UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo e implantación de un sistema de reconocimiento del habla para la elaboración de informes médicos

TRABAJO FIN DE MÁSTER

Máster en Ingeniería Informática

Autor: Feng Ho, Alex

Tutor: Sánchez Peiró, Joan Andreu

Curso 2019-2020

Resumen

Dentro de las tareas de los colaboradores de los hospitales o centros médicos, refiriéndose específicamente a los médicos de diagnóstico en laboratorios, la mayor parte del esfuerzo y tiempo recae en la redacción de informes médicos. Debido a ello, el número de pacientes que se pueden atender y el servicio que pueden ofrecer se ve reducido. En este trabajo se propone el diseño y desarrollo de un sistema de reconocimiento automático del habla (RAH) utilizando CMU Sphinx4 y Kylm para la elaboración de informes médicos y su implantación en el servicio de medicina nuclear del hospital Dr. Peset en Valencia. El desarrollo y adaptación de sistemas de RAH a problemas muy concretos es posible gracias a los avances en las últimas décadas en este campo y a la disponibilidad de datos accesibles en formato abierto. Los experimentos iniciales se realizan con un modelo acústico pre-entrenado en español del proyecto VoxForge y un corpus de texto que contiene un vocabulario con más de 18 000 palabras referentes a informes médicos de medicina nuclear. Como resultado de la experimentación, se ha obtenido un valor WER promedio de 26,6 % sin adaptación al locutor y se ha logrado mejorar a un WER promedio de 23,1 % con adaptación MAP indicando una reducción del error en 13,2 %. La aplicación y código fuente se encuentran disponible en https://sourceforge.net/projects/asr-for-medical-reporting/ bajo licencia GNU (GPLv3).

Palabras clave: Reconocimiento automático del habla, medicina nuclear, Sphinx4, informes médicos, HMM, modelo de lenguaje

Resum

Dins de les tasques dels col·laboradors als hospitals i centres mèdics, referint-se específicament als metges de diagnòstic en laboratori, la major part del esforç i del temps es dedicat a la redacció de informes mèdics. A conseqüència, el nombre de pacients que es poden atendre i el servei que es pot oferir es veu reduït. En aquest treball es proposa el disseny i desenvolupament d'un sistema de reconeixement automàtic de la parla (RAP) utilitzant CMU Sphinx4 i Kylm per a l'elaboració d'informes mèdics i la seua implantació en el servei de medicina nuclear de l'hospital Dr. Peset a València. El desenvolupament y adaptació de sistemes RAH a problemes molt concrets es possible gràcies als avanços en les últimes dècades en aquest camp i a la disponibilitat de dades accessibles de forma oberta. Els experiments inicials es realitzen amb un model acústic prèviament entrenat en espanyol del projecte VoxForge i un corpus de text que conté un vocabulari amb més de 18 000 paraules que fan referència a informes mèdics de medicina nuclear. Com a resultat de l'experimentació, s'ha obtingut un valor WER mitjà de 26,6% sense adaptació al locutor i s'ha aconseguit millorar a un WER mitjà de 23,1% amb adaptació MAP indicant una reducció del error en 13,2%. L'aplicació i el codi font estan disponibles a https://sourceforge.net/projects/asr-for-medical-reporting/ baix llicència GNU (GPLv3).

Paraules clau: Reconocimiento automático del habla, medicina nuclear, Sphinx4, informes mèdics, HMM, model de llenguatge

Abstract

Among the tasks of collaborators in hospitals or medical centers, specifically referring to diagnostic doctors in laboratories, most of the effort and time falls on the writing of medical reports. Due to this, the number of patients that can be attended and the service they can offer is reduced. This work proposes the design and development of an automatic speech recognition (ASR) system using CMU Sphinx4 and Kylm for the preparation of medical reports and their implementation in the nuclear medicine service of the Dr. Peset hospital in Valencia. The development and adaptation of ASR systems to very specific problems is possible thanks to the advances in the last decades in this field and the availability of open data. Initial experiments are performed with a pre-trained acoustic model in Spanish from the VoxForge project and a corpus of text containing a vocabulary of more than 18 000 words referring to nuclear medicine medical reports. As a result of experimentation, an average WER value of 26,6% has been achieve without speaker adaptation and it has been possible to improve to an average WER of 23,1% with MAP adaptation indicating a reduction of error in 13,2% . The application and source code are available in https://sourceforge.net/projects/asr-for-medical-reporting/ under GNU (GPLv3) license.

Key words: Automatic speech recognition, nuclear medicine, Sphinx4, medical report, HMM, languaje model

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Agradecimientos

Quiero agradecer profundamente a Elisa Caballero y Pedro Abreu por su gran disposición y colaboración durante el desarrollo de este trabajo de fin de máster.

A los colaboradores del servicio de medicina nuclear del hospital Dr. Peset por su disposición y ayuda en los experimentos que se llevaron a cabo en este trabajo.

A mi tutor, Joan Andreu Sánchez Peiró, por la oportunidad que me ha brindado para realizar este trabajo que es fruto de las orientaciones y sugerencias que me ha brindado. También por su disposición ante las dudas que surgieron y el profesionalismo con el cual me ha guiado durante el desarrollo de este trabajo.

Finalmente, quiero agradecer a mis familiares y amigos que me han brindado su apoyo a lo largo del máster.

CAPÍTULO 2
Introducción

A pesar de los grandes avances en tecnologías especificas de RAH, estos todavía no son accesibles a la gran mayoría de usuario. Si bien es cierto que existen herramientas y librerías de código abierto, se requiere que el usuario tenga un perfil técnico para poder aprovechar estas herramientas. Sí que es cierto que existen sistemas de RAH en el mercado pero tienen la particularidad de que son muy costosos y, en la mayoría de los casos, son desarrollados para que funcionen en un contexto general y no en uno específico como en el caso de la medicina nuclear.

Una de las tareas que más tiempo consume a los profesionales de la salud es la descripción de hallazgos en exámenes físicos u otras observaciones, lo que resulta en una gran cantidad de datos de diagnósticos que deben ser introducidos en los sistemas informáticos convirtiéndose en una tarea laboriosa. Además, cualquier error de transcripción puede pasar por desapercibido.

2.1 Motivación

En los últimos años, se han disparado los avances en la inteligencia artificial dando una clara visión del futuro en este campo. Tareas que antes creíamos que serían imposibles de realizar con el ordenador, hoy en día es posible, superando todas las expectativas. Sobre todo en el campo de la medicina, donde el impacto de estas tecnologías están revolucionando la forma en que, tradicionalmente, se diagnostica y detectan enfermedades.

Uno de los motivos por el cual he decidido aceptar el desarrollo de este proyecto es por el interés de aprender tecnologías punteras en el campo de inteligencia artificial y poner en práctica todo el conocimiento adquirido durante los estudios. Además, buscaba un proyecto novedoso que cumpliera con las características de un trabajo de fin de máster y que el trabajo resultante pudiera ser de utilidad a otras personas.

A nivel profesional, está claro que la realización de este trabajo también va encaminado a la construcción de un futuro laboral y especialización en el campo de la inteligencia artificial.

2.2 Objetivos

El objetivo general de este trabajo es desarrollar e implantar un sistema de reconocimiento automático del habla para la elaboración de informes médicos adaptable al contexto para el departamento de medicina nuclear del hospital Dr. Peset en Valencia.

El alcance del objetivo general está sujeto al cumplimiento de los siguientes objetivos específicos:

- Analizar y seleccionar herramientas de reconocimiento automático del habla con modelos acústicos en idioma español.
- Crear un modelo de lenguaje y léxico para el contexto de medicina nuclear.
- Crear una interfaz gráfica de fácil interacción para el usuario.
- Crear una aplicación que integre las herramientas de reconocimiento automático del habla y procesamiento del lenguaje natural.

2.3 Estructura de la memoria

Tomando en cuenta este capítulo introductorio, el resto de la memoria del trabajo de fin de máster se encuentra organizado de la siguiente forma.

El capítulo 2 presenta una revisión del estado del arte o contexto tecnológico sobre el cual se desarrolla este trabajo. Se documentan las distintas herramientas que existen en el ámbito del reconocimiento automático del habla y también se exponen aplicaciones ya existentes en el ámbito médico.

En el capítulo 3 se realiza un análisis de la problemática en general que se pretende resolver, así como los problemas relacionados con los datos que se disponen; y se propone una posible solución a estos problemas. También, se presenta la especificación de los requisitos utilizando la metodología propuesta por el estándar IEEE 830.

En el capítulo 4 se detalla todo lo relacionado con el diseño de la solución propuesta. Se describe la arquitectura general del sistema y su funcionamiento tanto a nivel de procesos como de interacción con el usuario. También se describen las tecnologías que se utilizaran para el desarrollo de la aplicación.

En el capítulo 5 se presenta el proceso de desarrollo de la solución siguiendo el diseño propuesto. Se detallan los componentes que se han desarrollado y las dificultades encontradas durante el desarrollo.

El capítulo 6 se describen los experimentos que se llevan a cabo para medir el rendimiento de la aplicación y se presentan los resultados de dichos experimentos.

El capítulo 7 se detalla el proceso de preparación del sistema para ser implantado y su implantación.

Finalmente, en el capítulo 8 y 9 se presentan las conclusiones obtenidas y relación del trabajo con los estudios cursados; y una reflexión sobre posibles mejoras a futuro respectivamente.

CAPÍTULO 3
Estado del arte

La comunicación entre humano y máquina se ha visto, en mayor medida, en la utilización de dispositivos como el teclado y el ratón. Sin embargo, en las últimas décadas, las interfaces orales [3] han tomado mayor relevancia gracias a su facilidad de utilización y a la velocidad en la comunicación con el computador. La mayor parte de la gente puede pronunciar más de 200 palabras por minuto, mientras que otros pueden escribir poco más de 60 palabras mediante un teclado en el mismo tiempo.

El desarrollo de sistemas RAH se remonta a los años 50. Los primeros trabajos en este campo lograban discriminar con cierta precisión los diez dígitos en inglés pronunciados de forma aislada por un único lector [4]. A partir de los años 60, comienzan a surgir trabajos de reconocimiento de palabras aisladas con el objetivo de llegar a sistemas capaces de reconocer palabras y frases de cualquier locutor, de forma continua y precisa.

3.1 Reconocimiento automático del habla

Los sistemas de RAH tienen como tarea principal convertir una señal de voz en transcripciones de texto [5] realizando sucesivas transformaciones sobre dicha señal en su representación fonética implícita [6]. El objetivo es estimar la mejor secuencia de palabras dado un modelo de reconocimiento del habla y una señal de voz.

Dicho lo anterior, el RAH se puede formular de la siguiente manera [7]: si A representa una secuencia de características acústicas proveniente de la señal de entrada, el sistema RAH debe producir la secuencia de palabras óptima $\hat{W}$ que coincida mejor con A, más formalmente, la secuencia de palabras que maximiza la probabilidad *a posteriori* dada la señal acústica de entrada:

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W|A) \tag{3.1}$$

donde W simboliza cualquier combinación de palabras del vocabulario.

Hay que tener en cuenta que se modela todas las posibles secuencias de palabras sobre el vocabulario [8] y se cubren todas las posibles frases que puede decir un locutor (siempre y cuando que las palabras formen parte del vocabulario del sistema).

La fórmula (3.1) es una aproximación estadística, por tanto, obliga a que todos los componentes del sistema RAH estén basados en probabilidades. Esta afirmación se expone en [9]. Al aplicar la regla de Bayes al argumento de la maximización, tenemos que:

$$P(W|A) = \frac{P(W)P(A|W)}{P(A)} \tag{3.2}$$

en donde $P(W)$ es la probabilidad de la secuencia de palabras W que viene dado por el modelo de lenguaje. $P(A|W)$ es la probabilidad de la señal acústica A observada dado una secuencia de palabras W la cual se obtiene a partir de un modelo acústico y $P(A)$ es la probabilidad de que un locutor emita la señal acústica A. Este último se puede eliminar ya que la probabilidad condicional que se quiere calcular asume que la señal acústica A es conocida. Por tanto, combinando las ecuaciones (3.1) y (3.2) tenemos que:

$$\hat{W} = \underset{W}{\operatorname{argmax}}\, P(W)P(A|W) \tag{3.3}$$

lo que resulta en una maximización que depende de la probabilidad de una secuencia de palabras (dado por modelo de lenguaje) y de la probabilidad de que un locutor haya emitido la señal acústica para la secuencia que se está considerando en ese momento (dado por el modelo acústico).

Los sistemas RAH están compuesto por dos fases: una fase de entrenamiento y otra de reconocimiento. El procedimiento para entrenar consiste en mapear la unidad fonética, ya sea fonemas o sílabas, a una observación acústica [6]. Para ello, la señal de voz debe grabarse y procesarse para crear la secuencia de características que luego se utilizará para crear, entrenar y almacenar un modelo oculto de Markov (HMM: del inglés Hidden Markov Hodel). El proceso de reconocimiento (ver Figura 3.1) inicia con la captura de la señal de voz y la obtención de la secuencia de características. Estas secuencias de características se comparan contra el modelo entrenado obteniendo la unidad fonética que mejor coincide. Las unidades fonéticas obtenidas se combinan siguiendo las reglas definidas por el modelo de lenguaje para intentar encontrar la frase pronunciada por el locutor.

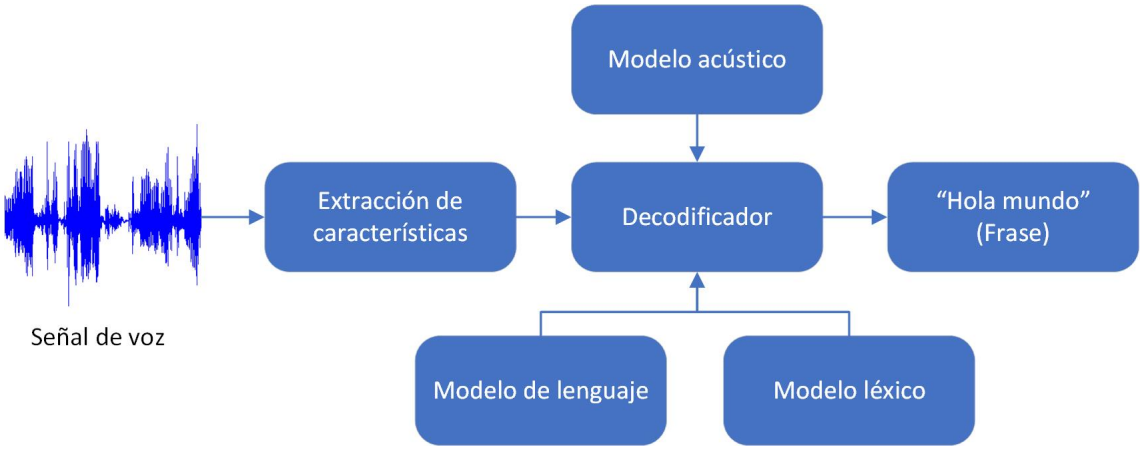

Figura 3.1: Diagrama de un sistema de reconocimiento automático del habla.

3.1.1. Procesado de la señal voz

Como ya sabemos que la entrada de los sistemas RAH son señales de voz es necesario aplicar un procesado a la señal. Esta es la primera operación que se realiza con el objetivo de extraer las características relevantes para el reconocedor. Este proceso requiere de mucha atención ya que el rendimiento del sistema depende mucho de este proceso [6].

La extracción de características consiste en extraer muestras de la señal de voz en intervalos regulares de tiempo en el rango de 10 a 25 ms. A cada muestra se le calculan parámetros que caracterizan la señal. LPC, MFCC, AMFCC, RAS, DAS, ΔMFCC [6] son diferentes técnicas para extracción de características. La mayoría de los sistemas RAH realizan análisis de banco de filtros a escala de mel y derivan coeficientes cepstrales llamados Coeficientes de filtro de escala Mel (MFCC: del ingles Mel Scale Filter Coefficients) [7].

3.1.2. Modelo acústico

El modelo acústico tiene como función predecir las frases hipótesis a partir de las características observadas de la señal de audio. La creación del modelo acústico involucra el uso de grabaciones de voz y su correspondiente transcripción textual. Aplicando el proceso de extracción de características que se describe en la sección anterior, se calcula la representación estadística para cada unidad fonética.

La representación estadística de las unidades fonéticas se estima como $P(A|W)$ en la ecuación (3.3). Esta estimación se realiza mediante HMM para modelar la probabilidad de transición entre cada unidad fonética y con modelos de mixturas gaussianas (o GMM del inglés Gaussian Mixture Model) para modelar la probabilidad de emisión de cada una [6].

En sistemas híbridos que combinan redes neuronales profundas (DNN, del inglés Deep Neural Network) y HMM, las DNN son entrenadas para aprender la probabilidad de emisión de los estados del HMM [10] reemplazando los modelos basados en GMM.

Modelos ocultos de Markov

En el contexto del reconocimiento automático del habla los modelos ocultos de Markov son los métodos más populares y ampliamente utilizados para modelar y combinar secuencias de unidades fonéticas [11, 7]. Dependiendo de la cantidad de unidades fonéticas a modelar, tendremos un HMM asociado a cada unidad. Según Rabiner [12] los HMM son un proceso doblemente estocástico, es decir, contiene un proceso estocástico que no es observable (que es oculto) pero que puede observarse a través de otro conjunto de procesos estocásticos que producen una secuencia de observaciones. Dichos procesos están conformados por conjuntos de estados conectados por transiciones. Cada transición se describe por dos tipos de probabilidades [11] que se aprenden mediante el algoritmo Baum-Welch [13]:

1. **Una probabilidad de transición**: el cual define la probabilidad de pasar de un estado a otro.

2. **Una probabilidad de emisión**: que define la probabilidad condicional de observar un conjunto de características acústicas cuando ocurra una transición. Se modela mediante modelos de mixturas gausianas (GMM, del inglés Gaussian Mixture Model).

Dicho lo anterior, los HMM se pueden representar como una máquina de estados finitos conformado por conjuntos de estados ocultos y conjuntos de muestras observables. En la Figura 3.2 se puede ver una representación en HMM de la palabra "mano", en donde los nodos superiores representan las unidades fonéticas y los nodos inferiores representan las muestras observables, en este caso, la secuencia de características acústicas. Las flechas horizontales representan la transición en la secuencia de unidades fonéticas.

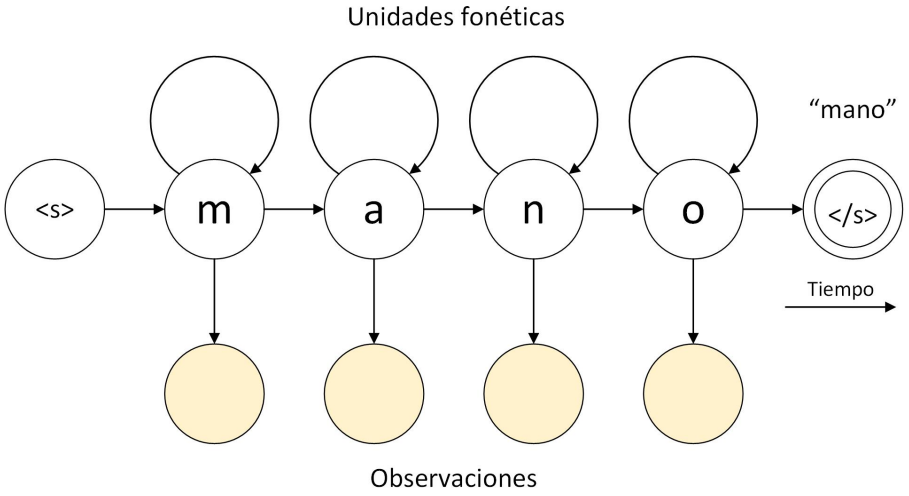

Figura 3.2: Representación de una palabra en unidades fonéticas dentro de un HMM.

