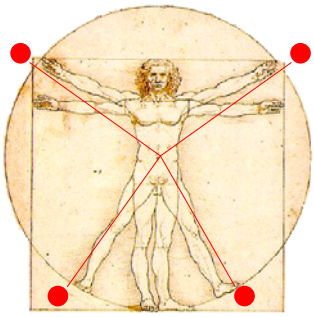
TECNOLOGÍ@ y *DESARROLLO*

Revista de Ciencia, Tecnología y Medio Ambiente

VOLUMEN IV. AÑO 2006

SEPARATA



SISTEMA DE ADQUISICIÓN Y RECONOCIMIENTO DE LA SEÑAL DE VOZ PARA APLICACIONES JAVA.

Antonio José Reinoso Peinado

UNIVERSIDAD ALFONSO X EL SABIO
Escuela Politécnica Superior

Villanueva de la Cañada (Madrid)

Escuela Politécnica Superior.
Universidad Alfonso X el Sabio.
28691, Villanueva de la Cañada (Madrid).
ISSN: 1696-8085
Editor: Julio Merino García tecnologia@uax.es

SISTEMA DE ADQUISICIÓN Y RECONOCIMIENTO DE LA SEÑAL DE VOZ PARA APLICACIONES JAVA.

Antonio José Reinoso Peinado[a]

[a)] Ingeniero en Informática,
Departamento de Electrónica y Sistemas, Escuela Politécnica Superior, Universidad Alfonso X el Sabio.
Avda. De la Universidad nº1, Villanueva de la Cañada, 28691 Madrid. España. Tlf.: 918105008.
email: areinpei@uax.es

RESUMEN: Los avances introducidos en las tecnologías informáticas así como en las técnicas de telecomunicación han hecho posible el desarrollo y consolidación de lo que se ha venido a denominar Sociedad de la Información. Este nuevo paradigma de acceso a la información pone a disposición de los usuarios un gran número de recursos, contenidos y servicios con una relativa facilidad y comodidad. Así, las nuevas tecnologías ofrecen un amplio abanico de posibilidades con unos requerimientos, en la actualidad, muy asequibles. Un computador personal dotado de elementos multimedia y una conexión a Internet son, en principio, los únicos elementos necesarios para la integración del individuo en esta nueva sociedad. Sin embargo, el carácter audiovisual con que la información es tratada y presentada plantea serios inconvenientes a personas con algún tipo de discapacidad que dificulte la interacción con un entorno de este tipo. Estas barreras resultan especialmente determinantes para las personas afectadas por algún problema o deficiencia visual. Este colectivo requiere de métodos alternativos y particulares de presentación y acceso a la información y, también, de recursos adicionales para el control de las aplicaciones que la tratan.

Dado que el objetivo perseguido es hacer posible la integración de toda la población en esta nueva Sociedad de la Información, sería muy deseable la introducción de elementos que contribuyan a salvar cualquier tipo de barrera que pueda derivarse de una discapacidad.

El presente artículo describe el trabajo de investigación realizado con objeto de introducir un mecanismo de control por voz en aplicaciones JAVA destinadas, en principio, al tratamiento y representación de contenidos WEB. Con ello se pretende aportar una mejora considerablemente significativa en las condiciones de acceso a la información presentada en Internet en presencia de algún tipo de discapacidad que impida la utilización de los elementos de control habituales. Si bien este ha sido el ámbito de desarrollo para el citado mecanismo, su utilización puede extenderse a cualquier aplicación de carácter más general sin mayores dificultades.

PALABRAS CLAVE: Reconocimiento Automático de la Voz, Accesibilidad , IVORY, JNI, Integración Java-C

ABSTRACT: Improvements in computer and telecommunication technologies have lead to the development and consolidation of what is called the Information Society. This new approach allows users to access, in an easy and comfortable way, to a great number of resources, contents and services, offering a huge number of possibilities with small requirements. The only elements necessary for the integration of the users in this new society are the access to a personal computer with multimedia equipments and an Internet connection. However,

the audiovisual treatment of the information can lead to serious difficulties for handicapped persons. These barriers are especially notorious for visually impaired people. This collective requires alternative ways of presentation and access to the information and, also, additional mechanisms to control the applications they are dealing with

.

