UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Implementación de un cliente de correo electrónico dirigido por voz

Proyecto Final de Carrera

Ingeniería Informática

Autor: Pablo Pellicer Javier

Director: Carlos David Martínez Hinarejos

Septiembre de 2013

Resumen

En esta memoria explicaremos los fundamentos utilizados y el proceso seguido en el desarrollo de un sistema que permita controlar un cliente de correo electrónico mediante órdenes de voz. Para ello se han utilizado tecnologías de reconocimiento automático del habla, así como diversos mecanismos para comunicar los diferentes componentes del proceso. Como resultado, obtenemos la posibilidad de utilizar la mayoría de funciones fundamentales de un cliente de correo electrónico mediante nuestra habla.

Palabras clave: modelos ocultos de Markov, reconocimiento automático del habla, correo electrónico, iATROS, Thunderbird.

Tabla de contenidos

1. Introducción

En este proyecto utilizamos la tecnología de reconocimiento de voz para permitir el manejo de un cliente de correo electrónico mediante la pronunciación de órdenes habladas. A continuación estableceremos algunos conceptos y aspectos básicos relevantes.

1.1 Reconocimiento del habla

El reconocimiento automático de voz, o reconocimiento automático del habla[9], consiste en el procesado de la voz emitida por el ser humano para obtener la información contenida, es decir, traducirla a texto.

Su uso ha ido aumentando estos últimos años, gracias al progreso de la tecnología necesaria para que tenga una precisión y velocidad aceptables, pero principalmente motivado por la comodidad que otorga. No solo eso, sino que permite hacer más accesibles algunas aplicaciones a aquellas personas no acostumbradas al uso de la tecnología, a la vez que posibilita su utilización a gente que por diversas razones o discapacidades no puede utilizar los dispositivos convencionales.

La inteligencia artificial tiene como objetivo (entre otros) permitir la comunicación hablada entre humano y máquina, para lo cual existen varias tecnologías:

- Reconocimiento automático del habla: Como hemos comentado, traduce el habla a texto. Esta es la tecnología que utilizamos para el presente proyecto.
- Procesamiento del lenguaje natural: No se limita a obtener las palabras pronunciadas, sino a analizarlas para otorgarles significado. Un ejemplo serían los modernos asistentes por voz de los teléfonos móviles (a los que se puede preguntar cosas como “¿Qué tiempo hará mañana en Nueva York?").

Para este proyecto utilizaremos el software iATROS[1] como reconocedor del habla. Hablaremos más sobre él y su funcionamiento en posteriores secciones.

1.2 Clientes de correo

Un cliente de correo es un programa de ordenador diseñado para leer y enviar mensajes de correo electrónico pertenecientes al usuario.

Dicho programa, que en ocasiones se trata de una aplicación web, se conecta a internet para obtener los mensajes del buzón de correo y mostrarlos al usuario, tras interpretar su formato. En nuestro caso hemos utilizado un programa de escritorio (Thunderbird), puesto que se puede manejar mediante atajos de teclado, el método que utilizaremos para darle órdenes.

Podemos diferenciar tres componentes imprescindibles que permiten la utilización eficaz de una cuenta de correo electrónico:

- Conexión a Internet: Dicha conexión, contratada mediante un ISP (Proveedor de Servicios de Internet), se trata de un componente imprescindible para que el cliente de correo pueda comunicarse con los servidores donde se aloja el correo.
- Cliente de correo: El cliente previamente descrito, que utiliza una conexión a internet para recibir y enviar los correos.

- Servidor de correo: Es en este servidor donde se alojan los mensajes que deseamos ver, y a través del cual enviamos nuestros mensajes. Puede ser accedido mediante el protocolo POP (Post Office Protocol) o mediante el protocolo IMAP (Internet Message Access Protocol), más moderno y con más posibilidades; en el caso de este proyecto no nos resultan de relevancia.

1.3 Objetivos

Nuestro objetivo con este proyecto es, principalmente, desarrollar un sistema que permita controlar mediante órdenes de voz las acciones más comunes de utilización de un cliente de correo electrónico, que repasaremos en la sección 4.1. Para ello, necesitaremos:

- Estudiar las bases del reconocimiento de voz y el funcionamiento de iATROS.
- Establecer los modelos léxicos y de lenguaje necesarios para que iATROS cumpla las funciones que necesitamos.
- Crear una aplicación, o intérprete, que, recibiendo como entrada las frases reconocidas por iATROS, ejecute las acciones correspondientes del cliente Thunderbird.
- Optimizar los parámetros del reconocedor para minimizar en lo máximo posible los errores de reconocimiento.

2. Reconocimiento del habla: bases

En este capítulo repasaremos la teoría tras la tecnología de reconocimiento de habla, el componente central de nuestro proyecto. Debemos destacar los siguientes aspectos:

- Modelos ocultos de Markov (HMM): Uno de los aspectos fundamentales del reconocimiento del habla, y en el que se basan los sistemas actuales. Se trata de modelos estadísticos que tienen como salida una secuencia de símbolos, a partir de la cual, generalmente, se trata de averiguar los parámetros ocultos.
- Algoritmo de Viterbi: Se trata de un algoritmo íntimamente relacionado con la utilización de modelos ocultos de Markov, pues se utiliza para hallar la secuencia de estados más probable en un HMM. En este caso, servirá para devolvernos la secuencia de símbolos (sonidos, palabras…) más probable, dada una observación.
- Aplicación de los HMM al reconocimiento del habla: Explicaremos cómo se utilizan los HMM, y el algoritmo de Viterbi en el contexto del reconocimiento del habla.
- Modelos del sistema: Léxico, acústico y del lenguaje. Estudiaremos sus aspectos teóricos, para luego explicar cómo los hemos utilizado en posteriores secciones.

A continuación, hablaremos sobre estos conceptos con algo más de profundidad.

2.1 Modelos ocultos de Markov

Un modelo oculto de Markov (o HMM, *Hidden Markov Model*) [8] se trata de un modelo estadístico de Markov en el que se asume que el sistema modelado es un proceso de Markov con parámetros no observados (ocultos).

Estos procesos de Markov que hemos mencionado son procesos estocásticos que cumplen la propiedad de Markov. Esta propiedad se cumple si los procesos carecen de memoria, lo que significa que la probabilidad para pasar de un estado *n* a un estado *m* no depende de ningún estado anterior a *n*.

Así pues, en un HMM nuestro objetivo será determinar los ya mencionados parámetros ocultos a partir de los observables (la salida). En nuestro caso, esto significará distinguir qué secuencia de estados resulta más probable.

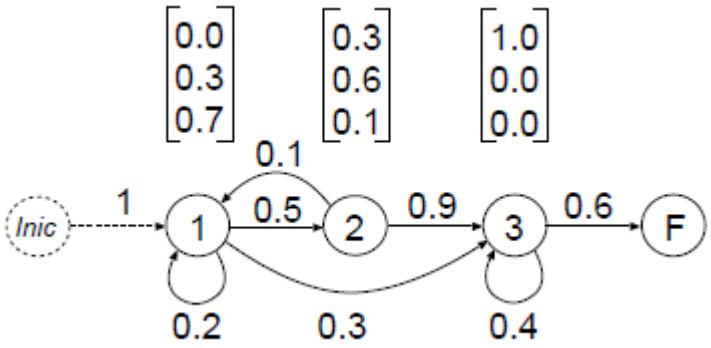
Los estados de un modelo oculto de Markov son inobservables, aunque sí podemos observar aquello a lo que influencian. Cada estado posee una serie de probabilidades de emisión para los posibles símbolos de salida, por lo que observar dicha salida puede proporcionarnos información sobre la secuencia de estados. En este caso, consideramos una palabra como cambios de fonemas, planteados como un HMM.

