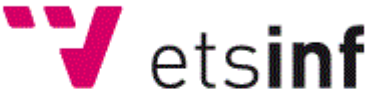
Proyecto fin de carrera

Navegación web usando la voz

Titulación: Ingeniería Informática
Autor: Marc Franco Salvador
Director: Carlos David Martínez Hinarejos



Escuela Técnica Superior de Ingeniería **Informática**

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Septiembre 2011

Agradecimientos

En primer lugar quiero agradecer a mi familia la paciencia y el apoyo que me han dado durante toda la realización del proyecto. A mis amigos que han tenido que soportar mis quejas y frustraciones en los peores momentos. A todos aquellos que me prestasteis vuestra voz para crear el corpus con el que realizar las pruebas.
Una mención especial a Carlos, mi director, sin el que este proyecto no hubiera sido posible. Gracias por contestar mis correos con tanta celeridad, por reunirse conmigo para revisar mi trabajo, y por darme ánimos en tantas ocasiones.
Finalmente a mi novia Diana, por su paciencia en cada momento, que ha sido la que más ha tenido que soportarme, y la que me ha animado a seguir adelante durante todos estos años.

Índice General

1. Introducción **7**
1.1. Reconocimiento del habla 7
1.2. Navegación web 8
1.3. Objetivos 9
2. Bases teóricas **10**
2.1. Reconocimiento del habla 10
2.1.1. Modelos ocultos de Markov 10
2.1.2. Modelos del sistema 12
2.1.3. Algoritmo de Viterbi 14
2.2. Tecnología web 15
2.2.1. El lenguaje HTML 15
2.2.2. Navegación web y el navegador 19
3. Aplicaciones **20**
3.1 iATROS 20
3.1.1. Módulos del sistema 20
3.1.2. Fichero de configuración del sistema 21
3.1.3. Fichero de configuración de la señal de audio 23
3.1.4. Ficheros de los modelos 24
3.2 Mozilla Firefox 29
3.2.1. Teclas de acceso rápido 29
3.2.2. Almacenamiento de la sesión 30
3.3 Xmacro 32
3.4 eutranscribe 33
4. Desarrollo de la aplicación **35**
4.1. Conjunto de órdenes 36
4.2. Diseño del reconocedor 38
4.2.1. Creación de los modelos del sistema 38
4.2.2. Ficheros de configuración 41
4.2.3. Ejecución 41
4.3. Diseño del intérprete 43
4.3.1. Diseño de órdenes normales 43
4.3.2. Agregar texto de los enlaces a los modelos 44
4.3.3. Diseño de otras órdenes especiales 47

4.4. Comunicación entre módulos 48
4.5. Otros módulos 49
5. Experimentación **51**
5.1. Diseño del experimento 51
5.2. El corpus de test 53
5.3. Experimentación 53
5.3.1. Primera fase 54
5.3.2. Segunda fase 56
5.3.3. Tercera fase 57
5.4. Análisis de los resultados 58
6. Conclusiones y trabajos futuros **60**
Bibliografía **62**

Capítulo 1

Introducción

El presente proyecto hace uso de la tecnología de reconocimiento del habla combinada con la navegación web para finalmente conseguir la navegación mediante el uso de la voz. En este capítulo introduciremos los aspectos generales del reconocimiento del habla y de la navegación web.

1.1. Reconocimiento del habla

El habla es el uso particular e individual que hace una persona de una lengua. Es característica única de la especie humana, y nuestra forma más natural de comunicación.
Por otro lado, la comunicación entre el hombre y la máquina se realiza mediante dispositivos tales como teclados, ratones, o pantallas táctiles. Dichos métodos, aunque puedan parecernos cómodos, requieren un aprendizaje previo para familiarizarse con el uso de todos estos dispositivos. Un aprendizaje que puede llegar a ser muy complicado dependiendo de la persona y la interfaz. Además, hay que tener en cuenta que son interfaces de comunicación no aptas para todos; ya sea por minusvalía u otros posibles problemas de la persona en cuestión, como pueda ser por ejemplo la edad. De ahí la necesidad de buscar otras formas de comunicación más naturales y accesibles, como pueda ser el habla.

Uno de los objetivos de la Inteligencia Artificial es permitir la comunicación hablada entre seres humanos y sistemas computacionales. Para ello existen dos direcciones:

- Comunicación desde el ser humano hacia la máquina:
    - **RAH: Reconocimiento Automático del Habla**. Genera la secuencia de palabras del lenguaje a partir de una señal vocal.
    - **PLN: Procesamiento del Lenguaje Natural**. Va más allá que el anterior, pues no trata de la comunicación por medio de lenguajes de una forma abstracta. A partir de una frase transcrita, trata de obtener su significado, verificando su corrección léxica, sintáctica y semántica.
- Comunicación desde la máquina hacia el ser humano:
    - **Síntesis del habla:** Creación de una voz artificial (no pregrabada), generada mediante un proceso de síntesis del habla a partir de un texto.

Para el reconocimiento de la lengua hablada, hay dos lineas de investigación diferentes que se están llevando a cabo:

- **Sistemas basados en el conocimiento** (Knowledge-based): El reconocimiento se lleva a cabo mediante reglas acústico-fonéticas que se basan en características de la forma de la onda de entrada. Dichas reglas son descritas por un experto humano.
- **Sistemas basados en la estadística o en los datos** (Data-based): No es necesario un experto humano que defina las reglas. Obtiene el "conocimiento" a partir de características que se extraen de la onda acústica mediante un análisis estadístico. Es necesario tener un corpus suficientemente grande para que pueda servir como referente el "conocimiento" extraído.

Por norma general, el primer método no obtiene buenos resultados, por la dificultad de los seres humanos para definir su conocimiento y expresarlo en forma de reglas en una máquina. El segundo método obtiene mejores resultados, en gran medida por la ausencia de componente humano en el proceso de extracción del conocimiento, fácilmente extraíble a partir de algoritmos estadísticos.

En el presente proyecto se ha hecho uso del reconocimiento automático del habla mediante técnicas estadísticas. Llegados a este punto es necesario definir los tres modelos que servirán de base para nuestro reconocedor del lenguaje: el modelo léxico, el modelo del lenguaje, y el modelo acústico.

- **Modelo léxico**: El modelo léxico incluye la pronunciación de cada una de las palabras que forman parte del lenguaje a reconocer. Normalmente las palabras son representadas mediante autómatas finitos deterministas (AFD) cuyas transiciones entre estados son los símbolos que representan los sonidos que forman las palabras, presentes en el modelo acústico.

- **Modelo del lenguaje:** El modelo está formado por todas las frases posibles que componen el lenguaje de nuestro reconocedor. Generalmente se representa mediante un AFD cuyas transiciones son las palabras que forman las frases, descritas en el modelo léxico. Además de los AFD, el modelo más popular para tareas de mayor envergadura son las n-gramas.

- **Modelo acústico**: El modelo acústico está formado por todos los sonidos posibles que pueden pronunciarse en nuestro lenguaje. Los sonidos son representados mediante modelos ocultos de Markov, y para cada sonido hay un símbolo que lo representa, que se corresponde con los símbolos del modelo léxico.

Queda claro que habrá que definir un lenguaje concreto para que nuestro sistema sea capaz de reconocerlo. En la sección 2.1 dedicada al reconocimiento del habla, hablaremos más detalladamente sobre los tres modelos anteriores y su implementación.

1.2. Navegación web

La navegación web está formada por el conjunto de técnicas, aplicaciones y tecnologías que utilizamos para acceder a páginas web en la red. A continuación definiremos los componentes básicos que forman la navegación: la conexión a internet, el navegador y las páginas web.

1.2.1. NAVEGACIÓN WEB

- **Conexión a Internet:** La conexión a Internet es el mecanismo de enlace con que una computadora o red de computadoras cuenta para conectarse a Internet. Se hace uso de la red telefónica, y es necesario contratarla mediante un proveedor de servicios de internet. De él depende la velocidad a la que descarguemos los archivos de Internet y, por tanto, la velocidad de navegación.

- **Navegador:** Un navegador es una aplicación que opera a través de Internet, interpretando la información de archivos y sitios web para que podamos leerla en nuestras pantallas. Es el encargado de interpretar el código de las páginas web y presentarlo en pantalla, permitiendo al usuario interactuar con su contenido y navegar hacia otros lugares de la red mediante enlaces e hipervínculos.
  El navegador dispone de herramientas para facilitar la navegación. Algunas de estas herramientas son: teclas de acceso rápido desde el teclado, funciones de navegación mediante ratón, o opciones presentes en la interfaz tales como “ver código fuente”, “página web anterior”, etc.

- **Páginas web:** Una página web es un documento adaptado para el *World Wide Web* que generalmente forma parte de un sitio web. Las páginas, una vez interpretadas por el navegador, pueden estar formadas por texto, imágenes y contenido multimedia.
  Las páginas web están desarrolladas en un lenguaje de marcado que provee la capacidad de manejar e insertar enlaces. Algunos lenguajes de codificación o programación de páginas web son HTML, PHP y Java.

En la sección 2.2. hablaremos más a fondo de los elementos anteriores.

1.3. Objetivos del proyecto

El objetivo principal del proyecto es realizar un sistema de navegación web controlado mediante la voz en una distribución Linux. El sistema deberá ser capaz de reconocer órdenes sencillas de navegación tales como “página anterior”, además de poder analizar el código fuente de la página web para añadir el texto de los enlaces al reconocedor, y así poder ejecutar ordenes como “abrir enlace noticias deportivas”. Para ello serán necesarios los siguientes pasos:

- Estudiar las bases teóricas del reconocimiento del habla: modelos ocultos de Markov, modelos del sistema y algoritmo de Viterbi.

- Estudiar la bases teóricas de la tecnología web: HTML, navegación web y el navegador.

- Implementar el reconocedor automático del habla a partir de las herramientas de iATROS (sección 3.1), y diseñar el modelo de lenguaje y léxico adecuados para la tarea.

- Crear las aplicaciones que servirán de puente entre el reconocedor y el navegador, para que el reconocimiento sea invisible a ojos del usuario.

- Obtener los parámetros de configuración óptimos de la aplicación.

Capítulo 2

Bases teóricas

En este apartado trataremos las bases teóricas necesarias para el desarrollo del proyecto. En primer lugar se tratará el reconocimiento del habla, profundizando en sus diferentes aspectos. A continuación hablaremos sobre la tecnología web y algunos de sus aspectos más destacados que influyen en el proyecto.

2.1. Reconocimiento del habla

Si hablamos del Reconocimiento Automático del Habla, hay tres aspectos fundamentales que no podemos ignorar. Dichos aspectos son los que comentaremos a continuación:

- **Modelos ocultos de Markov (HMM)**: Es un modelo estadístico basado en estados que sirve para predecir los parámetros ocultos a partir de los observables. En nuestro caso para, dado un sonido, determinar con qué palabra se corresponde.
- **Modelos del sistema**: Los modelos del sistema ya han sido presentados anteriormente: modelo léxico, modelo del lenguaje y modelo acústico. Ahora hablaremos más detenidamente sobre ellos.
- **Algoritmo de Viterbi**: El algoritmo de Viterbi permite encontrar la secuencia de estados más probable en un HMM a partir de una observación. Obtiene la secuencia óptima que mejor explica la secuencia de observaciones, lo cual nos servirá para, dado un conjunto de sonidos, predecir con qué palabra (o secuencia de palabras) se corresponden.

2.1.1. Modelos ocultos de Markov

Con el objetivo de representar el modelo acústico del que se compone cada fonema utilizaremos los modelos ocultos de Markov (HMM de sus siglas en inglés).

Para poder hablar de los HMM es necesario conocer primero los procesos de Markov. Un proceso de Markov es un proceso estocástico para el que se cumple la propiedad de Markov. Dicha propiedad dice que, estando en un estado *n*, la probabilidad de pasar a un estado *n+1* no depende de los estados por los que hayamos transitado anteriormente, sino sólo del actual. De ahí que se diga que un proceso de Markov es un proceso estocástico sin memoria.

Un HMM es un modelo estadístico en el que se asume que el sistema a modelar es un proceso de Markov de parámetros desconocidos. El objetivo es determinar los parámetros desconocidos (u *ocultos*, de ahí el nombre) de dicha cadena a partir de los parámetros observables. Los parámetros extraídos se pueden emplear para llevar a cabo sucesivos análisis.

En un modelo de Markov normal, el estado es visible directamente para el observador, por lo que las probabilidades de transición entre estados son los únicos parámetros. En un HMM, el estado no es visible directamente, sino que sólo lo son las variables influidas por el estado. Cada estado tiene una distribución de probabilidad sobre los posibles símbolos de salida. Consecuentemente, la secuencia de símbolos generada por un HMM proporciona cierta información acerca de la secuencia de estados.

Los HMM son ideales para modelar procesos que tienen cambios de estado a lo largo del tiempo. En nuestro caso, para una aplicación de reconocimiento del habla, podemos considerar una palabra como cambios de fonemas a lo largo del tiempo.

Una notación formal habitual de un HMM es la representación como una tupla (Q,V,π,A,B):

- **Q**: El conjunto de estados $Q = \{1, 2, \ldots, N\}$.
- **V**: El conjunto de posibles valores $\{v_1, v_2, \ldots, v_M\}$ observables en cada estado.
- **π** : Las probabilidades iniciales π = {π_i}, donde π_i es la probabilidad de que el primer estado sea el estado Q_i.
- **A**: El conjunto de probabilidades $A = \{a_{ij}\}$ de transiciones entre estados.
  - $a_{ij} = P(q_t = j \mid q_{t-1} = i)$, es decir, a_{ij} es la probabilidad de estar en el estado j en el instante t si en el instante anterior $t - 1$ se estaba en el estado i.
- **B**: El conjunto de probabilidades $B = \{b_j(v_k)\}$ de las observaciones.
  - $b_j(v_k) = P(o_t = v_k \mid q_t = j)$, es decir, la probabilidad de observar v_k cuando se está en el estado j en el instante t.

La secuencia de observaciones se denota como un conjunto $O = (o_1, o_2, \ldots, o_T)$.

A continuación vemos un ejemplo de un HMM en la figura 2.1.1.1.

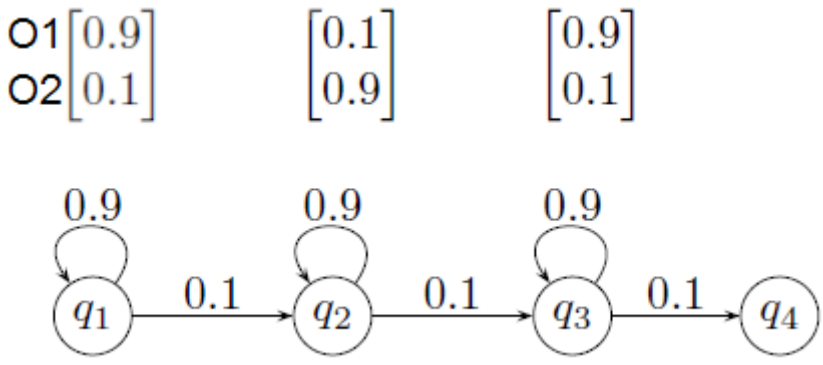


Figura 2.1.1.1: Ejemplo de un HMM con cuatro estados y dos posibles símbolos de emisión.

2.1.2. Modelos del sistema

Como ya hemos comentado anteriormente, para diseñar un reconocedor automático del habla es necesario disponer de tres modelos que definan el lenguaje que el sistema es capaz de comprender. Los tres modelos son: el modelo léxico, el modelo del lenguaje y el modelo acústico. En este apartado hablaremos más detenidamente sobre ellos.

