PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

RECONOCIMIENTO DE PALABRAS DEL IDIOMA ESPAÑOL CON JULIUS

FRANCISCA CAMILA MEDINA ALDERETE

NICOLE FERNANDA PIÑA RUSSEL

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO DE EJECUCIÓN EN INFORMÁTICA

DICIEMBRE, 2012

Pontificia Universidad Católica de Valparaíso
Facultad de Ingeniería
Escuela de Ingeniería Informática

RECONOCIMIENTO DE PALABRAS DEL IDIOMA ESPAÑOL CON JULIUS

FRANCISCA CAMILA MEDINA ALDERETE

NICOLE FERNANDA PIÑA RUSSEL

Profesor Guía: **Iván Mercado Bermúdez**

Profesor Co-referente: **Cristian Alexandru Rusu**

Carrera: **Ingeniería de Ejecución en Informática**

Diciembre, 2012

Después de muchos años de estudio y trabajo, agradezco a mis padres y hermanos por haberme apoyado y ayudado en todo este proceso de aprendizaje y crecimiento personal. Nicole.

ÍNDICE

Resumen

En el mundo de hoy las comunicaciones se expanden a un ritmo vertiginoso; sin embargo se mantiene la barrera del idioma en muchas partes del mundo. Para romper dicho impedimento la comunidad Language Grid provee servicios para facilitar la comunicación multicultural. Entre los servicios disponibles se encuentra Julius, un sistema de reconocimiento de voz, el cual es considerado una herramienta útil, ya que el reconocimiento de voz facilita la interacción entre usuario y sistema. Con la creación de un modelo de lenguaje en español Julius será capaz de reconocer ese idioma. Para ello es necesario utilizar las herramientas de desarrollo de modelo de lenguaje (HTK), investigar la transcripción del idioma español y la implementación de pruebas para mejorar la precisión del reconocimiento de palabras en los modelos creados.

Palabras claves: Language Grid, Julius, HTK, español, reconocimiento de habla.

Abstract

Nowadays, communications media are growing drastically; however, the language barrier remains high worldwide. In order to overcome such impairment the Language Grid community, which was developed at the Kyoto University, grants all sorts of language services to ease multicultural communication. Among the available services, the voice recognition system, Julius, is the only one on this field among the available services. This recognition system is considered a useful tool, since the voice recognition makes the human-computer interaction easier. Along with spanish language model, Julius would be able to recognize such language, and in order to achieve that, the toolkit for language model development (HTK) is required. A research of spanish transcription must also be performed. Afterwards, several tests are carried out so the accuracy for word recognition in language models can be improved.

Tags: Language Grid, Julius, HTK, spanish, speech reconigtion .

Glosario de Términos

Back-translation	:	Traducir al idioma original un texto previamente traducido, sin tener el documento fuente.
Baum-Welch	:	Algoritmo que permite estimar parámetros escondidos HMM, también es conocido como algoritmo fordward-backward.
Grafema	:	Letra, número u otro símbolo lingüístico que representa un fonema.
Paráfrasis	:	Explicación del contenido de un discurso para aclararlo en todos sus aspectos (muy utilizado en la traducción de un idioma a otro).
Toolkit	:	Equipo de herramientas, grupo de programas y rutinas que se utilizan como base para la programación de un nuevo sistema.
Viterbi	:	Algoritmo que soluciona algunos de los problemas de HMM.

Lista de Abreviaturas o Siglas

ASALE : Asociación de Academias de la Lengua Española.
EM : Expectación-Maximización.
GMM : Gaussian Mixture Models.
GPL : GNU General Public License.
HMM : Hidden Markov Model (Modelo Oculto de Márkov).
HTK : Hidden Markov Model Toolkit .
IPA : International Phonetic Alphabet (Alfabeto Fonético Internacional, AFI) .
LG : Language Grid.
LVCSR : Large Vocabulary Continuous Speech Recognition (Reconocimiento de voz continua de vocabulario amplio).
NECTEC : Thailand National Electronics and Computer Tecnology Center.
NICT : National Institute of Information and Comunications Technology (Instituto Nacional de Información y Comunicaciones Tecnológicas).
PUCV : Pontificia Universidad Católica de Valparaíso.
RAE : Real Academia Española.
SAMPA : Speech Assessment Methods Phonetic Alphabet
SRE : Software Recongition Engine (motor de reconocimiento de habla).

LISTA DE FIGURAS

LISTA DE TABLAS

1 INTRODUCCIÓN

El proyecto se enfocará en el desarrollo de un modelo de lenguaje, para colaborar con Language Grid, el cual es una plataforma de servicios multilingüe online que permite registros e intercambios de servicios de idioma de manera fácil, tales como diccionarios online, corporaciones bilingües, traducciones automáticas, entre otros.

Language Grid, es un software que proporciona una infraestructura para servicios multilingüe, fue desarrollada y lanzada como un software de código abierto por el proyecto de Language Grid del Instituto Nacional de Información y Tecnología de las Comunicaciones (NICT), desde abril de 2006. Actualmente está siendo usado por institutos de investigación, organizaciones sin fines de lucro y universidades, tales como la PUCV, en donde existe convenio.

Esta plataforma combina los recursos del idioma de un usuario y traductores automáticos para producir traducciones de alta calidad, que están configuradas para el campo que lo requiera. En base a lo anterior, se pretende colaborar con la comunidad trabajando en algún proyecto existente como Julius, que es un software de reconocimiento de voz en tiempo real. Actualmente Julius está en japonés, por lo cual se pretende llevar el proyecto al idioma Español.

Para lograr el reconocimiento de voz al español, se estudia y desarrolla un modelo de lenguaje del idioma español, en donde es necesario el análisis del funcionamiento de Julius (motor de reconocimiento de voz) y sus herramientas (para el desarrollo de modelo de lenguaje), además del estudio de la fonética y fonología del español (para la coherencia del idioma en la creación de los modelos).

Finalmente, el propósito de este documento es registrar el desarrollo del modelo de lenguaje junto con la explicación de las diversas herramientas a utilizar, para cooperar con futuros proyectos relacionados con el reconocimiento de voz.

2 DEFINICIÓN DE OBJETIVOS

Se presentarán a continuación el objetivo general y los objetivos específicos que se pretenden cumplir en el proyecto:

2.1 OBJETIVO GENERAL

Crear modelo acústico y lingüístico de un subconjunto de palabras del idioma español para el software Julius, con el fin de colaborar con dicho proyecto y Language Grid.

2.2 OBJETIVOS ESPECÍFICOS

- Estudiar el funcionamiento de Julius, junto a los proyectos y herramientas relacionados (Ej: Voxforge, HTK, etc).
- Desarrollar modelo lingüístico y modelo acústico del español, basado en el estudio de la trascripción fonética del idioma.
- Validar los modelos creados, mediante pruebas, para ajustar la precisión de reconocimiento de palabras.

3 PLAN DE TRABAJO

Para el desarrollo del proyecto y con el propósito de cumplir con los objetivos planteados, se definió cuales son los puntos importantes que se deben cumplir. Se comenzó con la recopilación de información necesaria, el análisis y diseño de la implementación del proyecto, junto con los planes, factibilidad y riesgos. Luego, utilizando la información recopilada y el diseño realizado, se desarrollo la implementación del proyecto, junto con el diseño del plan de pruebas.

Las tareas realizadas necesarias para cumplir con los objetivos especificados, está representada en la siguiente tabla:

Tabla 3.1 Tareas realizadas durante el desarrollo del proyecto.

Tarea	**Descripción**
Recopilación de información.	Representa todo el período en que se recopiló toda la información necesaria sobre de Languaje Grid, Julius, HTK, algoritmos, transcripción del idioma, etc.
Investigación sobre LG y sus servidores.	Se investigó para comprender el funcionamiento de la comunidad y los distintos servidores que ésta utiliza.
Análisis del proyecto a realizar (Julius al español).	Julius fue el servicio seleccionado de Languaje Grid, el cual está en idioma japonés. Se analizó su funcionamiento para lograr el reconocimiento de palabras en español.
Prueba de herramientas (Julius, HTK, modelado, etc.)	Para el desarrollo del modelo de lenguaje, se utilizaron herramientas como HTK, para modelar el idioma.
Diseño del proyecto.	Se especificaron las palabras que se utilizarán en el desarrollo del modelo de lenguaje, junto con la transcripción de cada palabra utilizada en los modelos.
Creación de modelo acústico y lingüístico.	Representa a todo el desarrollo del modelo de lenguaje, es decir, la grabación de voces, transcripción de palabras, aplicaciones de algoritmos e iteraciones con las herramientas de HTK, etc.
Estudio de la implementación de Julius en otras áreas, software, proyectos, etc.	Se investigó en que otros sistemas fue utilizado el software Julius, para su desarrollo.
Desarrollo e implementación del plan de prueba.	Son las pruebas realizadas a cada uno de los modelos desarrollados para identificar posibles problemas, precisión de palabras, etc.

Documentación de la implementación.	Representa a la documentación del proyecto, junto con las especificaciones para el desarrollo de modelo de lenguaje, planteamiento de posibles problemas y soluciones, pruebas, etc.

Respecto a la tabla 3.1 se puede señalar que se logró efectuar pruebas del HTK, como la realización de un sencillo modelo lingüístico y acústico del idioma español. En el desarrollo de este modelo se destaca el estudio de fonética del español y la transcripción fonológica de las palabras.

4 ESTADO DEL ARTE

Durante muchos años la "barrera del idioma" siempre ha separado unos países de otros, separando a las personas por la imposibilidad de comunicación, siendo también una de las causas de malentendidos y disputas. A pesar de que se considera que actualmente se está en una época de globalización, la barrera del idioma sigue siendo aún muy alta.

Además, los servicios de idiomas a menudo no son muy accesibles, por los derechos de propiedad intelectual y los precios, además no están estandarizados por lo que es muy complejo el trabajar con varios de estos servicios [2].

El profesor Toru Ishida del Departamento de Social de Informática en la Universidad de Kyoto, tiene como objetivo cruzar esta barrera mediante "Language Grid", desarrollando una poderosa herramienta para la formación de una sociedad multicultural, en el que cualquier usuario perteneciente a diversas culturas puedan trabajar en armonía participando fácilmente en los intercambios internacionales y culturales entre distintas naciones.[3]

4.1 LANGUAGE GRID (LG)

Language Grid es un software libre (sin fines de lucro) que provee una infraestructura de servicios multilingües, combinando recursos lingüísticos con traductores para producir traducciones de alta calidad, configuradas según el área o contexto. Fue creado como un software libre por el Proyecto Language Grid del Instituto Nacional de Información y Comunicaciones Tecnológicas en abril de 2006, luego en diciembre de 2007, el Departamento Social de Informática de la Universidad de Kyoto empezó a operar LG con propósitos de investigación.

La "Asociación Language Grid" está conformada por universidades, centros de investigación y organizaciones sin fines de lucro (como la asociación que tiene con Google desde el año 2009 y la Fundación Wikimedia con Wikipedia), las cuales pueden operar y utilizar los recursos lingüísticos que proveen distintos usuarios de todo el mundo (lo que implica una colaboración intercultural). Entre los colaboradores está la Pontificia Universidad Católica de Valparaíso (representada por el profesor Cristian Rusu), con el grupo llamado 'Use CV'-HCI.

Los stakeholders (grupo interés) que interactúan con LG, se clasifican en tres tipos:

- **Operador de Language Grid:** Gestiona los usuarios de LG y controla recursos de computación y lenguaje.

- **Proveedor del recurso de lenguaje:** Registra recursos lingüísticos a LG tales como traductores, analizadores morfológicos, diccionarios especializados, entre otros.

- **Usuario del servicio de lenguaje:** Organizaciones o individuos que usan los recursos lingüísticos y computacionales (en este tipo de stakeholders pertenece Use CV-HCI). [4]

4.1.1 ROL DE LANGUAGE GRID

En la actualidad LG está siendo operado por la Universidad de Kyoto con propósitos de investigación y sin fines de lucro. Las universidades y centros de investigación que participan en LG, proveen recursos del lenguaje tales como diccionarios y traductores automáticos, entre otros.

Cuando un usuario (organizaciones sin fines de lucro, escuelas y otros sectores) provee recursos lingüísticos, se pueden especificar derechos de autor e información de licencias en los perfiles de los recursos, como también restringir el acceso de otros usuarios. Gracias a estos aportes a los recursos lingüísticos, se logra avanzar en romper las barreras del lenguaje, al cubrir diversos campos de investigación para el apoyo y resolución de problemas en distintas regiones del planeta (como el control de desastres naturales, educación, cuidado médico, etc.).

Resumiendo lo mencionado anteriormente, LG es una plataforma que permite a los usuarios compartir recursos lingüísticos y apoyar actividades de colaboración intercultural, mejorando la usabilidad y accesibilidad de estos. Convierte los recursos lingüísticos en servicios lingüísticos, en donde los usuarios deben instalar y considerar el mantenimiento de los recursos, mientras que los proveedores deben lidiar con los problemas de propiedad intelectual. Además, estos recursos son acoplados en servicios web con interfaces estándar, que pueden ser utilizados por cualquier usuario en cualquier parte del mundo. Adicionalmente, con estos servicios se pueden componer diversos servicios lingüísticos que pueden ser compartidos en LG (mientras los proveedores mantengan el control y protección de su propiedad intelectual).

4.1.2 SERVICIOS LANGUAGE GRID

Los participantes del Language Grid pueden registrar sus propios diccionarios en una base de datos, los cuales pueden ser compartidos por todos los usuarios, mejorando la precisión de la traducción. La enciclopedia en línea Wikipedia, está escrita y modificada por diferentes usuarios, y se ha convertido en una enorme base de datos de conocimiento. Del mismo modo, la traducción de diccionarios puede evolucionar rápidamente a través de

"la fuerza combinada" o "inteligencia colectiva". El código fuente del software es abierto, y por lo tanto, los usuarios pueden desarrollar nuevas herramientas de forma independiente. [3]

Otro de los servicios es el de traducción automática, que difiere con los sistemas de traducción convencionales de internet en que los usuarios pueden mejorar la calidad de traducción utilizando LG. Existe la posibilidad de que los usuarios puedan usar los textos paralelos registrados mientras traducen; como por ejemplo, cuando un usuario ingresa una oración, aparecerán automáticamente ejemplos con significados similares, si encuentra una oración similar al significado que buscaba, mediante textos paralelos puede obtener una traducción precisa, en caso contrario, se puede ejecutar la traducción automática.

En las traducciones de textos en paralelo, generalmente el idioma eje para las traducciones es el inglés; es decir, el proceso para traducir de japonés a portugués será el siguiente: traducción de japonés a inglés y luego de inglés a portugués. Lo anterior es lo denominado "traducción multi-salto", que forma parte de un servicio más de LG.

Todos los servicios de LG están subdivididos en las siguientes 4 capas:

- **Infraestructura de red P2P:** La infraestructura combina múltiples servidores de internet para cumplir con las peticiones del usuario (combinando recursos lingüísticos de internet). Estos pueden añadir sus servidores a la red P2P, mediante el Service Manager de LG, en donde los usuarios tienen acceso a estadísticas de uso de los recursos lingüísticos y servidores que ellos proveen.

- **Recursos Lingüísticos:** Se proveen varios recursos lingüísticos para servicios web con interfaz estandarizada. También los usuarios pueden añadir nuevos recursos. En LG están disponibles los siguientes recursos lingüísticos:

    - Traductores Automáticos: (japonés, chino) (japonés, coreano) (japonés, inglés) (inglés, alemán) (inglés, español) (inglés, francés) (inglés, italiano) (inglés, portugués).

    - Analizadores Morfológicos: japonés, chino, coreano, inglés, alemán, español, francés, italiano, holandés, ruso, búlgaro.

    - Diccionarios Bilingües: términos de ciencia (life science) (japonés, ingles), términos de manejo de desastres (japonés, chino, coreano, inglés, francés, español), términos académicos (japonés, ingles). [4]

- **Servicios Lingüísticos:** Los recursos lingüísticos de pueden combinar con el flujo de trabajo del servicio web (Web Service Workflows). Estos servicios además son manejados por el Nodo Central de LG, en donde proveen búsquedas y composición de servicios lingüísticos. Los usuarios también pueden añadir nuevos servicios. Los siguientes tipos de servicios están disponibles en LG:

- Servicios Atómicos: un servicio web corresponde a cada recurso lingüístico. Por ejemplo, diccionarios bilingües, textos paralelos o analizadores morfológicos.

- Servicios Compuestos: un servicio avanzado descrito por un flujo de trabajo para componer servicios atómicos. Por ejemplo, traducciones especializadas o back-translation. [4]

- **Herramientas de colaboración intercultural:** Se desarrollan usando los servicios lingüísticos explicados anteriormente. El NICT y algunas universidades han desarrollado nuevas herramientas (como el Toolbox de LG), asimismo, las herramientas existentes han sido llevadas a múltiples idiomas. El Toolbox de LG tiene las siguientes funciones:

  - Traducción de texto: los usuarios pueden traducir contenidos de manera multilingüe y confirmar el resultado de la traducción con back-translation. También pueden dar click en cualquier parte de la oración para seleccionarla y ver sus resultados con back-translation. Pueden publicar sus mensajes usando sus lenguajes nativos y modificar los resultados de la traducción automática para mejorar localidad del contenido multilingüe.

  - Creación de diccionario multilingüe: los miembros de la comunidad pueden crear sus diccionarios multilingües y combinarlos con traductores automáticos para mejorar la calidad de traducción. Además de los diccionarios multilingües, también pueden crear textos paralelos multilingües, preguntas y respuestas, glosarios, etc.

  - Operación federada: NECTEC lanzó el centro operacional de Bangkok en enero del 2011. El centro cubre Asia sudeste, el centro de operación de Kyoto empezó la operación federara de LG con el centro de Bangkok. [4]

Actualmente Language Grid, utilizando todos los servicios mencionados anteriormente, ha desarrollado y seguirá dando apoyo a estudiantes, universidades, centros educaciones y de investigación, etc., para expandir el conocimiento de los diversos idiomas existentes. En muchos países ya se está utilizando LG para situaciones cotidianas, como en el caso de Japón, que utiliza el sistema de traducción automática para ayudar a extranjeros a rellenar el registro de pacientes del hospital de Kyoto (de la Universidad de Kyoto). [3]

Entre todos los proyectos disponibles en LG, se seleccionó uno que no estuviera en español. Se optó por el reconocimiento de voz, siendo el software Julius el único servidor que se dedica a este ámbito.

4.2 JULIUS

Durante años de estudio sobre LVCSR, se encontró la necesidad de optar por una plataforma común para trabajar en esta área. Para ello se consideraron algunos factores claves para la creación de una plataforma común, estos son: rendimiento, modularidad y disponibilidad, además de velocidad y precisión. Aun así, para realizar la plataforma se tiene que utilizar y comprender muchos tipos de modelos de cualquier mezcla, estados o fonemas.

También el motor de reconocimiento de voz y los modelos necesitan ser abiertos, para que cualquier usuario pueda construir sistemas de reconocimiento usando sus propios modelos, modificándolos y agregando elementos al motor existente. A pesar de que hay muchos sistemas de reconocimiento de voz disponibles (para propósitos de investigación o como productos comercializados), no satisfacen la necesidad de crear nuevos modelos o modificar lo existente.

Debido a esto, se desarrolló un motor de reconocimiento de voz continua de vocabulario amplio de dos pasos, en tiempo real y de código abierto llamado "Julius", el cual está enfocado para ser utilizados por investigadores y desarrolladores.

Julius es un Software decodificador de reconocimiento de voz LVCSR, desarrollado por el laboratorio de Kawahara en la Universidad de Kyoto, está afiliado a Language Grid como un servidor de reconocimiento de voz (pertenece a uno de sus servicios atómicos). Puede usar modelos de 2 y 3-gramas y se ejecuta casi en tiempo real en la mayoría de los equipos.

Este soporta varios formatos como el ARPA (desarrollado para los N-gramas), posee los modelos HMM y N-gramas, también trae incorporado varias técnicas de búsqueda tales como la poda de gauss, búsqueda insertada, selección de Gauss, lo que produce búsquedas más precisas.

Julius ha sido desarrollado y mantenido como parte del toolkit de software libre para LVCSR japonés [4] desde 1997, el trabajo en general aun continúa en proceso de alcanzar un óptimo reconocimiento de habla continuo, en Japón. El software está disponible de forma gratuita y con los códigos de fuente incluidos en el sitio web del mismo. [6]

A continuación, algunas imágenes del software Julius ejecutado en la terminal de Ubuntu:

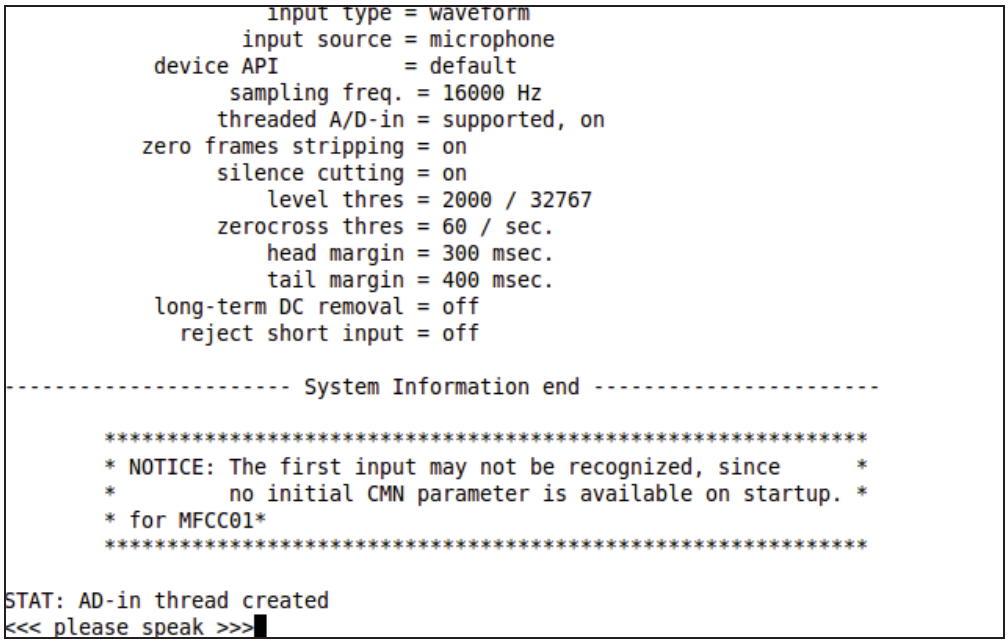

Figura 4-1 Vista desde la terminal de Ubuntu del software Julius.