Un HMM se define formalmente de la siguiente forma:

HMM = (Q,V,π,A,B), donde
Q: Conjunto de estados
V: Posibles símbolos de salida
π: Probabilidades iniciales. π_i es la probabilidad de que el estado inicial sea Qi.
A: Probabilidades de transición entre estados. A_{ij} es la probabilidad de transición del estado Q_i al Q_j. Esto es, la probabilidad de que si estamos en un momento dado en el estado Q_i, el siguiente sea Q_j.
B: Probabilidades de emisión de símbolos. $B_i(V_j)$ representa la probabilidad de observar V_j cuando estamos en el estado Q_i en un instante dado.

A su vez, la secuencia de salida obtenida para una entrada con T símbolos se representa como $O = (O_1, O_2, \ldots, O_T)$.

A continuación mostraremos la representación gráfica de un HMM sencillo como ilustración. Sobre de cada estado observamos las probabilidades de emisión asociadas a él, y entre ellos vemos las probabilidades de transición. El "estado inicial" aparece como ayuda visual, para representar que $\pi_1=1$, mientras que $\pi_2 = \pi_3 = \pi_F = 0$.



Hasta ahora nos hemos referido a modelos ocultos de Markov discretos: en ellos, las observaciones adoptan un valor de una serie de posibles valores. Sin embargo, se utilizan también modelos continuos. En este caso los posibles valores de las observaciones siguen un modelo continuo, que generalmente consiste en una distribución gaussiana. En el caso del reconocimiento del habla estos serán los que utilizaremos.

2.2 Algoritmo de Viterbi

El algoritmo de Viterbi[7] es ampliamente utilizado al trabajar con modelos ocultos de Markov, puesto que sirve para hallar la secuencia más probable de estados, también llamada "camino de Viterbi". En el reconocimiento automático del habla, obtener dicha secuencia es vital, puesto que nos indicará qué se ha dicho hablando.

Formalmente, nos permite hallar, a partir de la observación O = (O_1, O_2, …, O_T), la secuencia de estados más probable, S = (S_1, S_2, …, S_T).

Considerando $\delta_t(i)$ como la probabilidad del mejor camino hasta el estado Q_i tras t observaciones:

$$\delta_{t+1}(\mathrm{i}) = \left[\max_{1\leq i\leq N} \delta_t(i)\left(A_{ij}\right)\right] B_j(O_{t+1})$$

Adicionalmente, utilizamos la variable $\varphi_t(j)$ para almacenar el argumento que hace máxima la ecuación anterior para el instante t y el estado j.

El algoritmo, pues, seguirá estos pasos (en su versión recursiva):

1. Inicialización:

   $\delta_t(i) = \pi_i B_i(O_1)$, donde $1 \leq i \leq N$

2. Recursión:

   a. $\delta_{t+1}(\mathrm{i}) = \left[\max_{1\leq i\leq N} \delta_t(i)\left(A_{ij}\right)\right] B_j(O_{t+1})$

   b. $\varphi_{t+1}(j) = \arg\max_{1\leq i\leq N} \delta_t(i) A_{ij}$

   Donde t = 1, 2, …, T-1, $1 \leq i \leq N$ y $1 \leq j \leq N$

3. Terminación:

   $q_T^* = \arg\max_{1\leq i\leq N} \delta_T(i)$

4. Reconstrucción de la secuencia de estados más probable:

   En orden inverso, es decir, desde el final hasta el principio:

   $q_t^* = \varphi_{t+1}(q_{t+1}^*)$, donde t = T-1, T-2, …, 1

Sin embargo, aplicar este algoritmo recursivamente resulta prohibitivo en situaciones realistas, debido a su coste. Esto provoca que, en la práctica, se implemente siempre mediante técnicas de programación dinámica.

2.3 Aplicación de los HMM al reconocimiento del habla

Una vez explicados los conceptos de modelos ocultos de Markov y de búsqueda de Viterbi, queda ver cómo se aplican al reconocimiento de habla humana.

El caso de reconocimiento de palabras aisladas es sencillo, pues se puede ver como una aplicación relativamente directa de los conceptos antes explicados al concepto de habla.

Si la secuencia de fonemas pronunciados (la palabra hablada) es la salida del modelo oculto de Markov continuo, debemos encontrar la secuencia de estados (la palabra del vocabulario, o del modelo léxico, en este caso) más probable.

Los HMM modelados poseen estados no emisores al principio y al final, para poder delimitar correctamente la palabra en el caso de que pasemos al reconocimiento de habla continua.

Posteriormente, pasar del reconocimiento de palabras aisladas al reconocimiento del habla continua supone el encadenamiento de los HMM, sirviéndonos de los estados no

emisores antes mencionados para “juntar” los modelos. Al hacer esto, pasamos de modelar palabras completas (como sucedía en el reconocimiento de palabras aisladas) a modelar sonidos, por lo que necesitaremos algo que nos permita determinar qué secuencia de sonidos forma una palabra: el modelo léxico.

Por supuesto, estos casos son, en la práctica, más complejos. HTK[5] utiliza una reestimación de los parámetros del HMM mediante Baum-Welch, y una estimación del camino más verosímil mediante Viterbi, puesto que se adapta con más facilidad al caso de habla continua.

2.4 Modelos

Para definir un lenguaje reconocible por un reconocedor automático del habla es necesario establecer sus modelos acústico, léxico y de lenguaje. A continuación hablaremos sobre ellos:

2.4.1 Modelo acústico

El modelo acústico define el conjunto de sonidos que pueden pertenecer al lenguaje, y que, por tanto, deben ser reconocidos.

Cada uno de estos sonidos será representado por un modelo oculto de Markov continuo, lo que significa que sus salidas están modeladas por una distribución probabilística (como veremos posteriormente, en este caso se trata de distribuciones gaussianas).

Con el modelo acústico definido se podrá establecer el modelo léxico, estableciendo referencias a los sonidos contemplados en el primero. En la sección 3.1 veremos cómo funciona esto en el caso particular de iATROS, así como más detalles sobre los modelos que utilizamos en este caso.

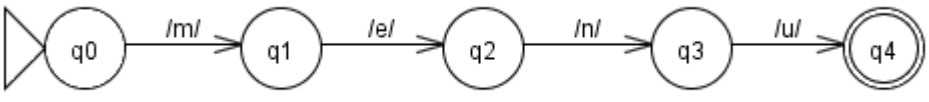
2.4.2 Modelo léxico

El modelo léxico, por su parte, es el encargado de establecer qué palabras pueden ser aceptadas por el reconocedor, para su posterior utilización en el modelo del lenguaje, y su secuencia de fonemas equivalente, pertenecientes en su totalidad al modelo acústico que hemos acabado de explicar.

Así pues, una palabra se puede contemplar como un autómata en el que las transiciones se ejecutan mediante los fonemas que la componen. Utilizando esta representación es posible contemplar diferentes pronunciaciones para una única palabra.

El modelo sería, por tanto, una colección de estos autómatas, cada uno representando una de las palabras aceptadas.

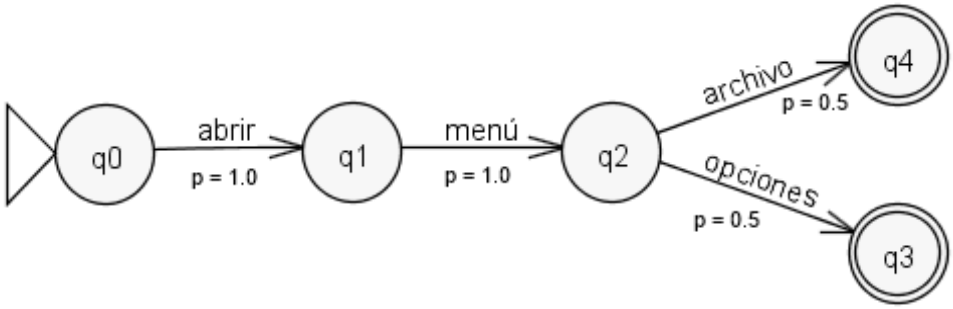
A continuación mostramos cómo se representaría, por ejemplo, la palabra “menú”:



2.4.3 Modelo del lenguaje

Finalmente, este modelo define todas las frases aceptadas por el reconocedor, utilizando palabras presentes en el modelo léxico. Para representar estas frases aceptadas se utiliza un único autómata en el que las palabras actúan como entrada, para causar las transiciones. Dichas transiciones tienen, además, una probabilidad asociada, que actúa como peso. En ocasiones se modela el silencio como una palabra más, que ha de aparecer al principio y al final de la frase.