Modelo léxico

El modelo léxico es el que define la pronunciación de cada una de las palabras que forman parte del lenguaje a reconocer. Cada palabra se representa mediante un autómata finito determinista independiente, de modo que dispondremos de un fichero con tantos autómatas como palabras tenga nuestro lenguaje. En dichos autómatas las transiciones entre estados son los símbolos que representan los sonidos (generalmente fonemas) que unidos forman las palabras. Cada uno de estos símbolos está presente en el modelo acústico. En la figura 2.1.2.1. tenemos un ejemplo de la representación de la palabra “abrir”.

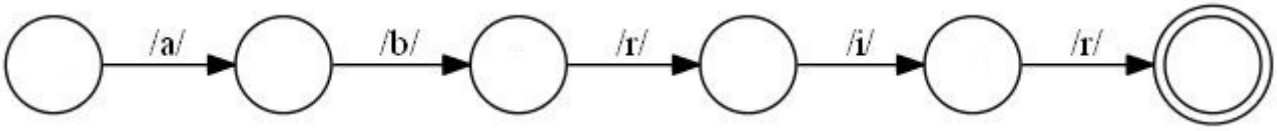


Figura 2.1.2.1: Ejemplo de un AFD para la palabra”abrir”.

Además podemos ver cómo quedaría representado en un fichero, utilizando el formato adecuado:

```
Name "abrir"
State 0 i=1
0 1 "a" p=1
1 2 "b" p=1
2 3 "r" p=1
3 4 "i" p=1
4 5 "r" p=1
State 5 f=1
```

El formato de representación de los tres modelos lo comentaremos en el apartado 3.1.4, dedicado a los ficheros de los modelos de la aplicación iATROS (sección 3.1).

Modelo del lenguaje

El modelo del lenguaje es el que define las frases que forman parte de nuestro lenguaje. Cada una de las palabras utilizadas en las frases tendrán que estar definidas en el modelo léxico. Para la representación de las frases se utiliza un AFD con un estado inicial y un único estado final, de modo que en el mismo autómata están todas las frases posibles del lenguaje. Las transiciones entre estados serán las palabras a reconocer, que además dispondrán de una probabilidad de emisión asociada. En la figura 2.1.2.2. tenemos un ejemplo para la representación de las frases “nueva ventana” y “nueva pestaña”.

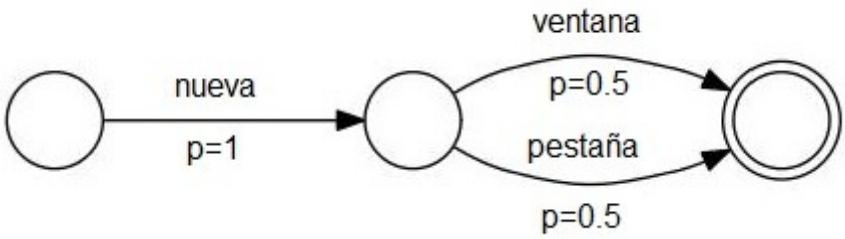


Figura 2.1.2.2: Ejemplo de la representación de las frases “nueva ventana” y nueva pestaña”.

El fichero con el modelo anterior sería el siguiente:

```
State 0 c = 1 i = 1
  0     1    "nueva" p = 1

State 1 c = 1
  1     2    "ventana"          p = 0.5
  1     2    "pestaña"          p = 0.5

State 2 c = 1 f = 1
```

Modelo acústico

El modelo acústico comprende todos los sonidos que pueden pronunciarse en nuestro lenguaje, almacenando las características de cada una de la secuencias acústicas permitidas. Cada uno de los sonidos está representado mediante un modelo oculto de Markov continuo, donde las salidas están modeladas por una distribución probabilística. Esta distribución suele ser una mixtura de distribuciones normales (gausianas), cada una con un peso asociado. Por ejemplo, en el modelo acústico empleado para el proyecto, tenemos 64 modelos para cada secuencia.

Cada secuencia acústica tendrá un identificador que la represente en nuestro modelo. Por ejemplo el fonema /a/ estará representado por “a”. Dichos identificadores serán utilizados en las transiciones entre estados del modelo léxico para representar los sonidos que forman las palabras. Hay que tener en cuenta que la transcripción fonética no se deriva directamente de la grafía de la palabra, sino que hay que interpretar esta fonéticamente. Así, la palabra “casa” se interpretaría como “kasa” en nuestro reconocedor. Hay otros ejemplos similares que serán comentados en la sección 3.4, dedicada a la aplicación *eutranscribe*, que sirve justamente para obtener su transcripción para nuestro modelo.

Podemos observar cómo sería uno de los modelos que representan el fonema /a/:

```
<MIXTURE> 1 3.474252e-03
<MEAN> 39
 1.785863e+04 1.299498e+03 -6.432632e+02 1.513383e+03 -1.147531e+02 1.304942e+03 5.227284e+00
1.238573e+03 -1.836611e+01 1.322657e+03 2.116885e+02 1.140614e+03 -1.483093e+02 4.423981e+02
6.415623e+02 1.492168e+02 3.146061e+01 1.392591e+02 2.472162e+01 1.126314e+02 5.481182e+01 8.754484e+01
5.021929e+01 1.099550e+02 7.642397e+01 7.027842e+01 1.855742e+02 -5.750425e+01 1.617091e+01 1.649160e+01
-3.283789e+01 1.179788e+01 -2.388708e+01 3.539843e+00 -2.320968e+01 2.741090e+00 -3.192752e+01
3.130974e+00 2.895082e-01
```

<VARIANCE> 39
5.879569e+05 1.443083e+05 5.697394e+04 5.408928e+04 3.555932e+04 3.183869e+04 4.928634e+04 1.966820e+04
2.380357e+04 1.743093e+04 2.286620e+04 2.281169e+04 1.893668e+04 6.287507e+04 2.368605e+04 8.469077e+03
8.191718e+03 4.942003e+03 5.167835e+03 3.632337e+03 3.156643e+03 3.855771e+03 3.091294e+03 3.360326e+03
2.176172e+03 2.106594e+03 9.947387e+03 3.104086e+03 1.455257e+03 1.118613e+03 7.147718e+02 4.974401e+02
5.415772e+02 4.094380e+02 3.893118e+02 3.348078e+02 3.334549e+02 3.642191e+02 3.403601e+02
<GCONST> 4.081876e+02

2.1.3. Algoritmo de Viterbi

Un problema de los modelos ocultos de Markov es que los estados que generan la secuencia de observaciones O queda oculta. Sin embargo para el RAH es muy importante conocer exactamente la secuencia de estados S que generan la secuencia O, ya que de este modo podremos determinar ante qué palabras y frases, definidas en los modelos léxicos y de lenguaje respectivamente, nos encontramos. Para ello utilizaremos el algoritmo de Viterbi.

El algoritmo de Viterbi permite encontrar la secuencia de estados $S = (q_1, q_2, \ldots, q_T)$ más probable en un HMM, a partir de una observación $O = (o_1, o_2, \ldots, o_T)$, es decir, obtiene la secuencia óptima que mejor explica la secuencia de observaciones. Formalmente queda del siguiente modo:

Consideremos la variable $\delta_t(i)$ que se define como:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1, q_2, \ldots, q_t = i, o_1, o_2, \ldots, o_t | \mu)$$

$\delta_t(i)$ es la probabilidad del mejor camino hasta el estado *i* habiendo visto las *t* primeras observaciones. Esta función se calcula para todos los estados e instantes de tiempo:

$$\delta_{t+1}(i) = \left[\max_{1 \leq i \leq N} \delta_t(a_{ij}) \right] b_j(o_{t+1})$$

Puesto que el objetivo es obtener las secuencia de estados más probable, será necesario almacenar el argumento que hace máxima la ecuación anterior en cada instante de tiempo *t* y para cada estado *j* y para ello utilizamos la variable $\varphi_t(j)$.

El algoritmo completo es el siguiente:

- 1. Inicialización:

  $\delta_1(i) = \pi_i b_i(o_1)$

  donde $1 \leq i \leq N$

- 2. Recursión:

  $$\delta_{t+1}(i) = \left[\max_{1 \leq i \leq N} \delta_i(a_{ij}) \right] b_j(o_{t+1}),$$

  donde:

  $t = 1, 2, \ldots, T-1, 1 \leq j \leq N$

$$\varphi_{t+1}(j) = \arg\max_{1 \leq i \leq N} \delta_t(i) a_{ij},$$

donde:

$$t = 1, 2, \ldots, T-1, 1 \leq j \leq N$$

- 3. Terminación. Obtenemos el estado final óptimo:

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

- 4. Reconstrucción de la secuencia de estados más probable. Para ello deshacemos el camino a partir del último estado:

$$q_t^* = \varphi_{t+1}(q_{t+1}^*),$$

donde:

$$t = T-1, T-2, \ldots, 1$$

Cabe decir que a pesar de que el algoritmo anterior ha sido formulado de manera recursiva, existe una versión iterativa mediante programación dinámica mucho más eficiente, con un coste O(NTB), siendo B el factor de ramificación efectiva del algoritmo dinámico y N el número total de estados.

2.2. Tecnología web

En este apartado trataremos los aspectos de la tecnología web que son de interés para el proyecto. En concreto estudiaremos los siguientes aspectos:

- **Lenguaje HTML:** Es el lenguaje predominante para el desarrollo de páginas web. Mediante este lenguaje podemos especificar todo el contenido y la forma de una página en forma de texto.
- **Navegación web:** Comprende la interacción entre el usuario e Internet, además de las técnicas utilizadas para ello. Generalmente se realiza mediante una aplicación conocida como navegador.
- **Navegador:** El navegador es una aplicación que interpreta el contenido de los archivos almacenados en Internet para mostrarlos en forma de web. Además incluye múltiples opciones para facilitar la navegación al usuario.

2.2.1. Lenguaje HTML

El HTML es un lenguaje de marcado basado en etiquetas utilizado para la codificación de páginas web. Describe el contenido y la apariencia de una página web, además de tener soporte para incluir dentro otros lenguajes, como puedan ser Javascript.

Uno de los componentes vitales del HTML son los elementos. Un elemento es contenido encerrado entre etiquetas para marcar su inicio y su final. Las etiquetas están encerradas entre <> y pueden contener atributos que en su mayoría son pares nombre-valor. Una etiqueta de final es exactamente igual a la de inicio salvo por un / antes del nombre. En la figura 2.2.1.1. podemos verlo más claramente mediante un ejemplo.

2.2.1. EL LENGUAJE HTML

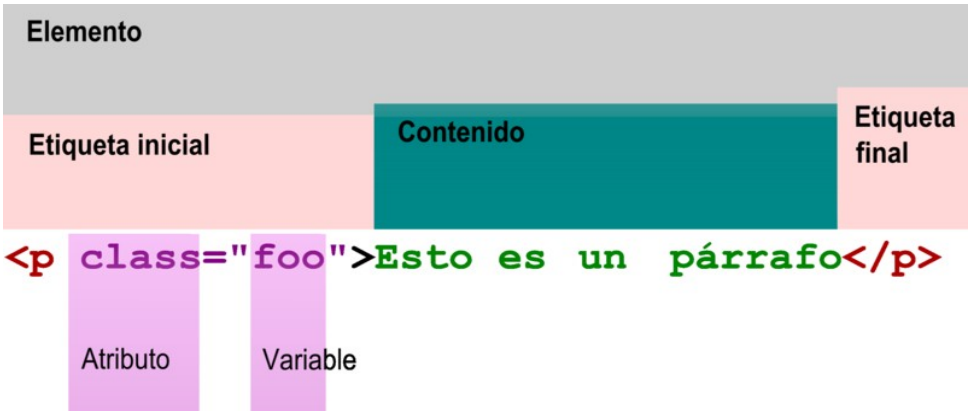


Figura 2.2.1.1: Ejemplo de un párrafo en HTML.

Podemos ver algunas de las etiquetas de uso más frecuente a continuación:

Etiqueta	Descripción
<html>	Define el inicio del documento HTML. Le indica al navegador que lo que viene a continuación debe ser interpretado como código HTML.
<script>	Incrusta o se llama a un script en una web.
<head>	Define la cabecera del documento HTML.
<title>	Define el título de la página.
<link>	Para vincular el sitio a hojas de estilo o iconos.
<style>	Para colocar el estilo interno de la página, usando un CSS o lenguajes similares.
<body>	Define el contenido principal o cuerpo de la página.
<table>	Define una tabla
<tr>	Representa la fila de una tabla.
<a>	Hipervínculo o enlace, dentro o fuera del sitio web. Lleva como atributo un *href=”nombre de la página”*.
<img>	Para insertar una imagen. Requiere un atributo *src* para indicar la su ubicación.
<b>	Texto en negrita.
<i>	Texto en cursiva.
<s>	Texto tachado.
<u>	Texto subrayado.
** **	Retorno de línea.

Cabe señalar que las etiquetas pueden estar anidadas y además que existen algunas, como
, que no requieren etiqueta de cierre. También comentar que cada documento HTML deberá comenzar con una etiqueta del estilo <!DOCTYPE …....> para indicar al navegador la versión de HTML utilizada. Por ejemplo *<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN" "http://www.w3.org/TR/html4/strict.dtd">* indica que está usando HTML 4.01.

Por último vamos a hablar de las entidades. Son caracteres especiales, como signos de puntuación, letras con tilde o diéresis que no se escriben tal cual, sino que se hace mediante un código que las representa. Aquí vemos una lista con algunos ejemplos:

Carácter	**Entidad HTML**	**Carácter**	**Entidad HTML**
á	á	Á	Á
é	é	É	É
í	í	Í	Í
ó	ó	Ó	Ó
ú	ú	Ú	Ú
à	à	À	À
è	è	È	È
ì	ì	Ì	Ì
ò	ò	Ò	Ò
ù	ù	Ù	Ù
ñ	ñ	Ñ	Ñ
ü	ü	¿	¿

Aquí tenemos un pequeño ejemplo de una página web con un texto “Hola mundo”:

```
<!doctype html>
<html>

<head>
<title>Hola mundo</title>
</head>

<body>
Hola mundo
</body>

</html>
```

2.2.2. Navegación web y el navegador

Hablamos de navegación web cuando existe una interacción entre un usuario y un navegador de Internet. El navegador es el encargado de interpretar el código fuente de una página web para mostrarlo por pantalla. Además, incluye todas las herramientas necesarias para una cómoda navegación web.

Mayoritariamente nos movemos de una página web a otra a través de enlaces e hipervínculos presentes en la propia web. Además, el navegador dispone de recursos como los "favoritos" o "marcadores", que permiten almacenar la dirección de una web que nos sea de interés. Por otro lado, hay herramientas para volver a la página web anterior, ver un historial de las páginas web visitadas, recargar la web actual por si ha habido un error o ha sufrido cambios, ...

En la figura 2.2.2.1. Podemos observar la interfaz típica de un navegador, con sus diferentes elementos resaltados.

Figura 2.2.2.1: Ejemplo de la interfaz del navegador Mozilla Firefox.