Dado un modelo HMM entrenado se requiere encontrar la secuencia de estados ocultos que maximicen la probabilidad de las características observadas. Este proceso se llama decodificación. La probabilidad de las características observadas se obtiene sumando todas las probabilidades de cada observación multiplicado por todas las posibles secuencia de estados, formalmente [12]:

$$P(O|\lambda) = \sum_{Q} P(O|Q,\lambda)P(Q|\lambda) \tag{3.4}$$

en donde λ es el modelo HMM, O representa las observaciones y Q representa la secuencia de estados.

La decodificación no puede utilizar la fórmula (3.4) de forma directa debido a su complejidad exponencial. Para solucionar este problema, se hace uso de algoritmos que buscan romper con la complejidad exponencial como los basados en programación dinámica. El algoritmo más utilizado para la decodificación es el algoritmo de Viterbi [14]. El objetivo de este algoritmo es el de encontrar el camino que maximice la probabilidad de la observación.

3.1.3. Modelo de lenguaje

El modelo de lenguaje tiene como función guiar y limitar la búsqueda entre las diferentes hipótesis de secuencias de palabras que se producen durante el reconocimiento. Refiriéndose a la fórmula (3.3), esta contiene un factor $P(W)$ que se estima con el modelo de lenguaje. Por tanto, frases que son correctamente construidas tendrán mayor probabilidad que aquellas frases mal construidas. Por ejemplo, la frase "*objetivan se colindantes nódulos*" es válida pero carece de sentido y tendría una probabilidad baja, mientras que la frase "*se objetivan nódulos colindantes*", sí que tiene sentido lo cual tendría una probabilidad mayor.

Generalmente, en los sistemas de RAH basados en HMM, se utilizan modelos de lenguajes basados en *n-gramas*. Los modelos basados en n-gramas predicen la probabilidad de la *n-ésima* palabra en base a la ocurrencia de las *n-1* palabras anteriores, formalmente [13]:

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1,w_2)...P(w_n|w_1,w_2,...,w_{n-1}) \tag{3.5}$$

En la fórmula (3.5) se da por hecho de que cuando una secuencia es suficientemente larga, solo se toman en consideración las últimas n palabras contando la actual y se asume que el resto de palabras no influye en el cálculo de las probabilidades.

Dependiendo del tamaño del corpus de texto que se utilice para construir un modelo de n-gramas, habrán secuencias de palabras que no se contemplan en los n-gramas. A esto se le conoce como el problema de frecuencia cero. La solución más común a este problema es la utilización de técnicas de suavizado. Estas técnicas buscan asignarles probabilidades a secuencias no vistas durante el entrenamiento, suficientemente bajas como para que no interfieran demasiado en el modelo [8].

3.1.4. Modelo léxico

El modelo léxico o también conocido como diccionario de pronunciación se representa normalmente como un listado de palabras con su transcripción fonética asociada. Estas palabras son tomadas del vocabulario de un corpus de texto y su transcripción fonética está compuesta por unidades fonéticas que dependen del lenguaje en que se esté trabajando.

Dentro de los sistemas de RAH, el modelo léxico tiene como objetivo servir de enlace entre las representaciones obtenidas del modelo acústico y la secuencia de palabras [15]. El modelo léxico tiene dos funciones principales: el primero, indica qué palabras son las conocidas por el sistema; y el segundo, provee de significado a las unidades fonéticas resultante del modelo acústico.

Las palabras dentro del modelo léxico puede contener múltiples pronunciaciones debido a la variabilidad de acentos y demás factores externos [16]. Por tanto, tener asociado todas las posibles pronunciaciones de las distintas palabras aumentaría el poder predictivo del sistema. Sin embargo, se debe tener mucha precaución ya que puede aumentar la ambigüedad de las secuencias de palabras.

La creación del modelo léxico se establece en dos partes principales: primero, se debe describir la pronunciación de las palabras en unidades fonéticas; y segundo, se debe representar las palabras como uno o varios conjuntos de unidades fonéticas. Durante la construcción del modelo léxico se debe prestar atención en ambas partes para alcanzar el mejor rendimiento posible del sistema [15]. La pronunciación de cada palabra suele tomarse de diccionarios de pronunciación existentes o generados automáticamente mediante programas de conversión de grafemas a fonemas mientras que, el listado de palabras, deriva del propio vocabulario asociado al corpus de texto.

3.1.5. Adaptación al locutor

La adaptación al locutor es un proceso en el cual se busca que un sistema de RAH pueda aprender las características acústicas de un locutor con el objetivo de mejorar el rendimiento del sistema para ese locutor específico [17]. Desde el punto de vista del locutor, los sistemas de RAH pueden ser de dos tipos: independiente del locutor y dependiente del locutor. En la Tabla 3.1 se realiza una comparativa entre los dos tipos de sistemas.

Independiente del locutor	**Dependiente del locutor**
▪ Puede ser utilizado por cualquier locutor sin tener que entrenar un modelo para cada uno.	▪ Requiere entrenar un modelo para que aprenda las características acústicas de todos los locutores.
▪ Son capaces de reconocer el habla de diferentes usuarios limitando el contexto del habla (palabras y frases).	▪ Generalmente pueden reconocer palabras y frases en contextos más amplios.
▪ La precisión del reconocimiento es baja debido a la variabilidad entre los distintos locutores.	▪ La precisión del reconocimiento es mayor porque tiene en cuenta la variabilidad de los locutores.

Tabla 3.1: Comparativa entre sistemas independientes del locutor y dependientes del locutor.

En el caso de los sistemas dependientes del locutor se requiere recolectar gran cantidad de datos para el entrenamiento [18]. Visto esta restricción lo deseable sería que se pudiera tomar una pequeña cantidad de datos de un locutor nuevo para ajustar el sistema independiente del locutor al nuevo locutor [19]. A esto se le conoce como técnicas de adaptación al locutor.

Las técnicas de adaptación al locutor que se utilizan en sistemas de RAH basados en HMM son: máxima probabilidad a posteriori (MAP, del inglés *maximun*

aposteri probability) y regresión lineal de máxima probabilidad (MLLR, del inglés *maximum likelihood linear regression*).

Maximum aposteriori probability (MAP)

La técnica MAP consiste en modificar los parámetros del modelo acústico de un sistema independiente del locutor de modo que se pueda adaptar a los datos de un nuevo locutor. MAP utiliza la probabilidad de la distribución de los parámetros previamente disponible. Con este conocimiento, se puede adaptar los parámetros con una limitada cantidad de datos de adaptación de modo que se pueda maximizar la probabilidad *aposteriori* [18]. Por tanto, los valores actualizados de los parámetros se define como [20]:

$$\hat{\lambda} = \text{argmax}\, p(o|\lambda)p(\lambda) \tag{3.6}$$

en donde $p(o|\lambda)$ es la probabilidad de la secuencia de características observadas o dado los parámetros del modelo λ; y $p(\lambda)$, es la distribución de la probabilidad de los parámetros.

Esta técnica solo actualiza los parámetros del modelo cuyo datos hayan sido observados en la adaptación [19]. Es por ello que se requiere una cantidad de datos considerables para obtener cambios significativos en el rendimiento del reconocimiento.

Maximun likelihood linear regression (MLLR)

La técnica MLLR consiste en estimar matrices con transformaciones que modifiquen los parámetros del modelo. MLLR toma algunos datos de adaptación de un nuevo locutor y actualiza los parámetros de *media* del modelo para maximizar la probabilidad de los datos de adaptación. Los otros parámetros del HMM no son adaptados ya que las principales diferencias entre locutores se caracterizan por los parámetros de las *medias* [19]. La adaptación de las *medias* del modelo está dado por [19]:

$$\hat{\mu} = W\mu \tag{3.7}$$

en donde $\hat{\mu}$ es el nuevo valor de la *media*; W, es una matriz $n(n+1)$ en donde n es la dimensión del vector de características observadas; y μ son los valores de *media* previos.

A diferencia con la técnica MAP, esta técnica funciona mejor cuando no se dispone de muchos datos de adaptación. Además, es más versátil ya que basta con calcular una matriz de transformación global para todos los modelos que existan en un sistema de RAH en vez de adaptar uno a uno, como es el caso de MAP.

3.2 Herramientas para el desarrollo de RAH

Los sistemas de reconocimiento automático del habla son complejos de desarrollar. Para ello existen librerías ya preparadas para realizar todo el trabajo de

un sistema de RAH desde el entrenamiento hasta la puesta en marcha. Muchas de estas herramientas son frutos del trabajo y colaboración de diferentes grupos de investigación en el campo del reconocimiento del habla.

Cada librería o herramienta tiene características únicas e implementan diferentes métodos que pueden mejorar el reconocimiento dependiendo del problema planteado.

3.2.1. CMU Sphinx toolkit

CMU Sphinx es un conjunto de herramientas y librerías de código abierto para sistemas de reconocimiento automático del habla desarrollado mutuamente entre el grupo Sphinx de la Universidad de Carnegie Mellon, Sun Microsystems Laboratories, Mitsubishi Electrica Research Lab (MERL) y Hewlett Packard (HP) [21]. Incluye una serie de programas que permite el reconocimiento del habla y el entrenamiento del modelo acústico utilizando HMM y GMM.

CMU Sphinx [22] recolecta cerca de 20 años de investigación en el campo y listan muchas de sus ventajas frente a otras herramientas:

- Implementación de algoritmos eficientes para el reconocimiento del habla. Las herramientas de CMU Sphinx están diseñadas específicamente para plataformas de bajos recursos.
- Diseño flexible. Permite agregar módulos de forma fácil.
- Las herramientas están enfocadas a aplicaciones prácticas.
- Soporte para múltiples idiomas como inglés, francés, alemán, mandarín, español y entre muchos otros. También permite la construcción de modelos para cualquier idioma.
- Dispone de soporte técnico para licencias comerciales.
- Contiene gran abanico de herramientas para propósitos como *keyword spotting*, alineación, evaluación de la pronunciación, etc.

Los lenguajes de programación utilizados para el desarrollo de las herramientas están basados en *C* y *Java*. Las herramientas más características de CMU Sphinx son *Sphinx4* y *pocketsphinx*. *Sphinx4* es un motor para RAH escrita en *Java* con el objetivo de facilitar la investigación en RAH. *Pocketsphinx* es una versión ligera del motor de RAH pensado para ser utilizado en dispositivos móviles o de bajos recursos.

3.2.2. Kaldi

Kaldi [23] es un conjunto de herramientas de código abierto para sistemas de RAH escrito puramente en lenguaje *C++* bajo la licencia Apache v2.0. El objetivo de Kaldi es mantener un código flexible y fácil de entender, modificar y extender. Kaldi es similar a HTK en cuando a objetivos y alcance. Entre sus características se pueden mencionar [23]:

- Integración con transductores de estados finitos con la librería OpenFst.
- Soporte para álgebra lineal utilizando rutinas estándar BLAS y LAPACK.
- Provee de algoritmos en su forma más general posible.
- Provee de recetas completas para construir sistemas RAH.
- Es una herramienta flexible, con todo el código estructurado de forma limpia.

El acceso a las funcionalidades de Kaldi es a través de línea de comando. Utilizando lenguaje de *scripting*, se puede construir y ejecutar un reconocedor del habla. La forma de utilizar Kaldi permite concatenar las entradas y salidas de las diferentes herramientas que dispone.

Una de los inconvenientes de Kaldi es que sus herramientas son difíciles de dominar ya que su uso esta dirigido a la investigación y, por tanto,su documentación es de nivel técnico.

3.2.3. HTK

Hidden Markov Model Toolkit o HTK [24] es un conjunto integrado de herramientas para construir y manipular HMM. Estas herramientas están desarrolladas en el lenguaje de programación *C* y se ejecutan, generalmente, en entornos basados en UNIX. HTK fue diseñado para ser utilizado con propósitos generales y enfocado al campo de la investigación en RAH. Actualmente, se puede utilizar en cualquier área del conocimiento cuyo problema esté enfocado como un proceso de modelado estocástico Markoviano [25].

Inicialmente, HTK fue desarrollado para el entrenamiento de HMM y decodificación utilizando el algoritmo de Viterbi [24], sin embargo, se ha estado actualizando la librería y ahora, incluye herramientas que permiten la extracción de características acústicas, construcción y entrenamiento de modelos acústicos; y también, permiten conducir pruebas sobre los modelos [26].

HTK también ha sido utilizado en otras áreas que no sean RAH, por mencionar algunas, reconocimiento de caracteres y formas gráficas, análisis de vibraciones mecánicas y determinación de secuencias de ADN humano.

El funcionamiento de HTK es controlado por módulos basados en una arquitectura flexible y autosuficiente [25]. La utilización de las herramientas se realizan a través de línea de comandos. Entre las herramientas que se pueden utilizar se destacan las siguientes [25]:

- HAudio: Controla la adquisición de audio.
- HDict: Control sobre el diccionario del reconocedor.
- HLM: Control sobre modelos de lenguaje.
- HModel: interpreta las definiciones de HMM.
- HTrain: Entrenamiento de modelos.

- HRec: Funciones para procesado en etapa de reconocimiento.

3.3 Sistemas de RAH en el ámbito médico

El desarrollo de sistemas de gestión hospitalario o de atención primaria ha permitido que el servicio de salud esté más organizado y accesible para el ser humano. La codificación automática de procesos y la gestión de solicitudes son funcionalidades que prescinden la gran mayoría de los hospitales en el mundo. Sin embargo, son pocos los sistemas que integren soluciones de ayuda al diagnóstico médico.

Con la reciente aceptación y adopción de los sistemas de historiales médicos electrónicos (EHR, del inglés Electronic Health Record) los médicos gastan, aproximadamente, 6 de 11 horas laborables en el EHR y 1,5 horas solo en documentación [27]. Por tanto, la utilización de sistemas RAH para transcripción médica puede acelerar el proceso de documentación ya que dictar notas suele ser más rápido que escribirlas por teclado. Esto permite que los médicos puedan tener más tiempo para el diagnóstico y atención de los pacientes [28].

Los sistemas de RAH son especialmente útiles para transcribir notas de voz al momento de realizar análisis sobre una enfermedad u observación. Su aplicación más frecuente es en el reporte y diagnóstico mediante la interpretación de imágenes y pruebas clínicas en laboratorios, en donde el médico puede dictar libremente mientras está utilizando un equipo de diagnóstico [29].

A pesar de las ventajas que ofrecen los sistemas de RAH su aplicación, en el ámbito médico, trae consigo algunos retos como la complejidad lingüística del dominio, la sonorización ambiental y el estilo del dictado [30]. La complejidad lingüística radica en la gran cantidad de terminología médica que complica la eficiencia de los modelos de lenguaje. Los diferentes tipos de dispositivos para captar audio y el ruido ambiental afecta en la calidad de las grabaciones. El estilo del dictado depende de cada médico produciendo frases ligadas cuando dicta muy rápido o introduciendo silencios largos entre frases.

En el ámbito médico se comercializan muchos sistemas de RAH el cual se describirán brevemente a continuación.

eScription One

eScription One es una herramienta de reconocimiento automático del habla para la transcripción de notas médica propietaria de Nuance Communications [31]. Esta herramienta está basado en la nube y automatiza los flujos de trabajo. Ya sea dictando en un dispositivo móvil, grabadora o teléfono, los médicos pueden brindar atención centrada en el paciente y grabar rápidamente narrativas de pacientes de alta calidad.

M*Modal Fluency for Transcription

Es un conjunto de módulos para la creación de documentos médicos propietaria de M*Modal [32]. utiliza la misma tecnología basada en la nube para que pueda documentar incluso en ausencia de una funcionalidad de documentación médica en el EHR. La precisión de reconocimiento de voz se basa en los perfiles de voz colectivos de más de 200 000 médicos y requiere un entrenamiento mínimo.

Amazon Transcribe Medical

Amazon Transcribe Medical [33] es un servicio de aprendizaje automático para capturar de manera rápida y eficaz las conversaciones entre el médico y el paciente en texto para luego analizarlo con el procesamiento de lenguaje natural o para ingresarlo en los sistemas de EHR ya que el servicio está formado para comprender la terminología y el estilo del lenguaje clínico.

CAPÍTULO 4
Análisis del problema

Una de las tareas que más tiempo consume para los expertos en diagnóstico del servicio de medicina nuclear del hospital Dr. Peset es la redacción de informes médicos. Este proceso se realiza de forma mecanizada mediante ordenador en donde el médico transcribe su diagnóstico utilizando el teclado y, por tanto, se convierte en una tarea tediosa y costosa en cuanto a tiempo por la gran cantidad de pacientes que requieren diagnóstico. De este proceso ineficiente los médicos consumen gran parte del tiempo redactando informes médicos ralentizando la entrega de resultados y atención de pacientes.

Si bien existen herramientas de RAH en el mercado estos se utilizan con propósitos generales y son muy costosos. Además, no permiten personalización o adaptación al vocabulario técnico utilizado. Como se trata del servicio de medicina nuclear el lenguaje y la terminología empleada en los informes médicos es muy específica y especializada en la medicina nuclear.

Como se puede intuir, el principal problema es la gran cantidad de tiempo que se emplea para la redacción de informes médicos. Es por ello que se hace necesario una herramienta que agilice el proceso y que permita al médico dedicarse, en mayor medida, a las labores de atención y diagnóstico al paciente.

4.1 Naturaleza de los datos

Los informes médicos son redactados con el sistema de historia de salud electrónica Orion Systems y consiste en llenar una plantilla que contiene secciones con distintas finalidades como son: información relativa al paciente e impresión diagnóstica.

4.1.1. Corpus de informes médicos

De todas las secciones que dispone una plantilla, nos enfocaremos en la sección de impresión diagnóstica al ser la más extensa. Para simplificar la manipulación y procesado de los mismos se ha generado un corpus con informes médicos.

Este corpus está compuesto por 4 721 566 palabras totales y 99 939 palabras únicas que incluyen números, caracteres especiales y letras. A continuación, se detallan las características del corpus generado.

Uso de lenguaje técnico

Debido a la naturaleza del informe médico, el lenguaje empleado está lleno de terminología científica y médica. En la Figura 4.1 podemos ver un fragmento del corpus con terminología como "*hipercaptación*" y "*gammagrafía*", muy común dentro del diagnóstico en la medicina nuclear.

Abreviaciones y unidades de medida

Para agilizar la redacción de informes médicos se suele abreviar palabras o frases largas. Estas abreviaciones no siguen convenciones ni estándares y pueden referirse a múltiples interpretaciones. La utilización de abreviaciones está sujeto a la forma en que un médico redacta sus informes. Por ejemplo, para la frase "*administración intravenosa*" se suele abreviar con "*administración iv*" o "*administración i.v.*".

Se utilizan muchas unidades de medidas como centímetros (cm), milímetros (mm) y gramos (gr) lo cual se deberán tener en cuenta por su gran frecuencia de uso en los informes médicos.

