
 </component>

 <component name="wordPruningLookaheadSearchManager"
   type="edu.cmu.sphinx.decoder.search.WordPruningBreadthFirst
LookaheadSearchManager">
   <property name="linguist" value="lexTreeLinguist"/>
   <property name="fastmatchLinguist"
value="allphoneLinguist"/>
   <property name="loader" value="acousticModelLoader"/>
   <property name="pruner" value="trivialPruner"/>
   <property name="scorer" value="trivialScorer"/>
   <property name="activeListManager"
value="activeListManager"/>
   <property name="fastmatchActiveListFactory"
value="fastmatchActiveList"/>

   <property name="growSkipInterval" value="0"/>
   <property name="buildWordLattice" value="true"/>
   <property name="keepAllTokens" value="true"/>

   <property name="lookaheadWindow"
value="${phoneticLookaheadWindow}"/>
   <property name="lookaheadPenaltyWeight"
value="${phoneticLookaheadWeight}"/>
   <property name="acousticLookaheadFrames"
value="${acousticLookaheadWeight}"/>
   <property name="relativeBeamWidth"
value="${relativeBeamWidth}"/>
 </component>

 <component name="activeList"
   type="edu.cmu.sphinx.decoder.search.PartitionActiveListFact
ory">
   <property name="absoluteBeamWidth"
value="${absoluteBeamWidth}"/>
   <property name="relativeBeamWidth"
value="${relativeBeamWidth}"/>
 </component>

 <component name="fastmatchActiveList"
   type="edu.cmu.sphinx.decoder.search.PartitionActiveListFact
ory">
   <property name="absoluteBeamWidth"
value="${absoluteBeamWidth}"/>
```

```
   <property name="relativeBeamWidth"
value="${phoneticBeam}"/>
 </component>

 <component name="activeListManager"
   type="edu.cmu.sphinx.decoder.search.SimpleActiveListManager
">
   <propertylist name="activeListFactories">
     <item>standardActiveListFactory</item>
     <item>wordActiveListFactory</item>
     <item>wordActiveListFactory</item>
     <item>standardActiveListFactory</item>
     <item>standardActiveListFactory</item>
     <item>standardActiveListFactory</item>
   </propertylist>
 </component>

 <component name="standardActiveListFactory"
   type="edu.cmu.sphinx.decoder.search.PartitionActiveListFact
ory">
   <property name="absoluteBeamWidth"
value="${absoluteBeamWidth}"/>
   <property name="relativeBeamWidth"
value="${relativeBeamWidth}"/>
 </component>

 <component name="wordActiveListFactory"
   type="edu.cmu.sphinx.decoder.search.PartitionActiveListFact
ory">
   <property name="absoluteBeamWidth"
value="${absoluteWordBeamWidth}"/>
   <property name="relativeBeamWidth"
value="${relativeWordBeamWidth}"/>
 </component>

 <component name="trivialPruner"
   type="edu.cmu.sphinx.decoder.pruner.SimplePruner"/>

 <component name="trivialScorer"
   type="edu.cmu.sphinx.decoder.scorer.SimpleAcousticScorer">
   <property name="frontend" value="liveFrontEnd"/>
 </component>

 <component name="alignerLinguist"
   type="edu.cmu.sphinx.linguist.flat.FlatLinguist">
   <property name="grammar" value="alignerGrammar"/>
   <property name="acousticModel" value="acousticModel"/>
```

```
    <property name="wordInsertionProbability"
      value="${wordInsertionProbability}"/>
    <property name="silenceInsertionProbability"
      value="${silenceInsertionProbability}"/>
    <property name="languageWeight" value="${languageWeight}"/>
    <property name="unitManager" value="unitManager"/>
  </component>

  <component name="allphoneLinguist"
    type="edu.cmu.sphinx.linguist.allphone.AllphoneLinguist">
    <property name="acousticModel" value="acousticModel"/>
    <property name="useContextDependentPhones" value="false"/>
    <property name="phoneInsertionProbability" value="0.05"/>
  </component>