Elemento	Descripción
1. Menú de Archivo	Diversas opciones como crear nuevas ventanas, pestañas, imprimir la página actual, guardarla en el PC, trabajar sin conexión...
2. Menú de Editar	Herramientas de edición de texto tales como Copiar, Pegar, Cortar, Deshacer, Rehacer, Seleccionar todo...
3. Menú de Ver	Tiene diversas opciones para cambiar la interfaz, ocultar menús, modificar el diseño y disposición de los botones...
4. Menú de Historial	Nos permite ver un histórico de las páginas web visitadas anteriormente, ya sea en el día, la semana o todo el mes.
5. Menú de Marcadores	Aquí se almacenan las direcciones de las páginas web que nos son de interés, para no tener que buscarlas en el futuro y acceder inmediatamente.

6. Menú de Herramientas	En este menú tenemos las opciones referentes a la configuración del navegador. Además dispone de herramientas para el desarrollo web, tales como ver el código fuente de la página actual.
7. Atrás	Volver a la página web anterior.
8. Adelante	Volver a la página web siguiente.
9. Página de inicio	Cargar la página web que tenemos de inicio en el navegador.
10. Barra de direcciones	En esta barra está la dirección de la web actual. Además, sirve para poder escribir y acceder a cualquier dirección web.
11. Recargar	Permite volver a cargar la página web actual. Sirve para ver posibles cambios que haya habido desde la última carga, o en caso de que se haya producido un error durante la carga.
12. Buscador	Busca en el proveedor de búsquedas por defecto la cadena de texto escrita en la barra de búsqueda.
13. Menú de Ayuda	Tenemos diferentes opciones de soporte sobre el navegador. Podemos ponernos en contacto con los desarrolladores, leer el manual, notas sobre la versión actual...
14. Pestaña actual	El navegador tiene la opción de tener varias páginas abiertas al mismo tiempo en una sola ventana. Las diferentes páginas se agrupan en pestañas. Así, seleccionando una, vemos la página web cargada para ésta.
15. Crear nueva pestaña	Abrimos una nueva pestaña.

Existen muchas otras órdenes u opciones adicionales, que comentaremos en la sección 3.2 que está dedicada al navegador utilizado en el desarrollo del proyecto, Mozilla Firefox.

Por último, cabe señalar que la mayoría de los elementos del navegador, aparte de poder ser accionados mediante botones, también pueden serlo mediante eventos de teclado. Esta información resultará muy útil en la sección 3.4, dedicada al desarrollo de la aplicación de reconocimiento de voz.

Capítulo 3

Aplicaciones

En este capítulo vamos a hablar de las aplicaciones externas al proyecto de las cuales haremos uso en el desarrollo de la tarea o bien que forman parte del sistema final desarrollado. En concreto las herramientas son cuatro:

- **iATROS:** Es un reconocedor del habla y texto manuscrito, con una estructura modular, para poder confeccionar diferentes sistemas a partir de sus herramientas.
- **Mozilla Firefox:** Uno de los navegadores más populares, de código abierto y disponible para múltiples sistemas.
- **Xmacro:** Una herramienta para grabar y reproducir eventos de teclado y ratón en un servidor X-Windows.
- **eutranscribe:** Dada una palabra, la analiza y nos devuelve su transcripción de manera que los fonemas que la componen se adaptan al modelo acústico.

3.1. iATROS

iATROS (del inglés *improved ATROS*) es una nueva implementación del ATROS, un reconocedor de voz anterior, que ha sido adaptado para ser utilizado tanto en el habla y el reconocimiento de texto manuscrito. Ha sido desarrollado por el PRHLT (Pattern Recognition and Human Language Technology), formado por miembros del DSIC (Departamento de Sistemas Informáticos y Computación) y DISCA (Departamento de Informática de Sistemas y Computadores) de la UPV, y del ITI (Instituto Tecnológico de Informática).

iATROS proporciona una estructura modular que puede ser utilizada para construir diferentes sistemas, en cuyo núcleo existe una búsqueda de Viterbi sobre una red de modelos ocultos de Markov. iATROS proporciona herramientas estándar de reconocimiento del habla *offline* y *online* (en base a módulos ALSA). Los módulos están completamente desarrollados en el lenguaje C y toda la aplicación es de código abierto.

Nosotros haremos uso de sus herramientas para el reconocimiento del habla; en modo *online* para la aplicación desarrollada, y *offline* para las pruebas de rendimiento.

Para conocer más información sobre la aplicación, puede visitarse la web oficial del proyecto en [2].

3.1.1. Módulos del sistema

El sistema se compone de tres módulos. Los dos primeros de preprocesamiento y extracción de características (para texto escrito o voz) y finalmente el módulo de reconocimiento. Los dos primeros proporcionan vectores de características al módulo de reconocimiento, que usando modelos ocultos de Markov y los modelos del sistema comentados en el apartado 2.1.2, aplica el algoritmo de Viterbi para encontrar la mejor hipótesis.

Vamos a explicar al proceso con más detalle. Podríamos decir que tenemos dos fases diferenciadas:

- **Preproceso acústico**: Comprende los módulos de preprocesamiento y extracción de características. Se extraen vectores de coeficientes cepstrales a partir de una señal de voz. Un coeficiente cepstral es un valor numérico que describe la energía que contiene un cierto rango de frecuencias de la señal del habla. Para la extracción de características se utiliza una ventana deslizante sobre la señal de entrada. Cada porción de la señal comprendida en la ventana se denomina *frame*. Los cepstrales serán extraídos de cada *frame* mediante un proceso que atraviesa diferentes fases. Dicho proceso se puede ver con más detalle en [16].
- **Decodificación lingüística**: Está formado por el módulo de reconocimiento. Hace uso de los modelos del sistema (modelo léxico, del lenguaje y acústico) creados para una tarea específica. Dichos modelos son utilizados, en una fase previa, para unirlos en una red de modelos ocultos de Markov (HMM) que comprenda todo el lenguaje.
  En esta fase, a partir de una secuencia de vectores de características dada, aplica el algoritmo de Viterbi sobre la red de HMM de nuestro lenguaje, para determinar la mejor hipótesis.

En la figura 3.1.1.1. tenemos un diagrama de los módulos que comprenden el sistema.

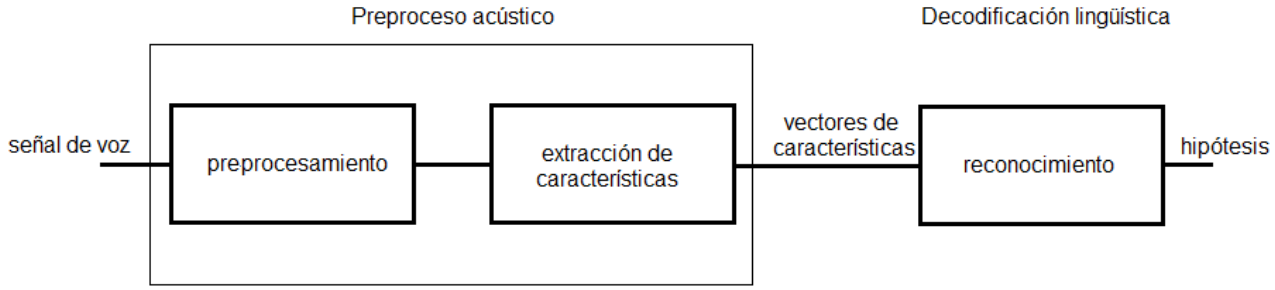


Figura 3.1.1.1: Diagrama módulos iATROS

Hay que señalar, que para la versión *online* del reconocimiento, los vectores de cepstrales de los *frames* son enviados al módulo de reconocimiento inmediatamente para que pueda ir realizando cálculos. De este modo no es necesario esperar al final de la pronunciación del locutor para iniciar el reconocimiento.

Como se ha comentado anteriormente, iATROS recibe una serie de ficheros de entrada. Además de los ficheros de los modelos del sistema, también necesita un archivo de configuración de la señal de audio de entrada y otro de configuración del sistema. En los siguientes apartados hablaremos de ellos.

3.1.2. Fichero de configuración del sistema

El fichero de configuración del sistema proporciona a iATROS la ubicación de los archivos de entrada, además del valor de los parámetros necesarios para su configuración. Aunque existen otros parámetros posibles o adicionales según el sistema desarrollado, nosotros vamos a comentar los que han sido de utilidad para el desarrollo del proyecto.

3.1.2. FICHERO DE CONFIGURACIÓN DEL SISTEMA

En la siguiente tabla se muestran los parámetros que nos son de interés:

Parámetro	Significado
samples	Indica la ubicación de un fichero que lista todos los ficheros con vectores de cepstrales a testear.
hmm	Ruta del fichero del modelo acústico.
lexicon	Ruta del fichero del modelo léxico.
grammar	Ruta del fichero del modelo del lenguaje.
grammar-type	Indica el modo en el que están construidos los ficheros de los modelos. Existen varias opciones: *FSM* (f*inite-state-machine*), *NGRAM* (*n-gramas*)...
beam	En el algoritmo de búsqueda se utiliza como criterio de poda, ya que se eliminarán las transiciones que superen la puntuación actual más el valor de *beam*. Los valores normales oscilarán entre 50 y 1000 dependiendo de los modelos del sistema.
grammar-scale-factor	El parámetro *GSF* (*grammar-scale-factor*) se utiliza en la fórmula siguiente: $P = \frac{MA + (GSF * LM)}{N}$ Siendo *P* la probabilidad final, *LM* la probabilidad del modelo del lenguaje, *MA* la probabilidad del modelo acústico y *N* un factor de normalización. Valores normales oscilan entre 0 y 50. Su efecto consiste en dar mayor o menor importancia al modelo del lenguaje.
Word-insertion-penalty	El parámetro *WIP* (*Word-insertion-penalty)* se utiliza en la fórmula siguiente: $-\log P = -\log MA + (GSF * LM) + WIP$ Valores normales oscilan entre -10 y 10. Su efecto consiste en penalizar o favorecer la decodificación de palabras cortas.
histogram-pruning	En el algorítmo de búsqueda, se utiliza como criterio de control, ya que si el tamaño del heap supera el tamaño definido en *histogram-pruning*, se elimina la hipótesis menos probable. Valores normales oscilan entre 5.000 y 15.000.

Podemos ver el aspecto de un posible fichero de Configuración del Sistema:

```
samples = ./lista_comandos_leidos

[decoder]
hmm = ./modelos/albayzin_iatros_64gs.hmm
lexicon = ./modelos/lexico_firefox.lx
grammar = ./modelos/automata_firefox.gr
grammar-type = FSM
beam = 65
grammar-scale-factor = 10
word-insertion-penalty = -9
histogram-pruning = 5000
```

En el capítulo 5, dedicado a la configuración del sistema, realizaremos pruebas variando los parámetros numéricos anteriores para optimizar el funcionamiento de la aplicación.

3.1.3. Fichero de configuración de la señal de audio

En el modo de reconocimiento *online* estamos realizando la adquisición de audio desde el propio iATROS, y será necesario un archivo de configuración que defina los parámetros de la señal de audio de entrada. Éste es precisamente el cometido del fichero de configuración de la señal de audio.

El fichero contiene diversos parámetros, pero nos centraremos solo en los más destacables:

Parámetro	Significado
SampleFrec	Indica la frecuencia de muestreo de la señal de audio. Valores normales oscilan desde 3.5 kHz hasta 33 kHz.
Channels	El número de canales de audio de entrada. Por ejemplo si queremos grabar en mono, sería solamente un canal.
Bits	El tamaño de la muestra empleado en la cuantificación del audio. El valor normal sería 16 bits.
InputDevice	Dispositivo de entrada de audio. Para utilizar la entrada por defecto del sistema operativo se puede dejar en *default*.

Vamos a ver el aspecto de un archivo de configuración de la señal de audio, que además incluirá los otros parámetros no comentados en la tabla anterior. El fichero sería así:

```
<filter>
TriangularFilters       23
</filter>
```

3.1.3. FICHERO DE CONFIGURACIÓN DE LA SEÑAL DE AUDIO

```
<parameters>
FrameSize      410
SampleFreq     16000
SubSampleFreq          100
FFTlength      512
FrameShift     160
WindowLen      0.025625
Factor 0.97
WindowAlfa     0.54
CepCoefNumb            13
Channels       1
Bits    16
Frames32
SecondsSilence         0.5
SilenceThreshold       50
Derivative     2
</parameters>

<buffers>
SizeSignal     100000
SizeCC         1000
</buffers>

<device>
InputDevice    default
OutputDevice default
</device>
```

3.1.4. Ficheros de los modelos

En el apartado 2.1.2 presentamos los modelos del sistema. Recordemos que los modelos son: modelo léxico, modelo del lenguaje y modelo acústico. En el presente apartado explicaremos con detalle la estructura que siguen los modelos del sistema en iATROS.

Modelo léxico

El modelo léxico incluye todas las palabras que se pueden pronunciar en nuestro lenguaje definido. Para cada palabra tendremos un AFD cuyas transiciones entre estados serán los símbolos que representan los fonemas almacenados en el modelo acústico.

El diseño del fichero es sencillo. Para cada autómata tendrá la siguiente forma:

Name “nombre”
State 0 i=1
0 1 “símbolo_fonema_0-1” p=1
1 2 “símbolo_fonema_1-2” p=1
…
n-1 n “símbolo_fonema_n-1-n” p=1
State n f=1

Donde *nombre* y *símbolo_fonema* serán el nombre que representará al AFD y por tanto a la palabra, y el símbolo que representa al fonema en el modelo acústico respectivamente.
Como podemos observar, en cada transición tenemos una probabilidad asociada, pero como el camino en el autómata es único, la probabilidad será 1 en todos los casos (ya que no vamos a emplear pronunciaciones alternativas). Además, vemos que el estado 0 (State 0) tiene al lado i=1, cuyo significado es que es el estado inicial (de ahí la i). Del mismo modo vemos en el estado n que tiene una f=1, que denota el estado final.

Vamos a ilustrar la teoría con un ejemplo. Digamos que queremos crear un fichero con autómatas para las palabras “nueva”, “ventana” y “pestaña”, y que los fonemas que representan los sonidos en el modelo acústico se escriben tal cual la letra. El fichero quedaría del siguiente modo:

Name “nueva”
State 0 i=1
0 1 “n” p=1
1 2 “u” p=1
2 3 “e” p=1
3 4 “v” p=1
4 5 “a” p=1
State 5 f=1

Name “ventana”
State 0 i=1
0 1 “v” p=1
1 2 “e” p=1
2 3 “n” p=1
3 4 “t” p=1
4 5 “a” p=1
5 6 “n” p=1
6 7 “a” p=1
State 7 f=1

Name “pestaña”
State 0 i=1
0 1 “p” p=1
1 2 “e” p=1
2 3 “s” p=1
3 4 “t” p=1
4 5 “a” p=1
5 6 “ñ” p=1
6 7 “a” p=1
State 7 f=1

Vemos que el diseño del modelo léxico es muy sencillo y que se podrán añadir palabras con facilidad. Hay que señalar que aquí no hemos tenido en cuenta detalles referentes a los símbolos de los fonemas tales como la letra *ñ*, la cual realmente será representada por la letra *h* en nuestro modelo acústico. Esto será explicado con detalle en este mismo apartado, en la parte dedicada al diseño del modelo acústico.

3.1.4. FICHEROS DE LOS MODELOS

Modelo del lenguaje

El modelo del lenguaje incluye todas las posibles frases que serán admitidas en nuestro lenguaje definido. Las palabras que las forman serán las presentes en el modelo léxico, y todas las frases formarán parte de un mismo AFD, usualmente con un único estado inicial y final.