As the aim of the present work is to make possible the integration of everybody in this new Information Society, it will be desirable to incorporate elements that can overcome any kind of handicap barrier.

This article describes the research work accomplished to design and develop a control mechanism voice to be incorporate in Java applications for dealing and representing web contents. The main objective is to improve the accessibility conditions to the information published on Internet for those people with sensorial handicaps, not able to use traditional control mechanisms. Although this is the main scope of the system, it has been designed in a flexible way for its extension to other general purpose applications.

*KEYWORDS: Automatic Speech Recognition, Accesibility, IVORY, JNI, JAVA-C Integration.*Mecanismos de control por voz para herramientas de mejora de la accesibilidad a Internet.

1. Introducción.

Las condiciones de accesibilidad a los contenidos WEB, en el caso de usuarios con discapacidades sensoriales, se verían notablemente mejoradas con la ayuda de herramientas adecuadas de presentación de la información apoyadas por dispositivos audio/táctiles.

En este tipo de aplicaciones resultaría de gran utilidad disponer de mecanismos que permitan al usuario interactuar con el sistema a través de su voz y ejercer mediante ella un control lo más completo posible. Esta alternativa supondría una mejora cualitativa en la forma de relación con el sistema, a la vez que aportaría un conjunto considerable de ventajas.

La capacidad por parte de una aplicación de responder a indicaciones verbales aumenta, en primer lugar, la facilidad y el grado de comodidad con que un usuario puede comunicarse con ella y conducirse a través de los elementos que la componen.

Las órdenes verbales constituyen un elemento de control totalmente natural y sencillo que no presentará mayores dificultades para ser adoptado por los usuarios.

Los beneficios de un sistema como el presentado redundarían en un conjunto muy amplio de usuarios, ya que todas aquellas personas con capacidad para expresarse verbalmente podrían aprovecharse de sus ventajas sin importar el sexo, la edad u otras consideraciones físicas.

Así mismo, la familiaridad inherente a esta forma de control reduce considerablemente el período de adaptación a la misma y facilita enormemente su aprendizaje por parte del usuario.

Por otro lado, un sistema que posibilite el control mediante el habla reduce considerablemente el tiempo invertido por el usuario en dar las instrucciones correspondientes para llevar a cabo una determinada tarea. De esta forma, una indicación verbal sustituye a un conjunto más complejo de acciones psicomotoras que habitualmente involucrarían además a diversos elementos como el puntero del ratón, menús, botones...

La amigabilidad y comodidad de esta posibilidad contribuirán decisivamente a conseguir una aceptación muy generalizada ya que si bien puede suponer un elemento relativamente indispensable en presencia de ciertas discapacidades, no cabe duda de que

ofrece una interrelación notablemente más rápida y favorable a cualquier tipo de usuario.

Para dar soporte a esta posibilidad de control a través de la voz debe disponerse, eso sí, de mecanismos ágiles y eficientes para la adquisición y tratamiento del habla.

Así pues, el presente artículo pretende detallar el trabajo de estudio e investigación realizado para diseñar y construir un componente de adquisición y reconocimiento del habla que pueda ser fácilmente integrado en aplicaciones JAVA, como las destinadas a trabajar con contenidos WEB.

Dicho componente deberá ofrecer una solución robusta y altamente portable y además ser fácilmente integrable en aplicaciones JAVA que pretendan ofrecer al usuario la posibilidad de responder a indicaciones verbales de un modo sencillo, preciso y ágil. Fundamentalmente, su especificación de requisitos consistirá en los siguientes puntos:

1. Total portabilidad con respecto sistema informático utilizado.
2. Alto grado de independencia con respecto al orador.
3. Capacidad de operación en tiempo real.
4. Facilidad para su incorporación en aplicaciones o sistemas más complejos

2. Descripción de la solución presentada.

El sistema presentado en este artículo consiste fundamentalmente en un módulo para la captura de voz y otro para su posterior tratamiento y reconocimiento, así como los mecanismos que hacen posible la correcta y eficiente integración de los mismos.