Formato y espaciados

Debido a que los informes médicos son exportados en texto plano, no se tiene garantía del formato. De igual forma, es muy difícil distinguir cuándo inicia y termina un informe médico. En la Figura 4.1 se puede observar que el corpus tiene espacios y tabulaciones entre líneas de texto, posee caracteres especiales e incluso se utiliza el carácter "ì" como indicador de saltos de línea.

```
Se realiza rastreo óseo completo sin evidenciar signos gammagráficos característicos de enfermedad ósea metastásica."


"       Se realiza gammagrafía ósea en dos fases para valoración de ì
prótesis de cadera izquierda que muestra una marcada dismetría entre ì
ambas caderas, con protusión acetabular y leve refuerzo de captación ì
acetabular en fase ósea (posiblemente degenerativo) sin observarse ì
depósitos del trazador que sugieran infección protésica.
       Incidentalmente existe marcada hipercaptación en articulación ì
coxofemoral derecha de etiología degenerativa."

"
Gammagrafía ósea en tres fases de caderas, realizada tras la administración i.v. de 99mTc-HDP.

No se observa captación patológica del radiotrazador en vástago femoral de la PTC derecha.
Evidenciamos un aumento difuso de captación, con leve hiperemia asociada, en cótilo derecho a controlar evolutivamente."

"
```

Figura 4.1: Fragmento de corpus con informes médicos

4.1.2. Corpus de audios en español

Como los informes médicos están escritos en español, se requieren grabaciones en el mismo idioma para construir el modelo acústico. Existen diferentes corpus de audios disponibles en Internet, sin embargo, la gran mayoría está disponible en idioma inglés, mandarín y alemán.

Voxforge

Este proyecto tiene como objetivo recolectar grabaciones de audio de distintas personas que quieran donar su voz para ser usadas con herramientas de reconocimiento de voz libre y de código abierto [34]. Para el corpus en español, las grabaciones fueron realizadas mediante una aplicación web desarrollado por el autor del proyecto. Los locutores son de distintos países hispanohablantes que suman un total de 40 horas de audio grabado.

Entre las características específicas del audio se deben destacar:

- Frecuencia de muestreo de 16 kHz.
- Calidad del muestreo de 16-bit mono.
- Audio en formato *MS WAV*.

Estas características son muy importantes y hay que tenerlas en cuenta al momento de realizar cualquier captación audio y que se vaya a incorporar a este corpus.

4.1.3. Modelo acústico pre-entrenado

El modelo acústico disponible para el idioma español está entrenado con el corpus que se describe en la Sección 4.1.2 con valores de parámetros por defectos. La experimentación del modelo acústico se realiza con un modelo de lenguaje cuyo vocabulario es de 20 000 palabras en español alcanzando una tasa de error por palabra del 26,8 %[1] en la decodificación sobre el conjunto de prueba.

4.2 Especificación de requisitos

En esta sección se realizará la especificación de los requisitos del *software* los cuales debe cumplir la herramienta a desarrollar para garantizar la solución del problema propuesto. La metodología utilizada para definir los requisitos es el estándar IEEE 830.

4.2.1. Introducción

En los siguientes apartados se definirá el propósito, ámbito del sistema y definiciones, acrónimos y abreviaturas con el que nos guiaremos durante el desarrollo del sistema.

Propósito

El propósito de esta especificación de requisitos es determinar las características y funcionalidades del sistema a desarrollar limitando el alcance del mismo.

[1] https://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/Spanish/

Ámbito del sistema

El sistema a desarrollar permitirá transcribir de voz a texto informes médicos para el servicio de medicina nuclear del hospital universitario Dr. Peset en Valencia a través de una aplicación de escritorio compatible con el sistema operativo *Microsoft Windows*.

Definiciones, acrónimos y abreviaturas

En la siguiente Tabla 4.1 se muestra la terminología que mayormente se utiliza en el proyecto con una breve explicación de los mismos.

Término	Definición
Usuario	Se refiere a la persona que utiliza la aplicación para transcribir de voz a texto.
MAP	Nombre de la técnica utilizada para realizar adaptación al usuario.
Adaptación	Se refiere al proceso que se lleva a cabo para ajustar los parámetros del modelo acústico.
Informe	Texto generado por el sistema de reconocimiento automático del habla.

Tabla 4.1: Términos y abreviaciones utilizados en el proyecto.

4.2.2. Descripción general

Funciones del producto

A continuación, se muestra el listado de funcionalidades que se podrán realizar dentro de la aplicación en grandes rasgos:

- Transcribir de voz a texto mediante un micrófono.
- Añadir y eliminar usuarios.
- Adaptación del modelo acústico al usuario.
- Captación de datos (audio y texto) para la adaptación al usuario.
- Ajustes de hiperparámetros en el reconocimiento del habla.
- Añadir informes médicos y re-entrenar modelo de lenguaje.

Características de los usuarios

El sistema puede ser utilizado por cualquier usuario con conocimientos básicos en el uso de un ordenador. También se caracterizará por manejar el vocabulario técnico que se utiliza en el ámbito de la medicina nuclear.

Restricciones

Dentro de las restricciones generales que caben dentro del proyecto se pueden listar las siguientes:

- El desarrollo de la aplicación debe utilizar el lenguaje de programación *Java*.
- Los archivos de texto que se suministren al sistema deben ser de extensión *.txt* y en formato UTF-8.
- Se debe utilizar herramientas de código abierto.

Suposiciones y dependencias

A continuación, se describen aquellos factores que, si cambian, pueden afectar a los requisitos:

- No existen roles definidos entre los usuarios dentro de la aplicación.
- La aplicación funcionará en sistemas operativos *Microsoft Windows*.
- Reconocimiento del habla solo en idioma español.
- El contexto de reconocimiento es en medicina nuclear.

Requisitos futuros

A continuación, algunos requisitos que se pueden abordar en versiones futuras como mejoras del proyecto:

- Permitir la integración de modelos acústicos en otros idiomas.
- Permitir el entrenamiento del modelo acústico.
- Corrección en tiempo real de vocabularios y diccionarios.
- Gestión de usuarios y roles de forma centralizada.
- Integración con otros sistemas de redacción de informes médicos.

4.2.3. Requisitos específicos

Este apartado contiene el detalle de todos los requisitos específicos que se deben cumplir durante el desarrollo del proyecto.

Requisitos de interfaz de usuario

RIU01: Requisito general de la interfaz de usuario

- Descripción: El diseño de la interfaz de usuario debe ser intuitiva y de fácil uso, con la posibilidad de moverse entre pantallas a través de atajos por teclado.
- Grado de necesidad: Alta.

Requisitos de funcionalidad

RF01: Añadir nuevo usuario

- Descripción: El sistema permitirá la adición de nuevos usuarios. Esto se realizará a través de un formulario con un único campo donde ingresará el nombre del usuario.
- Grado de necesidad: Alta.

RF02: Eliminar usuario existente

- Descripción: Se permitirá la eliminación de usuarios que están registrados previamente en el sistema.
- Grado de necesidad: Media.

RF03: Crear informe médico

- Descripción: El sistema permitirá al usuario redactar informes médicos de forma natural utilizando su voz y podrá editar, copiar y pegar texto dentro del mismo.
- Grado de necesidad: Alta.

RF04: Borrar informe médico

- Descripción: El usuario podrá borrar todo el texto del informe médico mediante una única acción dentro del sistema.
- Grado de necesidad: Baja.

RF05: Ajustes de hiperparámetros de reconocimiento

- Descripción: El usuario será capaz de ajustar los hiperparámetros iniciales del modelo acústico a sus necesidades.
- Grado de necesidad: Baja.

RF06: Grabación de audios

- Descripción: El sistema permitirá la grabación de audios y también permitirá su reproducción para verificación.
- Grado de necesidad: Media.

RF07: Editar la grabación de audio

- Descripción: El usuario podrá seleccionar la parte de audio que desea guardar como archivo de audio.
- Grado de necesidad: Media.

RF08: Generación aleatoria de oraciones

- Descripción: El sistema generara oraciones del corpus de texto aleatoriamente. El usuario podrá corregir las palabras con errores.
- Grado de necesidad: Media.

RF09: Adaptación del modelo acústico a la voz del usuario

- Descripción: El sistema permitirá adaptar el modelo acústico a la voz del usuario para mejorar la precisión del reconocimiento del habla.
- Grado de necesidad: Alta.

RF10: Cargar corpus de texto

- Descripción: El usuario podrá cargar al sistema un nuevo corpus de texto con información de nuevos informes médicos.
- Grado de necesidad: Media.

RF011: Generar modelo de lenguaje

- Descripción: El sistema permitirá la generación de nuevos modelos de lenguaje con corpus de texto cargados previamente para adaptarlo a nuevos contextos.
- Grado de necesidad: Media.

4.3 Solución propuesta

A partir de la revisión bibliográfica, especificación de requisitos y del análisis del problema que se han realizado en capítulos anteriores, se ha llegado a la conclusión de desarrollar una herramienta de reconocimiento automático del habla en español para la transcripción de informes médicos y que se pueda adaptar a cualquier contexto de forma sencilla a través de una interfaz gráfica.

La propuesta consiste en desarrollar una aplicación portable que utilice *CMU Sphinx4* como *toolkit* de reconocimiento automático del habla y que permita al usuario utilizar cualquier corpus de texto para adaptarlo a sus necesidades.

Para el modelo acústico se va a utilizar un modelo entrenado con el corpus de audio disponible en *VoxForge*. Este modelo se puede encontrar en el repositorio del proyecto *CMU Sphinx4* [35]. Esto nos permite ahorrar tiempo al no tener que entrenar un nuevo modelo acústico.

Para construir el modelo de lenguaje, se hace preciso limpiar y trasformar el corpus de informes médicos de modo que no existan palabras y abreviaturas mal escritas. Una vez limpio el corpus de texto, se utilizará la herramienta *Kylm* para entrenar un modelo de N-gramas y, a su vez, se construirá un diccionario con transcripciones fonéticas como modelo léxico.

Debido a que no se dispone de un corpus de audio de los médicos del servicio de medicina nuclear del Hospital Dr. Peset se habilitará una funcionalidad para que los usuarios puedan realizar grabaciones de voz leyendo oraciones que la propia aplicación generará aleatoriamente a partir del corpus de texto inicial. Esto permite adaptar el modelo acústico al usuario y mejorar el rendimiento del sistema en general.

CAPÍTULO 5
Diseño de la solución propuesta

Una vez analizado el problema y con una solución propuesta se pasa a realizar su diseño tratando de cumplir con todos los requisitos específicos definidos en el capítulo anterior.

En este capítulo se describe la arquitectura general del sistema y las distintas funcionalidades de los módulos. Además, se definen las interacciones entre el usuario final y la aplicación. También se explican detalles sobre la tecnología a utilizar para el desarrollo del sistema.

5.1 Arquitectura general del sistema

El sistema estará compuesto de 5 módulos. Esto nos permite aislar las diferentes funcionalidades y también permite la posibilidad de añadir mejoras en un futuro. Cada módulo cuenta con su propia interfaz gráfica y gestiona la interacción del usuario con cada módulo.

5.1.1. Módulo de reconocimiento automático del habla

Este módulo es el mas importante de todos. Además, utiliza la mayor parte de las funcionalidades que nos brinda *Sphinx4*.

Sus funciones principales son:

- Cargar el modelo acústico, modelo de lenguaje y modelo léxico.
- Cargar los ajustes de hiperparámetros.
- Cargar el modelo acústico adaptado al seleccionar un usuario.
- Iniciar y detener el proceso de reconocimiento del habla.
- Generación de informes médicos.

5.1.2. Módulo de captación de datos

Este módulo realiza todo lo referente a la captación de audio y texto que se utilizará posteriormente en la adaptación del modelo acústico al usuario.

Sus funciones principales son:

- Grabación y edición de señales de audio proveniente de una fuente externa(micrófono).
- Generación aleatoria de oraciones obtenidas del corpus de texto inicial.
- Exportar la señal de audio con las características requeridas por el modelo y con formato *.wav*.
- Exportar la oración generada en un archivo de texto con formato *.txt* y codificado en UTF-8.

5.1.3. Módulo de entrenamiento para el modelo de lenguaje

Para el modelo del lenguaje se utilizan modelos de N-gramas. Estos modelos serán construidos con la herramienta *Kylm*. También se utilizará la herramienta *Stanford CoreNLP* para tareas de procesamiento de lenguaje natural.

Sus funciones principales son:

- Entrenamiento y generación de modelos tri-gramas en formato arpa.
- Construcción del modelo léxico.
- Limpieza y tratamiento del corpus de texto.

5.1.4. Módulo de adaptación al usuario

Este módulo se encarga de todos los procesos para adaptar el modelo acústico. Se utiliza la técnica MAP que modifica los parámetros observados en el modelo acústico por defecto.

Sus funciones principales son:

- Crear el corpus de texto para la adaptación.
- Generar los archivos de características acústicas.
- Acumular las estadísticas de los datos de adaptación.
- Actualizar los parámetros del modelo acústico por defecto.

5.1.5. Módulo de gestión de usuarios

Realiza operaciones sencillas sobre gestión de usuarios. Los usuarios se guardan como directorios dentro de la carpeta de instalación de la aplicación.

Sus funciones principales son:

- Mostrar formulario para crear usuario.
- Mostrar listado de usuarios disponible para eliminar.
- Actualizar listado de usuarios disponibles.

5.2 Funcionamiento del sistema

En la sección anterior se ha determinado que la arquitectura del sistema se basará en distintos módulos. Dentro de cada módulo existen procesos o tareas que los definen y que se detallan a continuación.

5.2.1. Interacción de la aplicación con el usuario

La interacción del usuario y la aplicación se realiza mediante distintas pantallas gráficas el cual llamaremos *vistas*. Por cada módulo existen uno o varias vistas que controlarán la entrada y salida de datos, y la ejecución de procesos.

Vista creación de informes médicos

Esta vista es la primera que se muestra una vez iniciado el programa y, desde aquí, se navega entre las demás. Esta vista se encargará de interactuar con el usuario en todo lo referente con la creación de informes médicos y ajustes del sistema de reconocimiento del habla.

La forma en que se interactua con la vista se muestra en la Figura 5.1 con un diagrama de interacción. Una vez que se carga la vista, el usuario puede seleccionar entre tres opciones:

1. **Reconocimiento:** Permite iniciar o detener el reconocimiento del habla.
2. **Seleccionar usuario:** Permite seleccionar el modelo acústico adaptado al usuario.
3. **Modificar parámetros del sistema:** Permite ajustar los valores por defecto y cargar el sistema de reconocimiento del habla.

Vista captación de datos

Para captar datos nuevos, ya sea para entrenar cualquiera de los tres modelos que conforman un sistema de RAH (modelo acústico, léxico y lenguaje) o para

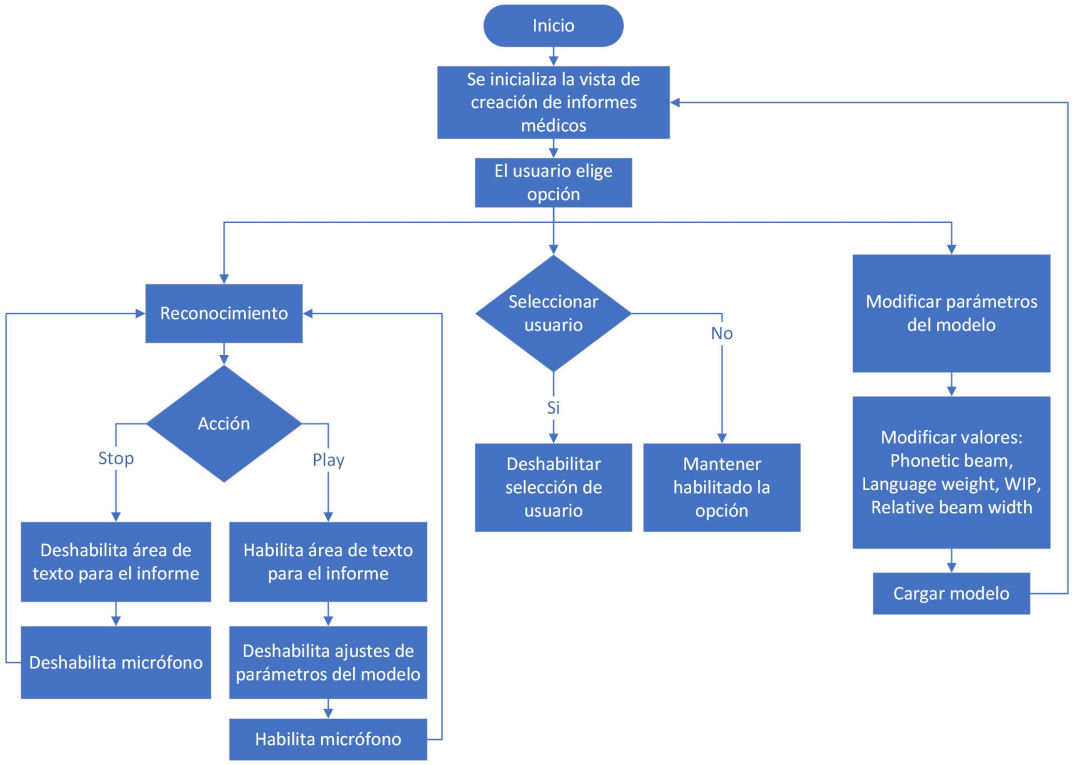

Figura 5.1: Diagrama de interacción para la vista de creación de informes médicos.

realizar adaptación del modelo existente al locutor, se dispone de una vista para dicha finalidad.

En la Figura 5.2 se muestra el diagrama de interacción. Una vez que el usuario está en la vista, el sistema muestra oraciones de forma aleatoria que el usuario tendrá que leer en voz alta al momento de realizar la grabación de audio. Al finalizar, el usuario podrá seleccionar la parte del audio que se guardara como grabación. Esto se hace para que el usuario pueda quitar los ruidos o silencios al inicio o al final de la grabación.

Figura 5.2: Diagrama de interacción para la vista de captación de datos.

Vista entrenar MAP

En esta vista el usuario podrá ejecutar los procesos relacionados con la creación del modelo acústico adaptado una vez se tengan datos para entrenar.

La interacción del usuario con la vista es muy sencilla. Como se puede ver en la Figura 5.3, el usuario ejecuta el proceso para actualizar los parámetros y crear el modelo acústico adaptado; y también visualiza el resultado del proceso.

Figura 5.3: Diagrama de interacción para la vista de entrenar MAP.

Vista gestión de usuarios

En esta vista se gestionarán los usuarios del sistema. En la Figura 5.4 podemos ver que se podrán ejecutar dos acciones:

1. **Crear usuario:** Mostrar formulario con un campo para introducir el nombre del usuario.
2. **Eliminar usuario:** Mostrar listado de usuarios disponibles.

Al ejecutar cualquiera de las opciones se debe actualizar el listado de usuarios en la vista de creación de informes médicos.

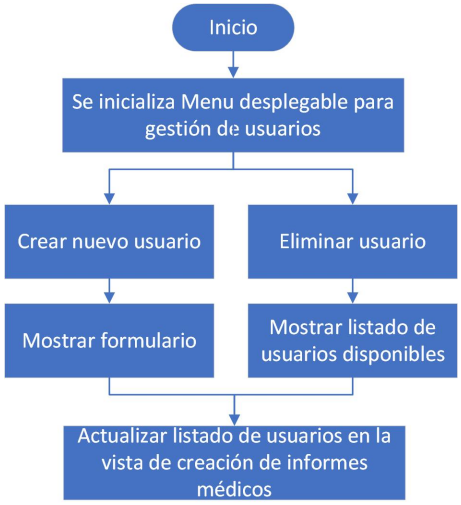

Figura 5.4: Diagrama de interacción para la vista de gestión de usuarios.

Vista modelo de lenguaje

En esta vista, el usuario podrá modelar el lenguaje utilizando cualquier corpus de texto. La interacción con esta vista se puede ver en la Figura 5.5 donde el usuario selecciona el corpus de texto a procesar y el nombre del modelo final. La vista irá informando al usuario a través de mensajes en un panel hasta que se culmine todo el proceso.