En la imagen 4.1 se puede visualizar parte del inicio de la interfaz de Julius en la terminal de Ubuntu (se muestra la licencia, entre otras informaciones importantes del sistema), finaliza con un mensaje pidiéndole al usuario que hable.

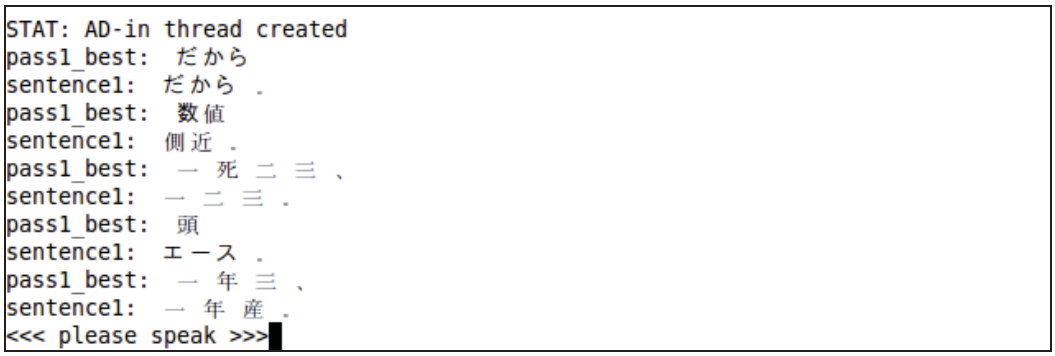

Figura 4-2 Vista desde la terminal de Ubuntu del funcionamiento del software Julius.

En el momento en que el micrófono del computador detecta la voz del usuario, Julius realiza la búsqueda de las palabras (utilizando los algoritmos que serán explicados más adelante), imprimiéndolas en la terminal. Entre las palabras mencionadas por el usuario están “ichi”, “ni”, “san”, que están en “sentence1” (las cuales fueron detectadas correctamente por el software). El programa finaliza con “Ctrl + c” o cerrando la terminal.

5 MARCO TEÓRICO

Para comprender mejor LG, es necesario aclarar el concepto de servicios de idiomas. Luego se definirán los modelos, algoritmos, técnicas de búsqueda, herramientas y conceptos más influyentes en el Software Julius.

5.1 SERVICIOS DE IDIOMAS

Los servicios de idiomas consisten en recursos lingüísticos, que incluyen diccionarios, glosarios, entre otros, y funciones de procesamiento de lenguaje, incluyendo el análisis morfológico, traducción y paráfrasis. Generalmente no son muy accesibles, por los derechos de propiedad intelectual y los precios [2], y como ya fue mencionado LG pretende solucionar esta problemática, al ser una plataforma con diversos servicios sin fines de lucro, los cuales pueden ser utilizados por diversos usuarios de distintas ubicaciones geográficas.

5.2 HIDDEN MARKOV MODEL (HMM, MODELO OCULTO DE MARKOV)

Es un modelo estadístico que contiene un conjunto finito de estados (o cadena de estados), en donde se pueden ver los resultados, pero no los estados de esta cadena (de ahí el nombre Modelo Oculto de Markov) esto permite estimar probabilidades de eventos no observados.

Para el reconocimiento de voz, los datos observados son las señales acústicas y las palabras(o fonemas, silabas) son los parámetros ocultos. Se representa cada palabra (o fonema, sílaba) del vocabulario del reconocedor con un modelo generativo (calculado en la fase de entrenamiento) y luego se calcula la probabilidad de la palabra a reconocer, cuando ha sido producida por cada uno de los modelos que están registrados en el reconocedor. Para ello, se asume que durante la pronunciación de una palabra, el aparato fonador puede adoptar solo un número finito de configuraciones aleatorias (o estados), y que desde cada uno de estos estados se producen uno o varios vectores de observación (puntos de la plantilla), cuyas características espectrales de cada fragmento de señal dependen del estado activo de cada instante, y la evolución del espectro de la señal durante la pronunciación de una palabra depende de la ley de transición de estados [8].

En la siguiente figura se puede visualizar una representación gráfica del modelo:

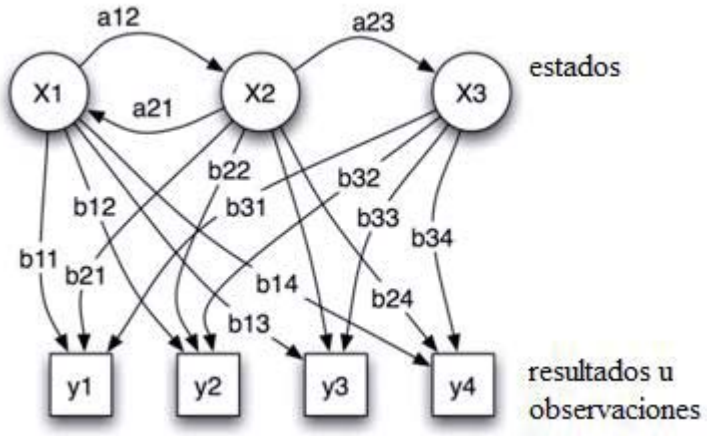

Figura 5-1 Representación gráfica de HMM

En la figura 5.1 se ve que los estados "x" (x1, x2 y x3) son los parámetros ocultos, "a" (a12, a21, a23) son las probabilidades de transición de estado, es decir la probabilidad de que el usuario emita otro sonido (fonema). Mientras que "y" (y1, y2, y3, y4) son las posibles observaciones de los resultados finales a los que podría llegar el modelo, y "b" (b11, b12, b21, etc.) son todas las probabilidades de ese resultado.

5.3 MODELO DE N-GRAMA

Un modelo de N-grama es un tipo de modelado probabilístico de lenguaje donde el N-grama es una secuencia continua de n objetos, los cuales pueden ser sílabas, fonemas, palabras, etc. dependiendo de cómo se separen los objetos, serán llamados unigramas (un objeto), bigramas (2 objetos), trigramas(3 objetos) o n-gramas(n objetos). [9]

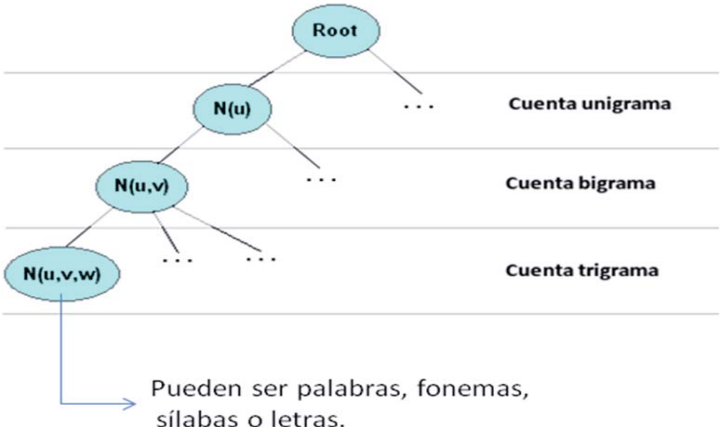

Figura 5-2 Modelo N-grama

La figura 5.2 resume cómo funciona el modelo de n-grama. Por ejemplo: La oración "el perro toma agua" se dividiría como un unigrama de la siguiente forma: "el", "perro", "toma", "agua", en cambio si fuera un bigrama seria: "el perro", "perro toma", "toma agua". También el objeto podría ser una sílaba donde en un unigrama seria "pe", "rro" o una letra donde seria "p", "e", "r", "r", "o".

5.4 ÁRBOL LÉXICO

Es una estructura de datos que permite compartir nodos (o estados) entre las palabras en el léxico. Los árboles consisten de palabras, fonemas o letras.

En un sistema, los árboles sólo se pueden usar para reconocimiento de palabras aisladas. Para incrementar la robustez de los árboles, se necesita alguna forma para distinguir el vocabulario. Dada esa descripción, se podría reconocer una palabra incluso en un contexto de habla continua. Por ejemplo, si una persona dijera "nací en febrero", entonces se podría reconocer la palabra "febrero" dentro del contexto del resto de las palabras. Para esto, se utilizan los siguientes métodos para modelado de voz de fondo: modelado de "cualquier" palabra y modelado de fonemas basura.

Una palabra cualquiera es un tipo especial de palabra donde los estados o fonemas que describen la palabra pueden hacer la transición a cualquiera de los estados o fonemas de la palabra. Como cualquier estado o fonema puede pasar a cualquier otro estado o fonema dentro del modelo "cualquiera", este modelo en particular será muy efectivo lidiando con el silencio de fondo, el silbido de fondo que normalmente suena como una "s" y el zumbido de fondo que se modela usando los n fonemas. Con el modelo de basura también se incrementa mucho la robustez. Esto significa que el resultado del fondo es más que la salida en "silencio" para el estimador de probabilidad de fonemas. Dos modelos de basura simples implementados son la mediana de las probabilidades superiores de los fonemas N-variados y el máximo del conjunto de fonemas probablemente tiene altas probabilidades en ruido. El modelo de basura ideal tendría mejor resultado en ruido y

eventos de habla fuera de vocabulario que cualquiera de las palabras de vocabulario. [10]

5.5 OTROS ELEMENTOS DE JULIUS

Para la formación de un LVCSR, se necesita los siguientes componentes (los cuales son necesarios para que el software Julius reconozca los fonemas del idioma):

- **Modelo Lingüístico:** Estos modelos contienen una lista muy larga de palabras y sus probabilidades de ocurrencia en una secuencia dada. Se utilizan en aplicaciones de dictado. Las gramáticas son archivos mucho más pequeños que contienen conjuntos de combinaciones de palabras predeterminadas. Cada palabra en un modelo de lenguaje o gramática, la cual tiene una lista asociada de fonemas para reconocer los distintos sonidos de cada palabra.[11]

- **Modelo Acústico:** Son modelos estadísticos que estiman la probabilidad de que cierto fonema sea pronunciado en un segmento de audio. Los modelos se entrenan por varias horas en una conversación pregrabada. [12]

- **Decodificador:** Es el software que toma el sonido emitido por el usuario y busca en el modelo acústico alguna coincidencia. Si la encuentra, el decodificador determina el fonema correspondiente al sonido. Este mantiene el rastro de los fonemas que coinciden hasta que llega una pausa de parte del usuario. Luego busca en el modelo lingüístico o archivo de gramática, una serie equivalente de fonemas. Si hay alguna coincidencia muestra el texto correspondiente a la palabra o frase. [11]

5.6 HTK (HIDDEN MARKOV MODEL TOOLKIT)

Es un kit de herramientas para construir y manipular HMM. Se utiliza principalmente en investigaciones de reconocimiento de voz. HTK consiste en un conjunto de librerías y herramientas disponibles en lenguaje C, las cuales otorgan facilidades sofisticadas para análisis de voz, entrenamiento de HMM, análisis de pruebas y resultados. [12]

5.7 PROYECTO VOXFORGE

Es un proyecto que fue creado para recolectar voz transcrita en motores de reconocimiento de voz (SRE) de código libre tales como Julius y HTK, se encarga de categorizar y tener disponibles todos los archivos de audio subidos y modelos acústicos

bajo la licencia GPL, esto debido a que estos recursos sólo están disponibles para software comerciales, y no para código abierto. [11] En este caso, es importante Voxforge ya que logró desarrollar el modelo lingüístico y acústico de Julius en inglés. En la siguiente figura se puede visualizar el funcionamiento del software en la terminal de Ubuntu:

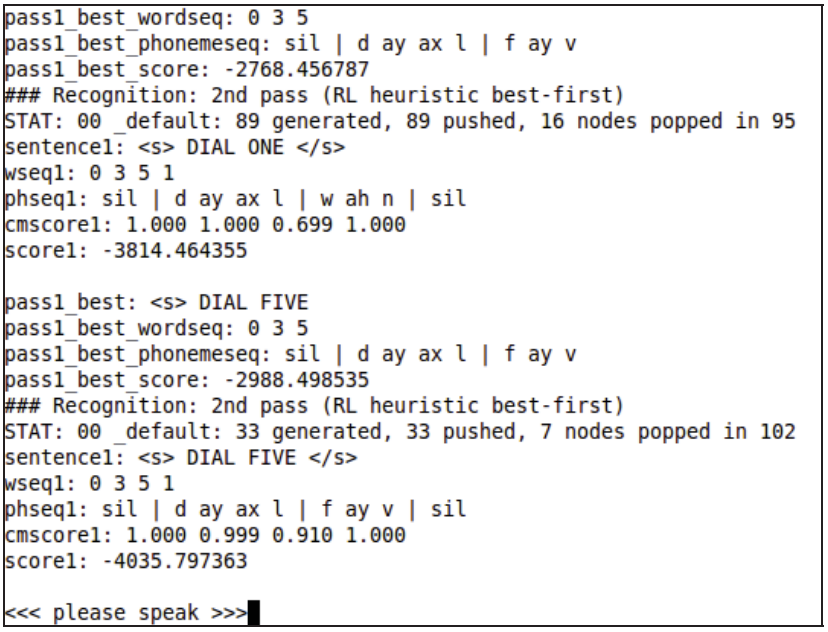

Figura 5-3 Vista desde la terminal de Ubuntu, del funcionamiento del proyecto Voxforge

El proyecto aún se mantiene en desarrollo, y puede solo detectar algunas palabras en inglés (es aún muy limitado). En la imagen anterior el usuario dijo “Dial Five” y fue detectado correctamente (solo detecta aquellas palabras que están en su registro).

5.8 ALGORITMOS DE BÚSQUEDA UTILIZADOS EN JULIUS

5.8.1 GAUSSIAN PRUNING

Es un algoritmo de búsqueda que se basa en GMM (Gaussian Mixture Models), esto es una función de probabilidad paramétrica representada como una sumatoria de los componentes. Los parámetros son estimados de los datos de entrenamiento usando el algoritmo iterativo de Expectación-Maximización (EM) el cual es una técnica usada para determinar los parámetros de una mezcla con un número a priori de componentes, es una forma particular de implementar la estimación máxima de probabilidad a el problema.

En resumen, Gaussian Pruning va podando el árbol léxico con el criterio de que cada vez que encuentre un estado que no supera cierto porcentaje de compatibilidad, lo retira.

5.8.2 BEAM SEARCH

Es un algoritmo de búsqueda que explora un grafo (en este caso un árbol) resultado de la optimización de la búsqueda mejor-primera (best-first search), que reduce requerimientos de memoria. La best-first es una búsqueda de grafos que ordena todas las soluciones (o estados) parciales de acuerdo a alguna heurística que intenta predecir que tan cerca está una solución parcial de la solución final (estado meta), pero en beam search sólo se mantiene un número predeterminado de mejores soluciones parciales como candidatos.

Beam search usa la búsqueda del ancho-primero (breadth-first search, usa un recorrido parecido a pre-orden, pero con la posibilidad de revisar los vecinos). Para construir el árbol de búsqueda, en cada nivel del árbol, genera todos los sucesores de los estados al nivel actual, ordenándolos incrementalmente a costo heurístico. Sin embargo, sólo almacena un número predeterminado de mejores estados en cada nivel llamado ancho de haz (beam width), luego solamente esos estados se siguen recorriendo, mientras más grande el ancho, menos estados se podarán. Si el ancho es infinito, ningún estado se podará y la búsqueda será idéntica a la del ancho-primero (breadth-first). El ancho conecta la memoria requerida para realizar la búsqueda. Como un estado meta podría ser podado, beam search no garantiza que el algoritmo encontrará la solución, ni tampoco que se encuentre la mejor solución. El beam puede ser arreglado o variable, un ejemplo podría ser que el ancho sea variable y parta con el ancho al mínimo, si no se encuentra solución el beam se agranda y se repite el proceso. [14]

5.9 TRANSCRIPCIÓN FONÉTICA Y FONOLÓGICA.

La fonética y la fonología son disciplinas de la lingüística encargadas de estudiar los sonidos del lenguaje. En la primera existe una base meramente acústica, mientras que en la segunda se tiende a considerar la imagen mental de lo que percibimos. Por ejemplo, el grafema “n” aparece en las siguientes palabras: enterrar, lenguaje, encima, sana. Se cree en estos casos que “n” siempre es igual, pero en realidad es lo contrario, porque esa “n” es una percepción mental llamada fonema y que es transcrita entre barras (/n/), donde sus sonidos reales, que son llamados fonos, son diferentes en cada una de las palabras del ejemplo. Por lo tanto, la fonología estudia los fonemas (que están clasificados en vocales y consonantes), mientras la fonética estudia los sonidos reales (fonemas y fonos). [15]

En base a los estudios de la fonología y fonética, nace la transcripción fonética la cual es un sistema de símbolos que representan el habla humana. Destaca el sistema AFI (Alfabeto Fonético Internacional, en inglés IPA International Phonetic Alphabet) por ser el sistema de transcripción fonética más antiguo y usado. Tiene como objetivo estandarizar y regularizar los distintos fonemas de todos los idiomas. En el idioma español, aún no está en desarrollo, debido a las distintas variaciones en las diversas regiones de Latinoamérica y España en que se utiliza este idioma, incluso en Chile el español (acá llamado castellano) varía a lo largo de su territorio, por esa razón, el AFI del idioma español está realizado en base a un español estándar (creado por la RAE y ASALE) y se mantiene en una constante

actualización (en el apéndice A esta detallado el actual alfabeto del AFI del idioma español y en el apéndice B esta detallado el alfabeto SAMPA) [16].

Dada la situación mencionada anteriormente, fue utilizado en el desarrollo del modelo de lenguaje un "español neutro", en el cual se utilizaron algunas de las reglas y simbologías planteadas en el AFI y SAMPA (que está basado en AFI).

En el anexo A, se encuentran las tablas de los fonemas utilizados para el desarrollo de un modelo acústico y lingüístico. Se muestra como muchos de los grafemas del idioma español son reducidos a un solo fonema que lo representa (por ejemplo: el fonema /k/, es utilizado para ca, co, cu, k- (a, e, i, o, u), qu- (e, i), esto es debido a la similitud de sus sonidos; aun así, existen distinciones en la articulación de cada uno de estos fonemas, esta es debido a que el sonido cambia dependiendo del movimiento fisiológico de cada persona (parte del estudio de la fonología), variando por los elementos, puntos y modo de articulación de las palabras, estas características de los fonemas están detallas en la columna "clasificación de fonemas" de la primera tabla; mientras en la segunda tabla, la transcripción fonética utilizada da mayor énfasis a los sonidos de la vocales más acentuadas.

Basado en los resultados obtenidos en la prueba de las "pruebas realizadas al finalizar un modelo" (explicado más adelante en el plan de pruebas), se identificó que el tener más fonemas en un modelo no hacia mejor la detección de la voz, pues utilizando la transcripción de las tablas del Anexo A y B, se obtuvieron resultados similares, por lo que se utilizaron para el modelo final la transcripción de las tablas del Anexo B (por ser menos fonemas). Lo que hace que una transcripción sea mejor que otra, es la clasificación de los fonemas involucrados, si la transcripción no está correctamente hecha, pueden surgir confusiones en el reconocimiento de estos (esta clasificación esta dada en el archivo tree.hed, explicado más adelante en el desarrollo del modelo de lenguaje).

6 ESTUDIO DE LA FACTIBILIDAD

Este estudio es necesario para confirmar la posibilidad de realizar el proyecto basado en el aspecto técnico, legal y económico. Una vez aprobado el estudio de la factibilidad, se dio paso a continuar con el desarrollo del proyecto.

6.1 ESTUDIO DE LA FACTIBILIDAD TÉCNICA

La Factibilidad Técnica consiste en realizar una evaluación de la tecnología existente y estuvo destinado a recolectar información sobre los componentes técnicos que posee la organización y la posibilidad de hacer uso de los mismos en el desarrollo e implementación del sistema propuesto.

6.1.1 HARDWARE

Cualquier computador con el mínimo de requerimiento de los sistemas actuales, que posea una entrada de micrófono externo o integrado de 16 bit.

6.1.2 SOFTWARE

- Sistema Operativo: Linux, Windows, Mac OS, FreeBSD, Sun Solaris.
- Julius.
- HTK.