A continuación mostraremos la visualización de un autómata sencillo que admite "abrir menú archivo" y "abrir menú opciones".



En el apartado 3.1.3 trataremos el formato mediante el cual se definen estos autómatas en iATROS.

Esta explicación del modelo del lenguaje corresponde a un modelo de tipo FSM (Finite State Machine), o lo que es lo mismo, a un modelo definido mediante autómatas de estados finitos.

También es posible, e incluso recomendable en otros proyectos, la definición de modelos de lenguaje mediante n-gramas.

Los n-gramas son colecciones de secuencias de palabras de longitud n, y su utilidad sería basar el modelo del lenguaje en la estadística[4], por lo que definir un modelo del lenguaje mediante su utilización necesitaría un corpus de frases muy grande. Esto, junto a la sencillez de la tarea (nuestros modelos del lenguaje más complejos tienen en torno a 50 frases), ha hecho que nos decantemos por los modelos a base de autómatas.

3 Aplicaciones

A continuación introduciremos los programas externos que han resultado clave para el desarrollo de este proyecto. En concreto, queremos destacar tres:

- iATROS: El reconocedor de habla utilizado.
- Thunderbird: El cliente de correo sobre el cual hemos trabajado.
- XMacro: La utilidad empleada para enviar eventos de teclado a Thunderbird y así poder manejarlo.

Adicionalmente, repasaremos otras utilidades y scripts que han sido de utilidad al desarrollar este proyecto.

3.1 iATROS

iATROS (improved ATROS)[1][6] es una nueva implementación de un reconocedor de voz ya existente, que ha sido adaptada para poder ser utilizada en casos de reconocimiento de texto manuscrito y habla. Se trata de una herramienta desarrollada por el grupo PRHLT (Pattern Recognition and Human Language Technologies)[12] de la Universitat Politècnica de València.

Con un núcleo que consiste de una búsqueda similar a la de Viterbi en una red de modelos ocultos de Markov, iATROS posee una estructura modular y ofrece herramientas estándar para el reconocimiento *off-line* y *on-line* de voz

En nuestro caso, dada la naturaleza del proyecto, no bastaría con utilizar la versión estándar de iATROS, puesto que, como no utilizaremos el mismo modelo de lenguaje para todas las pantallas y menús del cliente de correo, necesitamos poder cambiar de modelo al vuelo.

Para solucionar este problema, hemos utilizado una versión modificada de iATROS, que posee la capacidad de utilizar, en cada momento, un modelo de lenguaje u otro, según indicación del usuario o de programas que se comuniquen con el reconocedor. Por lo demás, utiliza el mismo formato de modelos. Sí que cambia ligeramente, sin embargo, el formato del fichero de configuración, que explicaremos más adelante.

A continuación hablaremos sobre los tres modelos que utiliza iATROS: el acústico, el léxico y el de lenguaje.

3.1.1 Modelo acústico

El modelo acústico, presente en el fichero “albayzin_iatros_64gs.hmm”, contiene varios modelos para cada fonema que pueda reconocer, representados por modelos ocultos de Markov continuos. En el desarrollo y entrenamiento de dicho fichero (explicado en [3]) se utilizaron grandes corpus con el objetivo de obtener un modelo acústico que representara la variabilidad en el habla que podría darse en casos prácticos: diferencias dialectales, de estado de ánimo, del ambiente de trabajo, etc. Por tanto, disponemos de un modelo acústico adecuado para el habla en español.

El modelo sigue esta estructura: para cada fonema, se define un modelo oculto de Markov que lo representa, para el cual se definen tres estados. Cada estado consta de 64 gaussianas de 39 componentes, correspondientes a 12 cepstrales + energía, con

sus derivadas y aceleración. La topología es estrictamente lineal, con una matriz de covarianzas diagonal.

Este fragmento está extraído del modelo acústico utilizado, y corresponde a parte de la definición del sonido “o”:

```
~h "o"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<NUMMIXES> 64
<MIXTURE> 1 3.052871e-03
<MEAN> 39
1.583073e+04 1.732582e+03 3.028561e+01 1.542297e+03 -2.699875e+02
8.402581e+02 2.143335e+01 1.157919e+03 2.505916e+01 1.089636e+03 -
4.196623e+01 9.205477e+02 2.482002e+01 1.675120e+03 6.343163e+02
1.106475e+02 2.509541e+02 9.810868e+01 4.317574e+01 9.599355e+01
1.198149e+02 6.918771e+01 1.165093e+02 4.311118e+01 1.041551e+02
5.712851e+01 -5.152733e+00 -4.390620e+01 -3.267732e+01 -
1.655329e+01 1.069001e+01 1.518766e+01 3.244548e+00 1.726676e+00 -
1.506276e+01 -7.406901e+00 -1.348508e+01 -2.249578e+00 -
5.251192e+00
<VARIANCE> 39
2.214582e+06 4.144199e+05 1.480758e+05 1.106104e+05 9.003127e+04
9.953895e+04 5.884056e+04 2.972439e+04 2.652506e+04 3.803510e+04
3.638104e+04 2.413211e+04 1.918245e+04 5.924761e+05 3.868758e+04
1.829826e+04 9.599272e+03 1.579836e+04 7.222038e+03 5.763008e+03
4.889083e+03 4.948525e+03 4.161598e+03 4.303112e+03 4.234914e+03
2.378068e+03 6.157296e+04 8.548314e+03 1.637003e+03 1.815245e+03
1.015218e+03 9.373586e+02 8.965078e+02 5.777987e+02 6.613397e+02
5.282754e+02 5.847682e+02 5.649480e+02 3.965851e+02
<GCONST> 4.313702e+02


{Aquí el resto de gaussianas, para este estado y los otros}


<TRANSP> 5
 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 3.819357e-01 6.180643e-01 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 5.101127e-01 4.898873e-01 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 4.782785e-01 5.217215e-01
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
<ENDHMM>
```

3.1.2 Modelo léxico

Este modelo incluye todas las palabras admitidas para el lenguaje definido, junto con su transcripción a una serie de fonemas (que referencian los fonemas definidos en el modelo acústico). Anteriormente, habíamos declarado que este modelo utiliza un autómata para cada palabra, permitiendo así la representación de pronunciaciones alternativas. En nuestro caso, no ha sido necesario aprovechar esta posibilidad, por lo que

hemos utilizado un formato simplificado, el utilizado por HTK. Mostramos a continuación la representación de la palabra “mensaje” en ambos formatos:

Formato v2 (estándar de iATROS)	Formato estándar de HTK
Name “mensaje” State 0 i = 1 0 1 “m” p = 1 1 2 “e” p = 1 2 3 “n” p = 1 3 4 “s” p = 1 4 5 “a” p = 1 5 6 “x” p = 1 6 7 “e” p = 1 State 7 f = 1	mensaje 1.0 m e n s a x e

Como podemos observar, se muestran diversas probabilidades. En el formato v2, representan las probabilidades de transición entre estados, mientras que en el formato de HTK, representan la probabilidad de pronunciación de una palabra. Como solo contemplamos una pronunciación para cada palabra, esta probabilidad será siempre de 1.

No solo necesitamos contemplar las palabras normales, sino que también hay que tener en cuenta el silencio. Así pues, las palabras “<s>” y “</s>”, empleadas por el modelo de lenguaje para el inicio y fin de frase, se traducirían como el fonema “S”.

3.1.3 Modelo del lenguaje

El modelo del lenguaje define todas las frases que han de ser admitidas en el lenguaje deseado, utilizando palabras presentes en el modelo léxico. Estas frases se representan en forma de un autómata, con un único estado inicial y final (correspondientes al silencio).