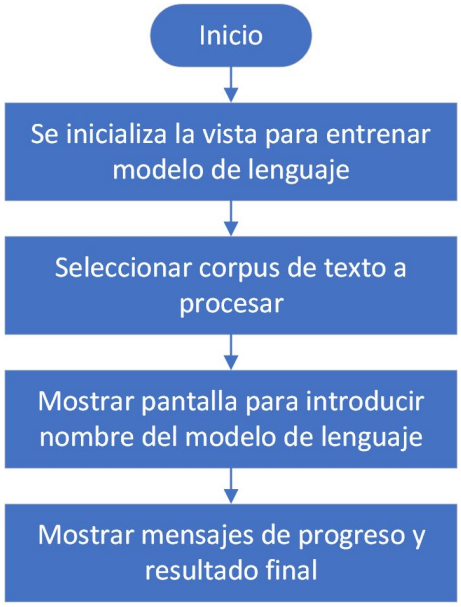

Figura 5.5: Diagrama de interacción para la vista de modelar lenguaje.

5.2.2. Ejecución de los procesos

A continuación, se detalla todo el funcionamiento de los distintos módulos propuestos en el diseño de la arquitectura general del sistema. Para tal objetivo se han desarrollado diagramas de flujos asociados a cada módulo.

Módulo de reconocimiento automático del habla

En la Figura 5.6 se muestra el diagrama de flujo referente a este módulo. La clase `App_recognizer` se encargará de gestionar los procesos de reconocimiento, ajustes de parámetros y selección de usuarios.

El proceso de reconocimiento es donde se utilizan los modelos acústico, léxico y de lenguaje para decodificar la señal de audio en texto. Para esto, se precisa cargar un archivo de configuración *XML* con los valores iniciales de todos los componentes que conforman la herramienta *Sphinx4*. Seguidamente, se cargan los modelos y se habilita la línea de entrada de audio para utilizar el micrófono. Una vez se tiene todo configurado, el sistema estará listo para iniciar el reconocimiento del habla.

El sistema permite el ajuste de parámetros iniciales del modelo. Estos parámetros son:

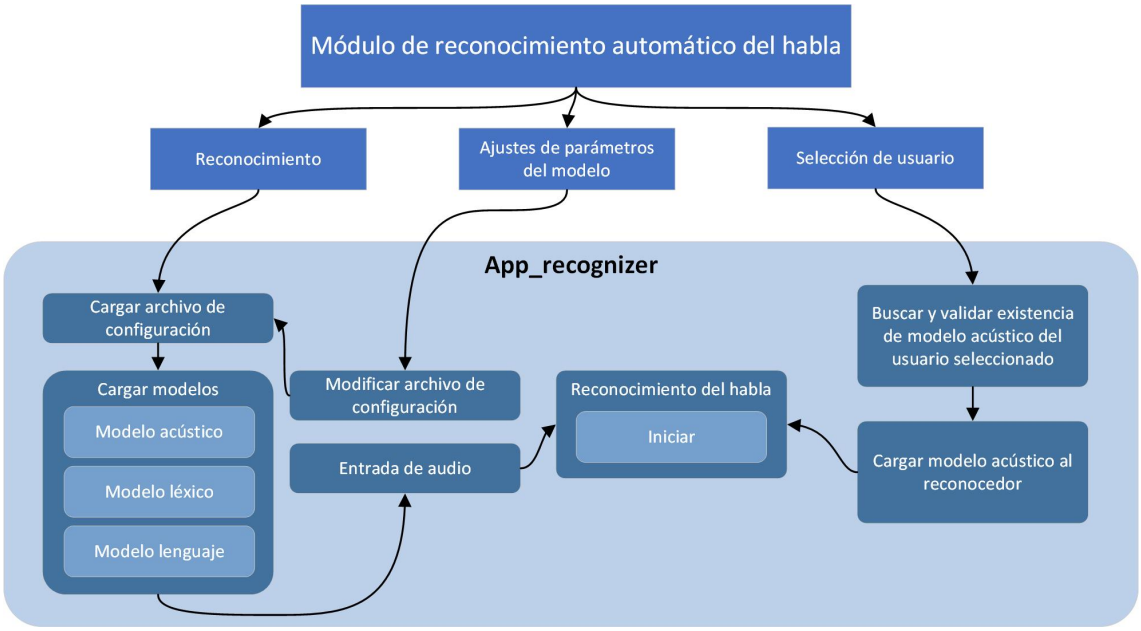

Figura 5.6: Diagrama de flujo del módulo de reconocimiento automático del habla.

- Phonetic beam.
- Language weight.
- Word insertion penalty.
- Relative beam width.

La modificación de los parámetros y la selección del modelo de lenguaje se debe cargar en tiempo de ejecución dando la posibilidad de probar diferentes configuraciones de forma fácil.

La selección de usuario permite cargar el modelo acústico adaptado. Para ello, se busca y valida la existencia de los archivos que conforman el modelo acústico adaptado en el directorio del usuario. Una vez se dispongan de los archivos, se cargarán en el sistema de RAH.

Módulo de captación de datos

Como se puede ver en la Figura 5.7, el módulo cuenta con múltiples funcionalidades. La primera, es la generación aleatoria de oraciones en la clase `Sentence_generation`, donde se carga el corpus de texto en memoria y se generan números aleatorios en un rango de `[0, lenght(corpus)-1]`.

Una vez mostrada la oración, se puede grabar el audio mediante la clase `Microphone`. Esta clase maneja todo referente a la gestión de entradas de audio. Para iniciar la grabación de debe verificar que la línea de entrada de audio esté disponible, luego se crea un hilo de ejecución que captará el audio. Para finalizar la grabación, simplemente se detiene el hilo de ejecución y se cierra la línea de entrada. Terminada la grabación, se dispara un evento que se encarga de dibujar el espectro de audio en un panel para su posterior edición.

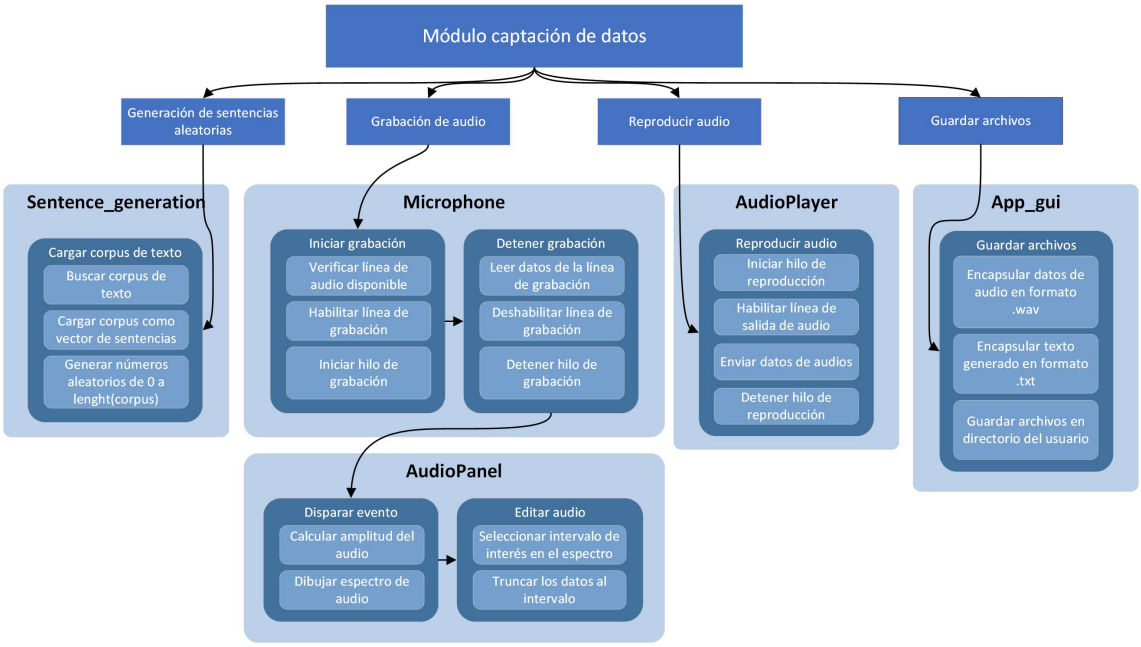

Figura 5.7: Diagrama de flujo del módulo de captación de datos.

El usuario podrá verificar y editar la grabación realizada previamente. La clase `AudioPlayer` gestiona todas las tareas de reproducción de audio. Cada vez que se reproduce un audio, se ejecuta un hilo que bloquea toda la aplicación hasta terminar la reproducción. La clase `AudioPanel` gestiona la visualización y modificación del audio grabado. La modificación se realiza seleccionando el intervalo de interés en el espectrograma de audio y truncando los datos a dicho intervalo.

Para finalizar, el usuario podrá guardar la oración generada y el audio grabado. Para ello, se generará un archivo en texto plano con extensión *.txt* para guardar la oración y se generará un archivo de audio con formato y extensión *.wav*. Estos archivos se guardarán en el directorio del usuario.

Módulo de adaptación al usuario

Para realizar la adaptación al usuario se deben tener todos los datos captados previamente. En la Figura 5.8 se muestra el diagrama de flujo para el módulo de adaptación al usuario en donde se puede ver la interacción entre sus componentes.

La clase `Generate_files` se encarga de generar los archivos en el formato necesario para que las herramientas en *Sphinx4* puedan procesarlo. Primero se crea un archivo *test.fileids* que contiene las rutas relativas de los archivos de audio. Luego se unifican todos los archivos de texto en uno solo (*test.transcription*) donde cada línea contendrá la transcripción y el nombre del archivo de audio.

Una vez se tenga todos los archivos en el formato correcto, se podrá ejecutar los procesos para actualizar los parámetros del modelo acústico por defecto y crear el modelo adaptado de cada usuario. En este proceso se puede distinguir tres pasos que se describen a continuación:

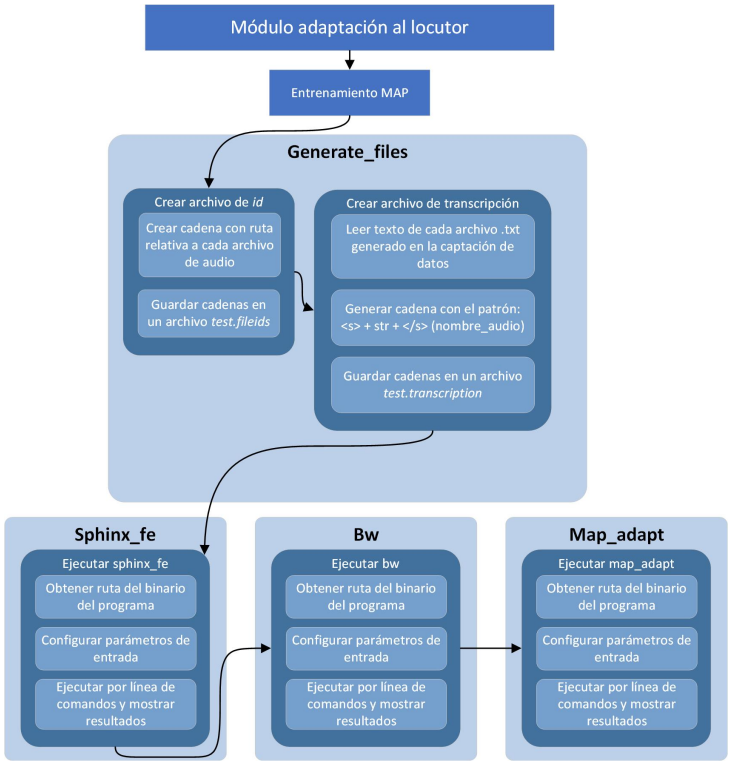

Figura 5.8: Diagrama de flujo del módulo de adaptación al usuario.

1. **Generar archivo de características acústicas.** La clase *Sphinx_fe* se encargará de generar estos archivos a partir de las grabaciones de audio y con los mismos parámetros que fueron utilizados para entrenar el modelo acústico.

2. **Acumular estadísticas de los datos.** La clase *Bw* se encargará de recolectar las estadísticas de los archivos generados en el anterior paso.

3. **Crear modelo acústico adaptado.** La clase *Map_adapt* se encargará de actualizar los parámetros y generar los archivos del modelo adaptado a partir de las acumulación de estadísticas. Como resultado, obtenemos los archivos *means*, *mixture_weights*, *variances* y *transition_matrices* con los parámetros actualizados a cada usuario.

Módulo de gestión de usuarios

Para la gestión de usuarios, se utilizará el sistema de directorios del sistema operativo para almacenar los datos referentes a los usuarios. En la Figura 5.9 se muestra el diagrama de flujo para el módulo de gestión de usuarios.

La clase `Directories` se encargará de manipular las acciones con el sistema de directorios del sistema operativo y consta de dos funciones principales:

1. **Crear nuevos usuarios.** Se mostrará un formulario para introducir el nombre. En caso de no existir, se creará una carpeta con el nombre del usuario.

2. **Eliminar usuarios.** Se mostrará un listado de todos los usuarios disponibles para eliminar del sistema.

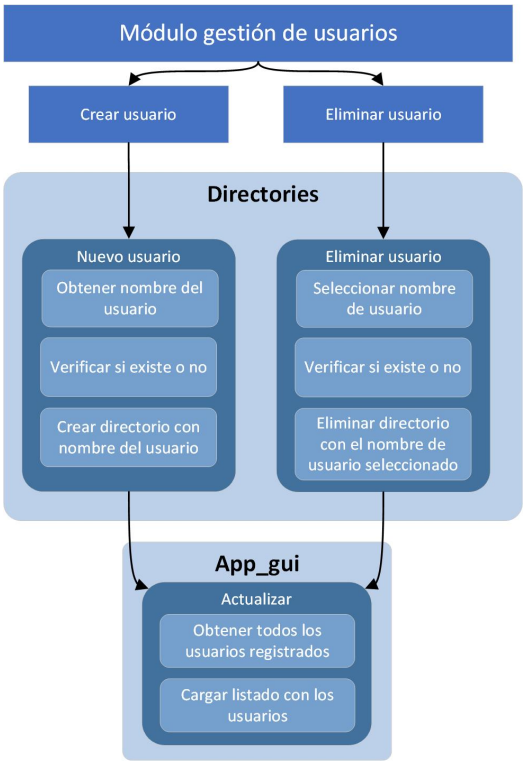

Figura 5.9: Diagrama de flujo del módulo de gestión de usuarios.

La clase `App_gui` actualiza toda la información para ser mostrada en la interfaz del usuario. Para ello, debe obtener todos los usuarios registrados y cargar las listas con la nueva información.

Módulo de entrenamiento para el modelo de lenguaje

El funcionamiento consiste en aplicar técnicas de procesamiento de lenguaje natural (NLP) a un corpus de texto y utilizar diccionarios previamente definidos para realizar correcciones ortográficas. En la Figura 5.10 se muestra el diagrama de flujo del proceso para construir un modelo de lenguaje y un modelo léxico.

La clase `LanguageModelBuilder` se encargará de tokenizar el corpus utilizando *CoreNLP* y se quitarán los caracteres no deseados con expresiones *regex*. Una vez se haya limpiado el corpus, se construirá el modelo léxico y el modelo de lenguaje. Para estimar el modelo de lenguaje se utilizará *Kylm*.

5.3 Tecnología utilizada

5.3.1. Sphinx4

El conjunto de herramientas que conforman *Sphinx4* [1] han sido diseñados para permitir un alto grado de flexibilidad y modularidad. La Figura 5.11 muestra la arquitectura general del diseño de *Sphinx4*. Cada componente representa un módulo que fácilmente se puede reemplazar.

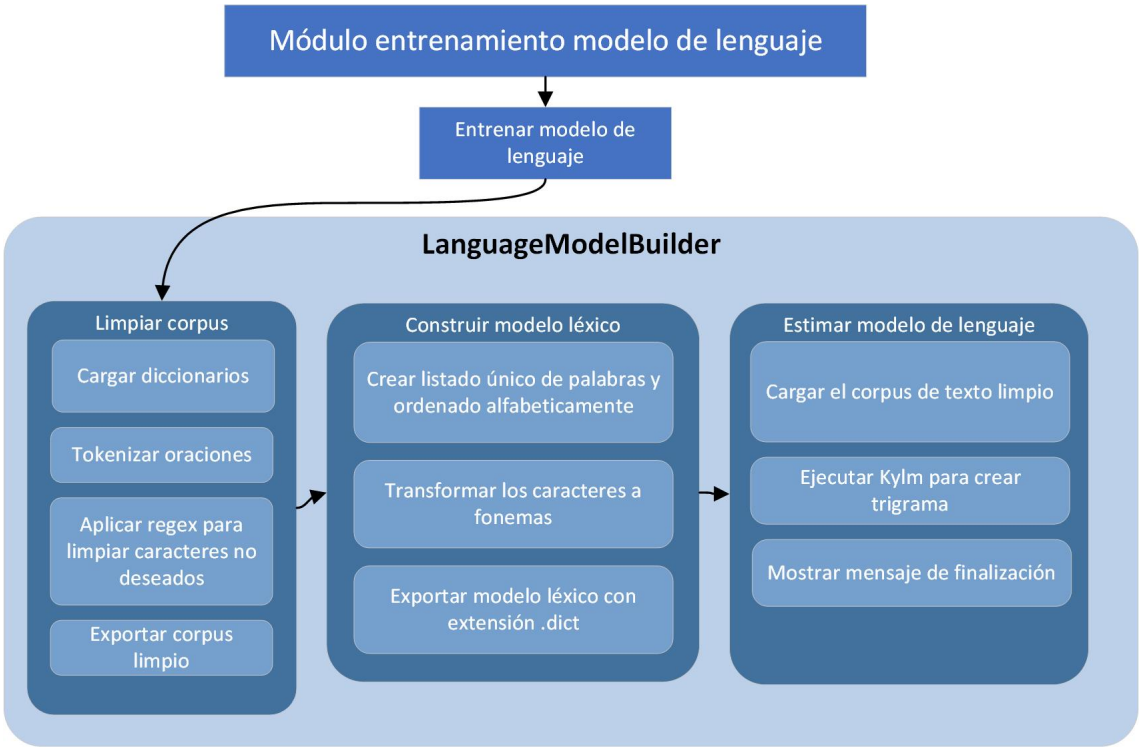

Figura 5.10: Diagrama de flujo del módulo de entrenamiento para el modelo de lenguaje.

Existen tres componentes principales: El *FrontEnd*, el *Decoder* y el *Linguist*. El *FrontEnd* se encarga de procesar la señal de audio, que puede ser un archivo o un *stream* de audio, y los parametriza en una secuencia de características (*Features*). En este caso, el procesado se realiza en tiempo real. El *Linguist* carga cualquier formato estándar de modelos lenguaje (*LanguageModel*), carga la información relacionada con la pronunciación de palabras (*Dictionary*) y carga los datos del modelo acústico (*AcousticModel*) para convertirlos en un grafo de búsqueda (*SearchGraph*). El gestor de búsqueda (*SearchManager*) que se encuentra en el *Decoder* toma las características de la señal de audio y el grafo de búsqueda para realizar la decodificación generando un resultado (*Results*).

Todos los parámetros se pueden configurar de forma dinámica y en tiempo de ejecución gracias al gestor de configuraciones (*ConfigurationManager*). Esto nos permite mayor flexibilidad a la hora de realizar ajustes en nuestro sistema de reconocimiento y probar nuevas configuraciones en menor tiempo y esfuerzo.

Sphinx4 también cuenta con una serie de herramientas y utilidades que facilitan tareas como la monitorización del rendimiento, generar *lattices*, calcular valores de confianza y uso de memoria.

5.3.2. Kylm

Para modelar el lenguaje se hace uso de librearías y herramientas que realizan este trabajo. *Kylm* [36] es un conjunto de herramientas escritas en el lenguaje de programación *Java*. Nos brinda mucha flexibilidad ya que permite configurar la estimación del modelo de lenguaje.