6.2 ESTUDIO DE LA FACTIBILIDAD LEGAL

La Factibilidad Legal consiste en realizar una evaluación legal de las herramientas y software que se utilizarán en la implementación del proyecto. Son detallados en la siguiente lista:

1. Los “Términos y Condiciones” de uso de Julius (Licencia Julius [27]), se cumplirán en su totalidad.
2. Ubuntu 10.04 está compuesto de múltiples software normalmente distribuidos bajo una licencia libre o de código abierto (GPL [16]).
3. HTK, licenciado por la Universidad de Cambrige, se obtiene gratuitamente desde el sitio web del proyecto (Licencia HTK [17]).
4. Uce-CV (grupo de la Pontifica Universidad Católica de Valparaíso) afiliado al Centro de Operación de Kyoto de Language Grid [4]. Además Language Grid es un software gratuito, sin fines de lucro [18].

5. Para el modelado en UML, se utilizará IBM Rational, el cual posee convenio de uso por la escuela de Ingeniería Informática de la PUCV.

6.3 ESTUDIO DE LA FACTIBILIDAD ECONÓMICA

Dada las características de este proyecto, no fue necesario realizar un estudio de factibilidad económica, ya que no se dio la necesidad de comprar algún software, hardware o herramientas involucradas en el proyecto, por ser gratuitas y por poseer el hardware necesario desde antes.

Como fue mencionado anteriormente, al cumplir con los factores técnicos, legales y económicos para el desarrollo del proyecto, se puede afirmar que el proyecto es factible. Fueron utilizados equipos que poseen una RAM de 3072 MB y 2048 MB, con sistemas operativos Ubuntu 10.04., características que se encuentran dentro de las factibilidades mencionadas anteriormente.

Aun así, para reafirmar la factibilidad del proyecto, se analizaron todos los riesgos posibles y por haber en el transcurso del proyecto, estos son explicados a continuación.

7 ANÁLISIS DE RIESGO, PLANES DE MITIGACIÓN Y CONTINGENCIA.

Los riesgos pueden ocurrir en el transcurso del ciclo de vida del proyecto, para tratarlos es necesario sean analizados (identificar su impacto y su probabilidad de ocurrencia) y en base a esto, realizar un Plan de Mitigación (prevención del riesgo) y un Plan de contingencia (que hacer cuando ocurre el riesgo). Para ello se analizaron los posibles riesgos del proyecto, estos son especificados en la siguiente tabla:

Tabla 7.1 Tabla de Riesgo, Planes de Mitigación y Contingencia del proyecto.

Riesgo	**Impacto**	**Probabilidad**	**Plan de Mitigación**	**Plan de Contingencia**
Integrante del grupo se enferma o se ausenta en etapas críticas del proyecto	Crítico	Media	Que el integrante afectado informe lo más rápido posible de su problema.	Se intentara contactar con el integrante ausente para que cumpla con sus tareas de algún otro modo (de no poder asistir presencialmente a las juntas de trabajo), como conectarse por internet, subir parte de sus tareas asignadas a Dropbox, etc.
Se ha subestimado el tiempo para desarrollo de proyecto 1.	Crítico	Medio	Se debería tener todo listo por lo menos antes de los días de entrega, para lograr lo anterior, se cumplirá la Carta Gantt.	Identificar lo que falta del proyecto, y dedicar más tiempo de trabajo para poder finalizar a tiempo.
No se alcanza a implementar todos los requerimientos.	Crítico	Medio	Trabajar los requerimientos como "prototipos", e identificar claramente aquellos que son requerimientos importantes del sistema y aquellos que no son tan importantes.	Al trabajar los requerimientos como prototipos, eliminarlos no implica un impacto tan grande, aun así, si se elimina o no se alcanza a implementar un requerimiento de gran importancia, se deberá analizar la situación actual y definir alguna solución.
Julius se convierte en un software comercial.	Crítico	Muy baja	Julius existe como software libre desde 1991 hasta el día de hoy, es muy baja la posibilidad de que se vuelva comercial ya que uno de sus objetivos es integrar y colaborar con	Dado que aun es período de la primera entrega del proyecto 1, existe la posibilidad de cambiar de software y enfoque del proyecto, aun así, esto requerirá una nueva investigación de las nuevas

			más países al proyecto.	herramientas y software, lo que significaría una gran pérdida de tiempo.
Bajo compromiso de un integrante del proyecto.	Crítico	Baja	Tener bien identificadas las tareas de los integrantes, realizar reuniones frecuentes para estar bien informados y comprometidos con el proyecto.	Conversar para recordar y estimular a aquel integrante que no esté cumpliendo con sus tareas. Es tarea de ambos integrantes llevar un flujo consistente de trabajo para evitar retrasos.
LG cierra el acceso al grupo de trabajo PUCV.	Crítico	Baja	Uno de los objetivos de este proyecto es colaborar con la comunidad LG, como grupo de trabajo PUCV, para ello, se confirmó antes de iniciar el trabajo que se tiene acceso al sitio.	Si llegara a ocurrir, se continuaría con el proyecto Julius, sin considerar la colaboración con LG.
El proyecto actual se hace imposible de implementar (ya sea por falta de tiempo o conocimiento).	Crítico	Media	Se tendrá definido (no formalmente en el informe) un Plan B con otra posibilidad de proyecto. También se intentará verificar que es factible en su totalidad el desarrollo del proyecto actual (y de ser posible también el del Plan B).	Se analizará y desarrollará el Plan B.

8 MODELO DE PROCESO UTILIZADO

El modelo que más se adecúa al proyecto es el "modelo incremental", ya que posee ventajas como:

1. Iteraciones que permiten comenzar de cero, si algún punto no funciona correctamente o es evaluado negativamente.
2. Gracias a que no es secuencial, se puede retroceder en las fases para arreglar problemas.
3. Le da gran énfasis a la etapa de diseño invirtiendo mucho tiempo, por lo que es de suma importancia.
4. Al combinar las partes estructuradas del modelo de cascada con las iteraciones del modelo evolutivo, se empieza con un análisis y diseño bien definidos, para luego arreglar las tareas rápidamente con los incrementos.
5. La parte de análisis y diseño es susceptible a posibles cambios, y al encontrarse en etapa temprana, esos cambios no son tan engorrosos de implementar.
6. Se reduce trabajo en la parte de desarrollo.
7. Permite priorizar los requisitos del sistema (dando menos prioridades a aquellos requerimientos que están menos claros).
8. Al implementar el primer incremento, se puede continuar implementando otros requerimientos y todos van evolucionando.

También como el proyecto esta subdividido en una parte como investigación de Language Grid y del software Julius (implementación del idioma español en Julius), fue necesario optar por requerimientos que podían ser modificados durante el avance del proyecto, sin tener que partir desde cero.

9 METODOLOGÍA DE DESARROLLO DE JULIUS Y HTK

La metodología en que fueron desarrollados Julius y la herramienta HTK se basa en la programación estructurada.

El software Julius está desarrollado y escrito en lenguaje C, el cual es un lenguaje propio de la programación estructurada. La lectura de los archivos (cuando se ejecutan los algoritmos de búsqueda durante el reconocimiento de voz) es realizada de forma secuencial. También cada uno de los archivos que conforman el software Julius posee funciones que definen el sistema.

La herramienta HTK también fue desarrollada y escrita en lenguaje C (junto con algunos scripts en Perl). Durante el desarrollo de los modelos acústico y lingüístico, las herramientas del HTK leen cada archivo de los modelos para ejecutar el algoritmo de HMM, de forma secuencial.

Al ser este proyecto muy particular, no se definió una metodología para su desarrollo, ya que el fin principal es desarrollar un modelo de lenguaje, el cual es considerado como una configuración al ya existente software Julius.

10 ESPECIFICACIÓN DE REQUERIMIENTOS

A continuación se especificarán los requerimientos para la realización del desarrollo del proyecto, tomando en primera instancia los requerimientos funcionales y no funcionales del proyecto. También se asignó una descripción y estado actual para cada uno de estos:

10.1 REQUERIMIENTOS FUNCIONALES

Tabla 10.1 Requerimientos funcionales, con su respectivo estado actual.

ID	Descripción	Estado
1.	**Customizar Julius para que reconozca el idioma español.** Julius es un software japonés que reconoce la voz. Se pretende que Julius pueda detectar el idioma español.	Aprobado
2.	**Generar modelos (acústico y lingüístico) de un conjunto de palabras del lenguaje español.** Se realizará un modelo acústico en español, es cual estará disponible para que cualquier usuario o desarrollador que requiera un reconocedor de voz lo pueda adquirir	Aprobado

10.2 REQUERIMIENTOS NO FUNCIONALES

Tabla 10.2 Requerimientos no funcionales con su respectivo estado actual

ID	Descripción	Estado
1.	**Desarrollado en Linux (Ubuntu)**	Aprobado
2.	**Desarrollar modelos con las herramientas del HTK.**	Aprobado

11 ANÁLISIS, DISEÑO

Para cumplir con el objetivo principal del proyecto, se pensó en el escenario en cual el usuario se enfrentaría al sistema. Este caso es modelado de la siguiente manera:

11.1 Diagrama Caso de Uso General

Un diagrama de Caso de uso general es preciso para explicar y representar gráficamente de una forma amplia el funcionamiento del sistema, junto a como interactúa el usuario con este.

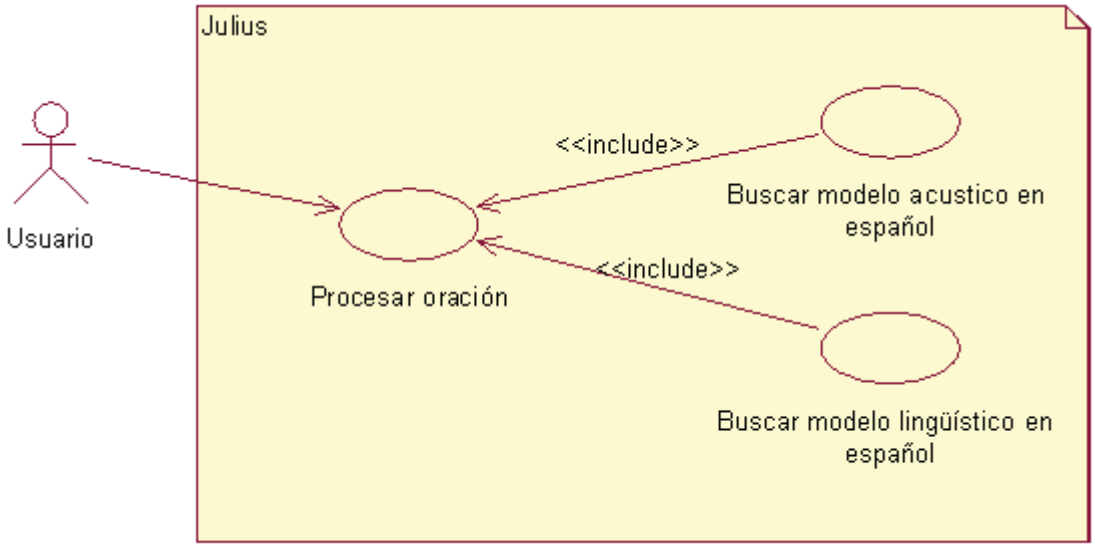

Figura 11-1 Diagrama de caso de uso general del proyecto.

Con los requerimientos ya especificados, se diseñó un diagrama de caso de uso del proyecto. Este diagrama representa como el usuario interactúa con el software Julius. Para el reconocimiento de la voz del usuario.

El usuario expresar alguna oración, la cual es procesada por el software, claro que antes la oración es procesada realizando la búsqueda del modelo acústico y lingüístico del idioma español de forma automática. A continuación se detalla el como Julius realiza este proceso.

11.2 Análisis del reconocimiento de voz en Julius

Una vez aclarada la intervención del usuario con el software, se necesita saber cómo funciona y está compuesto el software Julius, para identificar cuáles son las funciones principales, cuales son los procesos que se realizan para que la voz sea reconocida y así comprender mejor el comportamiento de los algoritmos, técnicas de búsqueda y herramientas que son usadas (como HMM, Beam Search, HTK, etc). Lo anterior es resumido en el siguiente diagrama:

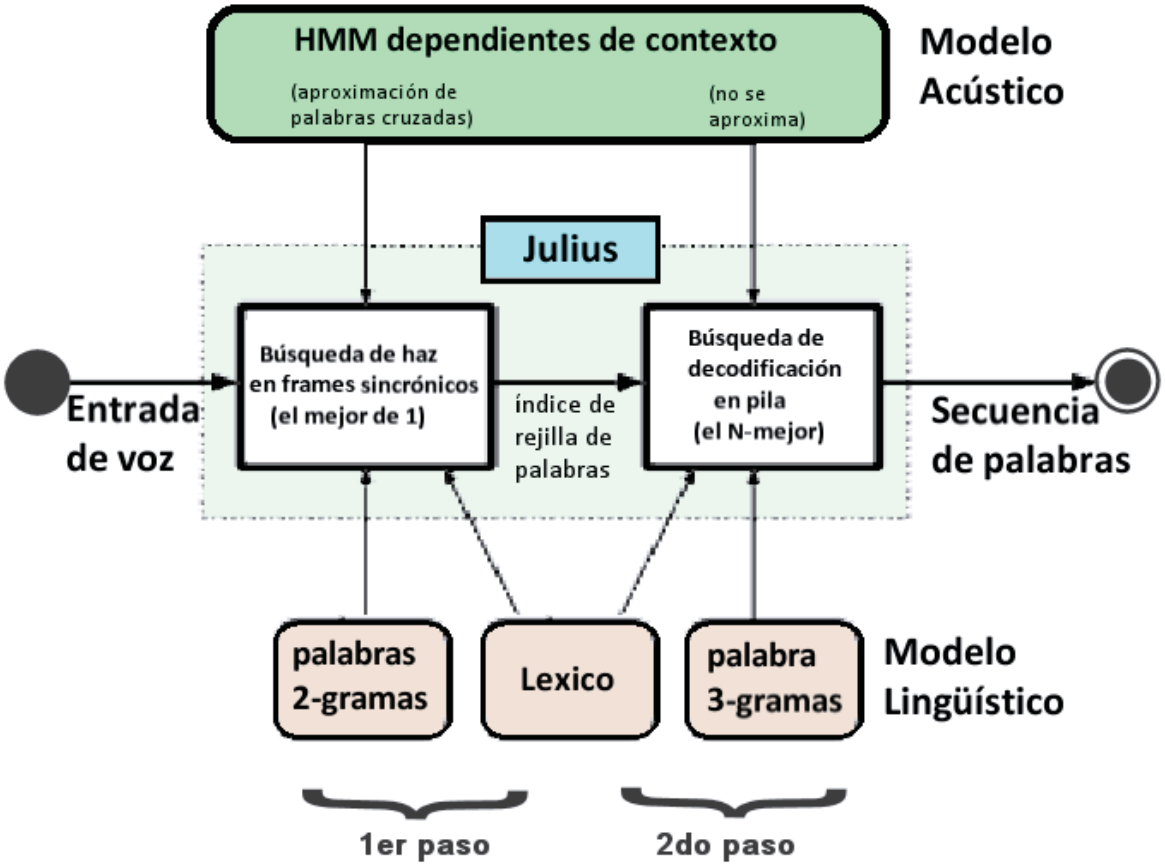

Figura 11-2 Diagrama que representa la estructura del reconocimiento de voz en Julius.

La figura 10.2 representa como Julius realiza el reconocimiento de voz, este es realizado mediante una búsqueda de 2 pasos, utilizando palabras de 2-gramas y 3 gramas. El primer paso realiza una búsqueda rápida de las mejores palabras, mientras el segundo realiza una búsqueda más precisa hasta encontrar la mejor palabra (o la más precisa, no necesariamente la mejor será la correcta). Estos pasos serán explicados a continuación:

11.2.1 Primer paso

Un árbol léxico asignado con probabilidades de un modelo lingüístico le es aplicado el algoritmo de Beam Search con un marco sincrónico. Asigna valores factoriales de 1-grama pre-computados a los nodos intermedios y aplica probabilidades de 2 gramas al nodo del final de la palabra. La dependencia de contexto de la palabra cruzada se maneja con la aproximación al mejor modelo de los registrados en el software. Como los valores de la factorización 1-grama son independientes de las palabras que siguen, se pueden computar estáticamente en un solo árbol léxico. Los valores obtenidos aun no son muy precisos (son solo una aproximación), por ello es necesario el segundo paso. Al finalizar esta búsqueda se genera un "índice de rejilla de palabras", que es un conjunto de nodos conformado con las palabras cruzadas que fueron encontradas, asignadas con un puntaje y puntos de partidas de cuadro por cuadro para dejar todo preparado para el siguiente paso.

11.2.2 Segundo paso

Se utiliza un modelo lingüístico de 3-gramas independiente del contexto de la palabra. La búsqueda se realiza en dirección inversa y se calcula un resultado preciso de las n-mejor palabras u oración independiente del índice de rejilla de palabras del primer paso. Se analiza nuevamente la entrada de voz para obtener nuevos resultados y conectar estos con los de la rejilla. Se realiza una búsqueda de decodificación en forma de pila, poniendo un número máximo de hipótesis de cada largo de la sentencia, para ello se utiliza el algoritmo de Gaussian Pruning, el cual "poda" la distancia de la probabilidad de registro a la mitad del largo completo del vector en el caso de no ser prometedor (es decir, se utiliza para descartar posibles resultados). Para reducir más los resultados obtenidos, en el modelo trifono, se realiza una nueva selección gaussiana, llamada selección de resultados de Gauss (Gaussian Mixture Selection). Se utiliza también HMM para la pre-selección de los estados trifono independientes del contexto, primero son procesadas las probabilidades de estado de los modelos en cada frame y luego se procesan los estados de los trifonos (cuyos monofonos correspondan a los clasificados entre los n-mejores). A los estados que no son seleccionados se les da la probabilidad de monofono.

El reconocimiento de voz de cada segmento de palabras es realizado de forma secuencial, se hace la decodificación del primer paso en paralelo con la entrada de voz. Empieza a procesar tan pronto como empieza un segmento de entrada, y cuando se detecta una pausa larga, termina el primer paso y continúa con el segundo. Como el segundo paso termina en poco tiempo, el retraso del resultado de reconocimiento es bastante corto. La salida es un resultado de un reconocimiento en una secuencia de palabras. Los N-mejores resultados se pueden mostrar. Se pueden generar secuencias de fonemas, resultados de probabilidad de registro y varias estadísticas de búsqueda. Pueden salir resultados parciales sucesivamente mientras se procesa el primer paso, aunque el resultado final se determina al final del segundo paso.

Para comprender los pasos explicados anteriormente, el siguiente diagrama representa un aspecto más bajo nivel del funcionamiento de Julius, dividiendo los archivos, librerías y funciones, respecto a los pasos explicados anteriormente:

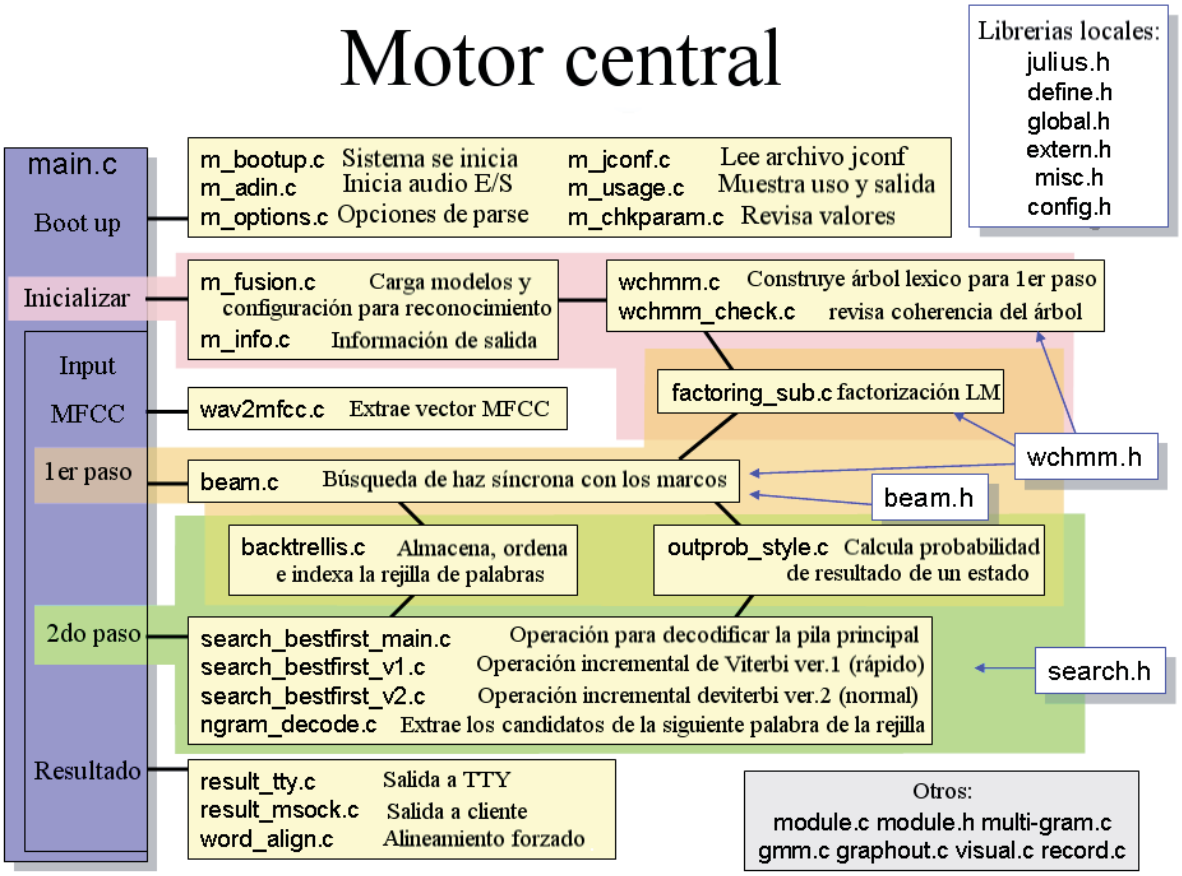

Figura 11-3 Componentes del software Julius

Respecto a la figura 11.3, se puede mencionar que el archivo que tiene todas las configuraciones para el reconocimiento de voz es el archivo jconf, que tiene todos los parámetros necesarios para nivelar el reconocimiento.