El fichero del modelo del lenguaje tendrá el siguiente diseño:

La cabecera del modelo formada por:

- *Name*: nombre del modelo.
- *NumStates*: número total de estados del modelo.
- NumEdges: número total de transiciones entre estados del modelo.
- *TotalChecksum*: Suma de las probabilidades de ser inicial de todos los estados.

A continuación vienen todos los estados y transiciones, definidos del siguiente modo:

State N_ESTADO c = CHECKSUM i = INICIAL f = FINAL
E_INI E_FIN "PALABRA_M_LÉXICO" p = PROB
… (se pueden poner tantas transiciones como se quiera)

Cuyos parámetros se explican en la tabla siguiente:

Parámetro	Significado
N_ESTADO	Número del estado. Comienza por el estado 0.
CHECKSUM	Suma de la probabilidad de las transiciones del estado y su probabilidad de ser final.
INICIAL	Valor real que utiliza para asignar al estado la probabilidad de ser inicial. Solo se suele poner en un estado, y en los demás no aparecerá.
FINAL	Valor real que utiliza para asignar al estado la probabilidad de ser final. Solo aparece si es final.
E_INI	Estado origen de la transición. Siempre será el propio estado que la contenga.
E_FIN	Estado destino de la transición.
PALABRA_M_LÉXICO	Palabra presente en el modelo léxico que representa la transición.
PROB	Probabilidad de la transición. En nuestro caso todas las transiciones serán equiprobables.

Vamos a ver un ejemplo. En este mismo apartado hemos diseñado un modelo léxico con las palabras “nueva”, “ventana” y “pestaña”. Digamos que queremos diseñar un modelo para el reconocimiento de las frases “nueva ventana” y “nueva pestaña”. El modelo quedaría así:

Name modelo1

NumStates 4

NumEdges 3

TotalChecksum 1

State 0 c = 1 i = 1
0 1 “nueva” p = 1

State 1 c = 1
1 2 “ventana” p = 0.5
1 2 “pestaña” p=0.5

State 2 c = 1 f = 1

Realmente, el modelo anterior ha sido simplificado. En la práctica, para que funcione bien el modelo, tendremos una palabra “<s>” que representará el fonema definido para el silencio, y que será la que transite del estado 0 al 1 y ahí será donde comiencen de verdad las frases del modelo del lenguaje.

Modelo acústico

El modelo acústico está formado por varios modelos que dan cuenta de cada posible fonema de nuestro lenguaje definido, con el objetivo de que funcione con cualquier locutor. Cada modelo será representado mediante un modelo oculto de Markov.

Utilizaremos el modelo Albayzin64gs [15], desarrollado a partir de las herramientas de HTK [1], que ha sido desarrollado a partir de frases equilibradas fonéticamente, frases correspondientes a una tarea de consulta a una base de datos geográfica, y habla producida en condiciones adversas. De este modo se intenta tener un corpus equilibrado compatible con el máximo número posible de locutores de habla española.

El diseño del modelo es como sigue. De forma abstracta, se podría decir que para cada fonema, la primera línea indica el nombre que le representa, el cual utilizamos en el modelo léxico. A continuación tenemos 64 gausianas diferentes del mismo fonema, para cada una de las cuales se especifica su media y su matriz de covarianza, representada mediante una matriz diagonal. Los modelos son HMM con 3 estados, con una topología lineal izquierda-derecha. Cada gausiana de un estado posee 39 componentes (compuestas por sus coeficientes cepstrales, energía, y sus primeras y segundas derivadas). Cada estado tiene un bucle sobre sí mismo y modeliza la probabilidad de emisión del símbolo mediante la mixtura de gausianas.

Además, hay que comentar que se han modelado silencios, representados por <s>, y que algunos

fonemas no están representados por la letra que parecería la más evidente. Es el caso de la */ñ/*, representada por la *h*, la */j/* representada por la *x*, la */rr/* representada por @, etc. En la sección 3.4 dedicada a la aplicación *eutranscribe*, veremos algunos ejemplos más, y cómo detectarlos mediante esa aplicación para añadir las palabras de forma correcta en el modelo léxico.

Vamos a ver como se representaría el fonema */c/*:

```
~h "c"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<NUMMIXES> 64
<MIXTURE> 1 3.894406e-03
<MEAN> 39
 1.080350e+04 5.014745e+01 -5.987255e+01 … … 1.201299e+01
<VARIANCE> 39
 5.304736e+05 7.025270e+04 1.943633e+04 … … 3.189388e+02
<GCONST> 3.882369e+02
<MIXTURE> 2 3.345787e-02
<MEAN> 39
 1.901917e+04 2.192118e+03 -3.230256e+02… …1.638872e+01
<VARIANCE> 39
 1.067590e+06 1.095485e+05 9.450711e+04 … … 3.933123e+02
<GCONST> 4.128390e+02
 …
 …
 …
<STATE> 3
<NUMMIXES> 64
<MIXTURE> 1 1.028401e-02
<MEAN> 39
 1.906498e+04 -4.576482e+02 -6.440861e+02 …. ... 3.119503e+01
<VARIANCE> 39
 8.471800e+05 2.809315e+05 1.055854e+05 … ... 3.538454e+02
<GCONST> 4.117584e+02
 …
 …
 …
<STATE> 4
<NUMMIXES> 64
 …
 …
 …
<TRANSP> 5
 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 8.043005e-01 1.956995e-01 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 7.331287e-01 2.668713e-01 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 7.631398e-01 2.368602e-01
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
<ENDHMM>
```

3.2. Mozilla Firefox

Mozilla Firefox es un navegador libre y de código abierto desarrollado por la Fundación Mozilla. Es multiplataforma y el segundo navegador más utilizado en Internet.

Sus características incluyen navegación por pestañas, corrector ortográfico, búsqueda progresiva, marcadores dinámicos, un administrador de descargas, navegación privada, navegación con georreferenciación, aceleración mediante GPU e integración del motor de búsqueda que desee el usuario. Además se pueden añadir funciones a través de complementos desarrollados por terceros.

En el apartado 2.2.2 ya pudimos conocer información sobre su interfaz, así que pasaremos directamente a presentar las opciones del navegador que más nos interesan.

3.2.1. Teclas de acceso rápido

Una de las características más comunes en los navegadores web, que además es la más importante para nuestro proyecto, son las teclas de acceso rápido para el accionamiento de órdenes o eventos. Mediante la pulsación de una o varias teclas simultáneamente podemos activar cualquiera de los eventos del navegador.

Vamos a ver una lista de los eventos que nos son de interés:

Orden	Teclas acceso rápido	Significado
Abrir en segundo plano	*Ctrl + Mayus + Enter*	Abre el enlace seleccionado en segundo plano.
Ampliar	*Ctrl + "+"*	Realiza un zoom sobre la pestaña actual.
Añadir marcador	*Ctrl + D*	Agrega la página actual a los marcadores.
Atrás	*Alt + Flecha izquierda*	Cargamos la página anterior.
Adelante	*Alt + Flecha derecha*	Cargamos la página siguiente.
Ayuda	*F1*	Despliega el menú de ayuda.
Bajar	*Flecha abajo*	Desplaza la vista sobre la página una línea hacia abajo.
Subir	*Flecha arriba*	Desplaza la vista sobre la página una línea hacia arriba.
Cierra pestaña	*Ctrl + W*	Cierra la pestaña actual.
Enlace anterior	*Shift + Tab*	Selecciona el enlace anterior en la página actual.
Enlace siguiente	*Tab*	Selecciona el enlace siguiente en la página actual.

3.2.1. TECLAS DE ACCESO RÁPIDO

Entrar	*Enter*	Abre enlace seleccionado o confirma una selección.
Escape	*ESC*	Cancelar / detener carga web actual.
Historial	*Ctrl + H*	Abre / Cierra el menú del historial web.
Marcadores	*Ctrl + B*	Abre / Cierra el menú de marcadores.
Nueva pestaña	*Ctrl + T*	Abre una nueva pestaña.
Nueva Ventana	*Ctrl + N*	Abre una nueva ventana.
Página inicio	*Alt + Home (Inicio)*	Carga la página de inicio del navegador.
Pantalla completa	*F11*	Activa / Desactiva la navegación a pantalla completa del navegador.
Recargar	*F5*	Recarga la página web actual.
Reducir	*Ctrl + "-"*	Reduce la vista en la página web actual.
Siguiente pestaña	*Ctrl + Tab*	Pasamos a la pestaña siguiente de la ventana actual.
Siguiente ventana	*Alt + Tab*	Pasamos a la siguiente ventana en el navegador.
Anterior ventana	*Alt + Shift + Tab*	Pasamos a la anterior ventana en el navegador.
Ver código fuente	*Ctrl + U*	Muestra el código fuente de la página actual.

Todas estas teclas de acceso rápido, combinadas con el programa Xmacro presentado en la sección 3.3, servirán para accionar las órdenes del navegador Firefox de modo invisible. Solo faltará diseñar en el capítulo 4 la aplicación que haga el reconocimiento de voz de la orden a ejecutar y envié las teclas a pulsar al Xmacro.

3.2.2. Almacenamiento de la sesión

En el apartado 4.3.2, donde hablaremos del diseño de las herramientas para agregar el texto de los enlaces de la web actual a los modelos del sistema, descubriremos que será necesario poder conocer la dirección de la página web actual desde fuera del navegador. Para ello extraeremos esa información del archivo donde se almacena el historial de la sesión actual del Firefox.

El archivo en cuestión se denomina *sessionstore.js*, y en su versión para Linux está almacenado en *directorio_instalación_Firefox/firefox/*.default/sessionstore.js*

El archivo almacena todas las páginas web visitadas en cada ventana y pestaña, además de otra información adicional que no nos es de interés y que no vamos a comentar.

La estructura con la que se almacena la información está formada por listas con diferentes campos. El diseño del archivo es el siguiente:

```
(
        {“windows”: [
                {”tabs”: [
                        {“entries”: [
                                {“url”:”dirección_web1”, ...}, {“url”:”dirección_web2”, …,
                                “formdata”,...}, ...
                                  ],  ....
                        },
                        {“entries”: [
                                {“url”:”dirección_web1”, ...}, {“url”:”dirección_web2”, …,
                                “formdata”,...}, ...
                                  ],  ....
                        },
                        …
                        {“entries”: [
                                {“url”:”dirección_web1”, ...}, {“url”:”dirección_web2”, …,
                                “formdata”,...}, ...
                                  ],  ....
                        }
                        ], … “selected”: PESTAÑA_SELECCIONADA …
                },
                …
                {”tabs”: [
                        {“entries”: [
                                {“url”:”dirección_web1”, ...}, {“url”:”dirección_web2”, …,
                                “formdata”,...}, ...
                                  ],  ....},
                        …
                        {“entries”: [
                                {“url”:”dirección_web1”, ...}, {“url”:”dirección_web2”, …,
                                “formdata”,...}, ...
                                  ],  ....}
                        ], … “selected”: PESTAÑA_SELECCIONADA …
                }
                    ], … “selected”: VENTANA_SELECCIONADA …
        }
)
```

Como podemos ver, tenemos una lista de ventanas, denominada *“windows”*, la cual contiene una lista de pestañas *“tabs”* por cada ventana abierta. Dentro de lista *“tabs”* existe una lista *“entries”* por cada pestaña abierta en esa ventana. Finalmente vemos que una lista *“entries”* puede contener varias *“url”*, ya que almacena todas las páginas visitadas. Además existe un campo *“selected”*

presente en cada lista *"tabs"* y *"windows"*, que contiene el número de la pestaña o ventana seleccionada en ese momento en el navegador. De un modo similar, existe un campo *"formdata"* indicador de que esa dirección web es la cargada actualmente en la pestaña.

Por último, es interesante conocer cómo aumentar la frecuencia de actualización del fichero *sessionstore.js*, ya que por defecto actualiza cada diez segundos. Los pasos son sencillos:

- 1º Abrir firefox, y escribir en la barra de direcciones: about:config
- 2º En el filtro de busqueda escribir: browser.sessionstore.interval
- 3º En la columna "valor", fijarlo por ejemplo en 2000 ms.

A partir de toda esta información será posible diseñar una aplicación que dado el archivo nos devuelva la dirección web que tenemos delante en pantalla, cosa que se hará en el apartado 4.3.2.

3.3. Xmacro

Xmacro es una sencilla aplicación de código abierto desarrollada en C++. Está diseñada para reproducir eventos de teclado y ratón en un servidor X Windows. La aplicación contiene dos ejecutables:

- **xmacrorec**: La aplicación muestra por pantalla el código de todas las pulsaciones realizadas. Dicho código será ejecutable mediante la aplicación xmacroplay. Se podría decir que nos sirve para grabar todas las pulsaciones y luego reproducirlas con la otra aplicación. Otra utilidad es conocer el código de algunas teclas.
- **xmacroplay**: Es la encargada de enviar al servidor X Windows todos los eventos almacenados en su fichero de entrada. Puede enviar pulsaciones simultaneas, cadenas de texto, pulsaciones de determinada duración, etc.

La aplicación que nos interesa es xmacroplay. La ejecución es sencilla:

xmacroplay :NUMERO_PANTALLA

Donde NUMERO_PANTALLA es la pantalla a la que queremos enviar los eventos. Normalmente será 0. Una vez ejecutada será solo escribir mediante su sintaxis los códigos de los eventos a ejecutar, y el programa se encargará de mandarlos al servidor X Windows.

La forma más adecuada de ejecutar la aplicación será almacenando el texto con los eventos en un fichero, y enviárselo a Xmacro mediante una tubería. Por ejemplo del siguiente modo:

echo "KeyStr Tab" | xmacroplay :0

La orden equivale a una pulsación de la tecla tabulación.

Vamos a ver una lista con la sintaxis más destacada de la aplicación:

Orden	Significado
KeyStr key_code	*KeyStr* envía al servidor X Windows una pulsación de la tecla *key_code*.
KeyStrPress key_code	*KeyStrPress* inicia la pulsación de la tecla *key_code*.
KeyStrRelease key_code	*KeyStrRelease* detiene la pulsación de la tecla *key_code*.
String cadena	*String* envíar al servidor X Windows la pulsación de las teclas que se correspondan con todo el texto presente en *cadena*.

Los códigos de las teclas están presentes en el archivo del sistema *X11/keysymdef.h*, quitando el prefijo *XK_*.

Un ejemplo de como ejecutar la pulsación de las teclas *Alt + Tab* sería el siguiente:

```
echo "KeyStrPress Alt_L
        KeyStr Tab
        KeyStrRelease Alt_L" | xmacroplay :0
```

Para conocer más información sobre la aplicación consultar [3].

3.4. eutranscribe

Anteriormente comentamos que en el modelo acústico no todos los fonemas están representados por una grafía equivalente. Para poder agregar al modelo léxico todas las palabras de forma adecuada será necesario conocer la transcripción correcta que representa todos sus sonidos en el modelo acústico. Para ello utilizaremos eutranscribe, una sencilla aplicación desarrollada en el lenguaje perl que dada una palabra nos devuelve su transcripción correcta para nuestro modelo.

Algunas de las tareas que realiza la aplicación son las siguientes:

- Elimina los acentos.
- Pasa de mayúsculas a minúsculas.
- Transforma las *'ñ'* se pronuncian con el fonema '*h'*.
- La doble *'r'* la transforma en *'@'*.
- Elimina signos de puntuación especiales y iniciales.
- Funde las cadenas de blancos.
- Sustituye *'j'* por *'x'*.
- La grafía *“qu”* se pronuncia como *'k'*.
- Las grafías *“gue”* y *“gui”* se cambia por *“ge”* y *“gi”* a no ser que lleven diéresis.
- La *'g'* antes de *'e'* e *'i'* se sustituyen por el fonema */j/* que en este caso sería *'x'*.