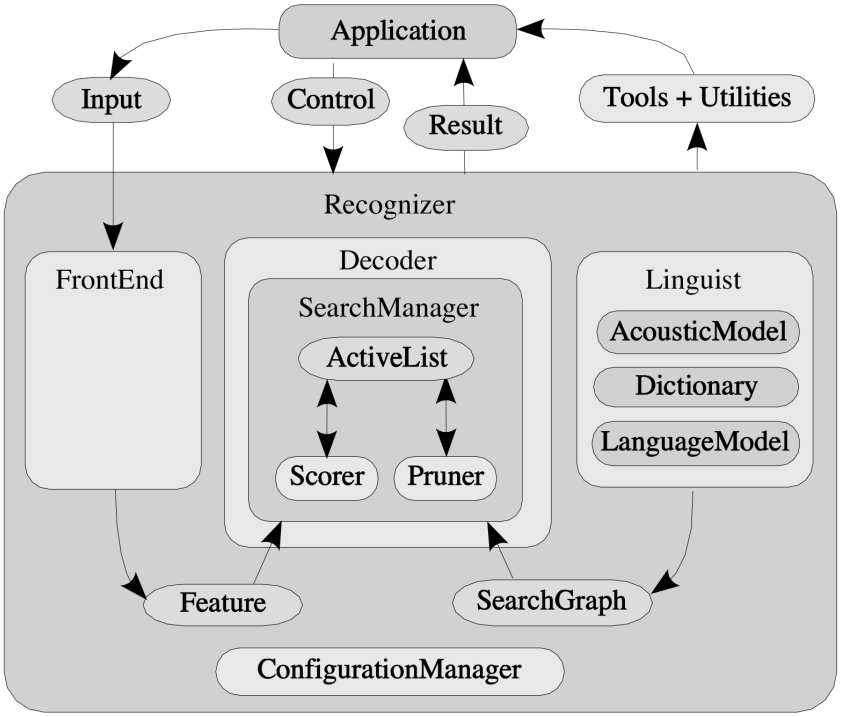

Figura 5.11: Arquitectura general de Sphinx4 [1].

Entre sus características podemos destacar:

- **Opciones de N-gramas.** Se puede definir el tamaño y nombre del N-grama.
- **Limitar símbolos y vocabulario.** Permite definir un vocabulario, indicar los símbolos que delimitan el inicio y final de una oración.
- **Técnicas de suavizado.** Permite establecer técnicas de suavizado como: *maximum likelihood*, *Good-Turing*, *Witten-Bell*, *absolute*, *Kneser-Ney* y *Kneser-Ney* modificado.
- **Formato estándar de salida.** Se puede elegir entre diferentes formatos como: *bin*, *wfst*, *arpa* y *neginf*.

5.3.3. Stanford CoreNLP

Stanford CoreNLP [2] es una herramienta desarrollada en el lenguaje de programación *Java* que permite realizar las tareas más comunes en el procesamiento del lenguaje natural. Esta herramienta es utilizada ampliamente por investigadores y empresas que necesiten de tecnologías de código abierto.

Debido a su simplicidad de uso, esta herramienta se puede integrar fácilmente en este proyecto. En la Figura 5.12 podemos ver cómo es el flujo de trabajo de *Stanford CoreNLP*. Se convierte el texto en un objeto de tipo `Annotation` y se pasa por una secuencia de procesos que ejecutan distintos análisis. El resultado es un conjunto de todos los análisis realizados que se pueden extraer como texto plano o en formato *XML*.

En la aplicación, se utiliza los anotadores `tokenize` y `ssplit` para tokenizar y separar párrafos de texto en oraciones respectivamente.

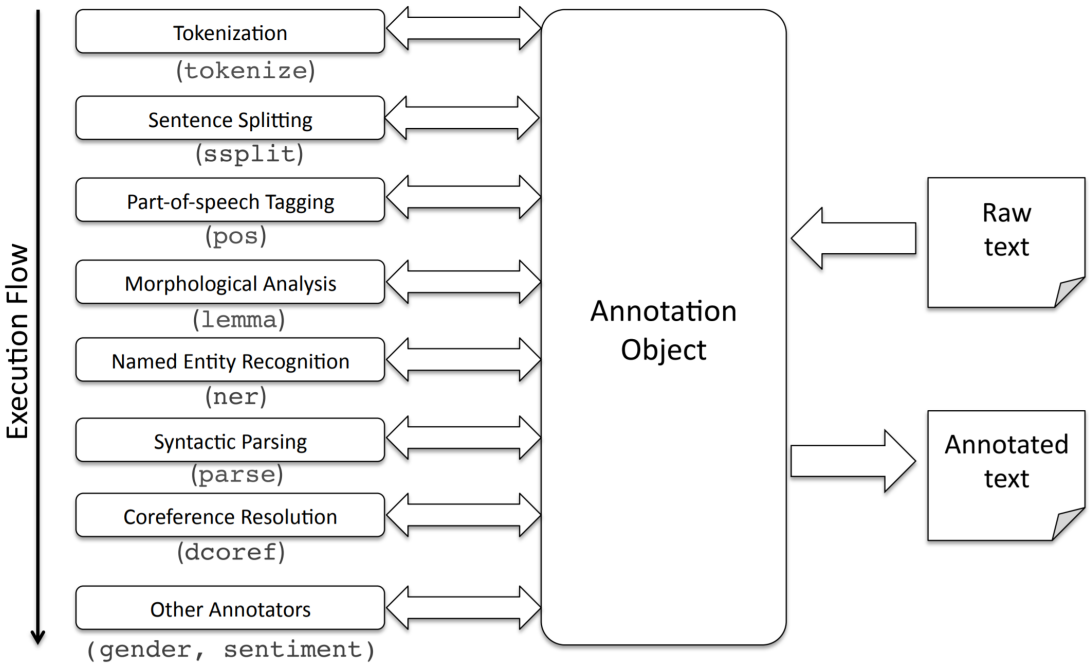

Figura 5.12: Arquitectura general de Stanford CoreNLP [2].

CAPÍTULO 6
Desarrollo de la solución propuesta

En este capítulo se describe el desarrollo de los diferentes componentes del sistema de reconocimiento automático del habla propuesto en el capítulo anterior. El desarrollo se divide en tres partes: procesamientos de los datos, lógica de negocio e interfaz de usuario. En el procesamiento de los datos se limpia el corpus de texto para construir el modelo de lenguaje y léxico. La lógica de negocio se encarga de gestionar todas las funcionalidades que utilizaremos de *Sphinx4* y también funciones propias. La interfaz de usuario es el punto de interacción entre usuario y la aplicación.

6.1 Procesamiento de los datos

Para la construcción del modelo de lenguaje y modelo léxico, disponemos de un corpus de texto con informes médicos. Este corpus, sin ser procesado, añade ruido y dificultades a la hora de construir los modelos. Para ello, debemos quitar las ambigüedades y errores ortográficos. También se precisa procesar todos los caracteres o palabras que dificulten el modelo, por ejemplo, los signos de puntuación.

Es aquí donde se establece qué palabras y caracteres serán admitidos en el sistema de reconocimiento automático del habla. También se deben definir las reglas para manejar el uso de siglas y abreviaciones. Para ello, se ha creado un diccionario de forma manual en donde se establece el formato de las siglas y abreviaciones junto a su pronunciación.

La corrección de palabras mal escritas se realiza construyendo un diccionario en formato `json` cuya llave (`key`) representa la palabra mal escrita y el valor (`value`) representa la corrección de la palabra. El proceso consiste en obtener un listado de palabras únicas y utilizar *Hunspell*[1] como corrector ortográfico para obtener una aproximación inicial. Luego, se revisa el resultado y se refina manualmente las palabras mal corregidas. Finalmente, se obtiene un diccionario con 12 048 palabras para corregir.

La clase `LanguageModelBuilder` carga los modelos en español y los anotadores `tokenize` y `ssplit` de *Stanford CoreNLP*. Además, contiene los métodos para

[1] https://hunspell.github.io/

la limpieza del corpus, creación del modelo de lenguaje y creación del modelo léxico.

6.1.1. Limpieza del corpus de texto

El proceso de limpieza y tratamiento del corpus de informes médicos lo realiza el método `cleanCorpus()` y se realiza de la siguiente manera:

1. Lee el fichero de texto que contiene el corpus de informes médicos línea a línea, omitiendo aquellas que están vacías, es decir, no contiene caracteres alfanuméricos y caracteres especiales.

2. Se crea un objeto de tipo `CoreDocument` con el corpus de informes médicos y se ejecuta el *pipeline* con los anotadores establecidos.

3. Se procesa cada `token` de cada oración detectada por *CoreNLP*.

4. Se utiliza expresiones *regex* para eliminar caracteres no deseados. Los patrones son:

   - `,+(?=\d)`: Estandariza el separador de número decimal "coma" por "punto".
   - `^\.+(?=\d)`: Añade "0." a los decimales que no tengan. Por ejemplo, ".35".
   - `[^a-zA-ZáéíóúüñÁÉÍÓÚÑ0-9\\)\(\.\%,\-\s//]`: Elimina caracteres no deseados.
   - `x{2,}`: Elimina ocurrencias de dos o más veces del carácter "x". Esto se utiliza múltiples veces en el corpus como línea separadora.
   - `\.{1,}`: Reemplaza los puntos suspensivos por un solo punto.
   - `\-{1,}`: Reemplaza guiones sucesivos por uno solo.
   - `(?<=[A-Za-záéíóú//\\-])\\.?(?=[0-9\\-\\.])`: Coloca un espacio en blanco a la izquierda de todos los números, puntos y guiones.
   - `(?<=[0-9\\-\\.])\\.?(?=[A-Za-záéíóú//\\-])`: Coloca un espacio en blanco a la derecha de todos los números, puntos y guiones.
   - `?=\\d)`: Detecta dígitos y los separa con espacios en blanco.

5. Se busca la palabra resultante en el diccionario de correcciones. Si existe, se reemplaza, sino mantenemos la misma palabra.

6. Se busca la palabra resultante en el diccionario de abreviaturas y siglas. Si existe, se reemplaza, sino mantenemos la misma palabra.

7. Comprobamos que el `token` no esté vacío y concatenamos todos los `tokens` para crear la oración.

8. Por cada oración, se normaliza las ocurrencias "i d" y "j d" por las siglas "id" y "jd" respectivamente. También eliminamos la ocurrencia "x x".

9. Por cada oración, se ajustan los nombres de fármacos, compuestos e isótopos. Por ejemplo, "1 8 F - FDG" a "18F-FDG". Se realiza de esta manera ya que, previamente, se han separado números y caracteres especiales. Por tanto, hace falta unificarlos nuevamente.

10. Finalmente, guardamos las oraciones en un archivo de texto. Cada línea representa una oración.

6.1.2. Creación del modelo de lenguaje

Hay que tener en cuenta que un informe médico puede estar constituido por múltiples oraciones. Para simplificar el modelo de lenguaje, se han separado todas las oraciones de tal forma que cada línea en el corpus represente una oración. Para realizar dicha tarea se ha utilizado `ssplit` de *CoreNLP*.

Una vez el corpus esté separado en oraciones, el siguiente paso es establecer los delimitadores de inicio y final de oración. Utilizaremos los símbolos `<s>` y `</s>` para indicar el inicio y final de la oración respectivamente. En este caso, no es necesario colocar dichos símbolos porque *Kylm* detecta automáticamente cuándo inicia y termina una oración.

Los parámetros que se utilizan para estimar el modelo de trigramas con *Kylm* son:

- **Ruta del corpus**: Ruta absoluta donde se encuentra el corpus de texto que se quiere modelar.
- **Número de elementos**: N-grama a utilizar. En este caso es tres (3).
- **Formato de salida**: Formato del modelo de lenguaje resultante (arpa).
- **Tipo de suavizado**: Se utiliza el suavizado Kneser-Ney modificado.

6.1.3. Creación del modelo léxico

Una vez se haya procesado el corpus de texto, se puede crear el modelo léxico. Para ello, se debe cargar el corpus de texto procesado y el diccionario de abreviaturas y siglas (*abc.json*). Luego, se obtiene un listado con palabras únicas ordenadas alfabéticamente. Se itera por todo el listado y se realiza lo siguiente:

- Reemplazar palabra que se está procesando si se encuentra en listado de abreviaturas y siglas (ver apéndice B).
- Reemplazar los caracteres por los fonemas (ver apéndice A).
- Guardar el resultado en un archivo con extensión *.dict*.

6.2 Lógica de negocio

El sistema se compone de 5 módulos principales tal como se describe en el Capítulo 4. En esta sección se realizará un análisis detallado de cada módulo describiendo su desarrollo y funcionamiento. Para tener una idea general de la organización del proyecto es conveniente describir la estructura de directorios de la aplicación.

El proyecto se organiza en 5 paquetes que contienen funcionalidades propias de la aplicación y 1 paquete que corresponde a la librería de *Sphinx4*. A continuación, se describe cada uno de los paquetes que se puede ver en la Figura 6.1.

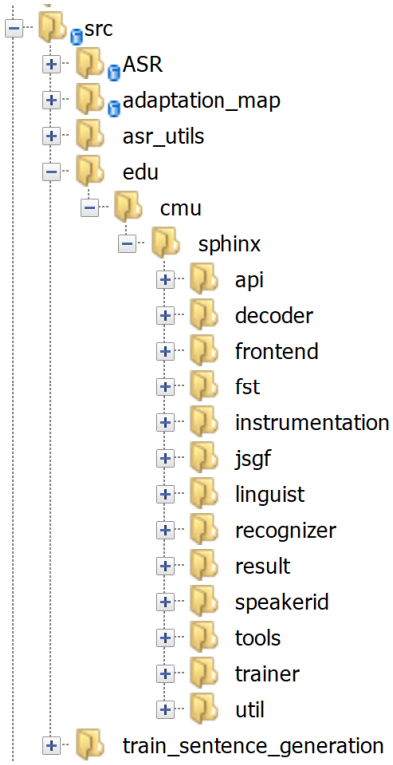

Figura 6.1: Estructura general de carpetas del proyecto.

1. **Paquete ASR**: Contiene las clases principales para iniciar y utilizar el sistema de reconocimiento automático del habla. Contiene las siguientes clases:
   - `AppGui`: Clase principal para la ejecución de la aplicación e interfaz gráfica.
   - `AppRecognizer`: Inicializa el decodificador de *Sphinx4*, abre hilo de ejecución para decodificar y realiza la carga de los hiperparámetros del reconocedor.
   - `LanguageModelBuilder`: Construcción de modelo de lenguaje, léxico y limpieza de corpus de texto.
   - `Microphone`: Clase que gestiona la interacción y comunicación con el micrófono.
   - `RecognizerConfiguration`: Obtiene el valor de los hiperparámetros desde el archivo de configuración *XML* inicial.

2. **Paquete adaptation_map**: Contiene las funcionalidades para ejecutar el proceso de adaptación al usuario con datos obtenidos previamente. Contiene las siguientes clases:
   - `Bw`: Interfaz para ejecutar la herramienta *bw.exe* de *Sphinx4* a través de línea de comando. Esta herramienta acumula estadísticas de las características acústicas.
   - `GenerateFiles`: Construye los archivos necesarios para que *Sphinx4* pueda realizar la adaptación al usuario
   - `Map_adapt`: Interfaz para ejecutar la herramienta *map_adapt.exe* de *Sphinx4* a través de línea de comando. Esta herramienta crea los archivos con los parámetros actualizados.
   - `Sphinx_fe`: Interfaz para ejecutar la herramienta *sphinx_fe.exe* de *Sphinx4* a través de ínea de comando. Esta herramienta crea los archivos con los vectores de características a partir de un archivo de audio.
   - `StreamGlobber`: Clase que redirige las salidas de texto de la línea de comando hacia un área de texto dentro de la aplicación.
3. **Paquete asr_utils**: Agrupa las clases de utilidades que facilitan el mantenimiento del código. Contiene las siguientes clases:
   - `Directories`: Clase para la gestión de usuario y tareas referente a accesos al sistema de directorios del sistema operativo.
   - `LoggerStatus`: Gestiona mensajes de estados de la aplicación para que se puedan visualizar en la interfaz gráfica.
   - `ResourceManager`: Gestiona todas las rutas a los recursos externos de la aplicación.
4. **Paquete train_sentence_generation**: Contiene la clase para la lectura de corpus de texto y generacion aleatoria de oraciones. Contiene la siguiente clase:
   - `SentenceGenerator`. Lee un corpus de texto y genera oraciones aleatorias.
5. **Paquete edu.cmu.sphinx**: Agrupa todas las funcionalidades de la herramienta *Sphinx4*. Los paquetes que se utilizan en el proyecto son:
   - `api`: Interfaz sencilla para iniciar todo el sistema de *Sphinx4* a través de un archivo de configuración XML. También incluye la gestión de las entradas y salidas de audio.
   - `tools.audio`: Herramienta para la captación, visualización y reproducción de archivos de audio.

6.2.1. Módulo de reconocimiento automático del habla

Este módulo se encarga de iniciar los parámetros y ejecutar las funciones principales de *Sphinx4* que hacen posible el reconocimiento automático del habla. A continuación se describen los componentes que lo forman:

AppRecognizer.java

Esta clase se encarga de iniciar todos los parámetros de configuración de los componentes de *Sphinx4* y también funciona como medio de comunicación entre *Sphinx4* y la aplicación. Para ello, nos apoyamos de una API que facilita poner en marcha todo el sistema de reconocimiento y, por ende, es mas fácil de manipular. La clase `LiveSpeechRecognizer` contiene los métodos para manipular el reconocedor.

Esta clase también carga los parámetros iniciales del reconocedor de *Sphinx4* y el modelo de lenguaje que el usuario haya creado previamente a través de la aplicación, todo en tiempo de ejecución. De este modo, se puede probar distintas configuraciones sin tener que modificar el archivo de configuración *XML* permitiendo mayor flexibilidad y facilidad de uso para el usuario.

Los pasos para iniciar *Sphinx4* son los siguientes:

1. El primer paso es cargar el recurso necesario para iniciar *Sphinx4*. Para ello, debemos indicar la ruta de los archivos correspondientes al modelo acústico, lenguaje y léxico; también el archivo de configuración *XML*.

   Para gestionar mejor los recursos, se ha creado una clase `Resource_manager` que contiene todas las rutas de los recursos y programas externos que se utilizaran dentro de la aplicación.

2. Cargar los parámetros de los componentes *FrontEnd, Linguist* y *Decoder*. Estos valores están en el archivo de configuración *XML*. La carga de parámetros lo realiza la clase `Context`.

3. Se configura las líneas de entrada de audio con la misma configuración con que fue entrenado el modelo acústico. La clase `Microphone` crea el objeto para la línea de entrada de audio con la configuración deseada. Los valores son:

   - Tasa de muestreo *(sampling rate)*: 16 kHz.
   - Calidad de muestre *(sample size)*: 16-bit mono.
   - *Endianness*: *Little-endian.*

4. Se inicia un hilo que ejecuta el proceso de reconocimiento del habla. Esto consiste en un bucle que siempre esta a la espera de entrada de datos.

Para realizar la carga de los hiperparámetros y el modelo de lenguaje se ha seguido los siguientes pasos:

1. Desde la interfaz gráfica, se lee el valor de los controles de hiperparámetros y se almacenan en un objeto de tipo `Map<String, String>` donde el primer valor es el nombre del hiperparámetro que se está modificando, y el segundo valor es el valor del hiperparámetro.

2. Se añade un nuevo atributo, que hace referencia al objeto creado en el paso anterior, en la clase `Configuration` de la librería *Sphinx4* y se crean los métodos `get` y `set`.

3. Se crea un nuevo método `setNewConfig(Map<String, String>)` en la clase `Context` que se encarga de modificar los valores en el `ConfigurationManager` de *Sphinx4*.