12 CREACIÓN DE MODELO DE LENGUAJE

Para poder crear el modelo del español fue necesario utilizar varias herramientas de HTK y algunos scripts que contiene el mismo HTK (escritos en lenguaje Perl). Entre las herramientas más utilizadas están:

- HDMan: herramienta que manipula archivos, sirve para editar archivos del diccionario.
- HLEd: herramienta que manipula archivos, edita capas y archivos de la capa superior (ejemplo: archivos que contengan fonemas).
- HCopy: herramienta que manipula archivos, sirve para convertir de un archivo de audio a otros tipos (también otros archivos).
- HCompV: herramienta de entrenamiento, al ingresar un prototipo de HMM computa la media general y la varianza.
- HERest: herramienta de entrenamiento, realiza entrenamiento de Baum-Welch incrustado al ingresar un modelo inicializado.
- HHEd: herramienta que manipula archivos, sirve para editar el modelo y archivos importantes macros (usado en HMM).
- HVite: herramienta de reconocimiento, realiza decodificación de Viterbi.

12.1 Preparación de Datos

Para crear el modelo, se necesita crear los siguientes archivos (allí se definen las palabras las cuales serán reconocidas una vez finalizado el modelo):

- prompts: un archivo con una lista de samples que contienen las palabras escritas que fueron grabadas con Audacity para entrenar el modelo.
- .grammar: un archivo con reglas escritas en EBNF para darle contexto y semántica a lo que se dice.
- .voca: un archivo que contiene categorías de las palabras del léxico usado.
- spanish_lexicon: un archivo que contiene todas las palabras del diccionario usado en el modelo junto con sus fonemas.

- samples.wav: archivos de audio que contienen las muestras para poder entrenar el modelo.

Se transforman los archivos .grammar y .voca (que contiene todas las palabras que se utilizarán en este modelo) a formato Julius (archivo .term y .dfa, respectivamente) mediante uno de los scripts en Perl en la terminal con el comando "mdkfa.pl sample" (siendo sample el nombre de ambos archivos). La gramática fue definida así (archivo .gramar):

S : NS_B LOOKUP NS_E
LOOKUP: NUMBER
LOOKUP: ADJ
LOOKUP: MISC
LOOKUP: TIME
LOOKUP: SUST
LOOKUP: VERB

Las palabras que serán utilizadas están definidas en el archivo sample.voca. La gramática también puede ser escrita de la siguiente manera:

S : NS_B LOOKUP NS_E
LOOKUP: WORD

Este último es utilizado siempre y cuando en el archivo "sample.voca" solo exista una única categoría llamada WORD (con todas las palabras). Es importante también destacar el archivo sample.dict se pueden modificar las palabras según como se quieran mostrar en pantalla, ya que este es el archivo que define como serán mostradas las palabras en pantalla. Por ejemplo la palabra "pequeño", posee el grafema "ñ", este carácter no es reconocido al ser utilizadas las herramientas del HTK durante la creación de los modelos (cambiándolo a "164" del formato ASCII), por lo que es remplazado en la mayoría de los archivos por "ny", entonces, en el archivo sample.dict se remplazan las ny a ñ, para mostrar la correcta escritura de la palabra por pantalla.

Las palabras que están en el archivo sample.voca deben ser añadidas al archivo prompts arbitrariamente (o en orden) y de manera que todas las palabras se repitan para que sean usadas para entrenar el modelo, como se muestra a continuación:

*/sample1 sabes nuestro hombre ahora me dice son gente
*/sample2 su padre dice ahora ellos toda ninya puedes haber desde
*/sample3 alguien dijo sobre ellos nunca son tan pequenyo
*/sample4 dijo alguien mira mucho puedes uno cero dos va
*/sample5 los anyos dios me gusta nuestro hijo padre cuatro
*/sample6 mi tipo dios hecho los anyos ven pero nosotros desde haber

Y así sucesivamente hasta crear un total de 25 samples más (se puede utilizar más de 25 samples), utilizando todas las palabras que se encuentran en el archivo sample.voca.

Estos samples deben de ser grabados en Audacity (con la configuración señalada en el manual de instalación del anexo C de este informe). Se grabaron 12 veces estos samples, con 12 voces diferentes (6 masculinas y 6 femeninas), logrando un total de 275 samples (mientras más samples sean grabados, con más variedad de voces, mejor será el reconocimiento de voz).

Nota: todas las compilaciones deben ser hechas desde la carpeta "manual" (que se encuentra en el cd del proyecto).

HTK incluye un script en perl que toma el archivo prompts que contiene los samples, saca el nombre del sample de la primera columna e imprime cada palabra en una línea sin que se repita en un archivo lista de palabras (wlist), este script se encuentra en una carpeta llamada "HTK_scripts". Esto se logra ingresando en la terminal el script de Perl "perl ../HTK_scripts/prompts2wlist prompts wlist" desde la carpeta "manual". Al ejecutarlo, el archivo wlist contendrá todas las palabras de los samples sin repetición. A este archivo se le debe agregar una palabra extra que se usará como silencio entre frases. En este informe llamaremos a esta palabra "silence" (en los modelos anteriores se utilizaron dos palabras (SENT-START y SENT-END) para denotar un silencio al principio de una frase y otro al final, pero después se descubrió que sólo bastaba con uno) esta palabra debe agregarse en orden alfabético dentro de wlist.

Antes de continuar, se debe crear un archivo llamado .ded que contiene lo siguiente (siempre dejar un espacio en blanco al final del archivo):

AS sp
RS cmu
MP sil sil sp

Se referirá a este archivo como global.ded, esta es una breve explicación de lo que hace:

- AS sp → Le agrega pausas cortas (short pauses, sp) a cada pronunciación.
- RS cmu → Quita marco de estrés. Actualmente el único marco de estrés que soporta es el usado en los diccionarios de Carnegie Melon University (su sistema es cmu).
- MP sil sil sp → Une cualquier secuencia de fonemas sil sp y las renombra como sil, esto para que no queden repetidas en caso de que termine una oración.

Es necesario añadir la información de pronunciación (estos son los fonemas que crean la palabra) a cada una de las palabras del archivo wlist, para así crear un diccionario de pronunciación. Para crear el diccionario de palabras y fonemas se usa la herramienta de HTK HDMan (con el comando "HDMan -A -D -T 1 -m -w wlist -n monophones1 -i -l dlog dict ../lexicon/spanish_lexicon"), este recorre el archivo wlist y busca su pronunciación en el archivo spanish_lexicon (el archivo que contiene las palabras y su pronunciación en la carpeta lexicon) y pone los resultados en un diccionario de pronunciación, en este caso el archivo dict. Este comando también genera el archivo monohpones1, el cual contiene todos

los monofonos utilizados en el modelo (incluyendo silencio (sil) y short pause (sp)) y el archivo dlog.

La herramienta HDMan puede explicarse de la siguiente forma:

HDMan -A -D -T 1 -m -w wlist -n monophones1 -i -l dlog dict ../lexicon/spanish_lexicon

Luego,

HDMan [options] newDict srcDict1 srcDict2...
(HDMan lee en los diccionarios srcDict1, srcDict2, etc.y genera un nuevo diccionario newDict(Wlist)).

Dónde:

- -A → Imprime los argumentos en la línea de comando.
- -D → Muestra variables de configuración.
- -T 1 → Pone el nivel de recorrido a 1.
- -m → Une las pronunciaciones de todos los diccionarios. Por default, HDMan genera una sola pronunciación para cada palabra. Si se tienen más de una pronunciación por palabra, se usa la primera. Habilitar esta opción causa que las pronunciaciones de esa palabra sean pronunciaciones de palabras distintas a esa (este problema se resuelve más adelante).
- -w {wlist} → Carga la lista de palabras del archivo {wlist}, sólo las pronunciaciones de estas palabras se van a extraer de los diccionarios.
- -n monophones1 → Muestra una lista de todos los fonemas distintos que fueron encontrados en el archivo monophones1.
- -i → Incluye como van a ser mostradas las palabras en dict (ejemplo: hola [hola] oo l a). O sea muestra los corchetes, que es como se verá la palabra al final por pantalla.
- -l {nombre log} → Escribe un archivo de log en {nombre log}. El archivo log incluye estadísticas del diccionario y una lista con el número de veces que aparece cada fonema.

De este último se debe destacar que para que HTK compile los audios y transcripciones del idioma en un modelo acústico, se necesita un diccionario fonéticamente balanceado con al menos de 30 a 40 oraciones con 8 a 10 palabras cada una. Si la gramática tiene menos frases o palabras que eso, o si no esta fonéticamente balanceada, habrá que agregarle más palaras al diccionario hasta que todos los fonemas se repitan al menos 3 a 5 veces. Este paso es esencial ya que aquí se definirá si el diccionario usado está lo suficientemente balanceado para continuar. Para verificar lo mencionado anteriormente, este archivo (dlog) que se crea con la herramienta HDMan, muestra la cantidad de veces que es repetido un fonema, en el archivo dlog del modelo actual muestra lo siguiente:

1. a : 25
2. oo : 13

3. r : 29
4. sp : 107
5. aa : 23
6. l : 8
7. g : 4
8. i : 15
9. e : 40
10. n : 27
11. t : 23
12. s : 43
13. ny : 4
14. o : 40
15. ii : 14
16. ll : 3
17. uu : 6
18. d : 24
19. b : 16
20. u : 10
21. ee : 29
22. k : 11
23. th : 8
24. x : 5
25. p : 13
26. ch : 3
27. m : 14
28. sil : 1

Como se ve, todos los fonemas están repetidos 3 o más veces, lo que significa que este diccionario de palabras es lo suficientemente balanceado. También es importante destacar que el fonema sil, no es necesario que se repita, ya que solo aparecerá al inicio y al final de una oración. Este archivo ahorra mucho tiempo de compilación ya que si se pasa a llevar en pasos futuros pueden aparecer múltiples errores debido a que el diccionario o la gramática no estaban bien balanceados. Otro detalle que muestra este archivo es si hay alguna palabra mal escrita o alfabéticamente desordenada en el diccionario, (spanish_lexicon) aparecerá al principio del archivo como “missing Word”.

Una vez que se sabe que el diccionario es balanceado, se pueden grabar las samples del archivo prompts (que deben ser grabadas con la configuración descrita anteriormente). Para mayor reconocimiento se recomienda repetir las samples y grabar con distintas voces, esto permite al modelo ser más flexible con cualquier tipo de voz, por ejemplo, si se graban puras voces agudas y se prueba el modelo con una voz grave, probablemente el modelo no reconozca muy bien las palabras o ni siquiera las reconozca, en cambio si se graban voces agudas, graves, normales, y muchas veces, el modelo tendrá más registros de como se dice cierta palabra. Se recomienda grabar las samples en un lugar sin ruidos externos, a una distancia prudente del micrófono y evitar hacer sonar la boca (o los labios) entre palabras, también evitar tomar aire entre palabras, es mejor tomar una cantidad de aire al principio y

decir toda la frase de la sample, así no quedarán registrados sonidos extras cuando grabe. Lo importante es que entre palabras, haya la menor cantidad de ruido posible (aunque algunas samples pueden tener ruidos menores de ambiente, ya que el modelo puede ser probado en una situación así).

Del archivo prompts se genera un archivo llamado words.mlf (Master Label File, un archivo que contendrá las transcripciones de prompts) con el comando "perl ../HTK_scripts/prompts2mlf words.mlf prompts", el cual contiene las palabras transcritas de los archivos de audio. También debe crearse otro archivo .led llamado mkphones.led (siempre dejar un espacio en blanco al final del archivo):

EX
IS sil sil
DE sp

El comando de expansión EX remplaza cada palabra de words.mlf por su correspondiente pronunciación en el archivo dict. El comando IS inserta un modelo de silencio sil al final y al principio de cada declaración. Por ultimo, el comando para borrar DE borra todos los sp (short pauses) que no se necesitan en las transcripciones (o que estén repetidos).

Luego con la herramienta HLEd se pueden transcribir las palabras de los archivos de audio en monofonos al archivo phones1.mlf, permitiendo más de una pronunciación por palabra, así solucionando el problema de las palabras con más de una pronunciación. El comando para HLEd es el siguiente:

HLEd -A -D -T 1 -l '*' -d dict -i phones0.mlf mkphones0.led words.mlf

Donde -l ->genera la ruta '*' en los patrones de salida.

Este comando crea el archivo phones0.mlf, que contiene las mismas transcripciones de words.mlf, pero con el fonema sil entre frases. Sin embargo, este archivo no contiene las pausas entre palabras (sp), por lo que se debe crear otro archivo .mlf llamado phones0.mlf (siempre dejar un espacio en blanco al final del archivo):

EX
IS sil sil

Y posteriormente ejecutar de nuevo HLEd -A -D -T 1 -l '*' -d dict -i phones1.mlf mkphones1.led words.mlf para obtener el archivo phones1.mlf, que contiene las correspondientes pausas entre cada palabra. Un ejemplo de cómo quedarían las palabras en cada archivo:

words.mlf -> "*/sample1.lab" aa l g i e n d ii x o
phones0.mlf ->"*/sample1.lab" **sil** aa l g i e n d ii x o **sil**
phones1.mlf ->"*/sample1.lab" **sil** aa l g i e n **sp** d ii x o **sil**

La parte final de la preparación de datos es parametrizar las ondas de habla en secuencias de vectores. HTK soporta análisis basado en FFT (Fast Fourier Transform) y LPC (Linear Prediction Cofficient), los cuales son para configurar espectros de habla. En este caso, se usarán MFCC (Mel frequency Cepstral Coefficients) que derivan del espectro basado en FFT.

Se puede crear el código usando la herramienta HCopy para convertir automáticamente las samples en vectores MFCC. Para esto, primero se necesita un archivo de configuración (llamado aquí wav_config) donde se especifican todos los parámetros de conversión. Una configuración razonable para los samples sería:

```
# Coding parameters, wav_config file
TARGETKIND = MFCC_0
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F
```

Algunos de estos parámetros son las configuraciones por default, pero se dan explícitamente para completar. En resumen, se especifica que los parámetros son para MFCC usando C0 como el componente de energía, el período del marco es 10 milisegundos (HTK usa unidades de 100 nanosegundos) la salida debe guardarse en formato comprimido y se le agrega a crc checksum. La configuración debe usar una ventana hamming y la señal debe tener preénfasis de primera orden aplicada usando un coeficiente de 0.97. El banco de filtro debe tener 26 canales y deben salir 12 coeficientes de MFCC. La variable ENORMALISE es verdadera (true) por default y realiza normalización de energía en los archivos de audio grabados. No se puede usar con audio en tiempo real y como el sistema es para audio en tiempo real, esta variable debe ponerse en falso (false). Notar que no es necesario explícitamente crear archivos de datos codificados, ya que la codificación se hace sola a medida que uno avanza con las herramientas necesarias de HTK. Sin embargo, crear estos archivos reduce la cantidad de pre-procesos requeridos durante el entrenamiento, el cual consume mucho tiempo.

Adicionalmente, se necesita de un archivo llamado codetrain.scp que contiene una lista con la ruta de las samples en formato .wav que serán luego convertidas a formato MFCC, el archivo será algo así:

```
../train/wav/sample1.wav ../train/mfcc/sample1.mfc
../train/wav/sample2.wav ../train/mfcc/sample2.mfc
```

../train/wav/sample3.wav ../train/mfcc/sample3.mfc

Hasta el número total de samples. Nótese que el nombre de los archivos mfcc finales es el mismo que el de los archivos de audio, en este caso los archivos de audio se llaman sample1.wav, sample2.wav, etc. pero dependen del nombre que se les haya puesto a los archivos de audio cuando se grabaron.

Después de tener listo el archivo de configuración y la lista con las samples ejecutar con la herramienta HCopy ("HCopy -A -D -T 1 -C wav_config -S codetrain.scp", no se detallaron los parámetros ya que son opciones estándar para cualquier herramienta de HTK o fueron descritas anteriormente) el cual mediante la lista que contiene la ruta de todas las samples de audio, convierte los archivos de audio a MFCC que consiste en puros 0 y 1, que describen las ondas de los archivos de audio guardados en otro formato. HTK se guiará de este formato para luego hacer las probabilidades de Markov.

12.2 Monofonos

El primer paso es crear monofonos planos para el inicio, luego de varias iteraciones y aplicando Markov estos datos irán cambiando. Hasta ahora todos los archivos se han generado en la carpeta "manual", en breve se crearán dentro de esta misma carpeta las iteraciones de Markov.

Primero se crea (aun en la misma carpeta) un archivo llamado proto, que contiene lo siguiente (siempre dejar un espacio en blanco al final del archivo):

```
~o <VecSize> 25 <MFCC_0_D_N_Z>
~h "proto"
<BeginHMM>
  <NumStates> 5
  <State> 2
    <Mean> 25
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0
    <Variance> 25
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0
 <State> 3
    <Mean> 25
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0
    <Variance> 25
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0
 <State> 4
    <Mean> 25
```

0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

<Variance> 25

1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0

<TransP> 5

0.0 1.0 0.0 0.0 0.0

0.0 0.6 0.4 0.0 0.0

0.0 0.0 0.6 0.4 0.0

0.0 0.0 0.0 0.7 0.3

0.0 0.0 0.0 0.0 0.0

<EndHMM>

Donde se tienen 5 estados con varianza 25 y media 25. Los valores de cada estado se irán restimando a medida que se hacen las iteraciones. También se necesita de otro archivo de configuración:

(config file)

TARGETKIND = MFCC_0_D_N_Z

TARGETRATE = 100000.0

SAVECOMPRESSED = T

SAVEWITHCRC = T

WINDOWSIZE = 250000.0

USEHAMMING = T

PREEMCOEF = 0.97

NUMCHANS = 26

CEPLIFTER = 22

NUMCEPS = 12

Donde,

- TARGETKIND → Tipo de parámetro del archivo.
- TARGETRATE → Período de tiempo del archivo en unidades de 100 nanosegundos.
- SAVECOMPRESSED → Guarda el archivo de salida en formato comprimido.
- SAVEWITHCRC → Adjunta un checksum al archivo de salida.

- WINDOWSIZE → Muestra el tamaño de la ventana de análisis en unidades de 100 nanosegundos.
- USEHAMMING → Usa una ventana Hamming.
- PREEMCOEF → Establece el coeficiente de preénfasis.
- NUMCHANS → Número de canales de filtro.
- CEPLIFTER → muestra el Cepstral liftering coefficient (una variable de MFCC).
- NUMCEPS → numero de parámetros.

Nota: estas variables se encuentran en la herramienta HParm y vienen con esos valores por default.

La variable TARGETKIND debe ser igual en los archivos proto y config.

Para continuar se necesita hacer un archivo que contiene la lista de archivos con la ruta de los archivos en formato MFCC creados anteriormente (estos archivos deberían encontrarse en la carpeta "train/mfcc") y listarlos para crear el archivo train.scp, que tiene un formato así (el nombre "sampleX" puede ser cambiado dependiendo de cómo se le llamó en los prompts y en los .wav):

../train/mfcc/sample1.mfc
../train/mfcc/sample2.mfc
../train/mfcc/sample3.mfc…

El número dependerá de cuantos samples tenga el modelo, en este caso son 275. Este archivo se debe modificar si posteriormente sale un error, ya que si no aparece la sample escrita, el modelo no la tomará en cuenta, por lo que si una sample da problemas de compilación, simplemente hay que quitarla de esta lista. Volviendo a la carpeta "manual", se crea ahí una carpeta llamada hmm0.

La herramienta CompV escanea un conjunto de archivos, calculando la media global, varianza y configura todos los cálculos gaussianos en un HMM dado para que tengan la misma media y varianza. Se invoca de la siguiente forma:

HCompV [options] [hmm] trainFiles ...

Donde hmm es el mismo hmm al que se le iniciarán sus parametros. Así, asumiendo que se almacena una lista con los archivos entrenados en train.scp, el comando "HCompV -C config -f 0.01 -m -S train.scp -M hmm0 proto" .

Donde,

- -C {config} ->usa el archivo de configuración {config}.

- -f {0.01} ->crea los límites inferiores de macros con valores iguales a 0.01 veces la varianza global. Se crea un macro por cada entrada y la salida se guarda en un archivo llamado vFloors.
- -m -> las covarianzas de los resultados de HMM siempre se actualizan, sin embargo se debe exigir explícitamente. Cuando se pone esta opción, HCompV actualice todas las medias de HMM con la que sale en las samples de los archivos de entrenamiento (las samples).
- -S {train.scp} ->usa el script {train.scp}.
- -M {hmm0} -> almacena el resultado del archivo macro HMM en la carpeta {hmm0}. Si esta opción no se pone, la nueva definición de HMM sobrescribe la existente.