Cada estado posee una línea para definirlo, indicando su número (desde el cero) y si se trata de un estado inicial o final (cuando corresponda). Dicha línea precede la lista de transiciones que salen del estado en cuestión. Cada transición sigue este esquema:

```
E. INICIAL     E. FINAL     “PALABRA”     p=PROBABILIDAD
```

Donde E. INICIAL es el estado en cuestión, E. FINAL es el estado destino, PALABRA es la palabra que activa la transición, y PROBABILIDAD es la probabilidad de que se produzca. En nuestro caso, hemos hecho que dicha probabilidad sea 1.0 / “numero de transiciones del estado”, resultando así en que las transiciones son equiprobables. En el apartado 4.2.2 veremos como ejemplo un modelo de lenguaje para uno de los menús del cliente de correo.

La versión de iATROS utilizada, además, requiere que tanto el estado inicial como el final sean únicos, y que una frase empiece o acabe con el silencio. Esto se verá representado en los autómatas, como veremos más adelante.

Adicionalmente, debemos tener en cuenta otros aspectos de iATROS, como sus módulos y sus parámetros de ejecución.

3.1.4 Ficheros de configuración

Existen tres ficheros principales que contienen parámetros de configuración de iATROS.

- Conf.feat: Este fichero contiene parámetros relacionados con la parte técnica del audio que utilizamos como entrada: canales, bits, frames, umbral de silencio, etc. Se utilizan para la adquisición y el preproceso de la señal, y en su mayor parte no han sido modificados.

Sí hemos modificado, sin embargo, los valores de "SecondsSilence" (los segundos de silencio que han de transcurrir para que tomemos la señal de silencio) y "SilenceThreshold" (el umbral de energía por debajo del cual se considera que se está escuchando silencio), para asegurar un funcionamiento correcto de iATROS en nuestro sistema y con nuestro micrófono.

- Fichero de gramáticas: El objetivo de este fichero es contener la lista de gramáticas (modelos del lenguaje) aceptadas por iATROS. La primera línea indica la cantidad de modelos que aceptamos, mientras que el resto del fichero está compuesto por una gramática por línea, indicando previamente su tipo. iATROS, antes de esperar que se pronuncie una nueva frase, buscará el índice (empezando por cero) del siguiente modelo de lenguaje que utilizará en el fichero /tmp/state. Este sería un fragmento de nuestro fichero de gramáticas:

```
90
FSM models/estadoEspera.aut
FSM models/libretaDirecciones.aut
FSM models/libretaDirecciones-MenuArchivo.aut
FSM models/libretaDirecciones-MenuArchivo-Nuevo.aut
FSM models/libretaDirecciones-MenuAyuda.aut
FSM models/libretaDirecciones-MenuEditar.aut
FSM models/libretaDirecciones-MenuHerramientas.aut
FSM models/libretaDirecciones-MenuVer.aut
FSM models/libretaDirecciones-MenuVer-BarrasDeHerramientas.aut
FSM models/libretaDirecciones-MenuVer-Disposicion.aut
{...}
```

- Config.cnf: Este es el fichero de configuración principal que hemos utilizado. Establece qué ficheros contienen sus diferentes modelos, así como diversos parámetros que modificaremos en el capítulo 6 para tratar de obtener un reconocimiento lo más fiable posible.

A continuación, mostramos una tabla con los diversos parámetros del fichero config.cnf relevantes a este proyecto:

Parámetro	Descripción
samples	Indica la ruta al fichero que indica qué ficheros .cc utilizar. Sólo es relevante para el reconocimiento offline.
hmm	Ruta al fichero del modelo acústico.
lexicon	Ruta al fichero del modelo léxico.
lexicon-type	Tipo de modelo léxico.
set-of-grammars	Solo utilizado con nuestro iATROS modificado: indica la ruta del fichero de gramáticas.
grammar	Solo utilizado con el iATROS original: indica la ruta del modelo del lenguaje.
grammar-type	Solo utilizado con el iATROS original (pues en el que admite múltiples modelos de lenguaje, este dato aparece en el fichero set-of-grammars): indica el tipo de gramática: FSM (Finite State Machine) o NGRAM (n-grama).
histogram-pruning	Utilizado como control durante la búsqueda. Si el tamaño del heap supera este valor, se elimina la hipótesis menos probable.
beam	Criterio de poda durante la búsqueda: las transiciones que superen la puntuación actual más este valor serán eliminadas. Cuanto más grande sea, realizaremos una búsqueda más amplia, por lo que obtendremos mayor precisión a costa de una menor velocidad.
grammar-scale-factor	Cuanto más grande sea, otorga mayor importancia al modelo del lenguaje con respecto al acústico.
word-insertion-penalty	Cuanto más alto, penaliza más la inclusión de palabras, por lo que favorece la decodificación de palabras largas.

3.2 Thunderbird

Desarrollado por la Fundación Mozilla, Mozilla Thunderbird[10] es un cliente de correo gratuito, multiplataforma y de código abierto, que, pese a haber detenido su desarrollo activo en 2012, sigue siendo considerado una opción válida.

Proporciona opciones como clasificación del correo basura, corrector ortográfico al redactar nuevos mensajes, soporte para IMAP y POP, o una interfaz basada en pestañas, entre otras.

Este programa, como sucede a menudo, posee diversos atajos de teclado para acelerar su utilización que, aunque suelen estar enfocados hacia los usuarios avanzados, nos permitirán controlarlo de forma externa. En la sección 4.1 (Diseño del conjunto de órdenes) veremos qué órdenes hemos implementado para las diversas pantallas del cliente.

3.3 XMacro

XMacro [2] es un paquete que contiene dos programas en C++:

- XMacrorec: Esta aplicación "graba" los eventos recibidos durante su ejecución, y muestra el código correspondiente. Se puede utilizar con el fin de grabar las pulsaciones de teclado recibidas durante su ejecución a un fichero, para así automatizar el proceso de definición de órdenes, o para conocer los nombres internos de teclas especiales, como Alt o Control.
- XMacroplay: Envía al servidor X los eventos definidos por su entrada.

Los eventos que utilizamos en nuestro sistema, que no son todos los admitidos por XMacro, son los siguientes:

Nombre	Descripción
KeyStrPress [ksname]	Se corresponde al presionado de una tecla, referenciada por su nombre.
KeyStrRelease [ksname]	Envía el evento de la liberación de la tecla *ksname*.
KeyStr [ksname]	Se corresponde a la pulsación (presionado+liberación) de la tecla referenciada por su nombre.
Delay [sec]	Esta orden introduce un retardo de los segundos indicados. Es utilizada al introducir secuencias de pulsaciones, para evitar enviar los eventos demasiado rápido y permitir que el programa responda correctamente.

Ilustraremos todos estos eventos con un ejemplo, el de abrir el menú archivo (Alt+A) y luego seleccionar la opción "Nuevo" (N). La orden de consola que enviaríamos para enviar esta secuencia de eventos sería la siguiente:

```
echo "KeyStrPress Alt_L
  KeyStr a
  KeyStrRelease Alt_L
  Delay 1
  KeyStr n" | xmacroplay :0.0
```

Como se puede observar, utilizamos “Alt_L” para referenciar la tecla Alt, puesto que corresponde al nombre que posee la tecla Alt izquierda. Podríamos haber utilizado de la misma forma “Alt_R”, pero nos decidimos por utilizar siempre la misma.

3.4 Otras utilidades

3.4.1 generarAutomata

“generarAutomata” es programa en Java, de desarrollo propio, cuya función es la de crear un autómata correspondiente al modelo del lenguaje definido en un fichero.

El fichero que recibe como entrada sigue el formato de una lista de las frases admitidas por dicho modelo, mientras, que, por su parte, el fichero de salida se adecúa al formato utilizado por iATROS y explicado en el apartado 3.1.3. También tiene la posibilidad de generar un modelo del lenguaje que no contemple explícitamente el silencio inicial y final, adecuado para la versión no modificada del iATROS que se puede utilizar para pruebas.