  <component name="flatLinguist"
    type="edu.cmu.sphinx.linguist.flat.FlatLinguist">
    <property name="grammar" value="jsgfGrammar"/>
    <property name="acousticModel" value="acousticModel"/>
    <property name="wordInsertionProbability"
      value="${wordInsertionProbability}"/>
    <property name="silenceInsertionProbability"
      value="${silenceInsertionProbability}"/>
    <property name="languageWeight" value="${languageWeight}"/>
    <property name="unitManager" value="unitManager"/>

    <property name="addOutOfGrammarBranch" value="true"/>
    <property name="outOfGrammarProbability"
value="${oogProbability}"/>
    <property name="phoneInsertionProbability"
value="${oogLoopProbability}"/>
    <property name="phoneLoopAcousticModel"
value="acousticModel"/>
  </component>

  <component name="lexTreeLinguist"
    type="edu.cmu.sphinx.linguist.lextree.LexTreeLinguist">
    <property name="acousticModel" value="acousticModel"/>
    <property name="languageModel" value="simpleNGramModel"/>
    <property name="dictionary" value="dictionary"/>
    <property name="addFillerWords" value="true"/>
    <property name="generateUnitStates" value="false"/>
    <property name="wantUnigramSmear" value="true"/>
    <property name="unigramSmearWeight" value="1"/>
    <property name="wordInsertionProbability"
value="${wordInsertionProbability}"/>
```

```
   <property name="silenceInsertionProbability"
value="${silenceInsertionProbability}"/>
   <property name="fillerInsertionProbability"
value="${fillerInsertionProbability}"/>
   <property name="languageWeight" value="${languageWeight}"/>
   <property name="unitManager" value="unitManager"/>
 </component>

 <component name="trieNgramModel"
   type="edu.cmu.sphinx.linguist.language.ngram.trie.NgramTrie
Model">
   <property name="location" value=""/>
   <property name="dictionary" value="dictionary"/>
   <property name="maxDepth" value="3"/>
 </component>

 <component name="simpleNGramModel"
   type="edu.cmu.sphinx.linguist.language.ngram.SimpleNGramMod
el">
   <property name="location" value="resource:/modelserver/es-
20k.lm"/>
   <property name="dictionary" value="dictionary"/>
   <property name="maxDepth" value="3"/>
   <property name="unigramWeight" value=".7"/>
 </component>

 <component name="largeTrigramModel"
   type="edu.cmu.sphinx.linguist.language.ngram.large.LargeTri
gramModel">
   <property name="location" value=""/>
   <property name="unigramWeight" value=".5"/>
   <property name="maxDepth" value="3"/>
   <property name="dictionary" value="dictionary"/>
 </component>

 <component name="dynamicTrigramModel"
   type="edu.cmu.sphinx.linguist.language.ngram.DynamicTrigram
Model">
   <property name="dictionary" value="dictionary"/>
   <property name="maxDepth" value="3"/>
   <property name="unigramWeight" value=".7"/>
 </component>

 <component name="alignerGrammar"
   type="edu.cmu.sphinx.linguist.language.grammar.AlignerGramm
ar">
   <property name="dictionary" value="dictionary"/>
```

```
    <property name="addSilenceWords" value="true"/>
  </component>

  <component name="jsgfGrammar"
type="edu.cmu.sphinx.jsgf.JSGFGrammar">
    <property name="dictionary" value="dictionary"/>
    <property name="grammarLocation" value=""/>
    <property name="grammarName" value=""/>
    <property name="addSilenceWords" value="true"/>
  </component>

  <component name="grXmlGrammar"
type="edu.cmu.sphinx.jsgf.GrXMLGrammar">
    <property name="dictionary" value="dictionary"/>
    <property name="grammarLocation" value=""/>
    <property name="grammarName" value=""/>
    <property name="addSilenceWords" value="true"/>
  </component>