Creará una nueva versión de proto en las carpetas hmm0 donde la media cero y las varianzas unitarias descritas arriba son remplazadas por las medias y varianzas globales del habla. Como se menciona en la opción –f, también se genera un archivo vFloors, con los valores base para el modelo.

Una vez creados estos archivos e la carpeta “hmm0”, hay que sacar de la carpeta “manual” el archivo monophones0 y copiarlo en “hmm0”. Se renombra este archivo como “hmmdefs” que funcionará como un archivo MMF (Master Macro File) el cual contiene todos los monofonos del modelo y se fusionará con proto, para así darles los valores planos a cada monofono. Este es el primer paso para empezar las iteraciones.

Primero, en el archivo hmmdefs se debe poner entre comillas dobles cada fonema (ejemplo: a -> “a”), luego anteponerle a cada fonema ‘~h ’ (incluyendo el espacio) donde cada fonema debe quedar así: ~h “a”.

Luego de aplicarle eso a cada fonema, se copia del archivo proto (que se encuentra en “manual/hmm0”) de la línea 5 en adelante, esto es desde <BEGINHMM> hasta <ENDHMM> y se pega todo ese contenido después de cada fonema en hmmdefs. El resultado será algo así (para cada fonema):

~h "a"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<MEAN> 25
2.495273e-07 1.105371e-07 2.588544e-08 -3.083722e-08 -4.340004e-08 -4.191975e-08
2.943552e-08 3.052282e-09 1.035418e-07 6.766110e-08 1.671550e-08 -7.776114e-08 -
7.355643e-03 -3.789790e-03 1.184971e-02 -7.089294e-03 -3.233539e-04 4.847068e-03 -
5.158428e-03 6.903479e-04 9.217368e-03 -6.614285e-03 -1.175333e-03 1.114921e-02
4.575740e-03
<VARIANCE> 25
6.580434e+01 3.732679e+01 3.525512e+01 4.770427e+01 4.332327e+01 4.544643e+01
5.620688e+01 2.553866e+01 4.001570e+01 3.416673e+01 2.128212e+01 2.660224e+01
1.668585e+00 1.700366e+00 1.616409e+00 1.768895e+00 1.718035e+00 2.098122e+00

2.326025e+00 1.677738e+00 2.010740e+00 1.595870e+00 1.417547e+00 1.510511e+00
1.447709e+00
<GCONST> 9.662931e+01
<STATE> 3
<MEAN> 25
2.495273e-07 1.105371e-07 2.588544e-08 -3.083722e-08 -4.340004e-08 -4.191975e-08
2.943552e-08 3.052282e-09 1.035418e-07 6.766110e-08 1.671550e-08 -7.776114e-08 -
7.355643e-03 -3.789790e-03 1.184971e-02 -7.089294e-03 -3.233539e-04 4.847068e-03 -
5.158428e-03 6.903479e-04 9.217368e-03 -6.614285e-03 -1.175333e-03 1.114921e-02
4.575740e-03
<VARIANCE> 25
6.580434e+01 3.732679e+01 3.525512e+01 4.770427e+01 4.332327e+01 4.544643e+01
5.620688e+01 2.553866e+01 4.001570e+01 3.416673e+01 2.128212e+01 2.660224e+01
1.668585e+00 1.700366e+00 1.616409e+00 1.768895e+00 1.718035e+00 2.098122e+00
2.326025e+00 1.677738e+00 2.010740e+00 1.595870e+00 1.417547e+00 1.510511e+00
1.447709e+00
<GCONST> 9.662931e+01
<STATE> 4
<MEAN> 25
2.495273e-07 1.105371e-07 2.588544e-08 -3.083722e-08 -4.340004e-08 -4.191975e-08
2.943552e-08 3.052282e-09 1.035418e-07 6.766110e-08 1.671550e-08 -7.776114e-08 -
7.355643e-03 -3.789790e-03 1.184971e-02 -7.089294e-03 -3.233539e-04 4.847068e-03 -
5.158428e-03 6.903479e-04 9.217368e-03 -6.614285e-03 -1.175333e-03 1.114921e-02
4.575740e-03
<VARIANCE> 25
6.580434e+01 3.732679e+01 3.525512e+01 4.770427e+01 4.332327e+01 4.544643e+01
5.620688e+01 2.553866e+01 4.001570e+01 3.416673e+01 2.128212e+01 2.660224e+01
1.668585e+00 1.700366e+00 1.616409e+00 1.768895e+00 1.718035e+00 2.098122e+00
2.326025e+00 1.677738e+00 2.010740e+00 1.595870e+00 1.417547e+00 1.510511e+00
1.447709e+00
<GCONST> 9.662931e+01
<TRANSP> 5
0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
0.000000e+00 6.000000e-01 4.000000e-01 0.000000e+00 0.000000e+00
0.000000e+00 0.000000e+00 6.000000e-01 4.000000e-01 0.000000e+00
0.000000e+00 0.000000e+00 0.000000e+00 7.000000e-01 3.000000e-01
0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
<ENDHMM>

Para así crear el inicio de los monofonos planos. Se destaca que estos valores son diferentes dependiendo del modelo y estos son los valores que irán variando a lo largo de las iteraciones. Para terminar con “hmm0” se crea un último archivo llamado macros el cual contiene las variables globales del modelo, y se crea combinando vFloors y proto:

a. crear un nuevo archivo llamado “macros”.

b. copiar todo el contenido de vFloors en macro.

c. copiar las tres primeras líneas de proto (esto es de ~o a <DIAG>) y pegarlos sobre los datos de vFloors.

El resultado será algo así:

~o
<STREAMINFO> 1 25
<VECSIZE> 25<NULLD><MFCC_D_N_Z_0><DIAGC> # datos de proto
~v varFloor1 #datos de vFloors
<Variance> 25
6.580433e-01 3.732679e-01 3.525512e-01 4.770427e-01 4.332326e-01 4.544643e-01
5.620688e-01 2.553866e-01 4.001570e-01 3.416672e-01 2.128212e-01 2.660224e-01
1.668585e-02 1.700366e-02 1.616409e-02 1.768895e-02 1.718035e-02 2.098122e-02
2.326025e-02 1.677738e-02 2.010740e-02 1.595870e-02 1.417547e-02 1.510511e-02
1.447709e-02

Luego de esto comienzan las iteraciones de estos valores. Se debe crear en la carpeta "manual" otras 9 carpetas hmm del 1 al 9 ("hmm1", "hmm2", etc.) comenzar a restimar con la herramienta HERrest así:

"HERest -A -D -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp -H hmmX/macros -H hmmX/hmmdefs -M hmm[X+1] monophones0"

Donde X es el número de la carpeta(es decir, 0, 1 y 2), y la opción –I carga el archivo .mlf. Esta herramienta crea 3 nuevas iteraciones, cada una de estas con nuevos valores para hmmdefs y utilizando los monofonos anteriormente definidos. Por lo que se debe ejetutar finalmente así:

"HERest -A -D -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp -H hmm0/macros -H hmm0/hmmdefs -M hmm1 monophones0

HERest -A -D -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp -H hmm1/macros -H hmm1/hmmdefs -M hmm2 monophones0

HERest -A -D -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp -H hmm2/macros -H hmm2/hmmdefs -M hmm3 monophones0"

Los cuales generan los archivos hmmdefs y macros para las carpetas "hmm1", "hmm2" y "hmm3" respectivamente. El paso anterior generó un HMM de izquierda a derecha de 3 estados para cada fonema pero no se incluía el fonema "sp". En la carpeta "hmm4" se arreglará el HMM para permitir la pausa corta (short pause,"sp"). Para esto se copian los contenidos de la carpeta "hmm3" en "hmm4". Al principio cuando se copió el primer hmmdefs se hizo con el archivo monophones0, este a diferencia de monophones1 no contiene el fonema "sp". Para añadirlo en el actual hmmdefs se copia todo el trozo desde '~h "sil"' hasta <ENDHMM>, no hay que borrar el actual modelo sil, sino que sólo copiarlo y pegarlo abajo. A este nuevo modelo sil se le hacen algunos cambios:

- Cambiar el nombre de "sil" a "sp".

- Quitar por completo los estados 2 y 4 (un estado comienza desde <STATE> hasta <GCONST> y el valor que aparezca en esa línea.) por lo que sólo quedará el estado 3.
- Cambiar la variable <NUMSTATES > de 5 a 3.
- Cambiar la variable que quedo de <STATE> de 3 a 2 (por lo que solo quedará el estado del medio).
- cambiar la variable de la matriz <TRANSP> de 5 a 3.
- cambiar por completo al matriz a la siguiente:

0.0 1.0 0.0
0.0 0.9 0.1
0.0 0.0 0.0

El nuevo modelo “sp” quedará más o menos así:

```
~h "sp"
<BEGINHMM>
<NUMSTATES> 3  #antes 5
<STATE> 2              #antes 3
<MEAN> 25
 -7.046570e+00 -3.262981e-01 -1.706483e+00 -1.080971e+00 -1.134529e+00
3.588506e+00 3.917166e+00 1.443405e+00 4.899211e+00 3.409961e+00 8.219168e-01
3.644213e+00 -7.641904e-02 -6.077167e-02 2.118241e-01 -8.631640e-02 3.686112e-02
8.506200e-02 -8.106526e-02 1.066912e-02 1.281262e-01 -1.437282e-01 -3.412217e-02
1.333326e-01 1.202221e-01
<VARIANCE> 25
 7.911258e+00 8.348815e+00 1.148870e+01 1.213321e+01 8.655976e+00 1.509970e+01
9.904381e+00 1.166922e+01 1.025182e+01 8.845907e+00 8.135198e+00 9.622693e+00
9.084668e-01 7.631339e-01 1.614822e+00 9.755048e-01 7.167343e-01 1.691362e+00
1.297928e+00 9.801642e-01 1.225108e+00 1.051384e+00 9.349809e-01 1.529028e+00
5.576642e-01
<GCONST> 7.411308e+01
<TRANSP> 3          #antes 5
 0.0 1.0 0.0
 0.0 0.9 0.1
 0.0 0.0 0.0
<ENDHMM>
```

La idea de esto es hacer que el modelo sea más robusto permitiendo que los estados individuales absorban los "ruidos extra" en los archivos de entrenamiento. El salto hacia atrás permite esto, sin que el modelo tenga que transitar a la siguiente palabra. Para esto se creó el fonema “sp” que actúa como un modelo de soporte, el cual tiene una transición directa del nodo de entrada al de salida. El "sp" tiene su estado atado al estado central del modelo de silencio. Después de incluir sp se debe ejecutar el editor de HMM HHEd para añadir las transiciones extra requeridas y amarrar el estado sp al estado central de sil. HHEd trabaja parecido a HLEd, aplica un conjunto de comandos para modificar el conjunto de HMMs, que deja “amarrado” al modelo el fonema “sp”. En este caso, se ejecuta así:

“HHEd -A -D -T 1 -H hmm4/macros -H hmm4/hmmdefs -M hmm5 sil.hed monophones1”

Donde sil.hed (que debe crearse en la carpeta “manual”) contiene los siguientes comandos:

AT 2 4 0.2 {sil.transP}

AT 4 2 0.2 {sil.transP}

AT 1 3 0.3 {sp.transP}

TI silst {sil.state[3],sp.state[2]} # ata los modelos sil y sp

Después de esto, el fonema sp queda amarrado al modelo (desde “hmm5” en adelante), luego se realizan 2 iteraciones más con HERest para reestimar este nuevo cambio, los cuales serían:

“HERest -A -D -T 1 -C config -I phones1.mlf -t 250.0 150.0 3000.0 -S train.scp -H hmm5/macros -H hmm5/hmmdefs -M hmm6 monophones1

HERest -A -D -T 1 -C config -I phones1.mlf -t 250.0 150.0 3000.0 -S train.scp -H hmm6/macros -H hmm6/hmmdefs -M hmm7 monophones1”

Que reestiman el modelo incluyendo a “sp” en las carpetas “hmm6” y “hmm7”. El siguiente paso es realinear los datos mediante la herramienta HVite, esta operación es similar al mapeo de palabra a fonema HLEd realizado anteriormente, solo que en este caso el comando HVite considera todas las pronunciaciones para cada palabra (en caso que una palabra tenga más de una pronunciación), y luego muestra pronunciación que mejor encaje del modelo acústico.

Como se dijo anteriormente, el diccionario contiene múltiples pronunciaciones en algunas palabras, los modelos creados hasta ahora se pueden usar para realinear la información de entrenamiento y crear nuevas transcripciones. Esto se logra con una ejecución de la herramienta HVite:

“HVite -A -D -T 1 -l '*' -o SWT -b silence -C config -H hmm7/macros -H hmm7/hmmdefs -i aligned.mlf -m -t 250.0 150.0 1000.0 -y lab -a -I words.mlf -S train.scp dict monophones1> HVite_log.”

Este comando usa los HMM almacenados en hmm7 para para transformar la palabras transcritas de words.mlf a un nuevo nivel de alienación aligned.mlf, usando las pronunciaciones del archivo dict. En el comando anterior, la opción -b se usa para insertar un modelo de silencio al principio y al final de cada frase (sample). Se usa el nombre "silence" asumiendo que en el diccionario hay una entrada "silence sil". También es necesario notar que el diccionario debe estar ordenado en primera instancia por mayúsculas y luego en orden alfabético.

La opción -t determina un nivel de poda (pruning) de 250.0 y la opción -o se usa para no mostrar por pantalla los nombres de las palabras, límites de tiempo y otros resultados en la salida del MLF.

Se genera un archivo llamado aligned.mlf que será usado desde ahora en vez de phones.mlf, ya que esta versión contiene operaciones de mapeo de palabra a fonema, lo cual ayuda a la precisión cuando hay más de una forma de pronunciar cierto fonema. También este comando genera un archivo llamado HVite_log, que al igual que dlog, debe ser revisado para poder continuar. En él se detallan todas las samples y si fueron procesadas exitosamente. Si una sample no pasó el proceso, se recomienda revisar el audio o si está bien escrito; si no hay errores, se debe quitar la ruta de los archivos train.scp y codetrain.scp y compilar nuevamente hasta este paso (si alguna de las palabras del diccionario aparece sólo en ese archivo, habrá también que sacarla del léxico y esto puede provocar que toda la gramática quede desbalanceada, por eso se reitera tanto en los primeros pasos que se deben tener varias palabras con la cantidad mínima de fonemas y luego repetirlas varias veces a la hora de grabar. Si esto ocurre se debe definir un nuevo diccionario conjunto de palabras y empezar desde cero). Aun no se sabe la causa de este posible error, pero puede tener que ver con la frecuencia de la voz que graba, o el ruido ambiental durante esa sample en particular. Con este nuevo archivo, se crean dos iteraciones más con HERest para generar dos hmmdefs con el nuevo archivo alineado:

"HERest -A -D -T 1 -C config -I aligned.mlf -t 250.0 150.0 3000.0 -S train.scp -H hmm7/macros -H hmm7/hmmdefs -M hmm8 monophones1

HERest -A -D -T 1 -C config -I aligned.mlf -t 250.0 150.0 3000.0 -S train.scp -H hmm8/macros -H hmm8/hmmdefs -M hmm9 monophones1"

Así concluye la creación de monofonos, este modelo puede ser ya usado en Julius, aunque no se garantiza su precisión.

12.3 Trifonos

Cuando se creó el archivo de diccionario, este contenía cada fonema para una palabra en forma de monofono (por ejemplo: ahora → a oo r a) en esta fase final se deben generar en formato de trifonos (ejemplo: ahora → A+oo A-oo+r oo-r+a r-a) donde el fonema anterior se escribe con un - y el fonema siguiente con un + (en los extremos la notación se invierte, ya que son bifonos). Se pueden crear trifonos dependientes de contexto clonando los monofonos y luego restimándolos usando transcripciones de trifonos. Esto último se debe crear primero usando HLEd, ya que su efecto secundario es generar una lista con todos los trifonos donde hay al menos un ejemplo en la información de entrenamiento (samples).

Primero se necesita un archivo .led que es similar al archivo .led anterior, solo que este se usará en esta parte para los trifonos:

```
WB sp
WB sil
TC
```

Los dos comandos WB definen a sp y sil como símbolos de límite de palabras. Estos bloquean la adición de contexto en el comando TI, visto en el siguiente script, que

convierte todos los fonemas (excepto los símbolos que sirven como límite de palabras) en trifonos. Por ejemplo:

sil aa l g i e n sp d ii x o sp ...

se vuelve

sil aa+l aa-l+g l-g+i g-i+e i-e+n e-n sp d+ii d-ii+x ii-x+o x-o sp ...

A este estilo de transcripción se le conoce como palabra interna. Nótese que algunos bifonos podrían generarse como contextos en los límites de palabras (al principio y al final), lo que solo incluirá 2 fonemas. Antes de continuar, se deben crear en la carpeta "manual" las carpetas "hmm10", "hmm11" y "hmm12".

Este archivo .led se llamará mktri.led. Luego con la herramienta HLEd ("HLEd -A -D -T 1 -n triphones1 -l '*' -i wintri.mlf mktri.led aligned.mlf") se pueden generar los archivos wintri.mlf y triphones1, los cuales contienen los mismos fonemas del modelo pero en formato de trifonos, esto significa que ahora wintri contiene lo escrito en las samples pero en forma de trifono, así como tripones contiene los mismos fonemas del archivo monophones1 en este formato. Junto con esto se necesita otro archivo llamado mktri.hed, que contiene un comando clon CL seguido de un comando TI para amarrar a todos los fonemas de las matrices de transición en cada conjunto de trifono, esto es:

CL triphones1

TI T_ah {(*-ah+*,ah+*,*-ah).transP}

TI T_ax {(*-ax+*,ax+*,*-ax).transP}

TI T_ey {(*-ey+*,ey+*,*-ey).transP}

TI T_b {(*-b+*,b+*,*-b).transP}

TI T_ay {(*-ay+*,ay+*,*-ay).transP}...

Este archivo se genera usando el script en Perl "maketrihed" que se encuentra en la carpeta "HTK_scripts" ejecutando lo siguiente:

"perl ../HTK_scripts/maketrihed monophones1 triphones1"

Después usando la herramienta HHEd se clona esto en "hmm10":

"HHEd -A -D -T 1 -H hmm9/macros -H hmm9/hmmdefs -M hmm10 mktri.hed monophones1"

Para luego restimarlo 2 veces en las carpetas "hmm11" y "hmm12" con la herramienta HERest:

"HERest -A -D -T 1 -C config -I wintri.mlf -t 250.0 150.0 3000.0 -S train.scp -H hmm10/macros -H hmm10/hmmdefs -M hmm11 triphones1

HERest -A -D -T 1 -C config -I wintri.mlf -t 250.0 150.0 3000.0 -s stats -S train.scp -H hmm11/macros -H hmm11/hmmdefs -M hmm12 triphones1“

Para terminar hay que hacer trifonos con estados "amarrados", esto se logra con la herramienta HDMan (“HDMan -A -D -T 1 -b sp -n fulllist -g global.ded -l flog dict-tri ../lexicon/spanish_lexicon”) que genera dict-tri y fulllist, este último siendo un conjunto con todos los monofonos y trifonos usados hasta ahora. Con estos archivos, se debe crear un nuevo archivo llamado “fulllist1” que tendrá los contenidos de los archivos fulllist y triphones1. Luego se debe crear el siguiente script de perl llamado fixfullist.pl en la carpeta “manual”:

```
#!/usr/bin/perl
##############################################################################
###     script name : fixfullist.pl
###         version: 1.0
###         created by: Ken MacLean
###         mail: contact@voxforge.org
###         Date: 2005.04.10
###     Command: perl ./fixfullist.pl
### Copyright (C) 2006 Ken MacLean
##############################################################################

use strict;
my ($line, $label, $filein, $fileout, %seen);
if (@ARGV != 2) {
  print "usage: $0 filein fileout\n\n";
  exit (0);
}
# read in command line arguments
($filein, $fileout) = @ARGV;

open (FILEIN,"$filein") || die ("Unable to open $filein for reading");
open (FILEOUT,">$fileout") || die ("Unable to open $fileout for writing");

%seen = ();
while ($line = <FILEIN>) {
  chomp ($line);
  unless ($seen{$line}) { # remove duplicate triphone names
              $seen{$line} = 1;
                print (FILEOUT "$line\n");
  }
}
close(FILEOUT);
```

Nota: este script no está incluido en la carpeta “HTK_scripts”.

Gracias a este script, el archivo fulllist contendrá todos los monofonos, y combinaciones de trifonos que aparecen en el modelo. Ahora se deben crear las últimas carpetas para la iteración del modelo, que serán “hmm13”, “hmm14” y “hmm15” en la carpeta “manual”.