Este programa toma el fichero de entrada como una lista de cadenas, a partir de la cual empieza a generar un autómata mediante una función recursiva:

Para cada una de las cadenas recibidas por la función de clasificación se coge la primera palabra, denominada “s”, de esta frase, y se obtiene la lista de frases (de aquellas recibidas) que comienzan por dicha palabra, habiéndosela quitado. A continuación, para cada una de las cadenas obtenidas, contempla otros dos casos:

- Si es una cadena vacía, significa que debemos añadir una transición al estado final mediante la palabra s.
- Si no lo es, agregamos esa cadena a una nueva lista.

La nueva lista del segundo subcaso es utilizada para la llamada recursiva, siempre que tenga una longitud mayor que cero.

Al finalizar la función recursiva, el programa convierte el autómata que ha ido rellenando al formato del fichero del lenguaje.

3.4.2 Eutranscribe

Eutranscribe se trata de un script escrito en Perl destinado a “traducir” las palabras del idioma español al alfabeto fonético reconocido por iATROS. Algunas de las operaciones y transcripciones realizadas son las siguientes:

- Elimina acentos.
- Pasa todo a minúsculas.
- Funde cadenas de espacio blanco.
- Transforma “rr” en “@”.

- Suprime los signos de puntuación iniciales, y convierte los finales en pausas.
- “x” se transcribe como “ks” entre vocales, y como “s” en otro caso.
- “y” se transcribe como “i” al final de palabra o entre consonantes.
- “hi” como comienzo de palabra se transcribe como “y”.
- “c” se transcribe como “z” antes de “e” o “i”, y como “k” en cualquier otro caso que no sea “ch” (que se transcribe como “c”).
- “j” se transcribe como “x”, así como la “g” de “ge” y “gi”.
- Se cambia “gue” y “gui” por “ge” y “gi” salvo que exista diéresis, y posteriormente la g se transcribe siempre igual.
- La “ñ” se transcribe como “h”.
- Etc.

Nosotros utilizaremos eutranscribe para transcribir las palabras aisladas que componen las frases que introduciremos en el modelo del lenguaje de iATROS. Por ejemplo:

- “abajo” pasa a ser “*abaxo*”
- “añadir” pasa a ser “*ahadir*”
- “correo” pasa a ser “*ko@eo*”

Estas transcripciones las utilizamos para generar el fichero del léxico que iATROS utilizará.

3.4.3 Setup

Este script, creado para este proyecto, realiza las acciones necesarias para obtener un autómata del modelo de lenguaje del iATROS, añadirlo a grammars.txt, y añadir el léxico correspondiente a lexico.lx. Realiza estas tareas de forma automática:

1. Elimina tildes y mayúsculas del fichero, obtiene una lista de las palabras únicas que contiene y la añade a una lista global (que contenga el léxico de todos los modelos de lenguaje) de la que posteriormente obtiene su transcripción gracias a eutranscribe.
2. Asocia las palabras de dicha lista a su transcripción, utilizando eutranscribe, y a partir de este resultado, genera el fichero del modelo léxico para iATROS siguiendo el formato necesario.
3. Genera el autómata del modelo del lenguaje utilizando el programa “generarAutomata” ya explicado en el apartado 3.4.1, y modifica el fichero “grammars.txt” para añadirlo. También genera el autómata y el fichero de configuración que serían necesarios para la ejecución de dicho modelo de lenguaje mediante la versión original de iATROS.

3.4.4 lanzarXMacro

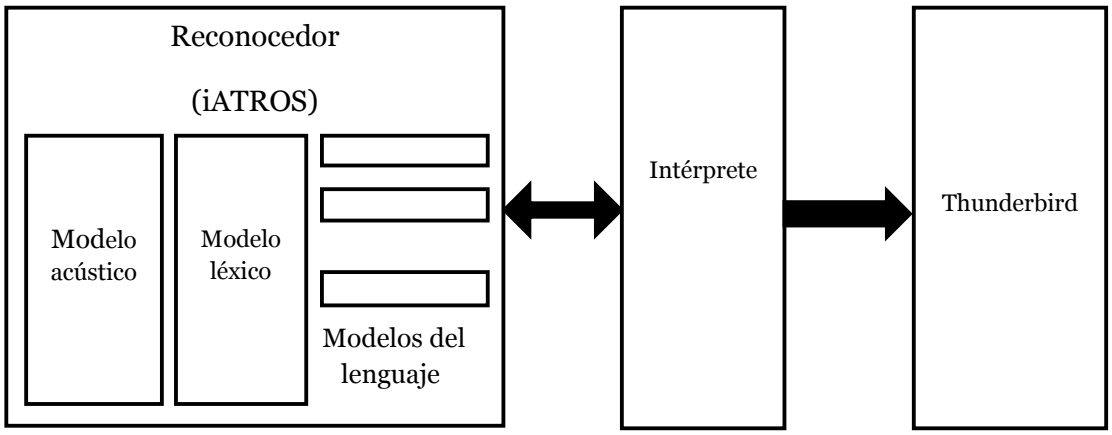
Se trata de un simple script que recibe una cadena como parámetro. Vuelca dicha cadena a un fichero temporal, para posteriormente utilizar ese fichero como entrada para XMacro.

4 Desarrollo

Llegados al momento de iniciar el desarrollo de nuestro sistema, podemos establecer tres partes diferenciadas:

1. Diseño del conjunto de órdenes que serán admitidas.
2. Puesta a punto y configuración del reconocedor para que acepte dichas órdenes.
3. Implementación del intérprete, que leerá la salida del reconocedor y lanzará los eventos de teclado correspondientes sobre el cliente de correo.

El sistema resultante, pues, debe permitir una comunicación bidireccional entre iATROS y el intérprete desarrollado por nosotros, así como posibilitar el envío de órdenes a Thunderbird por parte de nuestro intérprete. Este sería un esquema básico de dicho sistema:



Además, hemos grabado un vídeo demostrativo del funcionamiento del sistema. En él, abrimos la libreta de direcciones para enviar un email a uno de nuestros contactos, añadimos un segundo contacto como BCC, enviamos el correo (de forma retrasada) y cerramos el programa.

http://www.youtube.com/watch?v=r2Yzt0ohMwU

A continuación, describiremos el desarrollo de estas tres partes, junto con algunos aspectos adicionales.

4.1 Diseño del conjunto de órdenes

Para establecer el conjunto de órdenes que resultaría conveniente utilizar, decidimos que sería adecuado abrir el cliente de correo y observar qué acciones sería más probable lanzar desde el punto de vista del usuario, junto con los atajos de teclado que las ejecutan [11]. Para ello, además, dividimos las acciones en tres conjuntos, según la pantalla del programa desde la cual se lanzan: pantalla principal, ventana de la libreta de contactos y ventana de redacción de un nuevo mensaje.

Los menús (archivo, editar, ayuda, etc.) de cada una de estas ventanas también han sido tenidos en cuenta, pero no los comentaremos aquí debido a la cantidad de órdenes y submenús que contienen.

Órdenes admitidas en la pantalla principal:

Orden	Atajo de teclado correspondiente	Descripción
Abrir mensaje	Control+O	Abre el mensaje en una nueva pestaña.
Abrir mensaje en la conversación	Control+Shift+O	Abre el mensaje en una nueva pestaña, dentro del contexto de la conversación a la que pertenece.
Abrir menú Archivo/Editar/Etc. (o “Menú Archivo/Editar/Etc.”)	Alt + A/E/Etc.	Abre el menú correspondiente.
Anterior/Siguiente mensaje (o “Mensaje anterior/siguiente”)	Up/Down	Selecciona el mensaje siguiente/ anterior.
Anterior/Siguiente pestaña (o “pestaña anterior/siguiente”)	Control+Shift+Tab / Control+Tab	Navega entre pestañas.
Estrella	S	Marca o desmarca el mensaje con una estrella, para su clasificación.
Archivar	A	Quita el mensaje de la bandeja de entrada principal, sin eliminarlo.
Bajar/Subir	Av Pág/Re Pág	Avanza y retrocede dentro de un mensaje, para permitir su lectura.
Buscar	Control+K	Sitúa el foco en la caja de búsqueda.
Cambiar carpeta	Alt+I, A, A, Right	Muestra el menú de selección de carpeta (Menú Ir → Carpeta).
Cerrar pestaña	Control+W	Cierra la pestaña actual.
Cerrar programa/ Salir/Salir del programa	Alt+F4	Cierra Thunderbird.
Charlar	Alt+I, H	Abre la pestaña de chat (Menú Ir → Charlar)
Direcciones	Control+Shift+B	Abre la libreta de direcciones.