4. Se crea un nuevo método `loadConfig(Context)` en la clase `LiveSpeechRecognizer` que carga la nueva configuración en el decodificador de *Sphinx4*.

La clase `LiveSpeechRecognizer` contiene métodos para iniciar y detener el proceso de reconocimiento y el micrófono. A continuación se describen los métodos:

- **startRecognition**: Inicia la entrada de audio al sistema de reconocimiento.
- **openLineConnection**: Abre una línea de entrada de audio disponible.
- **initStartRecognition**: Inicia el sistema de reconocimiento y abre una línea de entrada de audio disponible.
- **stopRecognition**: Detiene la entrada de audio al sistema de reconocimiento.
- **closeRecognition**: Cierra la línea de entrada de audio y termina el proceso de reconocimiento.
- **loadConfig**: Carga los parámetros del reconocedor, el modelo acústico del usuario y el modelo de lenguaje en tiempo de ejecución.

RecognizerConfiguration.java

La clase `RecognizerConfiguration` obtiene los valores iniciales de los hiperparámetros que se encuentran en el archivo de configuración XML. De esta forma, podemos añadir o quitar hiperparámetros de una forma sencilla.

Debido a la gran cantidad de parámetros que se pueden configurar, se han elegido aquellos que determinan, en mayor medida, la precisión del reconocedor. Esto permite al usuario adaptarlo para conseguir un resultado óptimo.

Los hiperparámetros que se pueden modificar son:

- **Relative beam width**: Anchura de la poda en el algoritmo de Viterbi.
- **Word insertion penalty**: Penalización de la probabilidad de transición entre palabras.
- **Language weight**: Permite establecer el balance del impacto en las predicciones entre el modelo acústico y el modelo de lenguaje.
- **Phonetic beam**: Anchura de la poda aplicado a la transición entre fonemas.

6.2.2. Módulo de captación de datos

Tiene como objetivo principal obtener los archivos con los datos necesarios para entrenar un modelo acústico y permitir la adaptación de un modelo existente al usuario. Consiste en crear dos tipos de archivos de datos: un archivo de transcripción y un archivo de audio.

Para la captación y manipulación del audio, se realiza una adaptación de la herramienta `AudioTool` que se encuentra en el módulo `tools.audio` en *Sphinx4*.

A continuación se describen detalladamente sus componentes.

SentenceGenerator.java

Esta clase genera oraciones de forma aleatoria para el usuario. Las oraciones son tomadas del corpus de texto inicial que se utiliza para generar el modelo de lenguaje por defecto.

La clase contiene los siguientes métodos:

- **loadCorpus**: Este método carga en memoria las oraciones del corpus de informes médicos. Por defecto, solo carga oraciones con mas de 50 caracteres. Esto se hace para evitar oraciones muy cortas y para que el conjunto de datos resultantes sea significativo.
- **generateSentences**: Utiliza la clase `Random` para generar números aleatorios para seleccionar las oraciones.

Microphone.java

Esta clase forma parte de *Sphinx4*, se encuentra en el paquete `frontend.util` y gestiona todo lo referente a las entradas de audio. Se utiliza un archivo de configuración *XML* que contiene los parámetros que caracterizan la señal de audio.

A continuación, se describen los métodos que se utilizan en la aplicación:

- **startRecording**: Este método inicia la grabación de audio.
- **stopRecording**: Este método detiene la grabación de audio una vez que la aplicación haya leído todos los datos.
- **initialize**: Configura la línea de entrada de datos.

AudioPanel.java

Esta clase permite la manipulación y visualización de la señal de audio. Una vez captada la señal de audio a través del micrófono, se dispara un evento que ejecuta un proceso para procesar la señal y mostrarla en la interfaz gráfica.

A continuación, se describen los métodos que se utilizan en la aplicación:

- **paintComponent**: Dibuja la señal de audio en un panel editable.

- **setSelectionStart**: Indica el punto de inicio de la señal de audio.
- **setSelectionEnd**: Indica el punto de terminación de la señal de audio.
- **crop**: Selecciona los datos de la señal de audio en el rango establecido por `setSelectionStart` y `setSelectionEnd`.
- **stateChanged**: Evento que se dispara cuando se capta una nueva señal de audio.

AudioPlayer.java

Permite la reproducción del audio para verificar el resultado de la captación y manipulación. La reproducción se ejecuta en un hilo de procesamiento distinto.

Hay que tener en cuenta que este proceso es bloqueante, es decir, una vez ejecutado el proceso bloqueará todo el sistema hasta que termine la reproducción del audio. Por tanto, no se podrá realizar ninguna otra tarea.

Los métodos utilizados son:

- **play**: Inicia la reproducción de la señal de audio seleccionado en el `AudioPanel`.
- **run**: Ejecuta el hilo de procesamiento reproduciendo el audio por la salida de audio del ordenador.

AppGui.java

Esta clase contiene los la configuración y diseño de los componentes de la interfaz gráfica. El método que se utiliza para guardar los archivos se llama `saveAudioFile`. Este método exporta el texto en un archivo con extension *.txt* y convierte el audio en un archivo con formato *.wav*. Dichos archivos se deben guardan en el directorio del usuario seleccionado.

6.2.3. Módulo de entrenamiento para el modelo de lenguaje

Este módulo prepara un corpus de texto para crear un modelo de lenguaje y un modelo léxico. Además, contiene métodos y reglas para realizar el preprocesado de texto. La metodología utilizada para la limpieza del corpus, creación del modelo de lenguaje y creación del modelo léxico se describe en la Sección 6.1.

Antes de utilizar las librerías de *Stanford CoreNLP*, se debe descargar el modelo para el idioma español.[2]

A continuación, se describen detalladamente sus componentes:

LanguageModelBuilder.java

Esta clase contiene los métodos para limpiar un corpus de texto, construir el modelo de lenguaje y construir el modelo léxico.

[2] https://stanfordnlp.github.io/CoreNLP/

La clase contiene los siguientes métodos:

- **cleanCorpus**: Procesa un corpus de texto tokenizando y limpiando los caracteres no deseados.
- **buildVocab**: Construye el modelo léxico.
- **buildLm**: Construye el modelo de lenguaje en formato arpa.

6.2.4. Módulo de adaptación al usuario

Este módulo hace uso de herramientas que brinda *CMU Sphinx* para la creación y transformación de datos necesarios para la adaptación.

Antes de utilizar las herramientas, es necesario manipular los archivos de datos captados para transformarlos en un formato válido.[3] Para realizar la adaptación es necesario disponer de los archivos que se describen a continuación:

- **test.fileids**: Son archivos de textos que contienen las rutas relativas de las grabaciones de audio uno a uno. En la Figura 6.2 se muestra un ejemplo del contenido del archivo.

```
1  wav\user1\audio1
2  wav\user1\audio15
3  wav\user1\audio16
4  wav\user1\audio17
5  wav\user1\audio18
6  wav\user1\audio20
7  wav\user1\audio21
8  wav\user1\audio23
9  wav\user1\audio24
10 wav\user1\audio27
```

Figura 6.2: Ejemplo del contenido de un archivo *test.fileids*.

Cabe destacar que las rutas de los archivos de audio no incluye su extensión.

- **test.transcription**: Contiene el listado de todas las transcripciones para cada archivo de audio. La Figura 6.3 muestra un ejemplo del formato que debe seguir este archivo.

```
1  <s> el mayor de ellos es hipocaptante y está situado en el borde  </s> (audio1)
2  <s> no aparecen signos de infiltración ósea metastásica  </s> (audio15)
3  <s> captación focal diafisaria femoral izquierda a estudio  </s> (audio16)
4  <s> en cadera y rodilla derechas no se aprecian depósitos </s> (audio17)
5  <s> bocio multinodular con nódulo frío dominante izquierdo  </s> (audio18)
6  <s> masa en tercio medio de esófago con alta probabilidad  </s> (audio20)
7  <s> la gammagrafía con mibi no muestra hallazgos patológicos significativos  </s> (audio21)
8  <s> de forma incidental se aprecia aumento de captación </s> (audio23)
9  <s> tiroides de tamaño normal con ecogenicidad discretamente heterogénea </s> (audio24)
10 <s> aumento de captación en rama mandibular derecha </s> (audio27)
```

Figura 6.3: Ejemplo del contenido de un archivo *test.transcription*.

Es importante que cada oración o linea del archivo empiece con <s> y termine con </s> seguido del nombre del archivo de audio entre paréntesis. También se puede notar que el paréntesis contiene solo el nombre del archivo y

[3] https://cmusphinx.github.io/wiki/tutorialam/

no la ruta del directorio donde se encuentra. Por tal razón, es muy importante que exista coincidencia entre los archivos *test.fileids* y *test.transcription* de modo que el numero de lineas y el nombre de los archivos sean idénticos.

- **Modelo acústico**: Archivos con los parámetros aprendidos durante el entrenamiento.
  - **feat.params**: Valores de los parámetros utilizados para la extracción de características durante el entrenamiento del modelo acústico.
  - **mdef**: Archivo de definición del modelo.
  - **means**: Definición de medias gaussianas.
  - **variances**: Definición de variaciones gaussianas.
- **vocab.dict**: Debe contener una palabra por línea seguido de su transcripción fonética. En la Figura 6.4 podemos ver un extracto del vocabulario utilizado en la aplicación.

```
1  abajo a b a j o
2  abandonado a b a n d o n a d o
3  abandonar a b a n d o n a r
4  abarca a b a r k a
5  abarcan a b a r k a n
6  abarcando a b a r k a n d o
7  abarcar a b a r k a r
8  abdomen a b d o m e n
9  abdomina a b d o m i n a
10 abdominal a b d o m i n a l
```

Figura 6.4: Extracto de vocabulario *vocab.dict*.

- **test.filler**: Contiene el listado que representa sonidos que no forman parte del modelo de lenguaje. Como se puede ver en la Figura 6.5, solo se representan silencios.

```
1 </s> SIL
2 <s> SIL
3 <sil> SIL
```

Figura 6.5: Representación de silencios en el archivo *test.filler*.

- **test.phone**: Contiene un listado de los fonemas que se utilizan. En este caso, no se puede cambiar el listado ya que se está utilizando un modelo acústico que fue entrenado con estos fonemas. El listado completo se encuentra en el apéndice A.

Una vez se tienen los archivos preparados, se puede hacer uso de las herramientas *sphinx_fe*, *bw* y *map_adapt* de *CMU Sphinx*. La integración de estas herramientas con la aplicación, se hace mediante la creación de clases que ejecuten dichas herramientas a través de la línea de comando.

Sphinx_fe.java

Esta clase ejecuta la herramienta *sphinx_fe* mediante la línea de comando. *Sphinx_fe* convierte archivos de audio en archivos de características acústicas (*mel-frequency cepstral coefficients*).

Los parámetros mas importantes que se deben especificar son:

- **-argfile**: Ruta del archivo con los parámetros del modelo acústico (*feat.params*).
- **-samprate**: Frecuencia de muestreo (16 kHz).
- **-c**: Ruta del archivo que contienen las rutas relativas de las grabaciones de audio (*test.fileids*).

Bw.java

Esta clase ejecuta la herramienta *bw* mediante la línea de comando. *Bw* realiza una re-estimación Baum-Welch de los modelos ocultos de Markov dado un modelo y una secuencia de observación en un conjunto de múltiples secuencias de observaciones.

Los parámetros mas importantes que se deben especificar son:

- **-hmmdir**: Ruta del directorio que contiene los archivos del modelo acústico.
- **-moddeffn**: Ruta del archivo de definición del modelo (*mdef*).
- **-dictfn**: Ruta del archivo del modelo léxico (*vocab.dict*).
- **-ctlfn**: Ruta del archivo que contienen las rutas relativas de las grabaciones de audio (*test.fileids*).
- **-lsnfn**: Ruta del archivo de transcripciones (*test.transcription*).
- **-accumdir**: Ruta del directorio donde se guardará el resultado.

Map_adapt.java

Esta clase ejecuta la herramienta *map_adapt* mediante la línea de comando. *Map_adapt* actualiza los parámetros del modelo acústico por defecto y crea los archivos con los nuevos parámetros.

Los parámetros mas importantes que se deben especificar son:

- **-moddeffn**: Ruta del archivo de definición del modelo.
- **-meanfn**: Ruta del archivo que contiene las medias de densidad gaussianas (*means*).
- **-varfn**: Ruta del archivo que contiene las varianzas de densidad gaussianas (*variances*).

- **-mixwfn**: Ruta del archivo que contiene los parámetros de las mezclas de pesos gaussianos (*mixture_weights*).
- **-tmatfn**: Ruta del archivo que contiene la matriz de transiciones (*transition_matrices*).
- **-accumdir**: Directorio donde se encuentra las acumulaciones estadísticas.
- **-mapmeanfn**: Ruta del nuevo archivo que contiene las medias de densidad gaussianas.
- **-mapvarfn**: Ruta del nuevo archivo que contiene las varianzas de densidad gaussianas.
- **-mapmixwfn**: Ruta del nuevo archivo que contiene los parámetros de las mezclas de pesos gaussianos.
- **-maptmatfn**: Ruta del nuevo archivo que contiene la matriz de transiciones.

6.2.5. Módulo de gestión de usuarios

La aplicación permite adaptar el modelo acústico a diferentes usuarios. Para saber de qué usuario se trata la adaptación, es necesario una gestión de usuarios muy básica. La aplicación se apoya de las librerías de *Java* para la manipulación y acceso al sistema de directorios del sistema operativo.

La organización del directorio para la gestión de usuario se muestra en la Figura 6.6. El directorio `wav` contiene todos los directorios de los distintos usuarios (`user1, user2, user3`). Debe existir un directorio por cada usuario creado y, en dicho directorio, se guardarán todos los ficheros relacionados. A continuación, se describe la clase encargada de gestionar a los usuarios.

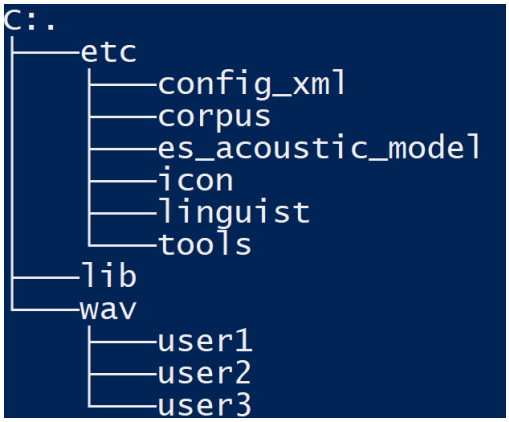

Figura 6.6: Jerarquía de directorios de la aplicación.

Directories.java

La gestión de los usuarios se realiza en la clase `Directories`. Se crea un directorio por cada usuario creado que contendrá todos los archivos que se generen para dicho usuario.

La clase contiene los siguientes métodos:

- **getAllSpeaker**: Devuelve el listado de todos los usuarios disponibles.
- **createSpeakerDir**: Crea un directorio con el nombre del usuario.
- **deleteSpeakerDir**: Elimina el directorio de un usuario específico.
- **isEmptyDir**: Valida que el directorio del usuario no esté vacío.
- **getAllLm**: Devuelve el listado de todos los modelos de lenguaje disponibles.

6.3 Interfaz del usuario

El desarrollo de la interfaz gráfica se realiza con ayuda de la librería `Swing` de *Java* y el *Swing GUI Builder* de Netbeans IDE. Todo el código referente a la interfaz gráfica se encuentra en la clase `AppGui`.

El esquema que se utiliza para organizar las diferentes vistas, que se detallan en la Sección 5.2.1, es un `CardLayout`. Este layout permite al usuario cambiar entre vistas a través de un componente `JMenu`.

En cada vista se utiliza un `GridBagLayout` para organizar los componentes que la conforman. De esta forma, la aplicación se ajusta al tamaño de la pantalla y no pierde su aspecto cuando cambia el tamaño de la ventana.

La interfaz gráfica se compone de tres partes fundamentales que podemos ver en la Figura 6.7:

1. **Barra de herramientas**: Compuesto de múltiples `JMenu` que permite la utilización de herramientas (cortar y seleccionar audio) dentro de la aplicación y cambiar a diferentes vistas. Además, incluye soporte de atajos por teclado para agilizar los accesos (ver Figura 6.8).
2. **Zona de trabajo**: Se despliegan `JPanel` correspondiente a la vista que el usuario haya seleccionado.
3. **Barra de estado**: Muestra mensajes de estados para que el usuario sepa lo que está ocurriendo en la aplicación.

6.3.1. Vista creación de informes médicos

Esta es la principal vista y es la que se muestra por defecto al iniciar la aplicación. En la Figura 6.9 se pueden ver los componentes de la vista y está conformado de cuatro partes:

1. Contiene componentes que permiten la selección de usuario (modelo acústico adaptado) y selección del modelo de lenguaje.
2. Es un `JTextArea` en donde se muestra el resultado del reconocimiento automático del habla y permite la edición del texto para correcciones manuales.

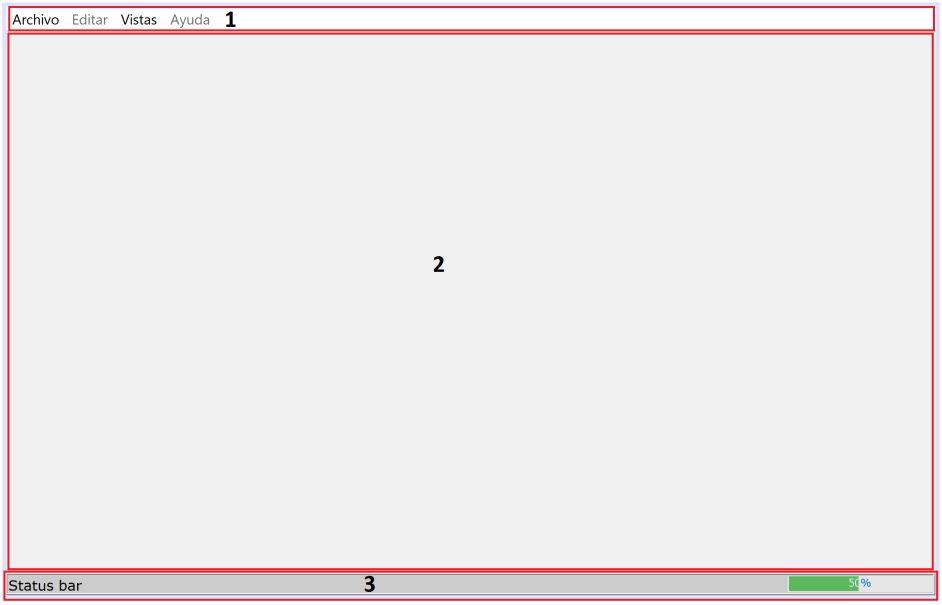

Figura 6.7: Organización de la interfaz de usuario.

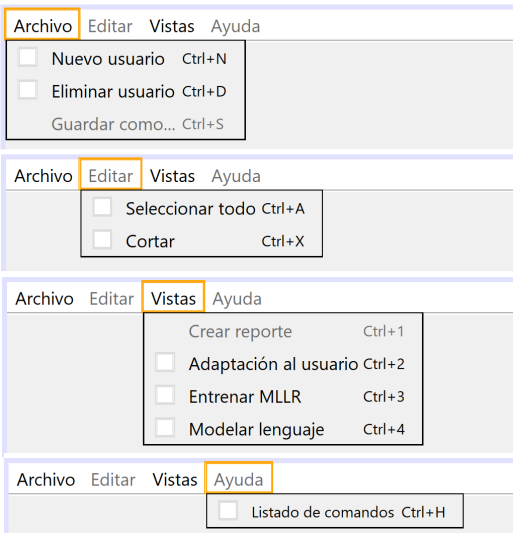

Figura 6.8: Opciones de la barra de herramientas.