Al tener esto, se debe crear un script llamado tree.hed que va a contener todos los monofonos y trifonos agrupados según el tipo de fonema (es aquí donde se agrupan los fonemas del español según su clasificación por articulación del fonema, esto es detallado en el punto 4.9 del informe), cabe destacar que mientras más clasificados estén los fonemas, mayor será la precisión con la que detectarán las palabras. El script tree.hed contiene las clasificaciones de los fonemas, mientras más QS (queries) hayan, mayor será la precisión con la que reconocerá las palabras. Algunos de los utilizados en el modelo contenido en el cd son:

```
QS  "R_"        { *+* }
QS  "R_Silence"         { *+sil }
QS  "R_Consonante"
  {+r,*+l,*+g,*+n,*+s,*+k,*+b,*+m,*+d,*+t,*+ny,*+p,*+ll,*+th,*+f,*+ch }
QS  "R_ConSonoras" { *+b,*+d,*+g,*+k,*+l,*+m,*+n,*+r,*+ll }
QS  "R_Sordas"          { *+f,*+k,*+p,*+s,*+th,*+t,*+x }
QS  "R_Bilabial"        { *+b,*+m,*+p }
QS  "R_Oclusiva"        { *+b,*+d,*+t }
QS  "R_Aproximante"             { *+b,*+d,*+g,*+k,*+r }
QS  "R_Africada"        { *+ch,*+Y }
QS  "R_Interdental"     { *+ch,*+t,*+th }…
```

Luego con el siguiente script de perl (que sí está incluido en la carpeta “HTK_scripts”): “perl ../HTK_scripts/mkclscript.prl TB 350 monophones0 >> tree.hed”, se generarán más reglas con los trifonos:

```
TB 350 "ST_a_2_" {("a","*-a+*","a+*","*-a").state[2]}
TB 350 "ST_A_2_" {("A","*-A+*","A+*","*-A").state[2]}
TB 350 "ST_s_2_" {("s","*-s+*","s+*","*-s").state[2]}
...
TB 350 "ST_a_3_" {("a","*-a+*","a+*","*-a").state[3]}
TB 350 "ST_A_3_" {("A","*-A+*","A+*","*-A").state[3]}
TB 350 "ST_s_3_" {("s","*-s+*","s+*","*-s").state[3]}
....
TB 350 "ST_a_4_" {("a","*-a+*","a+*","*-a").state[4]}
TB 350 "ST_A_4_" {("A","*-A+*","A+*","*-A").state[4]}
TB 350 "ST_s_4_" {("s","*-s+*","s+*","*-s").state[4]}
```

Para terminar con el archivo tree.hed, se debe agregar al final del archivo lo siguiente:

TR 1
AU "fulllist"
CO "tiedlist"

ST "trees"

Donde el comando CO se usa para compactar el modelo quitando los valores repetidos y amarrándolos de a 5, generando una nueva lista de modelos llamada "tiedlist". Una de las ventajas de agrupar el árbol es que permite sintetizar trifonos que no habían sido captados, para esto, el comando ST guarda los arboles en un archivo "trees". Si después se requieren trifonos no encontrados, por ejemplo en la pronunciación de una nueva palabra, se puede cargar el modelo existente en HHed, los árboles se van cargando de nuevo usando el comando LT que crea una nueva lista extendida de los trifonos creados. Finalmente, se crean en la carpeta "manual" las ultimas 2 iteraciones y se ejecuta HERest:

"HERest -A -D -T 1 -T 1 -C config -I wintri.mlf -s stats -t 250.0 150.0 3000.0 -S train.scp -H hmm13/macros -H hmm13/hmmdefs -M hmm14 tiedlist

HERest -A -D -T 1 -T 1 -C config -I wintri.mlf -s stats -t 250.0 150.0 3000.0 -S train.scp -H hmm14/macros -H hmm14/hmmdefs -M hmm15 tiedlist"

En la siguiente figura se resume los pasos explicados con anterioridad, señalando las respectivas carpetas involucradas con los monofonos y trifonos:

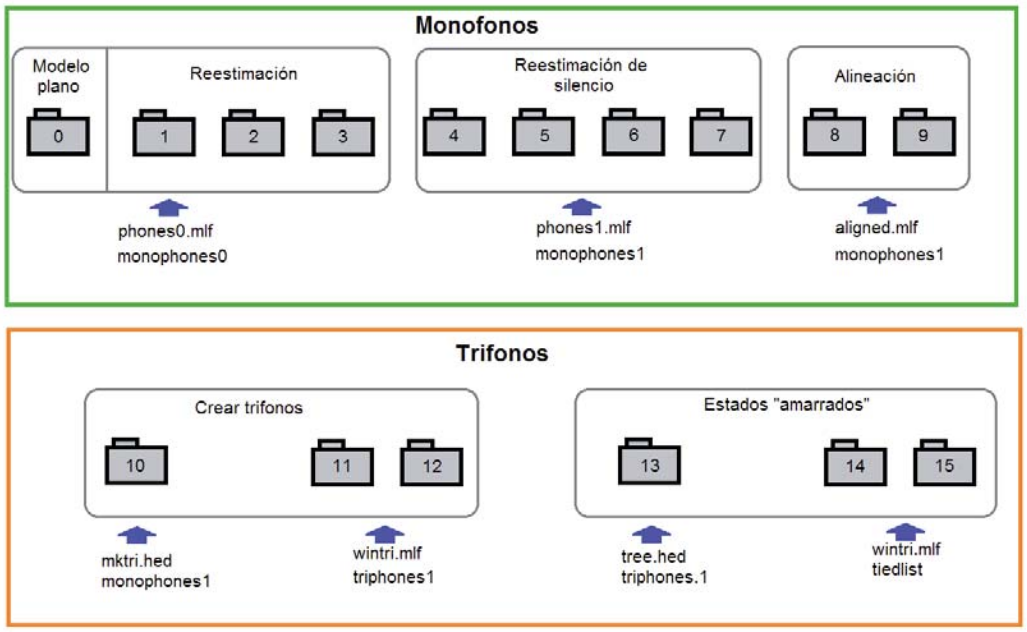

Figura 12-1 Carpetas involucradas en el desarrollo del modelo del idioma español

12.4 Ejecutar en Julius

Después de la creación de las 16 iteraciones de Markov (incluyendo la primera), el modelo está listo para ser probado en Julius, para eso se utilizan los archivos que fueron cambiados a formato Julius en un principio (los archivos .dfa y .term), luego son compilados en Julius. Para correr el modelo en Julius se necesita de un archivo .jconf que debe contener las siguientes configuraciones:

Forma normal de especificar una gramática.

-dfa sample.dfa

-v sample.dict

-h hmm15/hmmdefs

El modelo de trifonos necesita HMMList que mapea los trifinos lógicos en físicos.

-hlist ./tiedlist

Penalización por inserción de palabra

-penalty1 5.0 # primer paso

-penalty2 20.0 # segundo paso

(PTM/trifono) cambia el método de cálculo de IWCD en el primer paso

-iwcd1 max # asigna probabilidad máxima al mismo contexto.

selecciona el algoritmos de poda gausiano

Default: beam (configuración estándar), safe (otras).

-gprune safe # podas segura, precisa pero lenta.

-b2 200 # ancho de haz en el Segundo (numero de palabras)

-sb 200.0 # resultados del haz sobre el umbral

Especifica el nombre de la pausa corta para tratarla de manera especial.

-spmodel "sp" # nombre del modelo HMM

Para inserción de pausas libres de contexto, cortas y entre palabras.

-iwsp # añade un modelo sp que puede ser obviado al final de todas las palabras

-iwsppenalty -70.0 # transición de la penalización para los modelos sp añadidos.

-smpFreq 48000 # ratio de las samples (Hz)

Nota: estas son las opciones principales y necesarias por default para correr el modelo en Julius, existen muchas más para configurarlo de distintas formas [6].

La siguiente figura señala las herramientas del HTK que son utilizadas en Julius, para el reconocimiento de la voz:

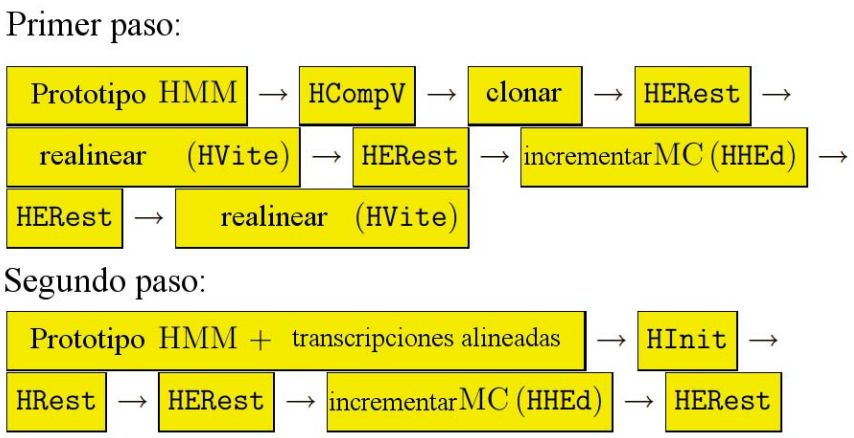

Figura 12-2 Herramientas del HTK que son utilizadas en Julius

Con el comando escrito en la terminal de Ubuntu "$julian -input mic -C julian.jconf" (donde julian.jconf es el archivo de configuración señalado arriba), se puede poner en marcha el modelo en Julius.

13 Plan de Prueba

13.1 Diseño de Plan de Prueba.

El diseño de Plan de Prueba estará basado en los requerimientos del proyecto especificados anteriormente, en el buen reconocimiento de las palabras que son utilizadas para el diccionario del idioma español (registrando el porcentaje de palabras acertadas) y la buena documentación del proyecto.

Serán evaluados los modelos acústicos y lingüísticos desarrollados del idioma español, para detectar cuáles son las palabras más reconocidas y las menos reconocidas, con el fin de especificar posibles falencias del modelo, para futuras correcciones.

Durante la implementación del modelo de lenguaje, en cada paso requerido, se realizaron pruebas de caja negra, esto quiere decir que es prueba en tiempo de ejecución de cada etapa del modelo, se revisó algunos archivos del HTK para verificar el problema de la codificación de las palabras, algunos caracteres, como la "ñ", "ü", tildes, etc. cambiaban a código ASCII al pasar (por ejemplo la ñ, pasaba a ser 164).

Como fue mencionado anteriormente, el diseño e implementación del proyecto es poco convencional; por lo tanto, el diseño de plan de pruebas también lo es, dada esta situación se consideran las siguientes pruebas:

13.1.1 Prueba modelo acústico y lingüístico

Es evaluado el reconocimiento de voz por cada palabra del diccionario del modelo acústico y lingüístico creado. Cada palabra tendrá que ser pronunciada 5 veces, por personas del sexo femenino y masculino respectivamente (los simples del modelo de lenguaje fueron grabados por voces de ambos sexos), que no estén involucradas en las grabaciones de los samples, los resultados serán registrados como porcentaje para reconocer aquellas palabras que fueron menos reconocidas. Luego, se analizarán las palabras y los fonemas de estas que fueron poco reconocidas, para identificar cual es el posible problema que requiere de mejor clasificación de fonema, ya sea más cantidad de grabaciones de samples, la existencia de otra palabra muy similar que produce confusión cuando reconoce la voz o por ser sonidos más complejos de identificar por Julius. Esta prueba fue utilizada en el modelo final de proyecto.

Tabla 123.12.1 Plan de Prueba de modelo acústico y lingüístico

NOMBRE	Prueba modelo acústico y lingüístico	**PRUEBA:**	P1
PROPÓSITO	Verificar el correcto reconocimiento de cada palabra del diccionario del modelo de lenguaje. En caso de encontrar palabras con bajo reconocimiento, identificar el fonema que está causando problemas.		
PRERREQUISITOS	• Modelo del español realizado. • Listado de palabras del diccionario del modelo de lenguaje.		

	• Hoja con resultados (la cual es llenada por la persona que realiza la prueba). • 2 personas (hombre y mujer).
UBICACIÓN	Carpeta del modelo de lenguaje.
ENTRADA	• Modelo acústico • Modelo lingüístico
PASOS	• Acceder a la carpeta en que se encuentra el proyecto (con ayuda del comando "cd nombre_carpeta"). • Ejecutar con el comando "julius -input mic -C julian.jconf", en la carpeta del proyecto. • Junto con el listado de palabras, mencionar cada una de ellas 5 veces, y anotar en la hoja de resultado si la palabra fue reconocida y por cual paso fue (paso 1 o 2). • Finalizar presionando "Ctrl+c". • Analizar los resultados y verificar los fonemas menos reconocidos,

13.1.2 Prueba de modelo parcial

Esta prueba es realizada por aquellas personas que realizan el modelo de lenguaje. Es similar a la prueba descrita con anterioridad; es decir, en una primera etapa, se pronuncia cada palabra del diccionario 10 veces, luego todos los resultados son registrados. La prueba fue realizada cada vez que se finalizaba un modelo de lenguaje, con el fin de encontrar falencias en el modelado tempranamente, como comparar distintos tipos de transcripción fonéticas. Si más del 70% de las palabras registradas en el diccionario son detectadas satisfactoriamente (es decir, sobre 6 veces), se da paso a concretar la "Prueba de modelo lingüístico y acústico".

Tabla 123.12.2 Plan de Prueba de modelo parcial

NOMBRE	Prueba de modelo parcial	**PRUEBA:**	P2
PROPÓSITO	Verificar el correcto reconocimiento de cada palabra del diccionario del modelo de lenguaje. Comparar distintas transcripciones fonéticas (al realizar esta prueba al mismo modelo, pero con diferente transcripción). Si el 70% de las palabras del diccionario pasan la prueba (es decir, ser detectadas sobre 6 veces), se considera que el modelo es lo suficientemente bueno como para realizarle P1.		
PRERREQUISITOS	• Modelo preliminar del español realizado. • Listado de palabras del diccionario del modelo de lenguaje. • Alguna de las personas que grabo su voz para el sample, debe realizar la prueba.		
UBICACIÓN	Carpeta del modelo de lenguaje.		
ENTRADA	• Modelo acústico • Modelo lingüístico		
PASOS	• Acceder a la carpeta en que se encuentra el proyecto (con ayuda del comando "cd nombre_carpeta"). • Ejecutar con el comando "julius -input mic -C julian.jconf", en la carpeta del proyecto. • Junto con el listado de palabras, mencionar cada una de ellas 10 veces.		

	• Anotar los resultados obtenidos. • Finalizar presionando "Ctrl+c". • Analizar los resultados.

13.2 Implementación de Plan de Prueba.

A continuación, se analizarán los resultados de las últimas pruebas realizadas.

13.2.1 Resultados de "Prueba de modelo parcial"

Durante el desarrollo del proyecto se realizaron 3 pruebas de modelos parciales (corregir), las cuales sirvieron como base de la creación del modelo final. Las 2 primeras pruebas fueron la comparación entre 2 modelos que contenían las mismas palabras, pero con diferente transcripción fonéticas (en el anexo A se puede ver las tablas de las distintas transcripciones fonéticas utilizadas en el proyecto), los resultados obtenidos indicaron que tener más fonemas no hace el modelo más preciso, si no más bien la buena y precisa clasificación de estos fonemas, lo cual es más importante en la creación de un modelo de lenguaje (es allí la importancia del estudio de la transcripción fonética del idioma español). También se detectó que palabras similares eran fácilmente confundidas entre sí en el modelo, para solucionar eso, es necesario utilizar palabras más variadas y mayor grabación de samples.

Respecto a la grabación de samples, se dio la necesidad de grabar muchos samples con voces de diferentes personas, este proceso conlleva un gran costo de tiempo, ya que no sólo se considera el tiempo de quienes crean los modelos, sino también de quienes tienen disponibilidad de tiempo para grabar estos samples. En el caso del último modelo desarrollado de 107 palabras, el tiempo mínimo empleado para grabar una persona fue de 30 minutos (si no ocurría ningún tipo de inconveniente durante la grabación, como ruido ambiental, problemas con el micrófono, etc.). También fueron grabadas 6 voces masculinas y 6 femeninas.

En esta prueba, se utilizó la voz de una de las personas que grabó su voz para el modelo. En el anexo D del informe, están los resultados por palabras del último modelo. Resumiendo los resultados, se obtiene la siguiente información:

Tabla 13.3 Resumen resultado prueba de modelo parcial

cantidad de palabras detectadas
10 veces = 60 palabras
9 veces = 13 palabras
8 veces = 15 palabras
7 veces = 4 palabras
6 veces = 3 palabras
5 veces = 3 palabras
4 veces = 1 palabra
3 veces = 1 palabra
2 veces = 2 palabras
1 vez = 2 palabras
0 veces = 3 palabras

Respecto a los resultados de la tabla 7, se destaca que 56,1% de las palabras del diccionario fueron detectadas totalmente bien, mientras que solo el 2,8% no fue detectado. Si se considera que la palabra reconocida 6 veces o más, es un resultado satisfactorio, se llega a que el 88,8% de las palabras cumplen con esta consideración; es decir, que el modelo es lo suficientemente bueno como para ser sometido a la prueba del modelo lingüístico y acústico.

13.2.2 Resultados de "Prueba de modelo lingüístico y acústicos"

Más del 70% de las palabras registradas en el diccionario fueron detectadas con la prueba anterior, por lo que el modelo es considerado lo suficientemente bueno como para hacer la prueba final. Los resultados obtenidos son los siguientes:

Tabla 13.4 Resumen resultado prueba modelo acústico y lingüístico

Voz femenina:	**Voz masculina**
5 veces = 57 palabras 4 veces = 13 palabras 3 veces = 10 palabras 2 veces = 3 palabras 1 vez = 5 palabras 0 veces = 19 palabras	5 veces = 24 palabras 4 veces = 26 palabras 3 veces = 23 palabras 2 veces = 10 palabras 1 vez = 12 palabras 0 veces = 12 palabras

En el anexo E se muestra el resultado de la prueba por palabra. Respecto a los resultados de la tabla 8, se destaca que el modelo detecta mejor la voz femenina que la masculina, logrando el 53,2% del diccionario de palabras acertadas correctamente, en comparación del 22,4% de acierto de la voz masculina. Si se considera que la palabra reconocida 3 veces o más, es un resultado satisfactorio, se llega a que el 74,8% de las palabras dichas por la voz femenina cumplen con esta consideración, mientras que 68,2% de las palabras dichas por la voz masculina cumplen con esta consideración.

A pesar de que la voz femenina es mejor detectada, la voz masculina tiene menos casos de palabras que no son reconocidas, es decir, solo 11,2% de las palabras del diccionario no fueron reconocidas con la voz masculina, mientras que el 17,7% de las palabras del diccionario no fueron reconocidas por la voz femenina. Entre las palabras que no fueron reconocidas por ambas voces, están: buen, casa, ha, ven, va, ya, hace, nada, padre, las cuales coinciden también en los resultados de la prueba parcial en que las palabras "ha, va y ya", no son detectadas de ninguna manera, probablemente ocurre este problema al ser palabras "similares", generalmente es confundida con la palabra "tan" (la cual es detectada satisfactoriamente por las 3 voces)

Este resultado puede mejorar si son grabados más samples con voces más variadas (para mejorar la precisión del reconocimiento de la voz). Aun así, es conveniente reconocer que un modelo de 107 palabras con samples de 12 voces diferentes, logran un modelo bastante bueno, en que logra un acierto sobre el 50% de las palabras registradas, con margen bajo el 20% de desacierto al detectar las palabras.

14 Julius en otros medios

Se han realizado varias aplicaciones usando a Julius como motor de reconocimiento de voz, algunas de estas incluyen gestores de diálogo, que son componentes para los sistemas de reconocimiento de voz, que se encargan de recibir lo que el usuario dice y dependiendo de la entrada el gestor verá la mejor solución o lo que se hará después. Por ejemplo, en un software de viajes se le pregunta al usuario adonde quiere ir y este puede decir "el lugar", "al lugar, por favor" o sólo "lugar". La tarea del gestor es estar preparado para cualquiera de estas respuestas.

Generalmente, donde más se han utilizados los gestores de diálogo son en aplicaciones de dictado y telefonía, Simon [20] y Kiku [21] por ejemplo, son programas de reconocimiento de voz open source usados para reemplazar mouse y teclado o líneas de comandos, por voz. Asimismo, Julius se ha visto usado para reemplazar comandos en PHP para ingresar consultas SQL, en Bash para usarlo en conjunto con scripts, en Python para dar comandos mediante reconocimiento de voz o controlando brazos robot [22] y en java para aplicaciones Android. Una de estas aplicaciones es "Shiri-tori intellectual, let's!" [23], una aplicación de Android que consiste en un juego educativo para preescolares, donde mediante el juego de shiri tori (un juego en donde se dice una palabra y uno debe empezar la siguiente palabra con la última sílaba de la palabra anterior) se dice la palabra y esta sale escrita.

Otro gestor de diálogo es Galatea [24], el cual fue creado con lib-Julius. Se ha visto también que varios usuarios o desarrolladores usan Julius para programar, dar comandos en sitios web o simplemente para dejar notas.

15 Futuras mejoras

Dentro de las mejoras que se le podrían hacer a este proyecto, la mayoría son iteraciones que harán al modelo más robusto y completo:

- Agregar más palabras, o crear un modelo con las palabras de una temática en especial que satisfagan las necesidades del usuario (ejemplo: un script que maneje un reproductor de video, necesitará palabras como play, stop, etc.)

- Agregar más samples, esto es, conseguir más voces para que así el modelo sea consistente con cualquier tipo de voz y no sólo con algunas agudas o algunas graves. También considerando los distintos ambientes donde será usado el modelo, por lo que las grabaciones deben ser hechas en distintos lugares, con distintas personas y con palabras dichas de distinta manera.

- Un error frecuente fue detectado en el paso de amarrar los monofonos, el error fue detectado con la herramienta HVite y se podía ver en el archivo HVite_log. El error consiste en que algunos archivos de audio no pasaban en ninguna medida la beam search, cuando eran revisados no tenían mayores diferencias que los otros archivos de audio, pero por alguna razón no eran procesados correctamente, por lo que al final fueron simplemente removidos.