- La *'c'* delante de *'e'* e *'i'* se cambia por *'z'*.
- La “ch” se cambia por 'c'.
- La *'c'* cuando no es *“ch”* es una *'k'*.
- Si concurren dos nasales lingualveolares, se pronuncia solo una.
- La partícula *“ps”* a principio de palabra se pronuncia *'s'*.
- La *'w'* se pronuncia *“gu”*.
- Cambiar la *'v'* por *'b'*.
- …

Podemos ver el resultado de la transformación de algunas palabras:

Palabra original	**Transcripción**
bajar	*baxar*
ventana	*bentana*
ocho	*oco*
pestaña	*pestaha*
web	*gueb*
cambiar	*kambiar*
cero	*zero*

Ahora ya podemos agregar adecuadamente las palabras al modelo léxico. Además, una vez presentadas todas las aplicaciones, ya estamos listos para pasar al desarrollo de la aplicación de navegación web por voz.

Capítulo 4

Desarrollo de la aplicación

En este capítulo trataremos el desarrollo de la aplicación de navegación web mediante el habla implementada. En principio, para su desarrollo, podríamos hablar de cuatro fases diferenciadas:

- **Definición del conjunto de órdenes:** Antes de comenzar, habrá que definir qué órdenes será capaz de ejecutar nuestra aplicación de navegación web.

- **Diseño del reconocedor:** Como comentamos anteriormente, para el reconocimiento del habla utilizaremos la aplicación iATROS, en su versión para el reconocimiento *online.* Tendremos que crear los modelos del sistema correspondientes y poner en funcionamiento la aplicación.
- **Diseño del intérprete:** Una vez conozcamos la orden a ejecutar, tiene que haber un programa que la interprete y se la envíe al navegador; esa es la función del intérprete. Será el encargado de una vez recibida una orden del iATROS, ejecutar todas las acciones previas necesarias y enviar las ordenes al navegador Firefox.
- **Comunicación entre módulos:** En este apartado trataremos todo lo relacionado con la comunicación entre los diferentes módulos. Utilizaremos tuberías con nombre y la aplicación Xmacro para envío de eventos de teclado a servidores X Windows.

En la figura 4.0.1. podemos ver un esquema básico del sistema a diseñar.

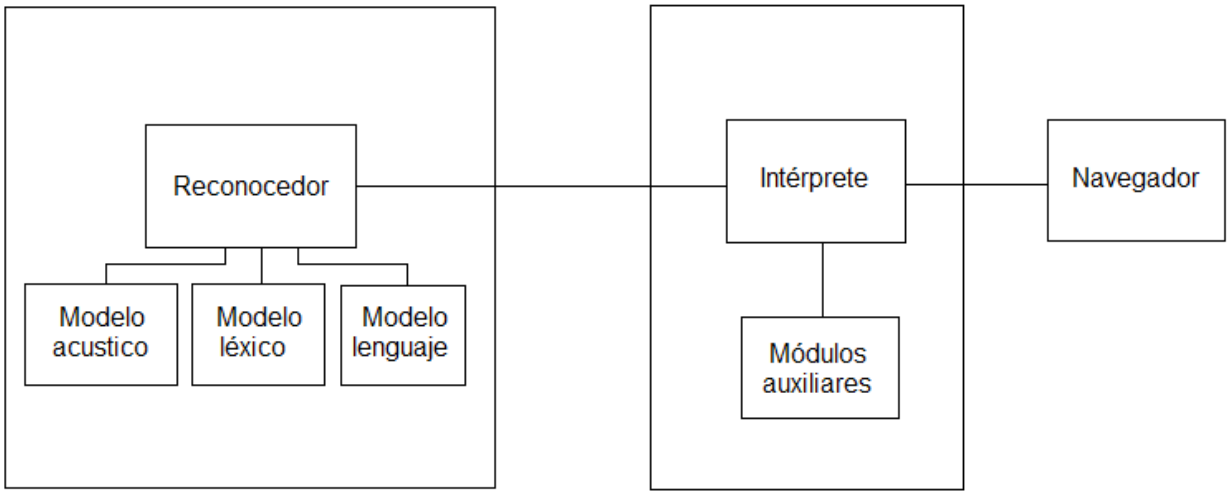


Figura 4.0.1: esquema sistema navegación web con voz.

4.1. Conjunto de órdenes

En primer lugar tenemos que definir un conjunto de órdenes suficientemente amplio para una navegación web cómoda. El texto de la orden será el que tendremos que pronunciar para ejecutarla una vez esté terminada la aplicación. Además de las típicas opciones de un navegador, tendremos que añadir otras más propias de una herramienta de reconocimiento de voz, las cuales irán referidas al intérprete.

En la siguiente tabla tenemos la lista de órdenes referentes al navegador que hemos definido:

Orden	**Significado**
abrir enlace + *"texto enlace"*	Abre el enlace que tenga como texto *"texto enlace"*.
abrir en segundo plano	Abre el enlace seleccionado en segundo plano.
ampliar	Realiza un zoom sobre la pestaña actual.
añadir marcador	Agrega la página actual a los marcadores.
atrás	Cargamos la página anterior.
adelante	Cargamos la página siguiente.
ayuda	Despliega el menú de ayuda.
bajar / baja	Desplaza la vista sobre la página 18 líneas hacia abajo.
subir / sube	Desplaza la vista sobre la página 18 líneas hacia arriba.
baja uno / baja dos / baja tres	Desplaza la vista sobre la página 1/2/3 líneas hacia abajo.
sube uno / sube dos / sube tres	Desplaza la vista sobre la página 1/2/3 líneas hacia arriba.
cierra pestaña	Cierra la pestaña actual.
código fuente	Muestra el código fuente de la página actual.
enlace anterior / anterior enlace	Selecciona el enlace anterior en la página actual.
enlace anterior + *"número"*	Se desplaza *X* enlaces hacia atrás en la web. El valor máximo de *X* será 99.
enlace siguiente / siguiente enlace	Selecciona el enlace siguiente en la página actual.
enlace + "número"	Se desplaza *X* enlaces hacia adelante en la web. El valor máximo de *X* será 99.
entrar / enter	Abre enlace seleccionado o confirma una selección.
escape	Cancelar / detener carga web actual.

4.1. CONJUNTO DE ÓRDENES

historial	Abre / Cierra el menú del historial web.
marcadores	Abre / Cierra el menú de marcadores.
nueva pestaña	Abre una nueva pestaña.
nueva ventana	Abre una nueva ventana.
página inicio	Carga la página de inicio del navegador.
pantalla completa	Activa / Desactiva la navegación a pantalla completa del navegador.
recargar	Recarga la página web actual.
reducir	Reduce la vista en la página web actual.
siguiente pestaña / pestaña siguiente	Pasamos a la pestaña siguiente de la ventana actual.
siguiente ventana / ventana siguiente	Pasamos a la siguiente ventana en el navegador.
ventana anterior / anterior ventana	Pasamos a la anterior ventana en el navegador.

Por otro lado, las órdenes referentes al intérprete serán las siguientes:

Orden	**Significado**
analizar web	Una de las opciones de nuestra aplicación será analizar el código fuente de una web, y extraer sus hiperenlaces con el texto que tienen asociado, para luego añadir ese texto a nuestros modelos del sistema, y que podamos acceder a ellos con la orden:" abrir enlace + "*texto enlace*"". El análisis de una página web no es automático, sino que lo activamos mediante la presente orden. De este modo ahorramos cálculos a la aplicación, y no complicamos el reconocimiento de voz, añadiendo palabras al lenguaje solo cuando sea necesario.
mostrar lista enlaces	Una vez ha sido analizada una web es posible que el texto original de los enlaces no sea el mismo a pronunciar, ya que eliminamos números y otros caracteres. Mediante esta orden vemos todos los enlaces listados en una sencilla página web, con un número al lado de manera que podamos combinarlo con la orden "enlace + "*número*"".
terminar	Termina la ejecución de la aplicación, cerrando todos sus procesos asociados.

pausar	Esta orden permite detener el reconocimiento de voz hasta que se vuelva a pronunciar. Resulta muy útil en caso de querer hablar de forma externa a la aplicación.
recargar modelos	Es una orden que recarga los modelos del sistema actuales por los generados con la orden "analizar web". Hay que señalar que su uso es solo para pruebas, pues la orden "analizar web" cambia los modelos automáticamente.
resetear modelos	Cambia los modelos del sistema actuales por los modelos por defecto del sistema.

Tenemos definido un total de cuarenta ordenes posibles. Ya podemos pasar a definir el lenguaje del sistema, y por tanto, crear nuestro reconocedor.

4.2. Diseño del reconocedor

En la sección 3.1. ya hablamos de iATROS en profundidad. Ahora es el momento de utilizarlo para crear un reconocedor adecuado para nuestra tarea. Para ello, en primer lugar vamos a crear los ficheros de los modelos del sistema a partir del vocabulario de la sección 4.1.

4.1.1. Creación de los modelos del sistema

Anteriormente, en el apartado 2.1.2, presentamos los modelos del sistema: modelo léxico, modelo del lenguaje y modelo acústico. Además, en el apartado 3.1.4 explicamos la estructura que tienen que seguir los ficheros de los modelos para su correcto funcionamiento en iATROS. Siguiendo esa estructura, y a partir del conjunto de ordenes definido en la sección 4.1, es muy sencillo diseñar los ficheros de los modelos léxico y del lenguaje (recordemos que para el modelo acústico utilizaremos uno externo al proyecto, el Albayzin64gs [15]).

Modelo léxico

El vocabulario a añadir al modelo es todo el que forman las órdenes de la sección 4.1. El proceso será el siguiente:

- Para cada palabra nueva de nuestro lenguaje:
    - **1º** pasar la palabra por la aplicación *eutranscribe* (sección 3.4), para obtener su correcta transcripción de acuerdo a nuestro modelo acústico.
    - **2º** crear el autómata correspondiente a esa transcripción siguiendo la estructura correspondiente, dejando la palabra original como identificador.

Hay que señalar el caso especial de los números. Tenemos que añadirlos del uno al noventa y nueve. Aunque un número se suele pronunciar seguido, vamos a añadir al léxico las palabras que forman los números, de modo que no haya que introducir los noventa y nueve.

4.1.1. CREACIÓN DE LOS MODELOS DEL SISTEMA

La lista de palabras añadidas para los números es la siguiente:

uno	seis	once	i	cincuenta
dos	siete	doce	veinte	sesenta
tres	ocho	trece	veinti	setenta
cuatro	nueve	catorce	treinta	ochenta
cinco	diez	quince	cuarenta	noventa

De este modo se podrán componer fácilmente todos los números. Por ejemplo:

10 = diez
16 = diez i seis
20 = veinte
25 = veinti cinco
46 = cuarenta i seis

Por último, añadir que además de todas las palabras mencionadas, para el correcto funcionamiento de la aplicación añadiremos las siguientes, cuya único símbolo de sonido en el modelo acústico será “S”, que representa el silencio: </s>, <unk>, <, >, <BACKOFF>, <s> y <UNK>.

Vamos a ver un pequeño fragmento del fichero con el modelo léxico:

Name "<s>"
State 0 i=1.0
0 1 "S" p=1.0
State 1 f=1.0

Name "abrir"
State 0 i=1
0 1 "a" p=1
1 2 "b" p=1
2 3 "r" p=1
3 4 "i" p=1
4 5 "r" p=1
State 5 f=1

Name "adelante"
State 0 i=1
0 1 "a" p=1
1 2 "d" p=1
2 3 "e" p=1
3 4 "l" p=1
4 5 "a" p=1
5 6 "n" p=1
6 7 "t" p=1
7 8 "e" p=1
State 8 f=1

Modelo del lenguaje

Ahora tendremos que crear un fichero con un modelo que incluya todas las frases de nuestro lenguaje. Las frases serán las que forman las órdenes de la sección 4.1 y los número del uno al noventa y nueve.

Siguiendo la estructura definida en el apartado 3.1.4, el proceso será el siguiente:

- Definir la cabecera del modelo, con el número de estados y transiciones totales del autómata.
- Crear un estado “0” cuya única transición sea un silencio etiquetado como “<s>”, que vaya al estado “1” donde comenzarán todas las frases.
- Para cada frase:
    - Añadir las palabras que la forman al autómata, creando un estado nuevo para cada una de sus palabras, evitando repeticiones en las transiciones y los estados (fusionando los idénticos). La última palabra de cada frase deberá transitar al estado final del modelo.
- Por último, para cada estado del autómata asegurar una distribución equitativa de la probabilidad de sus transiciones.

Aquí tenemos una pequeña muestra de cómo quedaría:

```
Name   kk

NumStates      25

NumEdges       125

TotalChecksum   1

State 0 c = 1 i = 1
  0     1    "<s>" p = 1

State 1 c = 1
  1     2    "abrir"          p = 0.03125
  1     12   "adelante"        p = 0.03125
  1     18   "anterior"        p = 0.03125
  1     12   "ampliar"         p = 0.03125
  1     15   "analizar"        p = 0.03125
  1     3    "añadir"          p = 0.03125
  1     12   "atras"           p = 0.03125
  1     12   "ayuda"           p = 0.03125
          ...
          ...
  1     17   "resetear"        p = 0.03125
  1     6    "siguiente"       p = 0.03125
  1     14   "subir"           p = 0.03125
  1     14   "sube"            p = 0.03125
  1     12   "terminar"        p = 0.03125
  1     10   "ventana"         p = 0.03125

State 2 c = 1
  2     11   "segundo"         p = 1
```

4.1.2. Ficheros de configuración

Recordemos que a iATROS hay que pasarle un fichero de configuración del sistema (apartado 3.1.2) y otro de configuración de la señal de audio (apartado 3.1.3) para el funcionamiento en modo *online*. El fichero de configuración de la señal de audio será el que se comenta en su correspondiente sección, mientras que el fichero de configuración del sistema será del que dependa la eficiencia de nuestra aplicación, y puede variar. Posee cuatro parámetros que podemos modificar: *beam, grammar-scale-factor, Word-insertion-penalty* y *histogram-pruning.*

En el capítulo 5 realizaremos un proceso de experimentación variando los parámetros anteriores, con el objetivo de encontrar la configuración óptima. De momento el fichero de configuración del sistema que usaremos será el siguiente:

```
samples = ./lista_comandos_leidos

[decoder]
hmm = ./modelos/albayzin_iatros_64gs.hmm
lexicon = ./modelos/lexico_firefox.lx
grammar = ./modelos/automata_firefox.gr
grammar-type = FSM
beam = 65
grammar-scale-factor = 10
word-insertion-penalty = -9
histogram-pruning = 5000
```

4.1.3. Ejecución

Una vez se haya instalado la aplicación iATROS siguiendo su correspondiente documentación y se hayan generado los ficheros de los modelos y sus ficheros de configuración, la ejecución de su reconocedor automático del habla en modo *online* es del siguiente modo:

iatros-speech-online -p F_CONFIGURACIÓN_AUDIO -c F_CONFIGURACIÓN_SISTEMA

Cuyos parámetros de ejecución son los siguientes:

- -p para especificar el fichero de configuración de la señal de audio a grabar.
- -f para especificar el fichero de configuración del sistema.

Una vez lanzada su ejecución, en la consola vemos lo siguiente:

Calculating noise...
Noise:180.534249
You can talk now...