3. Es un área de configuración en donde se agrupa los hiperparámetros que se pueden modificar.

4. Conjunto de botones que ejecutan las siguientes acciones:

   - **Play/Stop**: Inicia y detiene el sistema de reconocimiento.
   - **Cargar modelo**: Carga los hiperparámetros modificados y modelo de lenguaje seleccionado al sistema de reconocimiento.
   - **Borrar texto**: Borra todo el texto que se encuentra en el `JTextArea`.
   - **Copiar reporte**: Copia todo el texto que se encuentra en el `JTextArea` al portapapeles del sistema operativo.

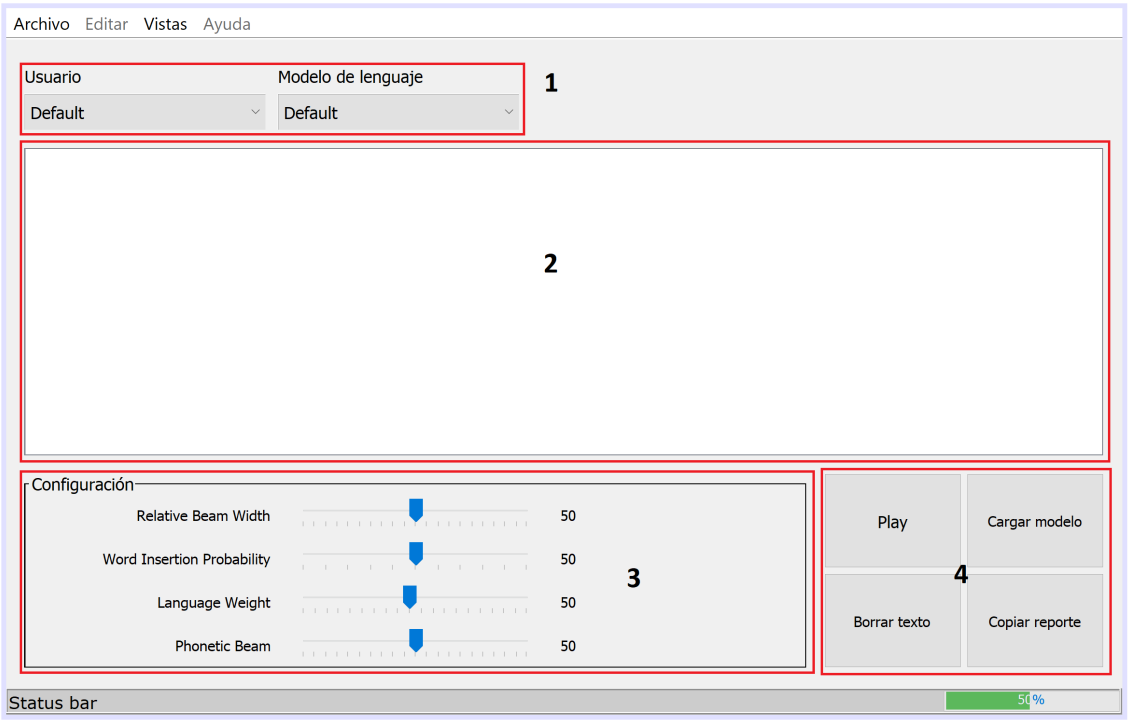

Figura 6.9: Vista creación de informes médicos.

6.3.2. Vista captación de datos

Permite al usuario generar los datos necesarios para la estimación de parámetros y para entrenar un modelo acústico. La grabación y visualización de la señal de audio se realiza adaptando las clases del paquete `tools.audio` de *Sphinx4* a las necesidades y diseño de la aplicación.

En la Figura 6.10 se puede ver la composición de la vista y su estructura está formado por tres partes:

1. Consiste en un `JTextArea` que muestra oraciones del corpus de texto que son seleccionadas aleatoriamente. El usuario puede corregir el texto manualmente.

2. Es un `JPanel` donde se pinta la señal de audio y el usuario puede seleccionar la parte del audio que desea guardar.

3. Conjunto de `JButton` para controlar la grabación de audio y visualización del tiempo de grabación. Los botones realizan las siguientes funciones:

   - **Generar**: Genera una nueva oración aleatoriamente.
   - **Record**: Inicia la grabación de audio.
   - **Play**: Reproduce el audio captado previamente para validación.
   - **Guardar**: Permite al usuario guardar los datos en un directorio específico.

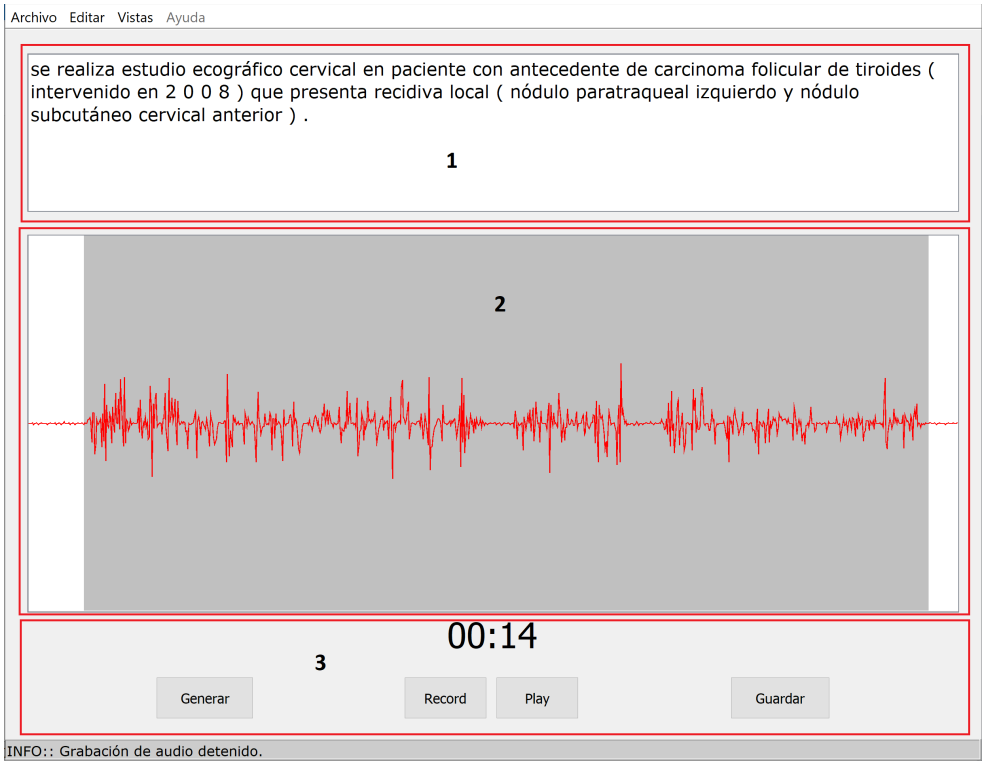

Figura 6.10: Vista captación de datos.

6.3.3. Vista entrenar MAP

Esta vista permite ejecutar los procesos para actualizar los parámetros y crear el modelo acústico de un usuario en específico. En la Figura 6.11 se puede ver que la vista está compuesta por dos partes principales:

1. Un `JTextArea` en donde se muestran los mensajes de las herramientas que se utilizan en el proceso. De este modo se puede saber si algo ha fallado o no.

2. Un `JComboBox` para listar los usuarios disponibles y un `JButton` para iniciar el proceso.

6.3.4. Vista gestión de usuarios

Debido a que la gestión de usuarios no requiere de llenar grandes formularios y tampoco se hace preciso mostrar mucha información, se ha optado por integrar la vista en la barra de herramientas. Esta vista es la única que no se encuentra en la zona de trabajo.

La vista se encuentra bajo el menú `Archivo` en la barra de herramientas (ver Figura 6.12) y está formado por dos opciones:

1. **Nuevo usuario**: Muestra una nueva ventana en donde se puede introducir el nombre del usuario a crear.

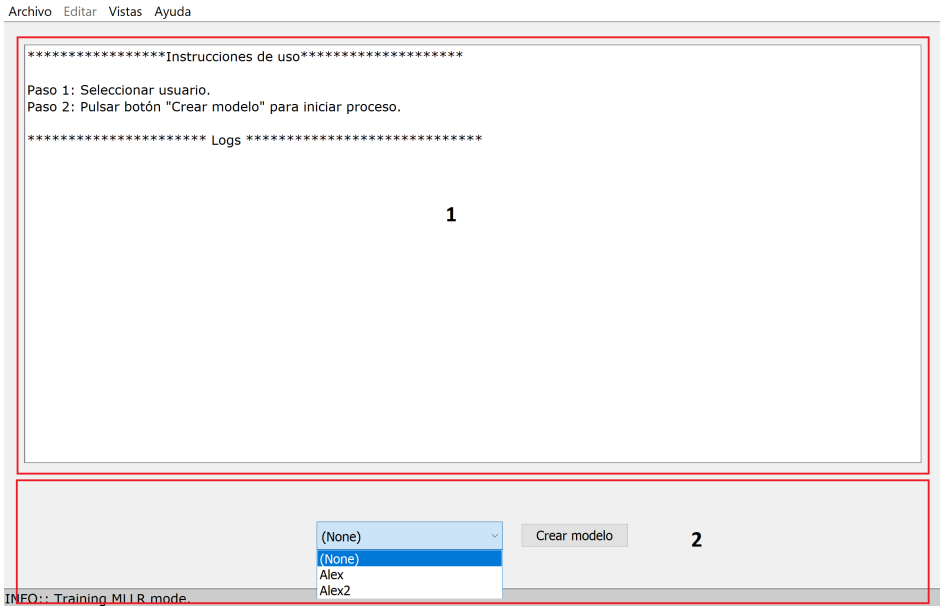

Figura 6.11: Vista entrenar MAP.

2. **Eliminar usuario**: Abre una ventana en la que se puede seleccionar el usuario a eliminar.

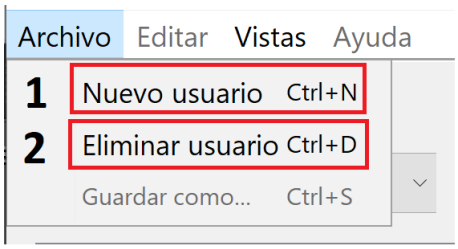

Figura 6.12: Vista gestión de usuarios.

6.3.5. Vista modelo de lenguaje

El diseño de esta vista es muy similar a la vista de entrenar MAP. El usuario selecciona el corpus de texto con el que quiera crear un nuevo modelo de lenguaje. Una vez seleccionado, se debe colocar un nombre al modelo para poder distinguirlo de los demás.

En la Figura 6.13 se puede ver que la vista esta formado por dos partes:

1. Un `JTextArea` en donde se muestran los mensajes de las herramientas que se utilizan en el proceso. De este modo se puede saber si algo ha fallado o no.

2. Un `JButton` que abre una ventana para seleccionar el corpus y otro `JButton` que inicia el proceso.

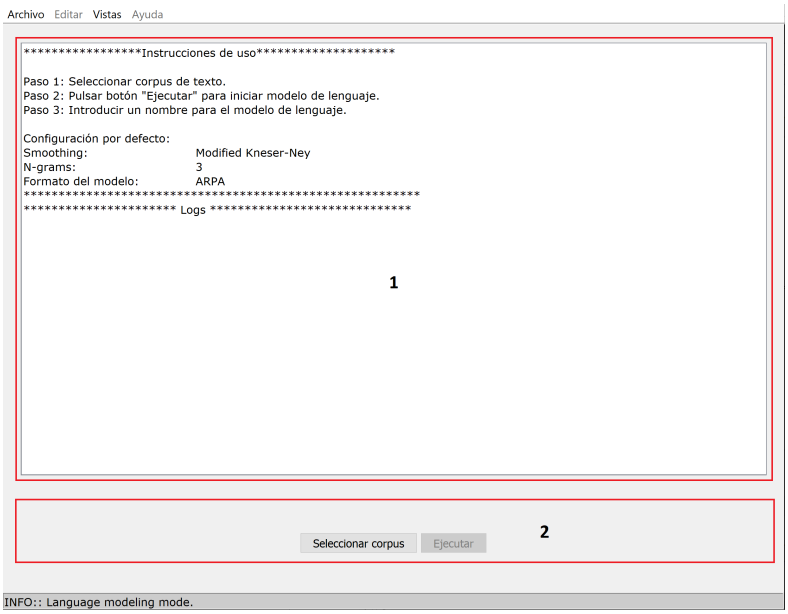

Figura 6.13: Vista modelar lenguaje.

CAPÍTULO 7
Implantación

En este capítulo se describe el proceso de empaquetado del sistema desarrollado para que sea una aplicación portable y que la implantación sea lo más sencillo posible.

Los pasos a seguir para empaquetar el sistema y crear archivos ejecutables para sistemas operativos *Microsoft Windows* son los siguientes:

1. Se debe compilar todo el código fuente del proyecto para generar el archivo ejecutable `.jar` y la carpeta `lib` que contiene las dependencias.
2. Crear una nueva carpeta y copiar los archivos generados en el paso anterior.
3. Copiar las carpetas `etc`, `lm` y `wav` en la carpeta creada en el paso anterior.
4. Descargar una versión *JDK* de *Java* igual o mayor a la 13.0.1. Se puede descargar en el siguiente enlace: https://www.oracle.com/java/technologies/javase-jdk14-downloads.html.
5. Instalar los archivos en la carpeta creada en el paso 2. Si ya se cuenta con una instalación previa, se debe copiar los archivos a dicha carpeta.
6. Descargar he instalar la aplicación *Launch4j*. Esta aplicación nos permite crear archivos ejecutables `.exe` y establecer algunas configuraciones. Se puede descargar desde el siguiente enlace: http://launch4j.sourceforge.net/
7. Una vez descargado *Launch4j*, se abre la aplicación y se debe introducir la siguiente información:
   - Basic >Output file: Colocar la ruta y nombre del ejecutable a generar.
   - Basic >Jar: Ruta del archivo ejecutable generado en el paso 1.
   - JRE >Bundled JRE path: Ruta del directorio que contiene los archivos del *JDK*.
8. Se ejecuta la aplicación para generar el archivo ejecutable.

Los pasos descritos solo se aplica cuando se realizan cambios en el código fuente del proyecto y se quiera liberar una nueva versión. La versión final de la aplicación se entrega junto a esta memoria se puede descargar con el siguiente enlace: https://sourceforge.net/projects/asr-for-medical-reporting/

Los requisitos mínimos del sistema para ejecutar la aplicación son los siguientes.

Para el reconocimiento del habla, adaptación al usuario y captación de datos:

- Sistema operativo Microsoft Windows 7 o superior.
- Procesador de 2.2 GHz o más rápido.
- 3 GB de RAM o superior.
- 1 GB de espacio disponible en disco duro.
- Interfaz entrada de audio requerido.

Para crear modelos de lenguaje:

- Sistema operativo Microsoft Windows 7 o superior (64-bits).
- 6 GB de RAM o superior.
- 1 GB de espacio disponible en disco duro.

CAPÍTULO 8
Evaluación y experimentación

En este capítulo se describen los detalles acerca de la metodología utilizada para la evaluación del sistema desarrollado y los resultados de la experimentación con usuarios finales.

La evaluación se realiza sobre el sistema de reconocimiento automático del habla en *Sphinx4* que incluye:

- Un **modelo acústico** independiente del locutor que reconoce vocabulario en idioma español.
- Un **modelo de lenguaje** en un contexto específico. En este caso, son informes de diagnóstico médico en medicina nuclear.
- Un **modelo léxico** construido a partir de los informes médicos.

Para facilitar la evaluación del sistema, se ha desarrollado una aplicación reutilizando los componentes ya desarrollados en la aplicación principal y se ha cambiado el modo de reconocimiento en tiempo real de *Sphinx4* al modo de reconocimiento por archivos de audio. De este modo, se agiliza el proceso de evaluación y captación de datos; y se tiene un mejor control sobre dichos procesos.

8.1 Metodología

La metodología para evaluar el rendimiento del sistema consiste en crear 2 corpus de texto compuestos de 10 oraciones cada una y seleccionadas aleatoriamente. El primer corpus (adaptación) se utiliza para realizar adaptación al usuario y se toman las oraciones del corpus de entrenamiento. El segundo corpus (test) se utiliza para realizar las pruebas del sistema y se toman las oraciones de un corpus que contiene un total de 12 informes médicos completos con una cantidad de 6 793 palabras y un vocabulario compuesto de 879 palabras obtenido gracias al aporte de médicos del servicio de medicina nuclear del hospital Dr. Peset en Valencia. Cada grabación realizada debe corresponder a una oración y también debe ser de habla continua. La grabación de audio se realiza para cada usuario participante. Se ha pedido la colaboración de 10 médicos del servicio de medicina nuclear en donde participan 5 hombres y 5 mujeres. Se ha pedido que dicten, de forma natural, las oraciones que se muestran en pantalla. El número de líneas

de texto, número de palabras totales, tamaño del vocabulario y tasa de OOV se muestran en la Tabla 8.3.

Para cada grupo se realizarán 3 experimentos con 5 personas diferentes. El primer experimento consiste en realizar reconocimiento con el modelo acústico por defecto. El segundo experimento consiste en realizar el reconocimiento con el modelo acústico adaptado mediante MLLR. El tercer experimento consiste en realizar el reconocimiento con el modelo acústico adaptado mediante MAP.

Se utilizan los hiperparámetros definidos por defecto y no se han explorado otras configuraciones para determinar los valores óptimos del sistema de RAH. Las técnicas de adaptación al locutor utilizadas están implementadas en *Sphinx4* por defecto.

8.2 Modelo de lenguaje

Para la construcción del modelo de lenguaje, se ha aplicado un proceso de limpieza con las reglas definidas en la Sección 6.1.1. Se utiliza un modelo de lenguaje basado en n-gramas en formato *arpa*. Para este corpus de texto, se utiliza un modelo de lenguaje de 3-gramas con suavizado Kneser-Ney modificado entrenado con la herramienta *kylm*. Se han utilizado un total de 26 fonemas para definir la pronunciación (ver apéndice A).

El corpus de entrenamiento utilizado para crear el modelo de lenguaje está compuesto de informes médicos en medicina nuclear. El número de líneas de texto, número de palabras totales y el tamaño del vocabulario respecto al corpus de entrenamiento se muestran en la Tabla 8.3.

8.3 Resultados

Los resultados del reconocimiento son presentados en la Tabla 8.1 y Tabla 8.2, en el cual se ha separado los resultados en hombres y mujeres, respectivamente. Los resultados obtenidos sobre el modelo acústico por defecto demuestran que el promedio en el WER, tanto el de hombres como el de mujeres, se asemeja al WER del 26,8 % tal como se ha explicado en la Sección 4.1.3.

Locutor	**WER Hombres**		
	Modelo (%)	**Modelo + MLLR (%)**	**Modelo + MAP (%)**
1-h	16.5	18.1	14.3
2-h	12.1	13.2	11.5
3-h	22.0	25.3	20.3
4-h	11.5	13.2	11.5
5-h	40.1	35.7	33.5
Promedio	**20.4**	**21.1**	**18.2**

Tabla 8.1: Comparación del WER entre el modelo acústico por defecto y el modelo acústico adaptado al locutor utilizando las técnicas MLLR y MAP para médicos hombres.

Locutora	WER Mujeres		
	Modelo (%)	Modelo + MLLR (%)	Modelo + MAP (%)
1-m	18.1	20.9	13.7
2-m	30.8	31.3	19.8
3-m	12.1	13.2	9.3
4-m	17.6	16.5	15.4
5-m	85.7	83.0	81.9
Promedio	**32.9**	**33.0**	**28.0**

Tabla 8.2: Comparación del WER entre el modelo acústico por defecto y el modelo acústico adaptado al locutor utilizando las técnicas MLLR y MAP para médicos mujeres.