- Mayor información sacada del HTKbook para mejorar el modelo, ya que el HTKbook contiene información adicional que no fue usada en este informe, la cual podría permitir mejorar o configurar de mejor forma el modelo.

- Utilizar el modelo para algún fin en específico, como el de una aplicación, etc. (como los descritos en el punto anterior, de Julius en otros medios).

16 CONCLUSIÓN

Luego de la investigación sobre el funcionamiento de Language Grid y la importante labor que cumple al ofrecer servicios que buscan una mejor comunicación intercultural (en distintos ámbitos, tales como medicina, control de desastres, educación, entre otros), se enfocó el trabajo en el área del reconocimiento de habla al español.

El software afiliado a LG, Julius, resulta ser un potente decodificador de reconocimiento de voz, que cuenta con varias técnicas de búsqueda que optimizan su rendimiento (como Hidden Markov Model y el modelo N-grama), logrando un reconocimiento de voz casi en tiempo real.

En un principio se pensó que la transcripción fonológica del japonés (visto en el diccionario del modelo japonés de Julius) iba a ser bastante parecida a la transcripción al español, pero no fue así, debido a que el español tiene muchas reglas y diversas formas de pronunciación que dificultó el proceso de la transcripción (invirtiendo la mayoría del tiempo de trabajo en esta parte, junto a la prueba de herramientas del HTK).

Como este proyecto es poco usual, se considera que el desarrollo mismo también lo es, debido a que se trabajó sobre un software existente. Por esta misma razón, la implementación de este proyecto fue enfocada a hacer que Julius funcione con el idioma español.

Se realizaron pruebas durante la finalización de cada uno de los modelos preliminares, lo cual sirvió como base para analizar la transcripción fonética realizada, y para analizar la cantidad de grabaciones mínimas necesarias para la creación de un buen modelo.

Se logró la integración de los fonemas “ñ”, “ü” y tildes (á,é,í,ó,ú), lo cual era la identificación de archivo en que se encontraban las palabras de salida del programa, este fue encontrado y aceptó los caracteres que no eran aceptados por el HTK durante las iteraciones.

Fueron creados varios prototipos para aprender con cada iteración qué problemas se pueden encontrar durante su desarrollo, entre ellas la implementación de la "ñ" y tildes, lo cual era la identificación del archivo en que se encontraban las palabras de salida del programa. Otro problema que se detectó fue la precisión con la que se reconocen las palabras, para ello se investigó sobre la fonética y fonología del idioma español y se procedió a grabar gran cantidad de samples con diversas voces masculinas y femeninas.

En el modelo final del proyecto, de 107 palabras, se utilizó una transcripción detallada, basado en los resultados obtenidos en las pruebas de los modelos parciales realizados anteriormente. Fueron utilizadas 12 voces distintas (6 femeninas y 6

masculinas), obteniendo un modelo que permite un acierto sobre el 80% de las palabras utilizadas en él.

Con los resultados obtenidos en las pruebas, ya se puede inferir como debe ser un buen modelo (que reconozca sobre el 50% de las palabras y con un margen de error bajo el 20%), al ser grabadas 12 voces diferentes se logra un modelo de buenas características, el cual, podría ser ampliado o ser utilizado para otros fines, como los investigados en la manera en que se puede utilizar Julius para otros fines (software, proyecto, etc.).

También se documentó los pasos para realizar un modelo de lenguaje en español, para que sea utilizado como apoyo para otros alumnos que necesiten crear uno en futuros proyectos.

Finalmente es importante y necesario mencionar, que este proyecto es una de las etapas claves y críticas para la finalización de la formación como futuros ingenieros de ejecución en informática, ya que fueron utilizadas muchas de las herramientas y técnicas que se han aprendido durante los años de estudio, y que se pretende aplicar en el futuro como profesionales.

17 REFERENCIAS

[1] Listado de frecuencia de palabras. RAE. Disponible vía web en http://corpus.rae.es/lfrecuencias.html

[2] Language Grid: Infraestructura para la colaboración intercultural. Toru Ishida, profesor del Departamento de Informática Social. Paper publicado el año 2005. Disponible vía web en http://www.ai.soc.i.kyoto-u.ac.jp/~ishida/pdf/ieee06i.pdf

[3] Toru Ishida, profesor del Departamento de Informática Social. Línea de Investigación "Proyecto Language Grid". Reseña escrita por Akira Miki el 11 de diciembre del 2009. Disponible vía web en http://www.astem.or.jp/virtual-lab/culture_en/research/r_kenkyu3

[4] The Language Grid. Información del año 2011. Disponible vía web en http://langrid.org/file/TheLanguageGrid-en.pdf

[5] T.Kawahara, A.Lee, T.Kobayashi, K.Takeda, N.Minematsu, S.Sagayama, K.Itou, A.Ito, M.Yamamoto, A.Yamada, T.Utsuro y K.Shikano. Free Software Toolkit for Japanese Large Vocabulary Continuous Speech Recognition. Publicado en el año 2000. Disponible vía web en http://www.ar.media.kyoto-u.ac.jp/EN/bib/intl/KAW-ICSLP00.pdf

[6] Julius. Sitio web oficial del software. Información del año 2011. Disponible vía web en http://julius.sourceforge.jp/en_index.php

[7] Akinobu Lee, Tatsuya Kawahara, Kiyohiro Shikano. Julius- an Open Source Real-Time Large Vocabulary Recongnition Engine. Publicado en el año 2001. Disponible vía web en http://julius.sourceforge.jp/paper/ri-eurospeech2001.pdf

[8] Marcos Antoine Jorquera Morales ex estudiando de la PUCV. Reconocimiento de voz. Memoria de título del año 2007, página 11-12.

[9] Daniël de Kok, Harm Brouwer. Natural Language Processing for the Working Programmer. Del año 2010-2011. Disponible vía web en http://www.nlpwp.org/nlpwp.pdf

[10] Johan Schalkyk. Word models, Lexical trees and grammars. Del año 1996. Disponible vía web en http://www.cslu.ogi.edu/toolkit/old/old/version2.0a/documentation/csluc/node7.html

[11] Proyecto Voxforge. Del año 2006-2012. Disponible vía web en http://voxforge.org/

[12] Andr´e Mansikkaniemi. Acoustic Model and Language Model Adaptation for a Mobile Dictation Service. Publicado en el año 2010. Disponible vía web en http://lib.tkk.fi/Dipl/2010/urn100143.pdf

[13] Departamento de Ingeniería de la Universidad de Cambrigde. HTK. Última

actualización en el año 2006. Disponible vía web en http://htk.eng.cam.ac.uk/

[14] British Museum Search. Disponible vía web en http://bradley.bradley.edu/~chris/searches.html/

[15] Xabier Frías Conde. Introducción a la fonética y fonología del español. Revista Philologica Romanica, Suplemento 04, 2001. Disponible vía web en http://www.romaniaminor.net/ianua/sup/sup04.pdf

[16] Free Software Foundation. Licencias de software. Licencia del 2011. Disponible vía web en http://www.gnu.org/licenses/license-list.es.html

[17] Departamento de Ingeniería de la Universidad de Cambrigde. Licencia del software HTK, 1992. Disponible vía web en http://htk.eng.cam.ac.uk/docs/license.shtml

[18] Departamento social de informática, Universidad de Kyoto, 2007-2011. Unirse a Language Grid. Disponible vía web en http://langrid.org/operation/en/procedure.html

[19] Winston Manrique Sabogal, Lo que hay que saber del Español, 2010. Disponible vía web en http://elpais.com/diario/2010/11/27/babelia/1290820336_850215.html

[20] "Simon", ProjectTeam Simon, Franz Stieger), 2007-2012. Disponible vía web en http://simon-listens.org/index.php?id=396&L=1

[21] "Kiku",Patrick Sebastien, 2010-2012. Disponible vía web en http://www.workinprogress.ca/kiku/

[22] "Robot arm with voice Control". Disponible vía web en http://www.aonsquared.co.uk/robot_arm_tutorial_1

[23] "Shiri tori and Android with Julius, Speech recognition engine", MTI corporation japan, 2011. Disponible vía web en http://www.moedroid.jp/11154.html

[24] "Galatea evolution and promotion of spoken dialogue technology", Galatea project, 2004. Disponible vía web en http://hil.t.u-tokyo.ac.jp/~galatea/

[25] José Luis Lara Carrascal, Instalar GCC desde cero, 2012. Disponible vía web en http://manualinux.heliohost.org/gcc.html

[26] “Audacity”, 2012. Disponible vía web en http://audacity.sourceforge.net/

[27] “Licencia de Software Julius”, 1999-2011. Disponible vía web en http://julius.sourceforge.jp/LICENSE.txt

18 ANEXOS

Anexo A: Lista de fonemas, que pueden ser utilizados en el modelo de lenguaje.

Vocales:

Tabla 18.1 Vocales para el modelo de lenguaje

Grafema	Fonema	Clasificación de fonemas	Ejemplo
a, á	/a/: otros casos. /:a/: antes de consonantes palatales (ch, /L/, /:n/, ñ, /Y/) y diptongo /ai/. /a:/: seguida de vocal posterior. Ante consonante fricativa velar (j) y en silaba trabada (au) por consonante lateral alveolar. /A/: en posición inicial antes de una consonante nasal (n, m, ñ) y entre dos consonantes nasales.	/a/: vocálica, central, abierta, no consonántico, no grave. /:a/: vocálica, central, abierta, palatalizada, no consonántico, no grave. /a:/: vocálica, central, abierta, velarizada, no consonántico, no grave. /A/: vocálica, nasalizada, no consonántico, no grave.	/a/: casa = k a s a /:a/: aire = :a J r e /a:/: flauta = f l a: W t a /A/: antena=A N t e n a
o, ó	/o/: otros casos. /o:/: en contacto con /r/, antes de /j/, en el diptongo, en silaba trabada, en el grupo a + ó + r/l. Antes de una consonante fricativa velar sorda (j) /O/: en posición inicial antes de una consonante nasal (n, m, ñ), y entre dos consonantes nasales.	/o/: vocálica, no consonántico, grave, posterior, semicerrada. /o:/: vocálica, no consonántico, grave, posterior, semiabierta. /O/: vocálica, no consonántico, grave, nasalizada.	/o/: coche = K o C e /o:/: ahora= a o: r a /O/ hombre= O m b r e
e, é	/e/: otros casos. /e:/: en contacto con /R/, /r/, antes de /j/, en diptongo, en silaba trabada por una consonante diferente a /m, n, s, d, z/ /E/: en posición inicial antes de una consonante nasal (n, m, ñ), y entre dos consonantes nasales.	/e/: anterior, semicerrada, vocálica, no consonántica, no grave. /e:/ anterior, semiabierta /E/: vocálica, nasalizada, no consonántico, no grave.	/e/: pescar= p e s k a r /e:/: perro= p e: R o /E/: entonces= E N t o n s e s
i, í, y	/i/: otros casos /i:/: en contacto con /R/, en silaba trabada, ante fricativa velar sorda (j, n). /I/: en posición inicial antes de	/i/: vocálica, no consonántica, no grave, anterior, cerrada. /i:/: vocálica, no consonántica, no grave, anterior, abierta /I/: vocálica, no consonántica, no	/i/: piso = p I s o /i:/ rico = R i: k o /I/: invierno = I N b J e: n o

	una consonante nasal (n, m, ñ), y entre dos consonantes nasales.	grave, nasalizada.	
u, ú	/u/: otros casos. /u:/ : ante fricativa velar sorda (j), en contacto con /R/. en silaba trabada. /U/: en posición inicial antes de una consonante nasal (n, m, ñ), y entre dos consonantes nasales.	/u/: grave, vocálica, no consonántica, posterior, cerrada, redondeada. /u:/: grave, vocálica, no consonántica, posterior, abierta, redondeada, /U/: nasalizada, vocálica, no consonántica, grave.	/u/: tuvo = t u b o /u:/ rubio = R u: b J o: /U: unto = U N t o

Diptongos:

Tabla 18.2 Diptongos para el modelo de lenguaje

Grafema	Fonema	Clasificación de fonemas	Ejemplo
Creciente: ie, ia, io, iu Decreciente: ei, ai, oi	/J/: creciente /J:/: decreciente	/J/: anterior, semiconsonante, creciente, sonora. /J:/: anterior, semivocal, decreciente, sonora.	/J/: tierra = t J e R a /J:/: rey= R e: J:
Creciente: ui, ue, ua, uo Decreciente: eu, au, ou	/W/: creciente /W:/: decreciente	/W/: creciente, sonora, posterior, semiconsonante. /W:/: decreciente, sonora, posterior, semivocal.	/W/: cuota = k W o t a /W:/ fauna= f a: W: n a

Consonantes:

Tabla 18.3 Consonantes para el modelo de lenguaje

Grafema	Fonema	Clasificación de fonemas	Ejemplo
b, v	/b/: precedida de consonante nasal, o inicio palabra, después de pausa, después de consonante nasal. /b:/: otros casos. /B/: posición final de silaba inicial.	/b/: sonora, bilabial, oclusiva /b:/: sonora, bilabial, aproximante. /B/: bilabial, aproximante, ensordecida.	/b/: balde = b a l d e /b:/: cabo= k a b: o /B/ absoluto: a B s o: l u t o
ch	/C/	Africada, interdental, sorda, palatal.	/C/: chancho = C a ~n C o

d	/d:/: precedida de una consonante nasal o latera. Inicio palabra. /d/: otros casos /D/: posición final de silaba aproximante, ensordecida.	/d:/: Sonora, dental, oclusiva /d/: Sonora, dental, aproximante /D/: dental.	/d:/: andes = a N d: e s /d/: cada= k a d a /D/ ciudad= s J u d a D
f	/f/	/f/: Fricativa, sorda, labiodental.	/f/: café= k a f e
G (a, o, u), gu (e, i)	/g/: precedida de una consonante nasal, o inicio palabra /g:/: posición inicial de silaba después de vocal o de consonante distinta de nasal. /G/: posiciónn final de silaba.	/g/: Oclusiva, sonora, velar, /g:/: velar, aproximante, sonora. /G/: velar, aproximante, ensordecida.	/g/: gato = g a t o /g:/ hago= a g: o /G/: zigzag= s i G s a G
c	/c/: antes de la m, n, ñ, t, v (los otros casos son con K), o si después l, r	Sonora, velar, aproximante.	/c/: recta = r e c t a /c/: claro = c l a r o
k, ca, co, cu, qu-	/k/	Oclusiva, sorda, velar.	/k/: blanco = b l a ~n k o
l	/l/: otros casos /:l/: seguida de consonante interdental (s, z) /l:/: seguida de consonante dental (d, t) /L/: seguida consonante palatal (ch, y)	/l/: lateral, sonora, alveolar. /:l/: lateral, interdental, sonora. /l:/: lateral, dental, sonora. /L/: lateral, palatalizada, sonora.	/l/: ala = a l a /l~/: alzar = a l~ s a r /l:/: alto = a l: t o /L/: colcha = k o L C a
m	/m/	Nasal, Sonora, bilabial.	/m/: amigo = a m I g o
n	/n/: seguida de consonante alveolar (r, s, l), vocal (a, e, i, o, u) o inicio	/n/: nasal, alveolar, sonora /n~/: nasal labiodental sonora. /n:/: nasal interdental sonora.	/n/: nene = n e n e /n~/: anfora = a n~ f o r a /n:/: anzuelo = a n: s W e l o

	palabra. /n~/: seguida de consonante labiodental (f,) /n:/: seguida de consonante interdental (z) /N/: seguida de consonante dental (d, t) /:n/: seguida de consonante palatal (ch, y) /~n/: seguida de consonante velar (cu, ca, g) /N~/: seguida de consonante uvular (j)	/N/: nasal dental sonora. /:n/: nasal palatizada, sonora. /~n/: nasal velarizada, velar, sonora. /N~/: nasal uvular sonora.	/N/: undido = u N D i d o /:n/: ancho = a :n C o /~n/: tango = t a ~n g o /N~/: enjuto = e N~ j u t o
ñ	/ny/: ñ	/ny/: nasal, palatal, sonora.	/ny/: cariño = k a r i ny o
p	/p/	Oclusiva, bilabial, sorda.	/p/: Japón = j a p o n
r, rr	/r/, /R/	/r/: vibrante, sonora, alveolar, rotica simple, aproximante /R/: vibrante, sonora, alveolar, rotica, múltiple.	/r/: tres = t r e s /R/: departamento = d: e p a R t a m e N t O
S, c (e, i)	/s/: inicio y final de palabra, seguida de consonante sorda (f, j, p) excepto /t/, /z/ o de pausa. /S/: posición final de silaba seguida de consonante sonora (t, n, /c/, b, /Y/), excepto /d/, o una palatal sonora (/:n). /s:/: posición final de silaba, seguida de /z/, o /t/. /:s/: posición final de silaba, seguida de /d/.	/s/: alveolar, fricativa, sorda. /S/: alveolar, fricativa, sonora. /s:/: dental, fricativa, sorda /:s/: dental, fricativa, sonora.	/s/: dos = d: o s /S/: asma = a S m a /s:/ hasta = a s: T a /:s/ desde = d: e :s d e /:s/ juzgar= j u :s g: a r

t	/t/: otros casos. /T/: inicia palabra con silaba, después de T	/t/: Oclusiva, sorda, dental. /T/: interdental, oclusiva, sorda	/t/: tirano = t i r a n o /T/: hazte = a s: T e
J, g- (e, i)	/j/	Fricativa, velar, sorda.	/j/: jamón = j A m O n
y- (a, e , i, o u), ll	/y/: otros casos. /Y/: posición inicial de silaba después de /n/ o /l/, o después de pausa	/y/: palatal, aproximante, fricativa, sonora. /Y/: Africada, Sonora, palatal, lateral.	/y/:ayer = a y e: r /Y/: yeso = Y e s o /Y/: ella = e Y a

Anexo B: Lista de fonemas, que fueron utilizados en el modelo acústico y lingüístico final.

Vocales:

Tabla 18.4 Vocales para el modelo de lenguaje final.

Grafema	Fonema	Clasificación de fonemas	Ejemplo
a, á	/a/: otros casos. /aa/: sonido más extendido.	/a/: vocálica, central, abierta, no consonántico, no grave. /aa/: vocálica, central, abierta, velarizada, palatalizada, no consonántico, no grave, nasalizada.	/a/: cosas = k oo s a s /aa/: sera = s e r aa
o, ó	/o/: otros casos. /oo/: sonido más extendido. .	/o/: vocálica, no consonántico, grave, posterior, semicerrada. /oo/: vocálica, no consonántico, grave, posterior, semiabierta, nasalizada.	/o/: años = a ny o s /oo/: hombre= oo m b r e
e, é	/e/: otros casos. /ee/: sonido más extendido.	/e/: anterior, semicerrada, vocálica, no consonántica, no grave. /ee/ anterior, semiabierta vocálica, nasalizada, no consonántico, no grave.	/e/: antes= aa n t e s /ee/: tres= t r ee s
i, í, y	/i/: otros casos /ii/: sonido más extendido.	/i/: vocálica, no consonántica, no grave, anterior, cerrada. /ii/: vocálica, no consonántica, no grave, anterior, abierta, nasalizada.	/i/: dios = d i oo s /ii/ día = d ii a
u, ú	/u/: otros casos. /uu/: sonido más extendido.	/u/: grave, vocálica, no consonántica, posterior, cerrada, redondeada. /uu/: grave, vocálica, no consonántica, posterior, abierta, redondeada, nasalizada.	/u/: lugar = l u g aa r /uu/ nunca = n uu n k a

Consonantes:

Tabla 18.5 Consonantes para el modelo de lenguaje final

Grafema	Fonema	Clasificación de fonemas	Ejemplo
b, v	/b/	/b/: sonora, bilabial, oclusiva, aproximante, ensordecida.	/b/: nueve = n u ee b e
ch	/ch/	Africada, interdental, sorda, palatal.	/ch/: hecho = ee ch o
d	/d/	/d/: Sonora, dental, oclusiva, aproximante.	/d/: padre = p aa d r e
f	/f/	Fricativa, sorda, labiodental.	/f/: café= k a f e
G (a, o, u), gu (e, i)	/g/	/g/: Oclusiva, sonora, velar, aproximante, ensordecida.	/g/: alguien = aa l g i e n
k, ca, co, cu, qu-	/k/	Oclusiva, sorda, velar.	/k/: quiero = k i ee r o
l	/l/	/l/: lateral, sonora, alveolar, interdental, dental, sonora, palatalizada.	/l/: lugar = l u g aa r
m	/m/	Nasal, Sonora, bilabial.	/m/: tenemos = t e n ee m o s
n	/n/	/n/: nasal, alveolar, sonora, labiodental, interdental, dental sonora, palatizada, velarizada, uvular.	/n/: con = k o n
ñ	/ny/: ñ	/ny/: nasal, palatal, sonora.	/ny/: años = aa ny o s
p	/p/	Oclusiva, bilabial, sorda.	/p/: tipo = t ii p o
r, rr	/r/, /R/	/r/: vibrante, sonora, alveolar, rotica simple, aproximante /R/: vibrante, sonora, alveolar, rotica, múltiple.	/r/: tres = t r ee s /R/: departamento = d e p a R t a m ee n t o
S, c (e, i), z	/s/: inicio y final de palabra, seguida de consonante sorda (f, j, p) excepto /t/, /th/ o de pausa. /th/: posición final de silaba seguida de consonante sonora (t, n, /c/, b, /Y/), excepto /d/, o una palatal sonora (/:n).	/s/: alveolar, fricativa, sorda, dental. /th/: alveolar, fricativa, sonora, dental.	/s/: dos = d o s /th/: vez = b e th

t	/t/	/t/: Oclusiva, sorda, dental, interdental.	/t/: antes = aa n t e s
J, g- (e, i)	/x/	Fricativa, velar, sorda.	/x/: trabajo = t r a b aa x o
y- (a, e , i, o u), ll	/y/	/y/: palatal, aproximante, fricativa, sonora, africada, lateral.	/y/: ya = y a

Anexo C: Manual instalación

Manual instalación

Pre-requisitos

Sistema operativo: Ubuntu 10.04

HTK

Es un kit de herramientas para construir y manipular HMM, las cuales otorgan facilidades para análisis de voz, entrenamiento de HMM (Modelo Oculto de Markov), y análisis de pruebas y resultados. Este kit es esencial para la creación de un modelo de lenguaje.