Esas tres líneas repetirán para cada orden a reconocer. Vamos a explicarlas con un poco de detalle:

- *Calculating noise... :* Indica que el sistema está calculando la potencia presente en la señal

de audio, cuyo valor lo usará para determinar el umbral a partir del cual se puede considerar que estamos hablando, y a partir del cual es silencio. Para que esto funcione correctamente, se deberá estar en silencio durante el proceso de cálculo.

- *Noise:180.534249 :* Nos da la lectura de la intensidad del ruido que ha captado en "silencio" por el micrófono. En este caso la lectura ha sido de 180.
- *You can talk now...* ; El programa nos avisa de que ya podemos comenzar a hablar. A partir de ese momento todo lo que se diga será descifrado por el reconocedor con el objetivo de devolver una hipótesis acertada del audio pronunciado.

Vamos a probar a accionar el reconocedor y darle la orden "ventana siguiente":

Calculating noise...
Noise:30.028530
You can talk now...
Start Recording...
tim 0.000000
-112896.601562 <s> ventana siguiente
Stop Recording...
End adquisition thread
End process thread
Calculating noise...
Noise:30.611119
You can talk now...

Vemos marcado en negrita como ha reaccionado el programa cuando hemos pronunciado la orden "ventana siguiente", que además ha descifrado correctamente. Veamos qué quieren decir esos mensajes:

- *Start Recording...* : El programa avisa de que comienza el proceso de grabación, pues el sonido captado ha superado el umbral de ruido.
- *Tim 0.000000* : Es el tiempo de proceso. En este caso ha sido 0.
- *-112896.601562 <s> ventana siguiente*: Vemos la puntuación de la mejor hipótesis en el algoritmo de búsqueda, seguido de su valor "<s> ventana siguiente". La puntuación o *score* será de más probabilidad cuanto más cercano a 0 sea.
- *Stop Recording...*
  End adquisition thread
  End process thread : El programa avisa de que el proceso de grabación ha terminado, y procede a borrar el hilo de adquisición y de proceso que había creado para la grabación. Dichos hilos se crean con el objetivo de que el programa pueda trabajar paralelamente decodificando la señal de audio, de modo que el modo *online* resulte más eficiente.

Al final del reconocimiento vemos cómo vuelve a calcular el silencio umbral y se prepara para captar otro mensaje. El proceso sería cíclico hasta que se termine el programa.

4.3. Diseño del intérprete

El intérprete será el encargado de, una vez conocida la orden a ejecutar, realizar los preparativos necesarios y enviársela al navegador. Para su desarrollo utilizaremos el lenguaje bash.

Su estructura básica será la siguiente:

```
bucle {
        orden_nueva = esperar_nueva_orden()

        case orden_nueva {
                orden_1:
                        ejecutar_orden_1...
                orden_2:
                        ejecutar_orden_2...
                …
                orden_n:
                        ejecutar_orden_n...
        }
}
```

Según el proceso de ejecución de las órdenes, podemos distinguir dos tipos:

- **Órdenes normales**: Lo único que requiere para su ejecución es enviarle al navegador los eventos de teclado asociados a su orden.
- **Órdenes especiales**: Son órdenes que requieren unos preparativos previos para su ejecución. Pueden no estar relacionadas con el navegador, siendo para el control de la aplicación de navegación web con la voz.

4.3.1. Diseño de órdenes normales

Las órdenes normales serán las más sencillas de ejecutar. Lo único que habrá que hacer es utilizar la aplicación Xmacro (sección 3.3) para enviar los eventos de teclado asociados (apartado 3.2.1) a las órdenes del navegador Mozilla Firefox.

La lista de órdenes normales será la siguiente:

abrir en segundo plano	ampliar	añadir marcador
atrás	adelante	ayuda
bajar / baja	subir / sube	baja uno / baja dos / baja tres
sube uno / sube dos / sube tres	cierra pestaña	código fuente
enlace anterior / anterior enlace	enlace siguiente / siguiente enlace	entrar / enter
escape	historial	marcadores
nueva pestaña	nueva ventana	página inicio

pantalla completa	recargar	reducir
siguiente pestaña / pestaña siguiente	siguiente ventana / ventana siguiente	ventana anterior / anterior ventana

A continuación vamos a ver unos ejemplos sobre como implementar algunas órdenes:

Orden	**Teclas de acceso rápido**	**Ejecución**
"ventana siguiente"	*Alt + Tab*	*echo "KeyStrPress Alt_L* *KeyStr Tab* *KeyStrRelease Alt_L" | xmacroplay :0*
"baja tres"	*Flecha abajo* *Flecha abajo* *Flecha abajo*	*echo "KeyStr Down* *KeyStr Down* *KeyStr Down" | xmacroplay :0*
"pantalla completa"	*F11*	*echo "KeyStr F11" | xmacroplay :0*

Hay que señalar que existen órdenes como "baja tres", "bajar" y "sube", que requieren de más de un evento de teclado, pero como hemos visto en el ejemplo anterior, su ejecución es igualmente sencilla.

4.3.2. Agregar texto de los enlaces a los modelos

Anteriormente habíamos comentado que una de las capacidades de nuestra aplicación será la de poder analizar una página web de manera que extraiga los enlaces y el texto asociado, para poder agregarlos a nuestros modelos del sistema y poder abrirlos mediante el habla con la orden "abrir enlace + "*texto_enlace*"". Dentro de este ámbito tenemos varias órdenes:

analizar web	recargar modelos	resetear modelos
abrir enlace + " texto enlace"	mostrar lista enlaces	

Dada la importancia de cada una de las órdenes, vamos a tratarlas por separado.

Analizar web

El objetivo de la orden es agregar todo el texto asociado a los enlaces de la web actual a nuestros modelos del sistema. Esto implica varios problemas, entre ellos conocer desde fuera del navegador Firefox en qué página web nos encontramos. Además tendremos que tener cuidado con el juego de caracteres del código de la página web, y también de los caracteres que figuran en el texto de los enlaces, como puedan ser los números. Por último, habrá que rehacer los ficheros de los modelos, no de forma manual como habíamos hecho hasta ahora, sino de forma automática mediante una aplicación.

Para implementar la orden ha sido necesario desarrollar una serie de aplicaciones, algunas en el lenguaje C++ y otras simplemente como scripts en bash. A continuación hablaremos de ellas.

4.3.2. AGREGAR TEXTO DE LOS ENLACES A LOS MODELOS

La lista completa de aplicaciones desarrolladas para la orden “analizar web” es la siguiente:

Nombre	Descripción
ultima_url	En el apartado 3.2.2 hablamos del fichero en el que el navegador Firefox almacena la información sobre la sesión actual: *sessionstore.js*. El programa *ultima_url* es una aplicación desarrollada en C++ que recibe como entrada el fichero anterior y, conociendo su estructura, busca y devuelve la dirección de la página web que tenemos enfrente en nuestro navegador.
w_extrae_enlaces	La aplicación recibe como entrada el código fuente de una página web y, conociendo la estructura del HTML(apartado 2.2.1), nos devuelve dos ficheros: la lista de enlaces de la página, y la lista de texto asociado a los enlaces de la página. Está desarrollada en C++, y la aplicación se limita a buscar las secciones del fichero con la cadena “href=” para tomar la dirección del enlace y, a continuación, buscar el texto asociado.
filtrar_caracteres	Es un script desarrollado en bash, que toma como entrada la lista de texto asociado a los enlaces y utiliza la aplicación de Linux *sed* para realizar una serie de tareas de limpieza sobre el texto: elimina números, signos de puntuación, espacios múltiples... Además, pasa cada una de las palabras del fichero por la aplicación *eutranscribe* (apartado 3.4), para que el texto esté listo para agregarse a nuestro modelo léxico.
agregar_lexico	La aplicación toma como entrada el fichero con el texto filtrado asociado a los enlaces y, también el modelo léxico original de nuestro sistema. En primer lugar almacena en una estructura de datos el modelo original y, a continuación, extrae cada una de las palabras del fichero con el texto de los enlaces para agregarlo también al modelo léxico. Como resultado, tenemos un nuevo modelo léxico que incluye el viejo vocabulario, además de todo el nuevo presente en el texto de los enlaces de la página web actual. La aplicación está desarrollada en C++.
rehacer_automata	Es un programa desarrollado en C++ que toma como entrada el modelo del lenguaje original de nuestro sistema, además del fichero con el texto filtrado asociado a los enlaces. En primer lugar lee el modelo original en una estructura de datos y, a continuación, lee las frases asociadas a cada enlace para agregarlas al nuevo modelo del lenguaje, poniendo como prefijo a cada frase: *abrir enlace*. Finalmente tenemos el nuevo modelo del lenguaje con todos los textos asociados a los enlaces de la web actual.

web_enlaces_html	Es un sencillo programa en C++ que genera una página web en HTML que posee una lista con todos los enlaces leídos de la página web actual, numerados y seguidos del texto necesario a pronunciar para acceder al enlace. La página creada será de utilidad para páginas web con muchos enlaces, o en el caso de que el texto original del enlace haya cambiado mucho al ser filtrado con la aplicación *filtrar_caracteres*. Posteriormente hablaremos de la orden "mostrar lista enlaces", que nos muestra la página que acabamos de crear.
recarga_modelos	Es un script desarrollado en bash que se encarga de cambiar los modelos del sistema viejos por los nuevos que acabamos de crear. Para ello cierra nuestra aplicación de navegación web con la voz, sus programas asociados, sustituye los viejos modelos por los nuevos y, finalmente, arranca de nuevo nuestra aplicación.

Una vez conocemos todas las aplicaciones creadas, vamos a resumir el proceso que sigue la orden "analizar web":

- 1º Recuperar la dirección de la página web actual con el programa *ultima_url*.
- 2º Utilizar la herramienta wget de Linux para descargar la página web.
- 3º Convertir el juego de caracteres a UTF-8 para evitar problemas.
- 4º Utilizar la herramienta *w_extrae_enlaces* para extraer todos los enlaces y textos asociados de la página web.
- 5º Filtrar todos los caracteres indeseados y adaptar las palabras para nuestro modelo léxico con la aplicación *filtrar_caracteres*.
- 6º Crear el nuevo modelo léxico que incluya las palabras del texto asociado a los enlaces, mediante la aplicación *agregar_lexico*.
- 7º Crear el nuevo modelo del lenguaje que incluya todas las frases asociadas a los enlaces, mediante la aplicación *rehacer_automata*.
- 8º Crear una página web en HTML que sirva como lista de enlaces y textos a pronunciar para su apertura. Usaremos la aplicación *web_enlaces_html*.
- 9º Cambia los modelos del sistema viejos por los nuevos con la aplicación *recarga_modelos*.

Recarga modelos

La función de esta orden es muy sencilla. Lanza el programa *recarga_modelos* que hemos comentado en esta misma página. Su funcionamiento no es de utilidad para el usuario, pues los modelos se cambian solos al analizar una web, pero está implementado para realizar pruebas.

Resetear modelos

La orden se encarga de llamar a un script desarrollado en bash, que restaura los modelos del sistema originales y reinicia la aplicación de reconocimiento de voz. Además de para realizar pruebas, podría resultar de utilidad en el caso de que hubiéramos analizado una página con demasiados enlaces y que estuviera aumentado mucho el error de nuestro reconocedor.

4.3.2. AGREGAR TEXTO DE LOS ENLACES A LOS MODELOS

Abrir enlace + “ texto enlace”

Una vez hemos analizado la página web, para poder abrir los enlaces pronunciado el texto asociado será necesaria esta orden.

El funcionamiento en el intérprete es el siguiente:

- Si estamos ante una orden “abrir enlace”:
    - Buscar en la lista de textos asociados a enlaces la cadena pronunciada.
    - Buscar el enlace asociado en la lista de enlaces.
    - Enviarle la dirección al navegador. Para ello accionamos el evento asociado a situarnos en la barra de direcciones y, a continuación, los eventos de teclado necesarios para escribir la dirección de la web:

        echo "KeyStr F6" | xmacroplay :0
        sleep 1
        echo "String "$dirección_web | xmacroplay :0
        echo "KeyStr Return" | xmacroplay :0

Anteriormente a la implementación de esta orden navegar entre páginas era complicado pues para acceder de una a otra teníamos que desplazarnos enlace por enlace y dar la orden “entrar“ en el momento adecuado. Ahora la navegación será mucho más cómoda, ya que si en una página web tenemos un enlace “noticias”, no habrá más que pronunciar la orden “abrir enlace noticias” y ya estaremos dentro.

Mostrar lista enlaces

Como hemos comentado anteriormente, al extraer el texto asociado a los enlaces se tiene que realizar un proceso de filtrado. En muchas ocasiones puede producir grandes cambios en el texto, de manera que pronunciar el texto original podría fallar. Para ello se crea esta orden, que muestra una sencilla página web en HTML con una lista de todos los enlaces leídos, con su texto original y el necesario a pronunciar para su apertura. Además, los enlaces están numerados para que sea compatible con la orden “enlace + “*número*”” y “enlace anterior + “*número*””, aumentando todavía más la comodidad de navegación.

4.3.3. Diseño de otras órdenes especiales

Además de todas las órdenes del apartado anterior, existen otras que también podríamos considerar “especiales” por su complejidad. Algunas son para el control de nuestra aplicación y otras para el navegador.

Las órdenes son las siguientes:

pausar	terminar
enlace + “número”	enlace anterior + “número”

A continuación vamos a hablar de cada una de ellas por separado.

Pausar

Al ejecutar nuestra aplicación no siempre tenemos por qué querer que esté escuchando y ejecutando lo que decimos, cosa que daría problemas si hablamos hacia otras fuentes. La orden "pausar" nos da la solución a ese problema, ya que, una vez pronunciada, nuestra aplicación no ejecutará ninguna otra orden hasta que volvamos a decir "pausar".

El funcionamiento es sencillo. Se agrega una sentencia de control antes de la lista de órdenes en el bucle principal del intérprete. Si estamos en pausa, tendremos una variable que así lo indique, y por tanto la orden recibida de entrada será borrada. Esta misma sentencia de control servirá para modificar la variable de pausa una vez se vuelva a pronunciar.

Hay que señalar que durante la pausa, puede ser necesario reiniciar nuestra aplicación de forma automática. Esto es debido a que aunque el intérprete no ejecute órdenes, el reconocedor sigue trabajando, y puede ocurrir que estemos hablando continuamente, mientras el realiza un test de silencio. De esta manera consideraría como umbral del silencio un valor muy elevado, correspondiente a la intensidad del ruido de nuestra voz. Para que al reiniciarse la aplicación sigamos en pausa será necesario almacenar el estado pausa en un fichero externo a la aplicación, de manera que el intérprete lo compruebe al arrancar. En la sección 4.5 hablaremos de la aplicación encargada de controlar la intensidad del "silencio" y reiniciar nuestra aplicación si es necesario.

Terminar

La orden se encarga de llamar a un script en bash que cierra nuestra aplicación y todos los procesos abiertos asociados, de manera que terminamos completamente su ejecución. Además, elimina todos los archivos temporales y restaura los modelos del sistema originales para que la sesión esté limpia en su próxima ejecución.

Enlace + "número"

Mediante esta orden, saltamos un "número" de enlaces hacia adelante en la página web. La complejidad reside en pasar de un número en texto a su expresión numérica. Para ello hemos creado la aplicación *dame_num*, que simplemente lo busca en una lista y nos lo devuelve. A continuación solo queda mandar al navegador, el número de veces que sea necesario, el evento asociado a pasar al siguiente enlace.