Los requisitos generales expuestos establecen una serie de características que resultarán determinantes tanto en el proceso de diseño y construcción de la aplicación de captura de la señal de voz, como en la elección de una metodología concreta para el reconocimiento de la misma.

Los requerimientos de portabilidad e independencia de los componentes software y hardware imponían la utilización de JAVA como lenguaje de codificación. El sistema JAVA resultaría, así, completamente independiente de las especificidades internas de las aplicaciones que pudieran beneficiarse de su integración, de la arquitectura de la plataforma de acceso empleada, así como de los diversos elementos software y hardware Uno de los principios de diseño del lenguaje JAVA es hacer posible el desarrollo de aplicaciones que puedan correr sobre cualquier plataforma que disponga de una máquina virtual adecuada (*JRE Java Runtime Environment*). De esta forma se consigue

independizar la compilación de las aplicaciones de las características particulares de una determinada arquitectura o sistema operativo subyacentes. Las aplicaciones JAVA son compiladas a un código especial conocido como código neutro (*bytecodes*) que es interpretado de igual modo por cualquier máquina virtual sin importar la plataforma sobre la que se encuentre instalada.

Para el tratamiento de la señal de voz se determinó la adopción de una metodología conocida como IVORY. Esta metodología, basada en un modelo probabilístico, estructura el proceso de reconocimiento en un conjunto de etapas que se aplican a la señal de entrada y se caracteriza por su eficiencia y alto grado de independencia del orador. Esta última característica constituye una ventaja de gran interés para las aplicaciones destinadas a un colectivo de personas muy amplio.

Cada fase de la citada metodología se traduce en una función con un conjunto bien definido de parámetros. De esta forma, ha sido necesario especificar de forma concisa el interfaz de cada una de estas funciones para conseguir el adecuado encadenamiento de las mismas.

Una característica fundamental añadida a esta metodología, consiste en la inclusión de información relativa a la situación actual de manejo de la aplicación, o contexto de trabajo, en algunas de las funciones de reconocimiento. Con ello se persigue limitar el conjunto de palabras que se pueden reconocer en cada instante de manejo de la aplicación. Esta característica permite un reconocimiento más fiable al reducir el tamaño del vocabulario. Esta reducción del vocabulario a reconocer en función de una determinada situación de trabajo con una aplicación permite, por tanto, realizar un reconocimiento más fiable al limitar considerablemente el conjunto de palabras candidatas.

Si con JAVA se solucionaban los requerimientos de portabilidad, la capacidad de ofrecer respuesta en tiempo real, por su parte, exigía una considerable agilidad de computación del sistema en su conjunto y, en particular, de la parte correspondiente a las funciones de reconocimiento por presentar una mayor carga de proceso. Este requisito, junto con la gran cantidad de movimientos de datos en memoria necesarios, aconsejaban para su implementación un lenguaje como C con el que obtener un código objeto compilado de mayor agilidad que JAVA.

En este punto se presentaban varios problemas importantes, por un lado, la complejidad de la interconexión de dos módulos construidos de esta forma. Por otro lado, podría resultar de interés poner a disposición de otras herramientas el propio sistema de

reconocimiento de forma separada al módulo de adquisición de voz sin que ambas partes se ofrecieran con un vínculo demasiado estrecho entre ellas. En este supuesto, se consideró que el grado de utilidad de una biblioteca de funciones C en el ámbito del desarrollo de aplicaciones WEB podría no ser demasiado alto. A las cuestiones anteriores habría que añadir la relativa a la pérdida de portabilidad con respecto a la plataforma que supone la utilización de código objeto compilado.

La potencia y prestaciones del lenguaje C en entornos de tiempo real hacen que resulte de interés mantener una biblioteca con la implementación de las principales funciones de reconocimiento escritas en este lenguaje. La inclusión del sistema de captura JAVA supone la importante mejora de añadir un nuevo interfaz a la biblioteca, de forma que ahora sus funciones también puedan ser invocadas desde aplicaciones JAVA o entornos Web. Todas las cuestiones relativas al intercambio de información entre las aplicaciones JAVA y la biblioteca C resultarán, por supuesto, transparentes a los usuarios. Así pues la biblioteca se hace accesible a aplicaciones escritas tanto en C como en JAVA.