  <component name="dictionary"
    type="edu.cmu.sphinx.linguist.dictionary.TextDictionary">
    <property name="dictionaryPath"
value="resource:/modelserver/es.dict"/>
    <property name="fillerPath"
value="resource:/modelserver/esp-ec-adapt/noisedict"/>
    <property name="unitManager" value="unitManager"/>
  </component>

  <component name="acousticModel"
    type="edu.cmu.sphinx.linguist.acoustic.tiedstate.TiedStateA
cousticModel">
    <property name="loader" value="acousticModelLoader"/>
    <property name="unitManager" value="unitManager"/>
  </component>

  <component name="acousticModelLoader"
    type="edu.cmu.sphinx.linguist.acoustic.tiedstate.Sphinx3Loa
der">
    <property name="unitManager" value="unitManager"/>
    <property name="location" value="resource:/modelserver/esp-
ec-adapt"/>
    <property name="topGaussiansNum" value="4"/>
  </component>

  <component name="unitManager"
    type="edu.cmu.sphinx.linguist.acoustic.UnitManager"/>
```

```
 <component name="liveFrontEnd"
type="edu.cmu.sphinx.frontend.FrontEnd">
   <propertylist name="pipeline">
     <item>DataSource </item>
     <item>dataBlocker </item>
     <item>speechClassifier </item>
     <item>speechMarker </item>
     <item>preemphasizer </item>
     <item>windower </item>
     <item>fft </item>
     <item>autoCepstrum </item>
     <item>liveCMN </item>
     <item>featureExtraction </item>
     <item>featureTransform </item>
   </propertylist>
 </component>

 <component name="DataSource"
            type="edu.cmu.sphinx.frontend.util.StreamDataSourc
e">
     <property name="sampleRate" value="16000" />
     <property name="bytesPerRead" value="3200" />
     <property name="bitsPerSample" value="16" />
     <property name="bigEndian" value="true" />
     <property name="signedData" value="true" />
  </component>



 <component name="dataBlocker"
type="edu.cmu.sphinx.frontend.DataBlocker"/>

 <component name="dataDumper"
type="edu.cmu.sphinx.frontend.util.DataDumper"/>

 <component name="speechClassifier"
   type="edu.cmu.sphinx.frontend.endpoint.SpeechClassifier">
   <property name="threshold" value="13" />
 </component>

 <component name="speechMarker"
   type="edu.cmu.sphinx.frontend.endpoint.SpeechMarker" >
 </component>

 <component name="preemphasizer"
   type="edu.cmu.sphinx.frontend.filter.Preemphasizer"/>
```

```
 <component name="windower"
   type="edu.cmu.sphinx.frontend.window.RaisedCosineWindower">
 </component>

 <component name="fft"
   type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTran
sform">
 </component>

 <component name="autoCepstrum"
   type="edu.cmu.sphinx.frontend.AutoCepstrum">
   <property name="loader" value="acousticModelLoader"/>
 </component>

 <component name="batchCMN"
   type="edu.cmu.sphinx.frontend.feature.BatchCMN"/>

 <component name="liveCMN"
   type="edu.cmu.sphinx.frontend.feature.LiveCMN"/>

 <component name="featureExtraction"
   type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtracto
r"/>

 <component name="featureTransform"
   type="edu.cmu.sphinx.frontend.feature.FeatureTransform">
   <property name="loader" value="acousticModelLoader"/>
 </component>

 <component name="speedTracker"
   type="edu.cmu.sphinx.instrumentation.SpeedTracker">
   <property name="showTimers" value="true"/>
   <property name="frontend" value="liveFrontEnd"/>
   <property name="recognizer" value="recognizer"/>
 </component>

 <component name="memoryTracker"
   type="edu.cmu.sphinx.instrumentation.MemoryTracker">
   <property name="recognizer" value="recognizer"/>
 </component>

</config>
```

ANEXO 13

CÓDIGO FUENTE SERVICIO WEB

1. Clase principal SpeechToText.java:

```
package com;

import java.io.IOException;
import java.util.Base64;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;