Enlace anterior + "número"

La orden funciona exactamente igual que la anterior, con la única diferencia de que ahora el evento de teclado enviado se corresponde con el de pasar al enlace anterior en lugar de al siguiente.

4.4. Comunicación entre módulos

Vamos a hablar de cómo hemos implementado la comunicación entre los diferentes módulos de nuestro programa. Para ello, en primer lugar vamos a describir uno de los métodos utilizados: las tuberías con nombre.

- Una tubería con nombre es creada explícitamente por una orden del sistema operativo y

persiste a posteriori de la finalización del proceso (a diferencia de las tuberías normales); además debe, ser borrada una vez que no va a seguir siendo utilizada. También resulta útil en nuestro caso para mandar mensajes entre programas que han sido ejecutados de forma independiente.

Ahora vamos a describir la comunicación entre los módulos de nuestro sistema:

- **Reconocedor → Intérprete**: Para este caso hemos utilizado las tuberías con nombre. Hemos modificado el código fuente del reconocedor en modo *online* del iATROS, de manera que escribe cada nueva hipótesis en una tubería que ha creado nuestro intérprete (llamada *comando_entrada*). De este modo nuestro intérprete, en cada iteración de su bucle principal, espera hasta que llega una nueva orden.

- **Intérprete → Navegador**: Como ya se dijo en la sección 3.3, todas las órdenes del intérprete al navegador pasan a través de la aplicación Xmacro, que envía eventos de teclado y ratón a un servidor X Windows. Existe un problema con esta opción, y es que realmente no estamos enviando las órdenes al navegador, sino a todo el entorno, de manera que podría influir en otros programas si no estuviéramos con el navegador en primer plano.

Podemos ver qué líneas han sido necesarias en iATROS para escribir en la tubería:

```
FILE *mipipe; //creamos un fichero para la tubería
mipipe=fopen("comando_entrada", "w"); // Lo abrimos como un fichero normal y corriente
// Con la siguiente línea escribimos en la tubería
fputs(sentence_str+4, mipipe);          //+4 para saltarse de la cadena el "<s> " , sentence_str
                                        // es la hipótesis del reconocedor
fclose(mipipe);       // Cerramos la tubería por este lado
```

4.5. Otros módulos

Por último vamos a describir otros dos scripts externos al reconocedor y al intérprete: *c_ruido* y *control_voz.*

c_ruido

El script está programado en bash y se ejecuta de forma paralela al resto, con el objetivo de controlar que el silencio umbral del reconocedor esté por debajo de cierto límite; en caso contrario reinicia la aplicación.

Para su desarrollo se han creado dos aplicaciones adicionales:

- *ruido_alto*: Es un programa desarrollado en C++, que nace por la incapacidad del lenguaje bash para trabajar con números en coma flotante. De ahí que para comparar si nuestro ruido (leído en decimales) es superior a cierto límite, hayamos tenido que crear una sencilla aplicación que realice la tarea.
- *resetea_aplicación*: Es un script en bash que cierra nuestra aplicación y sus procesos asociados y la vuelve a abrir, sin restaurar los modelos del sistema originales. La aplicación se queda igual que antes, pero con la ventaja de que el reconocedor vuelve a medir el “silencio”.

El funcionamiento de la aplicación *c_ruido* es el siguiente:

```
bucle {
        RUIDO_UMBRAL=  extraer_ultima_medición(LOG_ATROS)
        REINICIAR = ruido_alto RUIDO_UMBRAL RUIDO_LÍMITE
        si REINICIAR entonces
                resetea_aplicacion
        dormir 3 segundos
}
```

Donde:

- *extraer_ultima_medición()* es solo una combinación de los programas de Linux *cat*, *grep* y *tail.*
- LOG_ATROS es un fichero con la salida de toda la sesión de iATROS hasta el momento.
- RUIDO_LÍMITE es la variable que fija el máximo de la intensidad del ruido permitido (cuyo valor por defecto es 80).
- dormir 3 segundos es para evitar que esté consumiendo CPU continuamente.

La aplicación resulta muy útil en caso de activar la orden “pausar”, ya que el programa controlaría que nuestra habla no fijase un valor umbral de “silencio” demasiado elevado, además de otros casos en que nuestra aplicación haya podido captar algún ruido indeseado durante el test de “silencio”.

control_voz

Es un sencillo script en bash que se encarga del lanzamiento de la aplicación. Ejecuta el reconocedor, el intérprete y el programa *c_ruido*, los tres en *background*. El lanzamiento del navegador lo dejamos para el usuario, ya que puede tenerlo abierto con anterioridad.

Con esto ya tenemos una aplicación de navegación web con la voz completamente funcional, a falta solo de realizar las pruebas de configuración de parámetros en el siguiente capítulo.

Capítulo 5

Experimentación

En este capítulo trataremos el proceso de experimentación que ha sido necesario para encontrar la configuración de parámetros óptima de la herramienta iATROS. El proceso seguido ha sido el siguiente:

- **Diseño del experimento**: En primer lugar explicaremos en qué van a consistir los experimentos realizados. Hablaremos de cómo han sido diseñados, los parámetros a variar y del rango de valores que podrán tomar.

- **Obtención del corpus de test**: Para las pruebas será necesario tener muestras de audio con órdenes grabadas por diferentes locutores.

- **Fases de pruebas**: Realizaremos las pruebas sobre nuestro sistema aplicando a las diferentes configuraciones el corpus del test.

- **Análisis de los resultados**: Finalmente, analizaremos los resultados obtenidos en las pruebas anteriores.

5.1. Diseño del experimento

En el apartado 3.1.2. hablamos acerca de los parámetros de configuración de iATROS. En esta sección vamos a diseñar el esquema que seguiremos en la sección 5.2 para las diferentes pruebas.

Los parámetros que vamos a variar son los siguientes:

Parámetro	Rango de valores
beam	[50,1000]
grammar-scale-factor	[0,50]
word-insertion-penalty	[-10,10]
histogram-pruning	[5.000,15.000]

Para las pruebas utilizaremos las herramientas de reconocimiento *offline* de iATROS. Su archivo de configuración contendrá una lista de muestras a decodificar. Cada una de las muestras será un vector de coeficientes cepstrales, extraídos a partir de una grabación en formato .wav. La herramienta nos dará su hipótesis más probable para cada una de las muestras.

Veamos un ejemplo de cómo realizar una prueba:

- 1º Grabamos la orden “adelante” en un archivo ORDEN.wav.

- 2º Convertimos el fichero anterior en su vector de coeficientes cepstrales utilizando la herramienta de iATROS *iatros-speech-cepstral* , como vemos a continuación:
  iatros-speech-cepstral -c conf.feat -i ORDEN.wav -o ORDEN.CC
  Donde -c es para pasarle el archivo de configuración de la señal de audio (apartado 3.1.3), -i para indicarle el archivo de entrada, y -o para indicarle el de salida.

- 3º Testeamos el fichero con el reconocedor *offline* de iATROS, el *iatros-offline*:
  iatros-offline -c ./configuracion.cnf
  Con -c le pasamos a la aplicación el archivo de configuración del sistema (apartado 3.1.2), el cual contendrá en su lista de archivos a decodificar, el archivo ORDEN.CC. En el archivo de configuración será donde pondremos el valor de los parámetros a iterar en la investigación.

- 5º Por último solo queda observar y analizar la salida del programa:
  Start decoding...
  <s> adelante
  Vemos cómo comienza la decodificación y nos da la hipótesis. En este caso ha sido correcta.

Una vez visto el ejemplo anterior, es muy sencillo explicar cómo realizaremos los experimentos:

- 1º Obtendremos todas las muestras del corpus de test y las pasaremos a archivos de coeficientes cepstrales con un script en bash diseñado para ello: *crea_cepstrales.sh.*

- 2º Creamos un script principal *prueba_confs.sh* cuya tarea es variar los cuatro parámetros de configuración y almacenar cada nueva configuración en un fichero que le pasaremos a los scripts del punto siguiente.

- 3º Crearemos un script *testea_comandos.sh* que realice de forma automática las tareas de los pasos 2-5 del ejemplo anterior para todos las órdenes del corpus. Su salida será el porcentaje de aciertos de órdenes total para esa configuración.

Hay que señalar que realmente los scripts anteriores serán fraccionados en hasta 8 partes, de manera que los lanzaremos de forma simultánea para aprovechar toda la CPU. Cada uno probará un rango de configuraciones y, posteriormente, juntaremos todos los resultados en un solo fichero para analizarlos mejor.

El sistema utilizado en las pruebas ha sido el siguiente:

Intel Core i7-920 @ 4 Ghz
Memoria RAM 6 GB
Distribución Ubuntu 10.04 LTS x64

5.2. El corpus de test

El corpus de test está formado por todas las muestras que vamos a testear en nuestra aplicación. Todas las órdenes han sido grabadas con cuatro locutores diferentes para asegurar el desempeño óptimo de la configuración para cualquier locutor.

A pesar del reducido número de locutores, se ha intentado tener la máxima variabilidad entre las voces:

Locutor	**Sexo**	**Edad**	**Nacionalidad**
Locutor_1	Hombre	23	Española
Locutor_2	Mujer	23	Rusa
Locutor_3	Hombre	70	Española
Locutor_4	Mujer	50	Española

Para cada locutor hemos grabado cada una de las órdenes disponibles en la lista de la sección 4.1. Hay que señalar que no se ha incluido la orden "abrir enlace" + "*texto enlace*"", por su carácter dependiente del análisis previo de la web. Además, para el caso de las órdenes "enlace + "*número*"" y "enlace anterior + "*número*"", no se han probado todos los número del uno al noventa y nueve, sino algunos de ellos, asegurando la variabilidad:

Enlace cinco	*Enlace ocho*	*Enlace anterior seis*
Enlace cincuenta	*Enlace sesenta y seis*	*Enlace anterior trece*
Enlace cincuenta y nueve	*Enlace setenta y uno*	*Enlace anterior treinta*
Enlace noventa y cuatro	*Enlace anterior cuarenta*	*Enlace treinta y tres*
Enlace ochenta	*Enlace anterior cuarenta y siete*	*Enlace anterior veintisiete*
Enlace ochenta y siete	*Enlace anterior diez y nueve*	

5.3. Experimentación

Una vez ha sido diseñado el experimento y el corpus ya podemos pasar a realizar las pruebas con las diferentes configuraciones. El proceso de experimentación ha sido dividido en tres fases de experimentos diferentes:

- **Primera fase**: En primer lugar realizamos las pruebas variando los valores de la configuración a intervalos en todo su rango, sobre un conjunto de órdenes reducido del corpus.
- **Segunda fase**: Probamos el corpus completo sobre las mismas configuraciones que en la primera fase.
- **Tercera fase**: Finalmente, para encontrar la configuración óptima, afinaremos y profundizaremos en el rango de valores de los parámetros.

5.3.1. Primera fase

En primer lugar hay que señalar que esta fase de pruebas fue realizada antes de completar la aplicación, en una etapa inicial, cuando aun no estaban todas las órdenes. Fue necesaria para poder trabajar correctamente a lo largo de todo su desarrollo. En cualquier caso, ya había un número de órdenes implementadas elevado, en concreto treinta. Las órdenes testeadas fueron las siguientes:

Abrir segundo plano	*Código fuente*	*Página inicio*
Ampliar	*Enlace anterior*	*Pantalla completa*
Añadir marcador	*Enlace siguiente*	*Recargar*
Atrás	*Enter*	*Reducir*
Adelante	*Escape*	*Pestaña siguiente*
Ayuda	*Historial*	*Ventana anterior*
Bajar	*Marcadores*	*Ventana siguiente*
Baja uno / baja dos / baja tres	*Nueva pestaña*	*Sube uno / sube dos / sube tres*
Cierra pestaña	*Nueva ventana*	

Los valores de los parámetros han sido los siguientes:

Parámetro	**Valores**
beam	{100, 200, 400, 600, 800, 1000}
Grammar-scale-factor (GSF)	{0, 10, 20, 30, 40, 50}
Word-insertion-penalty (WIP)	{-10, -5. 0, 5, 10}
histogram-pruning	{5.000, 7.000, 9.000, 11.000, 13.000, 15.000}

Adelantamos que, en todas las pruebas, no hubo variabilidad en los resultados para el parámetro *histogram-pruning*, así que lo fijamos en 5.000 como valor óptimo y lo obviamos de los resultados para el resto de fases. Si no ha habido variabilidad ha sido porque 5.000 es un valor superior al máximo tamaño que tendrá nuestro *heap* en el proceso de búsqueda, pero no intentamos reducirlo porque hay que recordar que los modelos del sistema se amplían con los enlaces de las páginas (apartado 4.3.2), y no queremos que un valor reducido interfiera con los resultados en ese caso.

5.3.1. PRIMERA FASE

Podemos ver el resumen de los porcentajes de aciertos por configuración en la tabla siguiente:

			BEAM					
		WIP	**100**	**200**	**400**	**600**	**800**	**1000**
GSF	**0**	**-10**	95,8	79,1	67,7	59,3	57,2	57,2
		-5	**96,8**	79,1	67,7	59,3	57,2	57,2
		0	95,8	80,2	67,7	59,3	57,2	57,2
		5	95,8	81,2	66,6	59,3	57,2	57,2
		10	95,8	81,2	66,6	59,3	57,2	57,2
	10	**-10**	94,7	80,2	67,7	59,3	57,2	57,2
		-5	95,8	81,2	66,6	59,3	57,2	57,2
		0	95,8	81,2	66,6	59,3	57,2	57,2
		5	95,8	81,2	66,6	59,3	57,2	57,2
		10	94,7	81,2	66,6	59,3	65,2	57,2
	20	**-10**	94,7	81,2	66,6	58,3	55,2	56,2
		-5	95,8	81,2	65,6	58,3	55,2	56,2
		0	95,8	81,2	65,6	57,2	55,2	56,2
		5	95,8	81,2	64,5	57,2	55,2	55,2
		10	95,8	81,2	64,5	56,2	55,2	55,2
	30	**-10**	92,7	82,2	65,6	57,2	56,2	57,2
		-5	87,5	81,2	65,6	56,2	56,2	57,2
		0	80,2	81,2	65,6	56,2	56,2	57,2
		5	73,9	81,2	64,5	56,2	56,2	56,2
		10	5,2	81,2	64,5	56,2	56,2	56,2
	40	**-10**	0,0	81,2	63,5	55,2	55,2	56,2
		-5	0,0	81,2	63,5	54,1	55,2	56,2
		0	0,0	81,2	63,5	54,1	55,2	56,2
		5	0,0	81,2	63,5	54,1	55,2	55,2
		10	0,0	81,2	60,5	53,1	54,2	54,1
	50	**-10**	0,0	81,2	60,4	53,1	55,2	56,2
		-5	0,0	81,2	60,4	53,1	55,2	56,2
		0	0,0	81,2	58,3	52,0	54,1	55,2
		5	0,0	81,2	58,3	52,0	54,1	54,1
		10	0,0	81,2	58,3	52,0	54,1	54,1

En la tabla anterior podemos ver, que en general, cuanto más aumentamos el valor de los parámetros, más aumenta el porcentaje de fallos. Nuestra mejor configuración ha sido para {BEAM=100, GSF=0,WIP=-5} con un porcentaje de aciertos del **96,8%**.

5.3.2. Segunda fase

La segunda fase, al igual que la tercera, se llevó a cabo una vez se finalizó la aplicación completamente. Hay que tener en cuenta que ahora testearemos todas las órdenes del corpus, cosa que puede afectar drásticamente a los resultados.