Descarga de HTK:

- Registrarse en el sitio oficial de HTK: http://htk.eng.cam.ac.uk/register.shtml
- Crear una carpeta llamada "bin", en nombreusuario/bin
- Descargar "HTK source code (tar+gzip archive)" en la sección Download, en el sitio web de HTK.
- Descomprimir el archivo descargado en la carpeta creada (bin).

Instalación de HTK (*)

Opción 1:

- Verificar la versión dl gcc con el comando "gcc -v", si la versión es superior a la 4.0, se debe cambiar la versión a una inferior ([25] este manual de instalación desde cero de gcc podría servir).
- Configurar los archivos binarios con el comando "./configure – -prefix=/path/nombreusuario/bin/htk.3.3/"
- Escribir el comando "make all"

Opción 2:

- Abrir Synaptic (Sistema gestión de paquetes) e instalar el paquete "libx11-dev".
- Dirigirse al directorio en que se encuentra la carpeta de HTK, con el comando "cd /htk 3.3/"
- En la terminal de Linux, ingresar los comandos:
    - # ./configure

- # make all
- # make install

Para comprobar la buena instalación de HTK, ingresar en la terminal el comando "HVite –V", si se visualiza las opciones de de HVite, la instalación fue un éxito.

Para mayor información visite el sitio de HTK [13], Voxforge [11] o revise el HTKBook (disponible para descargar en el sitio web de HTK).

Julius (*)

Es un Software decodificador de reconocimiento de voz. Puede usar modelos de 2 y 3-gramas y se ejecuta casi en tiempo real en la mayoría de los equipos. Es utilizado para ejecutar el modelo de lenguaje creado con las herramientas de HTK.

Opción 1 (Synatic):

- Abrir Synaptic (sistema de gestión de paquete)
- Buscar "audacity"
- Descargar e instalar los paquetes de audacity.

Opción 2:

- Descargar el paquete binario de Julius en el sitio: http://julius.sourceforge.jp/en_index.php
- Descomprimir el archivo descargado en la carpeta creada (bin).
- Compilar julius con el comando "./configure – -prefix=/path/nombreusuario/bin/julius/"
- Ingresar el comando "make all" y luego el comando "make install"

Para comprobar la buena instalación de Julius, ingresar en la terminal "julius", si se visualiza el nombre de los creadores de Julius, quiere decir que se instaló correctamente.

Posibles errores durante la instalación:

Error 1: Cuando se utiliza el comando ./configure y aparece el siguiente error:
"configure: error: flex library not found! installation terminated"
"configure: error: ./configure failed for gramtools"

Solución: debe de instalar la libreria "flex" (en Synaptic), la cual es una herramienta para la generación de programas que realizan concordancia de patrones en textos. Es posible que ocurra un problema similar si no posee la librería "zlib" (librería de compresión de datos).

Para mayor información visite Julius Book [6] o Voxforge [11].

Audacity

Es un software editor de audio, multiplataforma, el cual es utilizado para grabar las samples (audios) necesarios para entrenar el modelo de lenguaje para el motor de reconocimiento de voz.

Opción 1 (Synatic):

- Abrir Synaptic (sistema de gestión de paquete)
- Buscar "audacity"
- Descargar e instalar los paquetes de audacity.

Opción 2 (terminal):

- Abrir terminal de Ubuntu.
- Ingresar el siguiente comando "apt-get install audacity"

Para mayor información visite el sitio oficial de Audacity [26].

Configuración:

Para grabar los samples es necesario configurar de la siguiente manera Audacity:

- En "Preferencias" (en el menú de Editar), ingresar al submenú "Calidad"
- Cambiar la frecuencia de muestreo predefinida a 48kHz, y el formato demuestra predeterminado a 16-bit.
- En el submenú "Dispositivo", cambiar el canal de grabación a mono.
- Todos los samples grabados deben ser guardados en formato .wav.

(*) La instalación podría variar, dependiendo de la versión del sistema operativo Ubuntu que posea, además de los paquetes y librerías que tenga instalados en su sistema. Este manual sólo cubre las especificaciones de los 2 equipos con Ubuntu 10.04, en que fueron probados estas herramientas y software. Aun así, estas herramientas pueden ser utilizadas en otras versiones más actuales o anteriores de Ubuntu.

Anexo D: Resultados prueba modelo parcial

Tabla 18.6 Prueba modelo parcial

Prueba de modelo parcial			
N°	**PALABRA**	**Confundida por…**	**Cantidad de veces dicha…**
1	debe	nueve	1/10
2	decir		9/10
3	dice		8/10
4	dijo	hijo, mejor	8/10
5	espera		9/10
6	estas		10/10
7	estoy		10/10
8	gusta		10/10
9	ha		0/10
10	haber		5/10
11	hablar		6/10
12	hace	parece	2/10
13	he		10/10
14	ir		10/10
15	mira		10/10
16	necesito		10 /10
17	podemos	años	10/10
18	puedes	antes	8/10
19	Quiero	cero	8/10
20	Sabes		10/10
21	será		6/10
22	son		10/10
23	tenemos	años	8/10
24	Tiene		10/10
25	Vamos	podemos	7/10
26	va	verdad	0/10
27	Ven	buen	1/10

28	así		10/10
29	Con		10/10
30	del		7/10
31	Don		8/10
32	en		10/10
33	eso		10/10
34	Hay	alguien	8/10
35	Hola		10/10
36	los		10/10
37	me		10/10
38	Mejor	por	9/10
39	menos	tenemos	9/10
40	mi	ni	7/10
41	Mucho		10/10
42	muy		10/10
43	Nada	verdad	5/10
44	ni		10/10
45	No		10/10
46	Nuestro		10/10
47	Oh		10/10
48	Parece		10/10
49	Pero		10/10
50	Poco		10/10
51	Por		10/10
52	Porque		10/10
53	Sea		10/10
54	Si		8/10
55	sobre		10/10
56	su		10/10
57	tal		10/10
58	tan		10/10
59	también		10/10

60	toda		10/10
61	tu		10/10
62	ahora	porque	6/10
63	antes		9/10
64	desde		10/10
65	nunca		10/10
66	siempre		9/10
67	cero		10/10
68	cinco		10/10
69	cuatro	porque	8/10
70	dos	los	9/10
71	nueve		10/10
72	ocho		10/10
73	seis	antes	9/10
74	siete		10/10
75	tres	tenemos	9/10
76	uno		10/10
77	alguien		10/10
78	años		8/10
79	casa		10/10
80	cosas		10/10
81	día		10/10
82	dios	menos	9/10
83	estado		10/10
84	ellos	años	8/10
85	gente		10/10
86	hecho	antes	8/10
87	hijo	dijo	9/10
88	hombre		10/10
89	lugar		10/10
90	nadie	nueve	8/10
91	niña		10/10

92	nosotros		10/10
93	padre		2/10
94	señor	desde	9/10
95	tiempo		10/10
96	tipo	tiempo	3/10
97	trabajo		10/10
98	usted		10/10
99	vida		8/10
100	vez	antes	4/10
101	verdad		8/10
102	buen		10/10
103	claro	pero	5/10
104	pequeño		9/10
105	seguro		10/10
106	ayuda	hay	7/10
107	ya		0/10
	Cantidad de palabras detectadas:	10 veces = 57 palabras 9 veces = 13 palabras 8 veces = 10 palabras 7 veces = 3 palabras 6 veces = 5 palabras 5 veces = 19 palabras 4 veces = 57 palabras 3 veces = 13 palabras 2 veces = 10 palabras 1 veces = 3 palabras 0 veces = 5 palabras	

Anexo E: Resultados prueba modelo lingüístico y acústico

Tabla 18.7 Resultado prueba de modelo lingüístico y acústico (con voz femenina)

Prueba de modelo lingüístico y acústico.			
		Sexo:	femenino
N°	**PALABRA**	**Confundida por**	**Cantidad de aciertos**
1	ahora	Padre y donde	3/5
2	alguien		5/5
3	antes	Alguien, padre, tan	2/5
4	años		5/5
5	así		5/5
6	ayuda	Hecho, alguien (3 veces), años	0/5
7	buen	Verdad, alguien	3/5
8	casa	Sera, parece, estado, lugar	0/5
9	cero		5/5
10	cinco		5/5
11	claro	Trabajo (5 veces)	0/5
12	con		5/5
13	cosas	Ocho, dos	3/5
14	cuatro	Por (2 veces) quiero, tan	1/5
14	debe	Desde (4 veces)	1/5
16	decir		5/5
17	del		5/5
18	desde		5/5
19	día		5/5
20	dice	Nuestro	4/5
21	dijo	Mejor	4/5
22	dios		5/5
23	donde		5/5
24	dos	los	4/5
25	ellos	Tenemos (4 veces), hace	0/5

26	en		5/5
27	eso		5/5
28	espera	Ellos	4/5
29	estado	estas	4/5
30	estas		5/5
31	estoy		5/5
32	gente		5/5
33	gusta		5/5
34	ha	Tan (5 veces)	0/5
35	haber	Parece (5 veces)	0/5
36	hablar	Tan, toda	3/5
37	hace	Parece (5 veces)	0/5
38	hay	Alguien (5 veces)	0/5
39	he	En (4 veces), pero	0/5
40	hecho		5/5
41	hijo	Dijo	4/5
42	hola		5/5
43	hombre		5/5
44	ir	Pero, quiero	3/5
45	los	menos	4/5
46	lugar		5/5
47	me		5/5
48	mejor		5/5
49	menos		5/5
50	mi		5/5
51	mira		5/5
52	mucho		5/5
53	muy		5/5
54	nada	Verdad (5 veces)	0/5
55	nadie	Nueve (3 veces), alguien, lugar	0/5
56	necesito		5/5
57	ni	Mi (2 veces)	3/5

58	niña		5/5
59	no	Uno (3 veces)	2/5
60	nosotros		5/5
61	nuestro		5/5
62	nueve	Donde, ahora	3/5
63	nunca		5/5
64	ocho	Por (3 veces)	2/5
65	oh	Poco (3 veces), con (2 veces)	0/5
66	padre	Parece (5 veces)	0/5
67	parece		5/5
68	pequeño		5/5
69	pero	quiero	4/5
70	poco		5/5
71	podemos		5/5
72	por		5/5
73	porque		5/5
74	puedes		5/5
75	quiero		5/5
76	sabes		5/5
77	sea	cero	4/5
78	seguro		5/5
79	seis	antes	4/5
80	señor		5/5
81	Será		5/5
82	si	Cinco (4 veces)	4/5
83	siempre		5/5
84	siete		5/5
85	sobre		5/5
86	son	Sobre, con	3/5
87	su	Sobre (2 veces), hijo (2 veces), poco	0/5
88	tal	Tan (3 veces), alguien	1/5
89	también		5/5

90	tan		5/5
91	tenemos		5/5
92	tiempo	Alguien (2 veces)	3/5
93	tiene		5/5
94	tipo	Cinco (3 veces), quiero	1/5
95	toda	padre	4/5
96	trabajo		5/5
97	tres		5/5
98	tu		5/5
99	uno	podemos	4/5
100	usted	Pero	4/5
101	va	Lugar, padre, niña (2 veces), verdad	0/5
102	vamos	Dos (5 veces)	0/5
103	ven	Buen (4 veces), haber	0/5
104	verdad		5/5
105	vez	Desde (2 veces), del (3 veces)	0/5
106	vida	Día (2 veces)	3/5
107	ya	Día (4 veces), gusta	0/5
	Cantidad de palabras detectadas:	5 veces = 57 palabras 4 veces = 13 palabras 3 veces = 10 palabras 2 veces = 3 palabras 1 veces = 5 palabras 0 veces = 19 palabras	

Tabla 18.8 Resultado prueba modelo lingüístico y acústico (con voz masculina)

Prueba de modelo lingüístico y acústico.			
		Sexo:	masculino
N°	**PALABRA**	**Confundida por**	**Cantidad de aciertos**
1	Ahora	Todos	3/5
2	Alguien		5/5
3	Antes	Parece	1/5
4	Años	Nosotros	3/5
5	Así	Parece	2/5
6	Ayuda	Otros	0/5
7	Buen	Donde	4/5
8	Casa	Parece, con	0/5
9	Cero	Seguro	3/5
10	Cinco	Seguro	1/5
11	Claro	Cero	2/5
12	Con	Tan, son	2/5
13	Cosas	Eso, podemos	3/5
14	Cuatro	Por, todos	3/5
14	Debe	Desde (4 veces)	1/5
16	Decir	Parece	4/5
17	Del	Haber	4/5
18	Desde	Parece	4/5
19	Día	Vida	3/5
20	Dice	Nuestro	4/5
21	Dijo	Gusta	4/5
22	Dios		5/5
23	Donde	Nueve	4/5
24	Dos	Tenemos	4/5
25	Ellos	Tenemos	1/5
26	En	Alguien	4/5
27	Eso	Antes	4/5
28	Espera	Donde	3/5

29	Estado	Estoy	4/5
30	Estas	Antes	3/5
31	Estoy	Antes	4/5
32	Gente		5/5
33	Gusta		5/5
34	Ha	Tan, claro, tal	0/5
35	Haber	Parece	1/5
36	Hablar	Hola, tal	3/5
37	Hace	Parece	0/5
38	Hay	Alguien	3/5
39	He	Del	2/5
40	Hecho		5/5
41	Hijo	Dijo	3/5
42	Hola		5/5
43	Hombre	Donde	4/5
44	Ir	Decir	4/5
45	Los		5/5
46	Lugar	Niña	4/5
47	Me	Buen, nueve	1/5
48	Mejor	Antes	4/5
49	Menos	Tener	3/5
50	Mi	Ni	1/5
51	Mira	Niña	3/5
52	Mucho		5/5
53	Muy		5/5
54	Nada	Los	0/5
55	Nadie	Nueve	3/5
56	Necesito		5/5
57	Ni	Nadie, decir	2/5
58	Niña	Mira	4/5
59	No	Los	0/5
60	Nosotros	Los	4/5

61	Nuestro	Antes	4/5
62	Nueve	Haber	3/5
63	Nunca	Antes	4/5
64	Ocho	Los, con	3/5
65	Oh	Con, dos	0/5
66	Padre	Alguien	0/5
67	Parece		5/5
68	pequeño		5/5
69	Pero	Cero	1/5
70	Poco		5/5
71	podemos		5/5
72	Por		5/5
73	Porque		5/5
74	Puedes	Antes	4/5
75	Quiero	Cero	3/5
76	Sabes	Cosas	4/5
77	Sea	Casa	3/5
78	Seguro		5/5
79	Seis	Señor	3/5
80	Señor	Antes	4/5
81	Será		5/5
82	Si	Casa	2/5
83	Siempre	Sea	2/5
84	Siete	Sea, seguro	1/5
85	Sobre		5/5
86	Son	Sobre	5/5
87	Su	Sobre	3/5
88	Tal	Tan, alguien	1/5
89	También		5/5
90	Tan	Con	4/5
91	Tenemos		5/5
92	Tiempo	Alguien	4/5

93	Tiene	Quien	1/5
94	Tipo	Cinco, tiempo	0/5
95	Toda		5/5
96	Trabajo		5/5
97	Tres	Antes	4/5
98	Tu		5/5
99	Uno	Mucho	2/5
100	Usted	Pero, gusta	3/5
101	Va		0/5
102	Vamos	Dos	3/5
103	Ven	Buen	0/5
104	Verdad	Hablar	3/5
105	Vez	Parece	0/5
106	Vida	Día	4/5
107	Ya		0/5

Cantidad de palabras detectadas:
5 veces: 24
4 veces: 26
3 veces: 23
2 veces: 10
1 vez: 12
0 veces: 12

19 APÉNDICES

APÉNDICE A: AFI DEL ESPAÑOL

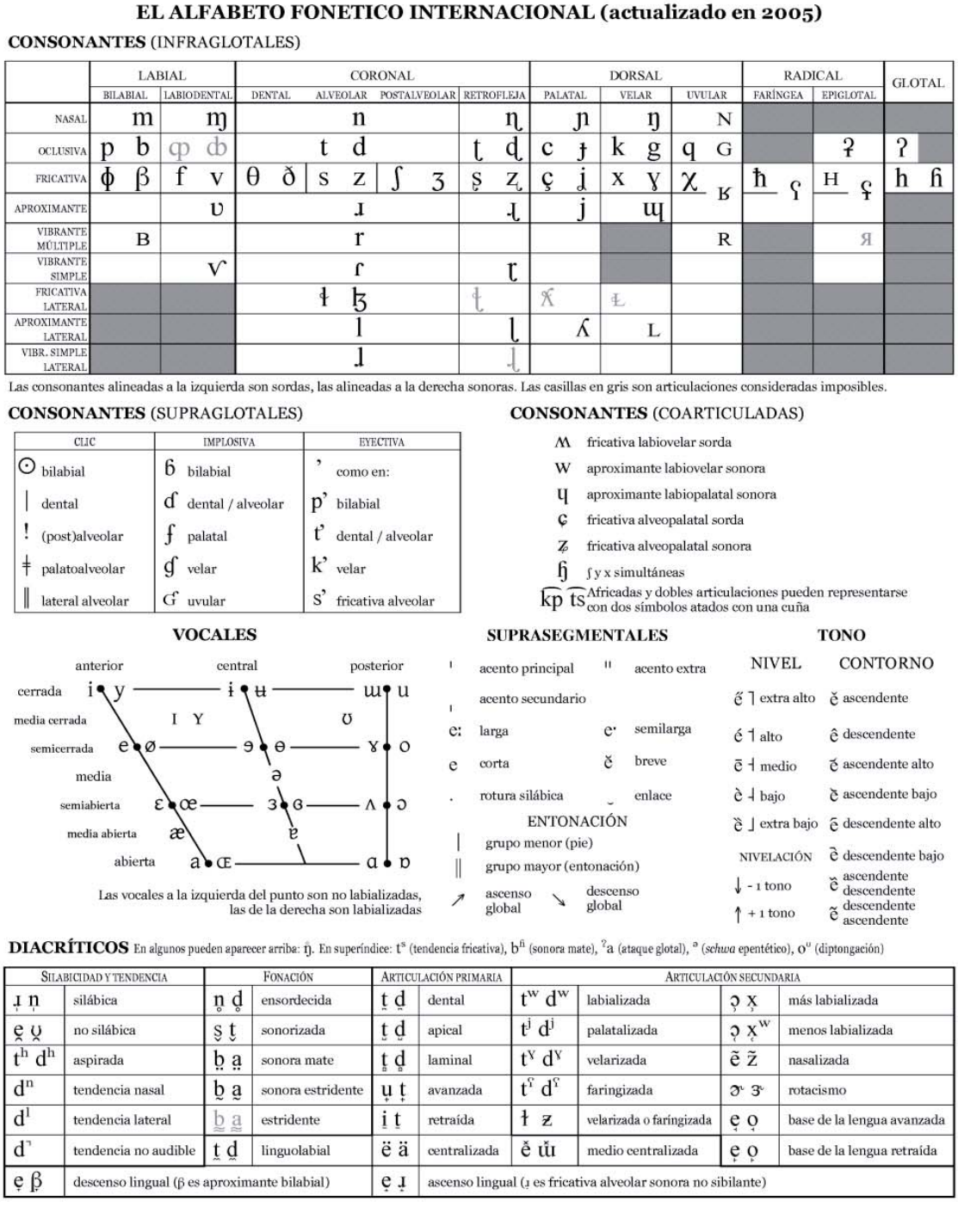

Figura 0-1 Alfabeto Fonético Internacional

APÉNDICE B: ADAPTACIÓN DEL ESPAÑOL DEL ALFABETO SAMPA

Consonantes:

Tabla 19.1 Diccionario SAMPA para el español (Wells 1998)

Símbolo	Ejemplo	Transcripción de ejemplo
p	padre	"paDre
b	vino	"bino
t	tomo	"tomo
d	donde	"donde
k	casa	"kasa
g	gata	"gata
tS	mucho	"mutSo
jj	hielo	"jjelo
f	fácil	"faTil
B	cabra	"kaBra (= /b/)
T	cinco	"Tinko
D	nada	"naDa (= /d/)
s	sala	"sala
x	mujer	mu"xer
G	luego	"lweGo (= /g/)
m	mismo	"mismo
n	nunca	"nunka
J	año	"aJo

l	lejos	"lexos
L	caballo	ka"baLo (o como jj)
r	puro	"puro
rr	torre	"torre
j	rei pie	rrej pje
w	Deuda Muy	"dewDa mwi
i	mico	"miko
e	pero	"pero
a	valle	"baLe
o	toro	"toro
u	duro	"dur