Eliminar/Eliminar mensaje	Supr	Elimina el mensaje seleccionado.
Etiqueta	Alt+M, J	Muestra el menú de selección de etiqueta correspondiente al mensaje seleccionado, para cambiarla (Menú Mensaje → Etiquetar).
Filtro rápido	Control+Shift+K	Lleva el cursor a la caja de búsqueda rápida.
Guardar archivos adjuntos	Alt+M, D	Abre el submenú correspondiente a los archivos adjuntos (Menú Mensaje → Adjuntos).
Guardar todos los adjuntos	Alt+M, D, G	Guarda todos los archivos adjuntos.
Legítimo	Shift + J	Marca el mensaje como deseado, o “legítimo”.
No deseado	J	Marca el mensaje como no deseado, o “spam”.
Recibir	F5	Recibe los mensajes correspondientes a la cuenta actual.
Recibir todos los mensajes nuevos	Shift+F5	Recibe los mensajes correspondientes a todas las cuentas.
Redactar	Control+N	Abre la ventana de redacción de un nuevo correo.
Reenviar	Control+L	Reenvía el mensaje seleccionado. Abre la ventana de redacción.
Responder	Control+R	Responde al remitente del mensaje seleccionado. Abre la ventana de redacción.
Responder a todos	Control+Shift+R	Responde tanto al remitente como al resto de destinatarios del mensaje seleccionado. Abre la ventana de redacción.

Órdenes admitidas en la ventana de la libreta de contactos:

Orden	Atajo de teclado correspondiente	Descripción
Abrir contacto	Return	Abre la ventana que contiene la información del contacto seleccionado.
Anterior/Siguiente contacto	Up/Down	Selecciona el contacto anterior/siguiente.
Anterior/Siguiente libreta	Shift+Tab, Up/Down, Tab	Cambia a la libreta de direcciones anterior/siguiente. Devuelve el foco a la lista de contactos.
Buscar	Control+K	Enfoca la caja de búsqueda.
Cerrar/Cerrar ventana	Control+W	Cierra la ventana, y vuelve a la pantalla principal.
Eliminar	Supr	Elimina el contacto seleccionado.
Nueva libreta/nueva libreta de direcciones	Alt+A, N, B	Crea una nueva libreta de direcciones (Menú Archivo → Nuevo → Libreta de direcciones).
Nueva lista	Alt+A, N, L	Crea una lista de correo (Menú Archivo → Nuevo → Lista de correo)
Nuevo contacto	Control+N	Abre la ventana de creación de un nuevo contacto.
Propiedades	Control+I	Abre las propiedades del contacto seleccionado.
Redactar	Control+M, Alt+Tab, Alt+F4	Abre la ventana de redacción de un correo, con el contacto seleccionado como destinatario. Cierra la ventana de la libreta de contactos.
Abrir menú Archivo/Editar/Etc. (o “menú Archivo/Editar/Etc.”)	Alt + A/E/Etc.	Abre el menú correspondiente.

Órdenes admitidas en la ventana de redacción:

Orden	Atajo de teclado correspondiente	Descripción
Abrir menú Archivo/Editar/Etc. (o "menú Archivo/Editar/Etc.")	Alt + A/E/Etc.	Abre el menú correspondiente.
Adjuntar	Alt+A, T, Right	Abre el menú para adjuntar mensajes (Menú Archivo → Adjuntar)
Adjuntar archivos	Shift+Control+A	Abre el cuadro de diálogo para adjuntar archivos.
Anterior/Siguiente campo	Shift+F6 / F6	Enfoca el campo anterior/siguiente.
Campo asunto	Alt+S	Enfoca el campo "asunto".
Campo de/remitente	Alt+D	Enfoca el campo "de".
Campo destinatario	Alt+D, Tab, Tab	Enfoca el campo "destinatario"
Campo mensaje	Alt+S, Tab	Enfoca el campo del mensaje.
Cerrar/Cerrar ventana	Alt+F4	Cierra la ventana de redacción y vuelve a la página principal.
Cifrar mensaje	Alt+P, E	Cifra el mensaje (Menú Opciones → Cifrar este mensaje)
Enviar/Enviar ahora/ Enviar mensaje	Control+Return	Envía ahora el mensaje.
Enviar más tarde	Control+ Shift+Return	Programa el envío de mensaje para más adelante, lo pone en la bandeja de salida.
Firmar digitalmente este mensaje	Alt+P, F, F, Return	Firma digitalmente el mensaje (Menú Opciones → Firmar digitalmente este mensaje).
Guardar/Guardar archivo	Control+S	Guarda el mensaje, en su estado actual, como archivo.

Guardar borrador	Alt+A, A, D	Guarda el mensaje, en su estado actual, como borrador (Menú Archivo → Guardar Como → Borrador).
Guardar plantilla	Alt+A, A, T	Guarda el mensaje, en su estado actual, como plantilla (Menú Archivo → Guardar Como → Plantilla).
Ortografía	Control+Shift+P	Revisa la ortografía del cuerpo del mensaje.
Seguridad/ Ver información de seguridad	Alt+V, S	Abre la ventana que muestra la información de seguridad (Menú Ver → Información de seguridad del mensaje).
Ver panel de contactos	F9, Alt+U, Tab	Abre el panel de contactos, y pasa al modelo de lenguaje correspondiente. Sitúa el foco de teclado en la lista de contactos.

Adicionalmente, la ventana de redacción tiene un panel de contactos, destinado a añadirlos como destinatarios. Dicho panel ha sido tratado como una ventana diferente para evitar problemas, puesto que altera el conjunto de órdenes posibles.

Se puede observar que los atajos de teclado que hemos asignado contienen más órdenes de lo habitual: esto se debe a que conviene mantener el foco de teclado sobre la lista de contactos, para poder navegar por ella cómodamente, y poder utilizar los botones de "añadir a ___". Esto implica que, tras ejecutar la orden adecuada, debemos volver a situar el foco en su lugar correspondiente.

Las órdenes que admite nuestro sistema para este panel son las siguientes:

Orden	**Atajo de teclado correspondiente**	**Descripción**
Anterior/Siguiente contacto	Up/Down	Selecciona el contacto anterior/siguiente.
Anterior/Siguiente libreta de direcciones	Alt+L, Up/Down, Tab, Tab	Cambia a la libreta anterior/siguiente y vuelve a enfocar la lista de contactos.

Añadir a “para”	Alt+P, Alt+U, Tab	Añade el contacto seleccionado como destinatario principal y vuelve a situar el foco en la lista de contactos.
Añadir a CC/“copia carbón”	Alt+C, Alt+U, Tab	Añade el contacto seleccionado como codestinatario y vuelve a situar el foco en la lista de contactos.
Añadir a BCC/“copia carbón oculta”	Alt+B, Alt+U, Tab	Añade el contacto seleccionado como codestinatario oculto y vuelve a situar el foco en la lista de contactos.
Buscar	Alt+U	Enfoca la caja de búsqueda.
Cerrar/Cerrar panel de contactos	Alt+D, F9	Sitúa el foco en el campo “de”, cierra el panel de contactos y vuelve al modelo de lenguaje de la redacción de mensaje.

También nos hemos encontrado con la necesidad de implementar unos “estados especiales” para tratar situaciones específicas:

- En ocasiones, se hace imposible seguir operando el cliente de correo únicamente mediante órdenes de voz (por ejemplo, cuando vamos a guardar un email). En estos momentos, el usuario deberá utilizar ratón y teclado, y, cuando haya acabado, ordenar “continuar” al reconocedor de voz. Es lo que hemos llamado “**estado Espera**”.