El requisito concerniente a la facilidad de incorporación a otras aplicaciones o sistemas más complejos determinó el diseño de todo el sistema de forma que pudiera correr bajo un hilo de control. Así, la aplicación más general no tendrá que permanecer a la espera de recibir alguna indicación verbal bloqueando mientras el resto de sus posibilidades, sino que una vez activado el hilo de control del sistema, la captura y reconocimiento de la voz se realizará concurrentemente con el resto de actividades que pueda desarrollar la aplicación. Se trata por tanto de hacer que el proceso de reconocimiento de voz se realice bajo un hilo completamente independiente de los que pueda activar la aplicación donde se integra.

3. Reconocedor de voz basado en la metodología IVORY.

El sistema de reconocimiento de voz utilizado en la solución presentada funciona aplicando cuatro fases a la señal de entrada (Gómez, 2000). Estas fases son las siguientes:

1- Transformación espectral

2- Extracción de parámetros

3- Cuantización vectorial

4- *Parser* de Markov

Los tres primeros pasos buscan extraer las características más significativas y, por tanto, más útiles para el reconocimiento de la señal de voz suministrada como entrada. El último paso compara estas características con los modelos estadísticos que se han

elaborado previamente de ciertas palabras (el *vocabulario* del reconocedor). Si las características de la señal de entrada se asemejan claramente a alguno de los modelos construidos, el sistema da como reconocida la palabra correspondiente a tal modelo, que es considerado como el modelo ganador en la comparación.

Los modelos estadísticos de cada palabra del vocabulario que se desee reconocer han de construirse previamente. Son necesarias unas muestras previas (*observaciones*) de cada palabra que forme parte del vocabulario. Esto constituye una *base de datos de palabras,* con las que construir los modelos estadísticos o modelos ocultos de Markov de cada palabra del vocabulario (Gómez, 1999).

Cuando el usuario pronuncia una palabra, ésta se recoge como una sucesión de valores que representan la intensidad del sonido. Si expresamos gráficamente esos valores a lo largo del tiempo obtenemos una representación como la siguiente:

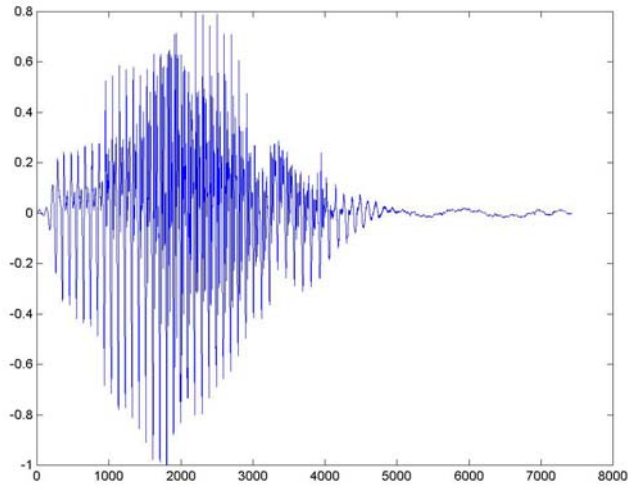


Figura 3.1. Representación gráfica de una señal de voz.

Entre las dificultades que presenta la utilización de gráficas similares a la anterior como técnica de inspección visual de la señal para su posterior reconocimiento está el ruido de fondo (Gómez, 1998), y que además en la representación gráfica se aprovecha menos el ancho del dibujo. Pero el principal inconveniente que plantea este tipo de análisis visual de la señal es que las vocales están pronunciadas con mucho énfasis (amplitud) y por tanto las consonantes están comparativamente silenciadas.

Por esta razón, para extraer características significativas de la señal de voz no se utilizan los valores de intensidad de la señal en el tiempo sino una ***transformación espectral por ventanas***. La señal se divide así en tramos de 256 muestras consecutivas, llamados *ventanas*. Con la hipótesis de que en cada ventana las características espectrales de la señal son estacionarias, se hace una transformación de Fourier [Rab99] de cada una de ellas.