Las pruebas han sido las mismas que en la fase anterior y podemos ver el porcentaje de aciertos:

			BEAM					
		WIP	**100**	**200**	**400**	**600**	**800**	**1000**
GSF	**0**	**-10**	80,9	63,9	41,5	21,6	11,8	11,4
		-5	**81,3**	64,4	41,5	21,6	11,8	11,4
		0	80,0	64,4	41,5	22,0	11,8	11,4
		5	80,0	64,8	41,1	22,0	11,8	11,4
		10	80,5	64,8	41,9	22,4	11,8	11,4
	10	**-10**	80,5	64,8	41,9	24,1	12,2	11,4
		-5	80,5	64,8	42,3	24,1	12,2	11,4
		0	80,5	64,4	42,7	24,1	12,7	11,4
		5	80,5	65,2	43,2	25,0	12,7	11,4
		10	80,5	66,1	43,2	25,4	12,7	11,4
	20	**-10**	80,5	66,9	44,9	27,1	13,5	11,4
		-5	80,5	66,5	44,0	26,6	14,4	11,0
		0	80,0	66,9	44,4	26,6	13,9	11,0
		5	80,5	68,2	44,0	26,6	14,4	11,0
		10	79,6	67,7	44,9	26,6	14,4	11,0
	30	**-10**	69,9	66,5	46,1	27,5	15,6	11,0
		-5	42,7	66,9	46,6	28,3	15,6	11,0
		0	3,3	67,7	47,0	28,8	15,6	11,0
		5	0,4	68,2	47,4	28,8	15,6	11,0
		10	0,0	68,2	47,8	28,8	15,6	11,0
	40	**-10**	0,0	67,3	48,7	30,0	16,1	11,4
		-5	0,0	66,9	47,8	30,0	16,5	11,4
		0	0,0	67,3	47,8	28,6	16,5	11,4
		5	0,0	67,7	47,8	30,0	16,5	11,4
		10	0,0	67,3	48,3	30,0	16,5	11,4
	50	**-10**	0,0	66,1	47,0	31,7	17,3	11,4
		-5	0,0	66,1	46,6	31,7	17,3	11,4
		0	0,0	66,1	46,1	31,3	17,3	11,4
		5	0,0	75,6	46,1	31,3	17,3	11,4
		10	0,0	65,2	46,1	30,5	18,2	11,4

5.3.2. SEGUNDA FASE

Como podemos ver, el porcentaje de aciertos general ha disminuido hasta un **81,3%**, y la configuración óptima se mantiene en {BEAM=100, GSF=0,WIP=-5}.

En vista de que el porcentaje de aciertos más alto se encuentra en el valor de BEAM más bajo, sería interesante probar con valores inferiores para ver si los resultados mejoran, que será uno de los objetivos de la tercera fase.

5.3.3. Tercera fase

En este fase vamos a intentar mejorar el porcentaje de aciertos de nuestra aplicación al máximo. Para ello vamos a aumentar los valores a intervalos más reducidos, y además vamos a probar un rango de *beam* que comprenda valores más bajos, ya que como hemos comentado en el apartado 5.3.2, nuestra mejor configuración estaba en su límite inferior.

La lista de valores que tomarán los parámetros es la siguiente:

Parámetro	**Valores**
beam	{25, 50, 75, 100, 125, 150}
Grammar-scale-factor (GSF)	{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
Word-insertion-penalty (WIP)	{-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10}

Ya que hay muchos resultados, vamos a mostrar una tabla con los porcentajes de acierto y los datos resumidos para el parámetro WIP:

		BEAM					
		25	**50**	**75**	**100**	**125**	**150**
GSF	**0**	69,0 (WIP=4)	79,6 (WIP=4)	**83,4 (WIP=-6)**	81,3 (WIP=-6)	79,2 (WIP=6)	75,2 (WIP=6)
	2	68,6 (WIP=-8)	80,0 (WIP=6)	**83,4 (WIP=-6)**	81,3 (WIP=-6)	79,2 (WIP=2)	76,2 (WIP=10)
	4	67,7 (WIP=-8)	79,6 (WIP=-2)	**83,4 (WIP=4)**	81,3 (WIP=-10)	79,2 (WIP=-6)	75,8 (WIP=8)
	6	65,6 (WIP=-10)	79,2 (WIP=-4)	**83,4 (WIP=-4)**	80,5 (WIP=0)	79,2 (WIP=-8)	75,8 (WIP=10)
	8	59,7 (WIP=-10)	78,3 (WIP=0)	**83,4 (WIP=-8)**	80,5 (WIP=0)	78,8 (WIP=0)	75,8 (WIP=4)
	10	28,3 (WIP=-10)	78,3 (WIP=-10)	**83,4 (WIP=2)**	80,5 (WIP=0)	78,8 (WIP=0)	75,8 (WIP=-2)
	12	0,8 (WIP=-10)	77,1 (WIP=-10)	**83,4 (WIP=-6)**	80,5 (WIP=0)	78,8 (WIP=0)	75,8 (WIP=2)
	14	0,0 (WIP=0)	73,7 (WIP=-10)	82,6 (WIP=-8)	80,9 (WIP=8)	78,3 (WIP=4)	75,8 (WIP=2)
	16	0,0 (WIP=0)	66,1 (WIP=-10)	82,2 (WIP=-6)	81,3 (WIP=8)	78,8 (WIP=0)	76,6 (WIP=-2)
	18	0,0 (WIP=0)	3,3 (WIP=-10)	82,2 (WIP=-10)	80,9 (WIP=2)	78,8 (WIP=6)	76,6 (WIP=-4)
	20	0,0 (WIP=0)	0,0 (WIP=0)	80,1 (WIP=-10)	80,5 (WIP=-8)	78,8 (WIP=-8)	76,6 (WIP=-8)

Hemos mejorado un poco los resultados, teniendo un **83,4%** de aciertos para una configuración de {BEAM=75, GSF=6,WIP=-4}.

Por último, vamos a realizar un proceso de experimentación en torno a nuestro mejor valor de *beam* (BEAM=75), con un aumento en los intervalos todavía más bajo:

Parámetro	**Valores**
beam	{60, 65, 70, 75, 80, 85, 90, 95}
Grammar-scale-factor (GSF)	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}
Word-insertion-penalty (WIP)	{-10, -9, -8, -7, -6, -5, -4, -5, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Como estamos ante muchas pruebas, vamos a ver una tabla resumen de los porcentajes de aciertos que recoge los mejores resultados:

		BEAM							
		60	**65**	**70**	**75**	**80**	**85**	**90**	**95**
GSF	**0**	83,8 (WIP=-5)	83,4 (WIP=-6)	83,4 (WIP=-9)	83,4 (WIP=-5)	82,6 (WIP=-6)	82,2 (WIP=5)	82,2 (WIP=9)	81,3 (WIP=0)
	5	83,4 (WIP=7)	84,3 (WIP=-7)	83,8 (WIP=-5)	83,4 (WIP=4)	82,6 (WIP=-5)	82,2 (WIP=0)	82,2 (WIP=5)	81,7 (WIP=-4)
	10	83,4 (WIP=-10)	**84,7 (WIP=-9)**	84,3 (WIP=-3)	83,4 (WIP=2)	82,6 (WIP=6)	82,2 (WIP=-7)	82,2 (WIP=10)	81,7 (WIP=8)
	15	80,5 (WIP=-9)	82,6 (WIP=-10)	82,6 (WIP=-8)	82,2 (WIP=-5)	82,6 (WIP=-9)	82,2 (WIP=-5)	81,7 (WIP=-9)	81,7 (WIP=5)
	20	41,1 (WIP=-10)	68,6 (WIP=-10)	75,4 (WIP=-10)	80,0 (WIP=-10)	80,9 (WIP=-10)	80,9 (WIP=-9)	80,9 (WIP=-2)	81,3 (WIP=-10)

Hemos mejorado nuestro porcentaje de aciertos hasta un **84,7%** para una configuración de {BEAM=65, GSF=10,WIP=9, histogram-pruning=5000}, la cual será nuestra configuración óptima.

5.4. Análisis de los resultados

Si analizamos los resultados, vemos como mientras que la aplicación inicial tenía un estupendo 96,8% de aciertos, ese porcentaje se ha visto drásticamente reducido, al ampliar el repertorio de órdenes, hasta un 84,7% (un 12,1% menos). Una vez tuvimos la mejor configuración, intentamos explicar el motivo de ese aumento del error.

En primer lugar creamos una matriz de confusión para las pruebas con la configuración óptima. Los resultados fueron reveladores. El 9,3% de nuestro porcentaje de error (es decir, el 61% de los errores) se produce con las órdenes "enlace + "*número*"" y "enlace anterior + "*número*"". Esto es debido a que tenemos noventa y nueve números en nuestro sistema, y hay muchos fácilmente confundibles entre si, como por ejemplo "ocho" y "ochenta", o "cuarenta y seis" y "cincuenta y seis". Si excluimos de las pruebas esas dos órdenes, tenemos un porcentaje de aciertos del 92.3% para nuestra configuración óptima, fallando una media de 3,3 órdenes por locutor de las cuarenta y dos que estamos probando.

5.4. ANÁLISIS DE LOS RESULTADOS

Se planteó intentar solucionar el problema de los números. Una posible solución hubiera sido reducir el número total que teníamos y no ponerlos todos, sino por ejemplo del uno al cinco, y luego a intervalos de cinco hasta el cincuenta. Finalmente se descartó porque realmente los números que menos problemas dan son los que hubiéramos dejado, y ya se pueden utilizar ahora.

Otras órdenes que producen algunos errores son las que llevan las palabras “ventana” y “pestaña”, como “nueva ventana” y “nueva pestaña”, o bien “siguiente ventana” y “siguiente pestaña”. Su onda acústica es similar y es posible que se equivoque, pero el error se produce en una medida muy reducida y no vale la pena intentar arreglarlo cambiando las órdenes por “nueva ventana” y “pestaña nueva”, por ejemplo

Es llamativo que el aumento del factor BEAM provoque una bajada de la calidad de reconocimiento, cuando lo usual es lo contrario. Posiblemente se deba a que los modelos de lenguaje modelan de forma equiprobable todas las palabras que se pueden dar entre dos estados, lo que hace que al podarse menos hipótesis aumente la posibilidad de confusión en el proceso de decodificación, y con ello baje el acierto de reconocimiento.

Por último, añadir que con la configuración conseguida, en las pruebas *online* de la aplicación se producen muy pocos errores, y no enturbian su funcionamiento, salvo quizá si pretendemos utilizar muy a menudo las órdenes con números comentados anteriormente.

Capítulo 6

Conclusiones y trabajos futuros

Durante el desarrollo de nuestro proyecto hemos afianzado conocimientos sobre el reconocimiento de formas, además de aprender cómo funciona realmente una aplicación de reconocimiento automático del habla, la herramienta iATROS. Además, hemos podido conocer mejor aspectos sobre la tecnología web, como el navegador y las páginas web.

La parte más complicada del desarrollo ha recaído sobre nuestra opción estrella en la aplicación: analizar el código fuente de las páginas web para poder acceder a los enlaces pronunciando su texto asociado. El principal problema ha sido recuperar la dirección de la página web del navegador de modo invisible para el usuario. Originalmente descargábamos el código fuente de la página mediante los eventos de teclado asociados que eran necesarios en el navegador, pero daba un aspecto poco profesional a la aplicación el ver cómo se manejaban los menús del navegador por sí solos. Una opción más compleja hubiera sido analizar el código fuente del navegador Mozilla Firefox con el objetivo de crear una opción invisible para la descarga del código fuente.

El aspecto más negativo de la aplicación es seguramente la tasa de errores que se produce con las órdenes “enlace + “*número*”” y “enlace anterior + “*número*””. El error producido por la similitud de la onda acústica de muchos números era difícilmente evitable siguiendo la metodología empleada. Habría que haber planteado otra clase de soluciones para solventar el problema.

En conclusión, se ha desarrollado una aplicación de navegación web con la voz que ofrece un rendimiento general muy satisfactorio. Dadas sus opciones, la aplicación podría ser de mucha utilidad a personas de movilidad reducida, a pesar de que existen aplicaciones profesionales más preparadas.

Existen muchas mejoras o ampliaciones que podrían llevarse a cabo en nuestra aplicación. En primer lugar se podría investigar cómo solucionar el problema del reconocimiento de números, además de pulir algunas de las aplicaciones utilizadas en nuestro programa. Por ejemplo, la aplicación encargada de extraer la dirección web actual del fichero *sessionstore.js* (apartado 3.3.2), se podría haber implementado en un lenguaje con soporte para patrones como Haskell, de modo que hubiera sido más eficiente y fiable. Por otro lado, hubiera sido interesante implementar una interfaz gráfica que mostrase la lista de enlaces extraídos de la página web con sus correspondientes herramientas de navegación para poder acceder a ellos. Una última mejora podría ser añadir la capacidad a nuestra aplicación de introducir texto en el navegador, cosa que hubiera facilitado la escritura de direcciones web y texto en las páginas. Si no se llevó a cabo fue porque no se consideró uno de los objetivos del proyecto.

En cuanto al reconocimiento del habla, aún queda mucho trabajo por hacer en sus diferentes líneas de investigación: sistemas de reconocimiento multidialectales y multilingües, sistemas robustos frente a ruidos y variaciones en el entorno, modelado de lenguaje, integración de sistemas de traducción oral, sistemas de reconocimiento automático del habla de gran vocabulario, comprensión del lenguaje... En definitiva, un largo trecho recorrido, y un todavía más grande trecho por recorrer en el campo de la inteligencia artificial.

Bibliografía

[1] Steve Young, Dan Kershaw, Julian Odell, Dave Ollason, Valtcho Valtchev, Phil Woodland, "*The HTK Book*", 2000, http://htk.eng.cam.ac.uk/.

[2] Página web de iATROS: http://prhlt.iti.es/page/projects/multimodal/idoc/iatros.

[3] Página web de Xmacro: http://xmacro.sourceforge.net/.

[4] Entrada wikipedia Inteligencia Artificial: http://es.wikipedia.org/wiki/Inteligencia_Artificial.

[5] Entrada wikipedia Estadística Computacional: http://es.wikipedia.org/wiki/Ling%C3%BC%C3%ADstica_computacional .

[6] Entrada wikipedia del Habla: http://es.wikipedia.org/wiki/Habla .

[7] Entrada wikipedia Síntesis del habla: http://es.wikipedia.org/wiki/S%C3%ADntesis_de_habla .

[8] Entrada wikipedia Reconocimiento del habla: http://es.wikipedia.org/wiki/Reconocimiento_del_habla .

[9] Entrada wikipedia Procesamiento de lenguajes naturales: http://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales .

[10] Entrada wikipedia Conexión a internet: http://es.wikipedia.org/wiki/Conexi%C3%B3n_a_Internet .

[11] Entrada wikipedia Navegador internet: http://es.wikipedia.org/wiki/Navegador_web .

[12] C.J. Leggetter y P.C. Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density hidden markov models, " *Computer Speech and Language*", vol. 9, pp.171–185, 1995.

[13] G. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.

[14] Estándar lenguaje HTML: http://www.w3c.es/.

[15] 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.

[16] Míriam Luján-Mares, Vicent Tamarit, Vicent Alabau, Carlos-D. Martínez-Hinarejos, Moisés Pastor, Alberto Sanchis, Alejandro Toselli. "*iATROS: a speech and handwriting recognition system*", V Jornadas en Tecnología Web.