- Por otra parte, tenemos un subconjunto de las ocasiones en las que utilizamos el estado de espera. En dichas ocasiones, es factible utilizar órdenes sencillas de voz, como “arriba”, “abajo”, “aceptar”, etc. para manejar el programa, como sucede en algunos menús en los que no hay definidos los suficientes atajos de teclado (Menú Ir → Carpeta, por ejemplo). Es lo que hemos llamado “**modo manual**”.

Una vez tratadas estas situaciones especiales, podemos, junto con las órdenes definidas de forma tradicional, utilizar la mayor parte de funciones de Thunderbird mediante diversas órdenes de voz, reduciendo la necesidad de la utilización de la entrada tradicional (ratón y teclado) a los casos en los que es estrictamente necesaria.

4.2 Reconocedor

iATROS utiliza tres modelos, como ya hemos explicado anteriormente: el modelo del lenguaje, el modelo acústico y el modelo léxico. Puesto que ya disponemos del modelo acústico, que es independiente de las órdenes que queramos implementar, ya que está adaptado al idioma español, sólo es necesario implementar los otros dos.

4.2.1 Modelo léxico

El modelo léxico será global, es decir, admitirá las palabras contenidas en todos los modelos del lenguaje. Para ello, nos servimos del script "setup", explicado con anterioridad, que, entre otras cosas, va rellenando la lista de palabras admitidas, y, a partir de ella, genera el fichero lexico.lx.

Aquí mostramos unas pocas líneas pertenecientes a dicho fichero:

```
<s> 1.0 S
</s> 1.0 S
a 1.0 a
abajo 1.0 a b a x o
abreviatura 1.0 a b r e b i a t u r a
abrir 1.0 a b r i r
aceptar 1.0 a z e p t a r
acerca 1.0 a z e r k a
acercar 1.0 a z e r k a r
acronimo 1.0 a k r o n i m o
actividad 1.0 a k t i b i d a d
actual 1.0 a k t u a l
actualizar 1.0 a k t u a l i z a r
acuse 1.0 a k u s e
enriquecido 1.0 e n @ i k e z i d o
entrar 1.0 e n t r a r
entrega 1.0 e n t r e g a
enviados 1.0 e m b i a d o s
enviar 1.0 e m b i a r
errores 1.0 e @ o r e s
```

4.2.2 Modelo del lenguaje

Para definir las frases aceptadas por el reconocedor para uno de los estados del autómata del intérprete, debemos crear un modelo del lenguaje que las acepte, tomando las palabras como entrada para efectuar las transiciones. Dicho autómata, por razones técnicas, debe partir de un único estado inicial, que recibe como entrada "<s>" (correspondiente al silencio), y acabar en un estado final que reciba "</s>", correspondiente al silencio, para finalizar la detección.

Para este proceso utilizamos el script setup, explicado en el apartado 3.4.3., que obtiene dicho modelo llamando a generarAutomata, añade el léxico previamente no utilizado al modelo léxico, y finalmente modifica el fichero de gramáticas para incluir este último modelo del lenguaje.

Aquí mostramos el modelo del lenguaje para uno de los menús más sencillos, que soporta las frases “cancelar”, “correo y noticias”, “exportar” e “importar”.

```
State 0 i = 1
0 1 "<s>" p = 1.0

State 1
1 4 "cancelar" p = 0.2500
1 2 "correo" p = 0.2500
1 4 "exportar" p = 0.2500
1 4 "importar" p = 0.2500

State 2
2 3 "y" p = 1.0000

State 3
3 4 "noticias" p = 1.0000

State 4
4 5 "</s>" p = 1.0

State 5 f = 1
```

Volviendo a la propia aplicación de iATROS, hemos tenido que modificar el fichero online.c, que contiene el bucle de ejecución del reconocimiento *on-line*, para que el reconocedor vuelque su hipótesis más probable (la frase reconocida) en el fichero “salida_reconocedor”, que será leído por el intérprete.

4.3 Intérprete

El intérprete se trata de otra aplicación Java que consta, por una parte, del autómata que representa las pantallas y menús de Thunderbird, y por otra, de las frases que sirven para pasar de unos estados a otros (con las acciones correspondientes asociadas a las transiciones).

Dicho intérprete ejecuta dos acciones principales:

- Al iniciarse, crea el autómata correspondiente a las pantallas y los menús de Thunderbird, con sus transiciones, que representan las acciones que debemos enviar para operar con el cliente de correo.
- Tras crear el autómata, el intérprete pasa a un bucle infinito que sigue el siguiente esquema:

```
estado_actual = estado_inicial
iteración = 1
mientras(true){
    escribir iteracion y estado_actual en /tmp/state
    mientras no exista “salida_reconocedor”{
        ver si existe “salida_reconocedor”
    }
    Leer iteración y frase de “salida_reconocedor”
    Comprobar que la iteración concuerda.
        Si no -> vuelta a esperar salida
        Si concuerda -> continuar

    aplicar transición correspondiente (cambia estado actual)
    iteración = iteración + 1
}
```

Las acciones se crean con una función llamada “crearTeclas” que, a partir de una cadena de caracteres, crea la orden que ha de transmitirse a XMacro. Esta función ha sido creada para facilitar al máximo la escritura y modificación de las acciones. Aquí tenemos un ejemplo de su funcionamiento:

Cadena	**crearTeclas(cadena)**
"Alt+A, N”	new String[]{"lanzarXMacro", "KeyStrPress Alt_L KeyStr a KeyStrRelease Alt_L Delay 1 KeyStr n"}

El array de cadenas resultante se ejecutará mediante la función “Runtime.getRuntime().exec(orden)”. Por su parte, “lanzarXMacro” es un sencillo script, que, dado un parámetro, lo redirecciona a xmacroplay, cuyo funcionamiento ya hemos visto en la sección 3.3.

Estas acciones se lanzan como parte del proceso de transición de un estado a otro.

4.4 Comunicación entre los componentes

La comunicación entre el analizador iATROS y nuestro intérprete ha sido realizada mediante ficheros externos: un programa escribe el dato necesario, mientras que el otro está esperando a que aparezca el fichero con la información que va a recibir. Una vez leída dicha información, se elimina el fichero.

Esto se utiliza con dos objetivos: informar a iATROS del modelo de lenguaje que debe cargar, y pasar al intérprete la frase reconocida por iATROS.

Sin embargo, esta implementación mostraba algún problema, puesto que los programas, en ocasiones, tomaban como recibido un dato recibido con anterioridad en lugar del nuevo dato. Para solucionarlo hicimos que, junto al dato que se necesita transmitir de un programa a otro, apareciera el índice de la iteración actual (empezando en 1). De esta forma, los programas pueden distinguir cuándo han tomado un dato incorrecto y así continuar la espera del dato real.

Esta solución, además, nos ha servido para crear la posibilidad de ordenar a iATROS que finalice su ejecución cuando le indicamos que estamos en la “iteración 0”. Así, cuando el intérprete llegue a un estado en el que sabe que el cliente de correo ha finalizado, enviará, de esta forma, dicha orden al reconocedor, y posteriormente finalizará él mismo. Esto dará como resultado la finalización de todos los componentes del sistema.

5 Experimentación

Para finalizar, detallaremos la experimentación seguida para obtener una configuración de iATROS que minimice los errores. El proceso seguido consta de estas partes:

- **Diseño**: decidimos cómo hacer las pruebas y qué parámetros cambiar entre qué valores.
- **Experimentación**: tomaremos los valores de la tasa de error obtenidos mediante la ejecución de iATROS.
- **Resultados**: Analizaremos los resultados de la experimentación, y nos decantaremos por una configuración óptima.

5.1 Diseño

Ya que queremos minimizar la aleatoriedad de las pruebas, utilizaremos para ellas la función de reconocimiento *off-line* de iATROS. De esta forma, la entrada será la misma para todas las pruebas que hagamos, por lo que podremos valorar mejor los efectos de cambiar los parámetros.