Para mejorar el rango dinámico de estos datos, es decir, para que no haya mucha diferencia entre los valores, se suele tomar el logaritmo. Así se mejora la resolución en las pequeñas amplitudes:

La función del reconocedor encargada de llevar a cabo esta transformación presentaría un prototipo como el siguiente:

```
void IVORY_espectro(float *sennal, float *espectro, float *penergia)
```

Dicha función recibe como parámetro de entrada un vector de 256 valores reales correspondientes a una ventana de la señal (`sennal`). Tras aplicar un filtro de preénfasis obtiene la representación espectral de la ventana mediante la aplicación de la transformada rápida de Fourier (FFT). Esta representación consiste en un vector de 128 valores reales (`espectro`) que será devuelto como parámetro de salida. La función calcula además la energía presente en la ventana mediante la suma de los armónicos obtenidos. Este valor también será devuelto como parámetro de salida (`penergia`).

Las características propias de cada vocal se manifiestan más claramente tras esta transformación espectral por ventanas (*short-term fourier transform)* y hasta cierto punto también las consonantes. Sin embargo las bandas que contienen la señal es una característica dependiente del hablante. Incluso una misma vocal pronunciada en distinto tono de voz estará contenida en distintas bandas de frecuencia.

Es necesario, por tanto, definir un conjunto de parámetros que tomen el mismo valor en pronunciaciones distintas del mismo fonema. En el sistema de reconocimiento IVORY se utiliza un método para detectar la "forma" del fonema independientemente de su "tamaño" y los parámetros utilizados se basan en el contenido en frecuencia en las siguientes 20 bandas (un círculo marca el inicio de la banda, y un asterisco el final):

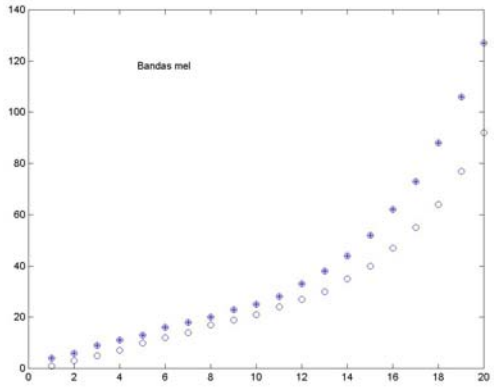


Figura 3.2.: Bandas Mel para una ventana.

Esta partición lineal-exponencial del espectro se conoce como *escala Mel*.
Para cada ventana de tiempo se calcula cuánta energía esta contenida en estas bandas del espectro. Así los 128 valores por ventana se ven reducidos a 20 (un valor por cada banda).

Es necesario ahora definir unos parámetros, a partir de estas 20 cifras, que den independencia respecto al hablante. Se usa una cierta transformación que consiste en tomar el logaritmo de estos datos y después hacer una transformada inversa de Fourier. Esta transformada inversa del logaritmo se conoce como *transformación cepstral* (Gómez, 2000). El resultado, al aplicarlo a los 20 valores de la energía en bandas, son 10 coeficientes con valores reales.

De esta forma, el conjunto de parámetros que resume las características de una ventana se denomina ***plantilla***. En el sistema IVORY, cada una de estas plantillas consistirá en un vector de 20 elementos. Los diez primeros valores constituyen la transformación cepstral descrita arriba, mientras que los diez segundos consisten en los promedios de las transformaciones cepstrales que resultan de las ventanas anteriores y posteriores a la que se está considerando.

Esta fase es implementada en el reconocedor mediante una función con una cabecera como la siguiente:

```
void IVORY_plantilla(float *espectro, float
promediados[AVERAGE_X][AVERAGE_Y], float *plantilla)
```

El espectro correspondiente a la ventana cuya plantilla se va a construir es recibido como parámetro de entrada (`espectro`). A partir de dicho espectro y de las transformaciones cepstrales correspondientes a las ventanas anteriores y posteriores (`promediados`), la función elabora la plantilla para la ventana actual que se devuelve como parámetro de salida (`plantilla`). La matriz de transformaciones cepstrales queda actualizada de forma que incluya la nueva plantilla ya que ésta será necesaria para el cálculo de la plantilla correspondiente a la ventana siguiente.