Durante la inspección de las grabaciones de audio de todos los locutores, se ha detectado que el volumen de la voz, el acento y claridad de las pronunciaciones afecta el reconocimiento. Haciendo referencia a la locutora "5-m", presenta un WER muy elevado debido a que el volumen de su voz es muy bajo. Mientras que los locutores "2-m", "3-h" y "5-h", presentan un WER entre el 20 % y 31 %, cuya fuente de error son los acentos regionales marcados en la forma de hablar y la claridad de las pronunciaciones.

Los resultados obtenidos con la adaptación del modelo acústico muestran que el rendimiento del reconocedor mejora en 10,8 % para hombres y 14,7 % para mujeres utilizando la técnica de adaptación MAP. Mientras que utilizando la técnica MLLR, se reporta una reducción del rendimiento en el reconocimiento de 3,2 % para hombres y 0,3 % para mujeres. Se ha encontrado que ambos grupos se benefician de la adaptación mediante la técnica MAP.

Para evaluar mejor el rendimiento del sistema se ha realizado el cálculo de la tasa de error ocasionado por las palabras fuera de vocabulario (OOV, del inglés *Out Of Vocabulary*). Los OOV son palabras que aparecen en el corpus de test pero no existe en el vocabulario del corpus de entrenamiento. La Tabla 8.3 reporta el número de palabras OOV y la tasa de OOV en el corpus de test.

	Entrenamiento	Test	Adaptación
# de líneas de texto	264 342	10	10
# de palabras totales	5 727 223	1 830	2 220
Tamaño del vocabulario (palabras)	18 624	103	123
# de palabras OOV	–	50	–
Tasa de palabras OOV (%)	–	2.7	–

Tabla 8.3: Estadísticas de los corpus de textos utilizado para los experimentos en el RAH.

La Tabla 8.4 muestra algunos ejemplos donde el sistema de RAH no es capaz de reconocer las palabras OOV que se encuentran en el texto de referencia. En este caso las palabras "*aprox*", "*TRxAP*" y "*:*" son palabras OOV. Queda demostrado que el error causado por las palabras OOV afecta en la precisión del reconocimiento de las palabras que se encuentran en el vocabulario de reconocimiento.

Transcripción decodificada	**Texto de referencia**
lesión hipermetabólica en CIE de mama derecha (SULmax 9 . 6 , dados los 2 x 3 0 mm n isoecoico) , en relación con neoplasia a estudio .	lesión hipermetabólica en CIE de mama derecha (SULmax 9 . 6 , de aprox 2 2 x 3 0 mm TRxAP) , en relación con neoplasia a estudio .
- glucemia basal 3 . 2 9 8 mg / dl	- glucemia basal : 9 8 mg / dl

Tabla 8.4: Ejemplos de transcripciones decodificadas directamente por el sistemas de RAH donde no se reconoce los OOV en el texto de referencia.

CAPÍTULO 9
Conclusiones

En este proyecto se ha desarrollado un sistema de reconocimiento automático del habla utilizando el conjunto de librerías *CMU Sphinx* para la elaboración de informes médicos adaptado al lenguaje utilizado en medicina nuclear e implantado en el servicio de medicina nuclear del hospital Dr.Peset en Valencia.

El desarrollo de este proyecto a sido completado de manera satisfactoria cumpliendo con los objetivos generales y específicos planteados en la Sección 2.2 y atendiendo a los requisitos captados durante el análisis del problema.

Para completar el objetivo específico de analizar y seleccionar herramientas de RAH, se ha realizado una investigación sobre las diferentes herramientas que existen de código abierto y seleccionar la que mejor se ajuste al problema teniendo en cuenta los requisitos iniciales. Para crear un modelo de lenguaje se ha utilizado la librería *Kylm* y, el modelo léxico, se ha construido a partir de expresiones regulares. El objetivo específico de crear una interfaz gráfica intuitiva se ha logrado utilizando la interfaz de diseño integrado en *Netbeans IDE*. La integración de todos los componentes del sistema de RAH se ha logrado utilizando el lenguaje de programación *Java*; y el procesamiento del lenguaje natural se ha logrado utilizando la librería *Stanford CoreNLP*, específicamente, las funcionalidades para separar un texto en oraciones y el tokenizador.

De acuerdo a los resultados obtenidos durante la realización de los experimentos se ha demostrado que la adaptación de sistemas de RAH con modelos acústicos pre-entrenados al ámbito de la medicina nuclear es posible gracias a la utilización de modelos de lenguaje. Se ha visto que la adaptación a locutor mediante la técnica MAP es efectiva para este sistema en concreto mostrando mejoras en la precisión del reconocimiento. También queda demostrado el impacto que tienen las palabras OOV, muy comunes en medicina, en el resultado final del reconocedor.

En el desarrollo de este proyecto, se han integrado con éxito las herramientas mencionadas previamente para resolver la problemática planteada. Para ello, ha sido necesario aplicar el conocimiento de distintas áreas de la informática, en especial, el desarrollo de software y la inteligencia artificial.

Las dificultades encontradas durante el desarrollo del TFM han sido varias. El aprendizaje del funcionamiento de *CMU Sphinx* ha requerido de tiempo y muchos experimentos antes de trabajar directamente con la herramienta. Esto es debido a la falta de documentación precisa y ejemplos de cómo realizar tareas con-

cretas, por lo que se ha optado por analizar el código fuente. Otra dificultad en el proyecto fue el procesado y limpieza del corpus de texto. Este corpus de texto contiene muchos errores de tipo ortográfico. También utiliza abreviaciones y formatos varios que se tuvieron que traducir y normalizar, por ejemplo, el uso de diferentes separadores de números decimales y unidades de medidas. La limpieza del corpus y las correcciones ortográficas se realizaron con ayuda de correctores automáticos y revisión manual.

Uno de los problemas que se encontró para implantar la aplicación en los ordenadores del servicio de medicina nuclear del hospital Dr. Peset fue que dichos ordenadores mantienen políticas de uso muy estrictas, lo que dificulta la instalación de dependencias y programas. Para solventar este problema, se creó una aplicación totalmente portable con ayuda de la herramienta *Launch4j* que empaqueta todos lo recursos necesarios para que la aplicación funcione desde cualquier sitio.

Finalmente, con el desarrollo del TFM se ha aprendido a trabajar e integrar las herramientas que dispone *CMU Sphinx*. Se ha profundizado en los conceptos teóricos sobre sistemas de RAH y se pusieron en práctica mediante el uso de librerías como *Stanford CoreNLP* y *Kylm*. Se han aplicado los conocimientos sobre planificación y gestión de proyectos adquiridos durante el curso del máster. Se ha logrado desarrollar una aplicación flexible y disponible para cualquier persona, el cual puede adaptar a sus necesidades.

9.1 Relación del trabajo desarrollado con los estudios cursados

Para el desarrollo de este proyecto se han requerido la aplicación de los conocimientos adquiridos durante el curso del máster en ingeniería informática de los cuales se pueden mencionar: conocimientos en programación, desarrollo de interfaces, gestión y planificación de proyectos, programación multihilos, razonamiento probabilísticos, procesamiento de datos, funcionamiento de sistemas de información en medicina e integración de aplicaciones. Con esto queda demostrado que dichos conocimientos han servido para dar solución al problema planteado y que el contenido de este TFM ha sido desarrollado conforme a los estudios cursados.

Para concluir esta sección se destacan aquellas competencias transversales que se han requerido y puesto en practica para la elaboración de este TFM.

- **Comprensión e integración.** Ha sido necesario para integrar el conocimiento adquirido durante el curso y concretarlos en este trabajo.

- **Aplicación y pensamiento práctico.** Se han aplicado los conocimientos teóricos, se han llevado a cabo experimentos y se han analizado los resultados.

- **Análisis y resolución de problemas.** Se han analizado diferentes posibilidades y se ha identificado una solución viable.

- **Diseño y proyecto.** Se ha diseñado la aplicación y se ha planificado el proyecto hasta su culminación.
- **Conocimiento de problemas contemporáneos.** Se identificaron e interpretaron los problemas en el ámbito de la medicina nuclear con respecto a la elaboración de informes médicos.
- **Aprendizaje permanente.** Se ha requerido de aprender y profundizar en nuevas técnicas y herramientas para la realización de este TFM.

CAPÍTULO 10
Trabajos futuros

En este capítulo se presentan las líneas de trabajo a futuro que se pueden seguir con el objetivo de mejorar diferentes aspectos de la solución propuesta; y algunas propuestas de funcionalidades que son oportunas para el proyecto.

Como primera mejora, se podría optimizar el proceso de limpieza del corpus de texto. Para ello seria necesario la ejecución de los procesos en paralelo, ya sea en CPU o en GPU.

Otra de las mejoras que se pueden hacer es el re-entrenamiento del modelo acústico con más datos de audio. Se puede incorporar datos de distintas fuentes, incluso desde VoxForge, ya que cada vez más personas realizan donaciones de audio. También se puede cambiar el paradigma de GMM-HMM utilizado para construir modelos acústicos a uno basado en redes neuronales profundas. En este caso, se suele utilizar redes neuronales recurrentes. La combinación de HMM y redes neuronales profundas se como sistemas de RAH híbridos.

Por otra parte, las palabras OOV, es un problema que no se llega a abordar en este TFM. La solución a esto sería construir un modelo de lenguaje no de palabras, sino de caracteres; y realizar la decodificación en base a caracteres. Lidiar con el OOV es de mucha importancia en el campo médico debido a la complejidad del vocabulario utilizado, sobre todo, en los nombres de los fármacos. Una solución a corto plazo sería crear modelos de lenguajes utilizando corpus de texto que contengan un vocabulario mucho más amplio.

Para mejorar las transcripciones de los informes médicos, se puede aplicar técnicas de transcripción asistida. Lo que se propone es que el usuario final, en este caso el médico, pueda corregir, con el menor esfuerzo, las frases resultantes de la decodificación. En este caso, se podría utilizar el *lattices* que genera el sistema de RAH como recurso de transcripción asistida.

Bibliografía

[1] P. Lamere, P. Kwok, W. Walker, E. Gouvêa, R. Singh, B. Raj, and P. Wolf, "Design of the cmu sphinx-4 decoder.," 2003.

[2] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, pp. 55–60, 2014.

[3] H. L. Rufiner and D. H. Milone, "Sistema de reconocimiento automático del habla," *Ciencia, Docencia y Tecnología*, 2004.

[4] M. E. Esparza Arellano and J. B. Avalos Briseño, "Reconocimiento de voz," *Conciencia Tecnológica*, 2003.

[5] X. Lu, S. Li, and M. Fujimoto, "Automatic speech recognition," in *SpringerBriefs in Computer Science*, 2020.

[6] W. Ghai and N. Singh, "Literature Review on Automatic Speech Recognition," Tech. Rep. 8, 2012.

[7] Samudravijaya K, "Automatic Speech Recognition," tech. rep.

[8] E. Sanchis Arnal and J. A. Gómez, "Desarrollo de un sistema de reconocimiento automático del habla," tech. rep., 2010.

[9] D. X. Sun and F. Jelinek, "Statistical Methods for Speech Recognition," *Journal of the American Statistical Association*, 1999.

[10] K. Veselý, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks," tech. rep., 2013.

[11] U. Shrawankar and A. Mahajan, "Speech: A Challenge to Digital Signal Processing Technology for Human-to-Computer Interaction," tech. rep., may 2013.

[12] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[13] D. Jurafsky and J. Martin, "Speech and Language Processing," tech. rep., 2014.

[14] G. D. Forney, "The Viterbi Algorithm," *Proceedings of the IEEE*, 1973.

[15] M. Adda-Decker and L. Lamel, "The Use of Lexica in Automatic Speech Recognition," in *Lexicon Development for Speech and Language Processing*, pp. 235–266, Springer, Dordrecht, 2000.

[16] M. Obedkova, *Data-Driven Lexicon Generation for ASR*. PhD thesis, Universidad del País Vasco, 2019.

[17] O. Saz, E. Lleida, and A. Miguel, "Combination of acoustic and lexical speaker adaptation for disordered speech recognition," tech. rep., 2009.

[18] T. Ramya, S. L. Christina, P. Vijayalakshmi, and T. Nagarajan, "Analysis on map and mllr based speaker adaptation techniques in speech recognition," in *2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]*, pp. 1753–1758, 2014.

[19] C. J. Leggetter and P. C. Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models," tech. rep., 1995.

[20] C. . Lee and J. . Gauvain, "Speaker adaptation based on map estimation of hmm parameters," in *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, pp. 558–561 vol.2, 1993.

[21] H. Prakoso, R. Ferdiana, and R. Hartanto, "Indonesian automatic speech recognition system using cmusphinx toolkit and limited dataset," in *2016 International Symposium on Electronics and Smart Devices (ISESD)*, pp. 283–286, 2016.

[22] CMUSphinx, "Cmusphinx open source speech recognition." https://cmusphinx.github.io/wiki/about/, (consultado el 9 de junio de 2020).

[23] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz, J. Silovský, G. Stemmer, and K. Vesel, "The kaldi speech recognition toolkit," *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*, 01 2011.

[24] S. J. Young, "The HTK hidden Markov model toolkit: Design and philosophy," tech. rep., 1994.

[25] R. Carrillo Aguilar, "Diseño Y Manipulación De Modelos Ocultos De Markov , Utilizando Design and Manipulation of Hidden Markov Models Using Htk Tools . a Tutorial," *Ingeniare. Revista chilena de ingeniería*, 2007.

[26] Z. Hatala, "Practical Speech Recognition with HTK," tech. rep.

[27] C.-C. Chiu, A. Tripathi, K. Chou, C. Co, N. Jaitly, D. Jaunzeikare, A. Kannan, P. Nguyen, H. Sak, A. Sankar, J. Tansuwan, N. Wan, Y. Wu, and X. Zhang, "Speech recognition for medical conversations," tech. rep., 2018.

[28] T. Dodiya and S. Jain, "Speech Recognition System for Medical Domain," *International Journal of Computer Science and Information Technologies*, vol. 7, no. 1, pp. 185–189, 2016.

[29] B. Hnatkowska and J. Sas, "Application of automatic speech recognition to medical reports spoken in polish," 01 2008.

[30] W. Salloum, E. Edwards, S. Ghaffarzadegan, D. Suendermann-Oeft, and M. Miller, "Crowdsourced continuous improvement of medical speech recognition," in *AAAI Workshop - Technical Report*, 2017.

[31] N. Communications, "Medical transcription solutions." https://www.nuance.com/healthcare/provider-solutions/medical-transcription, (consultado el 11 de junio de 2020).

[32] M*Modal, "Advanced medical speech recognition software & ehr solutions." https://mmodal.com/speech-solutions/, (consultado el 11 de junio de 2020).

[33] Amazon, "Amazon transcribe medical." https://aws.amazon.com/es/transcribe/medical/, (consultado el 11 de junio de 2020).

[34] M. Ken, "Voxforge.org." http://www.voxforge.org/home, 2006 (consultado el 27 de marzo de 2020).

[35] C. M. University, "Cmu sphinx project resources." https://sourceforge.net/projects/cmusphinx/, (consultado el 29 de marzo de 2020).

[36] G. Neubig and X. Yao, "The kyoto language modeling toolkit." http://www.phontron.com/kylm/, 2009 (consultado el 21 de abril de 2020).

APÉNDICE A
Listado de unidades fonéticas

a
b
ch
d
e
f
g
gn
i
j
k
l
ll
m
n
o
p
r
rr
s
t
u
x
y
z
SIL

APÉNDICE B
Listado de abreviaturas y siglas


```
{
  "0": "cero",
  "1": "uno",
  "2": "dos",
  "3": "tres",
  "4": "cuatro",
  "5": "cinco",
  "6": "seis",
  "7": "siete",
  "8": "ocho",
  "9": "nueve",
  "a": "a",
  "b": "be",
  "c": "ce",
  "d": "de",
  "e": "e",
  "f": "efe",
  "g": "gramos",
  "h": "ache",
  "j": "j",
  "k": "ka",
  "L": "litros",
  "m": "eme",
  "n": "ene",
  "ñ": "eñe",
  "o": "o",
  "p": "pe",
  "q": "cu",
  "r": "ere",
  "s": "ese",
  "t": "te",
  "u": "u",
  "w": "dobleuve",
  "y": "y",
  "z": "zeta",
  ".": "punto",
```

```
"-": "comandoguion",
"/": "comandobarra",
"(": "abreparentesis",
")": "cierraparentesis",
"%": "porciento",
"FE": "comandoefee",
"spect": "espek",
"ECG": "electrocardiograma",
"PTHio": "comandopeteachei",
"18F-FDG": "diesciochoefedege",
"99mTc-DMSA": "noventaynuevedeemeesea",
"99mTc-DTPA": "noventaynuevedetepea",
"99mTc-ECD": "noventaynueveecede",
"99mTc-MAA": "noventaynueveemeaa",
"99mTc-HDP": "noventaynueveachedepe",
"99mTc-MAG3": "noventaynueveemeagetres",
"123I-Ioflupano": "cientoveintitresioflupano",
"123I-MIBG": "cientoveintitresemeibege",
"FDG": "comandoefedege",
"cps": "cepeese",
"PSA": "peesea",
"EPOC": "comandoepoc",
"ROLL": "comandorrol",
"TNM": "comandoteeneeme",
"ATPO": "comandoatepeo",
"BSGC": "comandobeesegece",
"RUSS": "comandorrus",
"OSNA": "comandoosna",
"SDRC": "comandoesedeerrece",
"OCFA": "comandooceefea",
"vs": "versus",
"TSHr": "teeseacheerre",
"IECA": "comandoieca",
"Nx": "comandoeneequis",
"Mx": "comandoemeequis",
"idx": "comandoideequis",
"mCi": "milicurios",
"mm": "milimetros",
"pg": "picogramos",
"Kg": "kilogramos",
"mcg": "microgramos",
"cm": "centimetros",
"mg": "miligramos",
"dl": "decilitros",
"ml": "mililitros",
"ng": "nanogramos",
"id": "comandoide",
"x": "comandoequis",
",": "coma",
```

```
"i.v.": "intravenoso",
"PET-CT": "pecete",
"HDP": "achedepe",
"TSH": "teeseache",
"CT": "comandocete",
"PET": "comandopet",
"TC": "tac",
"MSI": "comandoemeesei",
"RM": "erreeme",
"SUV": "suv",
"LSI": "comandoeleesei",
"LSD": "comandoeleesede",
"LDI": "comandoeledei",
"LID": "comandoeleide",
"CIE": "comandocie",
"4R": "comandocuatroerre",
"C1": "comandoceuno",
"C2": "comandocedos",
"C3": "comandocetres",
"C4": "comandocecuatro",
"C5": "comandocecinco",
"C6": "comandoceseis",
"C7": "comandocesiete",
"D1": "comandodeuno",
"D2": "comandodedos",
"D3": "comandodetres",
"D4": "comandodecuatro",
"D5": "comandodecinco",
"D6": "comandodeseis",
"D7": "comandodesiete",
"D8": "comandodeocho",
"D9": "comandodenueve",
"D10": "comandodediez",
"D11": "comandodeonce",
"D12": "comandodedoce",
"L1": "comandoeleuno",
"L2": "comandoeledos",
"L3": "comandoeletres",
"L4": "comandoelecuatro",
"L5": "comandoelecinco",
"S1": "comandoeseuno",
"S2": "comandoesedos",
"I": "romanouno",
"II": "romanodos",
"III": "romanotres",
"IV": "romanocuatro",
"V": "romanocinco",
"VI": "romanoseis",
"VII": "romanosiete",
```

```
  "VIII": "romanoocho",
  "IX": "romanonueve"
}
```