Dichos parámetros son los explicados en el apartado 3.1.4, y detallaremos a continuación los valores que hemos probado:

Parámetro	Rango de valores
histogram-pruning	5000
word-insertion-penalty	{-10, -5, 0, 5, 10}
grammar-scale-factor	{5, 10, 15}
Beam	{200, 500, 1000}

En un principio habíamos planeado modificar también "histogram-pruning", pero observamos que no tenía ningún efecto. Esto se debe a que nuestros modelos de lenguaje utilizado son sencillos, y no llegaríamos a tener que efectuar dicha poda ("pruning").

A continuación decidimos centrar los experimentos en el reconocimiento de las frases admitidas en la página principal de Thunderbird, nuestro modelo de lenguaje más grande. Se graba a tres locutores pronunciando todas las órdenes, y posteriormente se comprueba cuál es el error cometido al intentar reconocerlas todas.

Para ello, se ha creado un sencillo script que lee todas las líneas del fichero que contiene todas las frases de la página principal (48), y solicita al usuario que las grabe una por una en formato .wav. A continuación, se utiliza iATROS para convertir estos ficheros de audio en vectores de coeficientes cepstrales, y finalmente se utiliza el modo *off-line* para obtener las frases reconocidas.

Los locutores antes mencionados son los siguientes:

- Hombre de 22 años.
- Mujer de 57 años.
- Hombre de 62 años.

5.2 Experimentación

A continuación mostraremos una tabla con los porcentajes de error obtenidos durante la experimentación:

			Word-Insertion-Penalty				
		GSF	-10	-5	0	5	10
Beam	200	5	**4.166667**	**4.166667**	**4.166667**	**4.166667**	**4.166667**
		10	**4.166667**	**4.166667**	**4.166667**	**4.166667**	**4.166667**
		15	**4.166667**	**4.166667**	**4.166667**	**4.166667**	**4.166667**
	500	5	4.861111	4.861111	4.166667	4.861111	4.861111
		10	4.861111	4.861111	4.166667	4.861111	4.861111
		15	4.861111	4.861111	4.166667	4.861111	4.861111
	1000	5	4.861111	4.861111	4.166667	4.861111	4.861111
		10	4.861111	4.861111	4.166667	4.861111	4.861111
		15	4.861111	4.861111	4.166667	4.861111	4.861111

Como podemos observar, la variabilidad del error es muy pequeña. Esto es principalmente provocado por el pequeño tamaño del modelo de lenguaje, así como el del corpus de muestra.

Viendo que el único factor que ha afectado el resultado es el beam, pasamos a efectuar unas pocas pruebas más, esta vez con valores cercanos a 200. Utilizaremos los siguientes valores:

- Histogram-pruning = 10000
- Grammar-Scale-Factor = 10
- Word-Insertion-Penalty = -10

	Beam				
	50	100	150	200	300
Error (%)	13.888889	4.861111	4.166667	4.166667	4.861111

5.3 Resultados

Tras efectuar todas estas pruebas, podemos concluir que, en este modelo, estos parámetros afectan poco a su correcto funcionamiento, principalmente debido a su sencillez. El porcentaje de aciertos conseguido ha sido bastante alto, entorno al 95.83%, y la variabilidad del error no ha superado el 1%, excepto cuando hemos disminuido demasiado el parámetro beam.

Sin embargo, sí que debemos destacar dos aspectos:

- El factor beam, en este sistema, tiene un efecto contrario al que esperábamos inicialmente. Por una parte, puede deberse a la escasa complejidad del lenguaje, y por otra, es posible que sea debido a que, puesto que nuestro modelo presenta transiciones equiprobables, una poda más permisiva provocaría mayor confusión entre hipótesis.
- Aunque no lo hemos representado en estas tablas, la mayoría de confusiones se centran en un subconjunto de frases. Por ejemplo, “menú ver” y “menú ir”. Este problema es comprensible, puesto a que los sonidos emitidos por una persona pronunciando estas frases son considerablemente similares.

Concluyendo, para el correcto funcionamiento de este sistema utilizaríamos los siguientes parámetros:

Grammar-Scale-Factor	Indiferente (utilizaremos 10)
Word-Insertion-Penalty	Indiferente (utilizaremos -10)
Histogram-Pruning	Indiferente (utilizaremos 10000)
Beam	200

Hemos escogido “200” como valor adecuado para el factor Beam, puesto que se trata de un buen punto medio, y no querríamos bajarlo mucho más por si ampliáramos el modelo del lenguaje y pasara a perjudicarnos haberlo disminuido tanto.

6 Conclusiones

El desarrollo de este proyecto nos ha servido para afianzar conocimientos sobre el reconocimiento de formas (concretamente, el reconocimiento automático del habla), así como sobre el desarrollo de aplicaciones que han de sincronizarse entre sí. La toma de contacto con el reconocedor iATROS ha sido satisfactoria, debido a la necesidad de tratar con modelos léxicos y de lenguaje.

El resultado final es el de un cliente de correo con un control por voz poco fluido, pero con muchas de sus funciones al alcance de unas pocas frases habladas. Algunas de las posibilidades de mejora habrían sido analizar el título de los mensajes más recientes para su más fácil acceso, o hacer lo mismo con los contactos. Sin embargo, las opciones que teníamos a nuestro alcance para actuar sobre Thunderbird se limitaban a los atajos de teclado, por lo que no resultaba factible.

Por tanto, conseguir una mayor integración con el cliente de correo sería quizás la principal vía de mejora en trabajos futuros. Para ello se podría analizar y modificar el código fuente de Thunderbird, de forma que existan nuevas formas de interactuar con una aplicación externa, o incluso, en situaciones específicas con los recursos y requisitos adecuados, desarrollar un cliente completamente nuevo (opción desechada aquí puesto que disponíamos de un cliente muy competente).

La otra principal mejora posible sería la introducción de texto. Sin embargo, el dictado de texto dista de la teoría utilizada, y merecería su estudio dedicado.

Por su parte, el ámbito del reconocimiento del habla está en auge recientemente, dadas sus posibilidades en dispositivos móviles. Estos avances, sin embargo, pertenecen a la rama del reconocimiento de lenguaje natural, dependiente de lo que hemos estudiado aquí, pero con objetivos que van más allá.

Así pues, nos gustaría concluir que el resultado conseguido, pese a los diversos aspectos en los que se podría profundizar más para mejorar, es satisfactorio, y presenta una forma accesible a personas con dificultad para utilizar periféricos (como los ratones) de manejar la mayoría de funciones cotidianas de un cliente de correo electrónico.

Bibliografía

[1] Página de iATROS: https://prhlt.iti.upv.es/page/projects/multimodal/idoc/iatros

[2] Página de XMacro: http://xmacro.sourceforge.net

[3] F. Casacuberta, R. García, J. Llisterri, C. Nadeu, J.M. Pardo, A. A. Rubio. “Desarrollo de corpus para investigación en tecnologías del habla (Albayzin)”, Procesamiento del Lenguaje Natural. 12. 35-42. 1992.

[4] Christopher D. Manning, Hinrich Schütze, Foundations of Statistical Natural Language Processing, MIT Press: 1999.

[5] The HTK Book: http://htk.eng.cam.ac.uk/docs/docs.shtml

[6] Luján-Mares, Mıriam, et al. "iATROS: A speech and handwritting recognition system." V Jornadas en Tecnologıas del Habla (VJTH’2008) (2008): 75-78.

[7] Forney Jr, G. David. "The viterbi algorithm." Proceedings of the IEEE 61.3 (1973): 268-278.

[8] Wikipedia: Hidden Markov models http://en.wikipedia.org/wiki/Hidden_Markov_model

[9] Wikipedia: Speech recognition http://en.wikipedia.org/wiki/Speech_recognition

[10] Página de Mozilla Thunderbird http://www.mozilla.org/es-ES/thunderbird/

[11] Ayuda de Thunderbird: Atajos de teclado https://support.mozillamessaging.com/es/kb/accesos-directos-de-teclado

[12] Página web del PRHLT: https://prhlt.iti.upv.es