Parte importante del diseño de unas plantillas es que éstas se agrupen por zonas próximas según el sonido a que corresponden. Si consideramos que una base de datos de entrenamiento, consistente en una serie de realizaciones variadas de cada palabra de la que se intenta construir un modelo estadístico, puede constar de 100 realizaciones de un vocabulario de 10 palabras, el número de plantillas es de ~50000. A las muestras de entrenamiento se les suele denominar *observaciones*.

Este número resulta excesivo si se pretende comparar las plantillas obtenidas de una palabra a reconocer con cada una de las 50000 de muestras de entrenamiento. Por eso se ***cuantiza***, es decir, se reduce ese conjunto tan amplio.

El proceso de cuantización en el espacio de plantillas no es un proceso sencillo porque se trata de un espacio vectorial de 20 dimensiones. El conjunto de muestras de entrenamiento se divide en zonas, cada una con un punto privilegiado o *centroide*, del mismo modo que en el ejemplo anterior se dividía el conjunto de muestras en subintervalos, cada uno con un entero privilegiado. Así basta comparar el dato con los centroides, y no con todas las muestras. La zona que, a efectos de comparación, se ve substituida por el centroide, se llama *cluster* (Díaz, 2001). En el reconocedor que se describe se utilizan 256 clusters, cada uno con su centroide.

El resultado es el *codebook* –libro de códigos- y cada centroide es el código de una región del espacio de plantillas.

Con la obtención de una serie de centroides concluye el proceso de extracción de características a partir de la señal original de entrada. Con la aplicación de este proceso de cuantización se consigue que cada ventana de 256 muestras de voz (256 valores reales) quede representada por un solo centroide (un valor entero entre 0 y 255) del libro de códigos. Una grabación de una palabra de dos sílabas se verá así reducida a una sucesión de 60 centroides. Será en la siguiente fase donde se lleve a cabo el reconocimiento propiamente dicho.

La función del reconocedor encargada de implementar esta fase tiene una cabecera como la siguiente:

```
void IVORY_cuantizar(float *plantilla, int contexto, byte *pcentroide)
```

La plantilla a cuantizar es suministrada como parámetro de entrada (`plantilla`). La función compara los 20 valores que constituyen la plantilla con las 20 coordenadas de cada uno de los centroides que constituyen el libro de códigos y devuelve como parámetro de salida el centroide más cercano o ganador (`pcentroide`). El libro de códigos que se utilizará será el correspondiente al vocabulario asociado al contexto especificado (`contexto`).

A partir de la base de datos de entrenamiento se construye un modelo estadístico de cada palabra del vocabulario que se pretende reconocer. Estos modelos se denominan *modelos ocultos de Markov* (Hidden Markov Model o HMM).

Cada uno de estos modelos constituye un conjunto de sucesiones de centroides. Estas sucesiones son similares a las que se obtienen de las transformaciones paramétricas de las señales reales de voz pronunciadas por diversos hablantes. Por esta razón se denomina modelo, porque modela o representa a todas las realizaciones de una determinada palabra con independencia del orador que la pronuncie.

Inversamente, dada una sucesión de centroides, puede calcularse también la probabilidad de que un modelo de Markov la haya producido. Así, ante una sucesión de centroides correspondientes a una palabra desconocida, puede calcularse para cada modelo del vocabulario cuál es el mejor (el más probable) para tal sucesión. La palabra correspondiente a tal modelo es la que el reconocedor proporciona como respuesta a la señal de voz de entrada.

Calcular la probabilidad de que una sucesión de centroides haya sido producida por un modelo HMM guarda similitud con el análisis léxico de una cadena de símbolos por un autómata. Por eso a esta fase se le suele denominar ***parser*** y los modelos de Markov pueden describirse como autómatas estocásticos.