@WebService(serviceName = "SpeechToText")
public class SpeechToText {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "speech2text")
    public String speech2text(@WebParam(name = "audioData")
String audioDataString64) throws IOException {
        //TODO write your implementation code here:


        byte[] audioData =
Base64.getDecoder().decode(audioDataString64);


        ClientSpeechEngine clientSpeechEngine = new
ClientSpeechEngine("MAC", "localhost", 7777);
        clientSpeechEngine.init();
        clientSpeechEngine.txBytesToServer(audioData);

        String res = clientSpeechEngine.rxHypothesis();
        System.out.println(res);

        return res;
    }
}
```

2. Clase ClientSpeechEngine.java:

```
package com;

import java.io.BufferedReader;
```

```
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;

public class ClientSpeechEngine {

   private String clientName;//= "MAC";
   private String ipAddress;
   private int port;

   private OutputStream out;
   private DataOutputStream dos;

   private BufferedReader in;

   private Socket s;//= new Socket("192.168.2.68", 7777);

   public ClientSpeechEngine(String clientName, String
ipAddress, int port) {
        this.clientName = clientName;
        this.ipAddress = ipAddress;
        this.port = port;
   }

   public void init() throws IOException {
        s = new Socket(ipAddress, port);
   }

   public void txBytesToServer(byte[] data) throws IOException
{
        out = s.getOutputStream();
        dos = new DataOutputStream(out);

        dos.writeInt(data.length);
        dos.write(data, 0, data.length);
   }

   public String rxHypothesis() throws IOException {
        BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()));
        return in.readLine();

   }

}
```

ANEXO 14

ARCHIVO WSDL DEL SERVICIO WEB

```
This XML file does not appear to have any style information
associated with it. The document tree is shown below.

 <!--

   Published by JAX-WS RI (http://jax-ws.java.net). RI's
   version is Metro/2.3.2-b608 (trunk-7979; 2015-01-
   21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-
   API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-
   b141020.1521 svn-revision#unknown.

 -->

 <!--

   Generated by JAX-WS RI (http://jax-ws.java.net). RI's
   version is Metro/2.3.2-b608 (trunk-7979; 2015-01-
   21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-
   API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-
   b141020.1521 svn-revision#unknown.

 -->

 <definitions xmlns:wsu="http://docs.oasis-
 open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
 1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
 xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"x
 mlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tns="http://com/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://schemas.xmlsoap.org/wsdl/"targetNamespace="htt
 p://com/" name="SpeechToText">

   <types>

     <xsd:schema>

       <xsd:import namespace="http://com/"
       schemaLocation="http://localhost:8080/Speech2TextWebSer
       vice/SpeechToText?xsd=1"/>

     </xsd:schema>

   </types>
```

```
<message name="speech2text">

  <part name="parameters" element="tns:speech2text"/>

</message>

<message name="speech2textResponse">

  <part name="parameters"
  element="tns:speech2textResponse"/>

</message>

<message name="IOException">

  <part name="fault" element="tns:IOException"/>

</message>

<portType name="SpeechToText">

  <operation name="speech2text">

    <input
    wsam:Action="http://com/SpeechToText/speech2textRequest
    " message="tns:speech2text"/>

    <output
    wsam:Action="http://com/SpeechToText/speech2textRespons
    e" message="tns:speech2textResponse"/>

    <fault message="tns:IOException" name="IOException"
    wsam:Action="http://com/SpeechToText/speech2text/Fault/
    IOException"/>

  </operation>

</portType>

<binding name="SpeechToTextPortBinding"
type="tns:SpeechToText">

  <soap:binding
  transport="http://schemas.xmlsoap.org/soap/http"
  style="document"/>

  <operation name="speech2text">

    <soap:operation soapAction=""/>
```

```
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="IOException">
        <soap:fault name="IOException" use="literal"/>
      </fault>
    </operation>
  </binding>
  <service name="SpeechToText">
    <port name="SpeechToTextPort"
    binding="tns:SpeechToTextPortBinding">
      <soap:address
      location="http://localhost:8080/Speech2TextWebService/S
      peechToText"/>
    </port>
  </service>
</definitions>
```