Esta fase de reconocimiento propiamente dicho es llevada a cabo por una función que presenta una cabecera como la siguiente:

```
void IVORY_parser(byte centroide, float energia, int contexto,
                  state_t estado, byte *pganador)
```

Esta función implementa un autómata de reconocimiento y se va llamando con el centroide obtenido como resultado de la cuantización vectorial de cada ventana (`centroide`), así como con el nivel de energía de la misma (`energia`). Este parámetro de entrada se utiliza para detectar si la ventana corresponde al final de una palabra y validar así la comparación de la serie de centroides recibidos con los pertenecientes a los distintos modelos de que se dispone. En otro caso, el centroide suministrado es tratado como una nueva entrada al autómata y hará que éste evolucione a través del conjunto de estados que lo definen. Además del propio centroide, esta transición vendrá determinada por el estado actual en que se halle el autómata. El estado del autómata después de la recepción de cada centroide va quedando reflejado en una estructura que actúa como parámetro de entrada/salida (`estado`). Por último, el índice del modelo ganador se devolverá a través del correspondiente parámetro (`pganador`)

Un tercer parámetro de entrada indica al reconocedor el contexto en que se encuentra trabajando el usuario y se utiliza para determinar el conjunto de modelos donde buscar el correspondiente a la palabra que se trata de reconocer. La utilización del contexto en la tarea de reconocimiento se explica en detalle en el siguiente apartado. La función devuelve como parámetro de salida el modelo al más próximo a la sucesión de centroides obtenidos de la señal de voz de entrada. La palabra asociada a ese modelo será la reconocida por el sistema.

3.1. Modificaciones introducidas en las funciones del reconocedor: Introducción del contexto de trabajo.

En la interacción con una determinada aplicación, lo habitual es que, en cada instante, el usuario tenga a su disposición una serie de opciones, activas como resultado de alguna de las posibilidades de evolución que ofrecen los componentes de dicha aplicación. En ese mismo momento, sin embargo, carecerá de sentido la utilización de muchas otras de sus opciones. Por ejemplo, el menú edición de cualquier procesador de textos mostrará distintas opciones en función de que haya o no texto previamente seleccionado.

Definimos **contexto de trabajo** como la situación actual en el manejo de una aplicación. Cada contexto de trabajo determina un conjunto de acciones posibles. Por tanto, en un instante dado, el contexto de trabajo en que se halle el usuario determinará las órdenes que pueden ser aceptadas como válidas. De esta forma, a cada contexto se le puede asociar un determinado vocabulario, denominado **vocabulario activo**, constituido por

las indicaciones verbales correspondientes a las órdenes con validez en la situación representada por el contexto.

El contexto juega, pues, un papel fundamental en la tarea de reconocimiento ya que determina el conjunto de modelos o, dicho de otro modo, el vocabulario que puede ser reconocido en un instante determinado. Las ventajas de este enfoque, frente a la posibilidad de reconocer primero la palabra de entre un conjunto general y comprobar después la validez de su aplicación en un instante concreto, se basan en que cuanto más pequeño sea el conjunto de modelos de que se dispone más precisa resulta la tarea de reconocimiento.

Esta forma de llevar a cabo el reconocimiento exige que algunas funciones del reconocedor tengan conocimiento en todo momento del contexto de trabajo en que se encuentra el usuario y deberán recibir esta información como parámetro de entrada.

La función del reconocedor encargada de realizar la cuantización vectorial de una ventana de voz y la que implementa el *parser* o reconocedor propiamente dicho reciben como parámetro de entrada un valor entero entre uno y tres que indica el contexto de trabajo actual.

La **función de cuantización** utilizará el contexto de trabajo para determinar el libro de códigos con las coordenadas de los centroides que se van a comparar con las de la plantilla correspondiente a la ventana procesada.

El ***parser*** o autómata de reconocimiento utiliza la información del contexto actual para elegir el conjunto de modelos de Markov correspondiente a las palabras que pueden reconocerse. Esta elección servirá para determinar las matrices de probabilidad de emisión de centroides y de transición de estados que van a utilizarse.

4. Aplicación JAVA para la captura del discurso hablado y alimentación de las funciones de reconocimiento.

4.1. Descripción de la aplicación.

La aplicación diseñada se compone principalmente de tres módulos o clases que se describen detalladamente en los apartados siguientes.

Una de estas clases (*ParserState*) se utilizará para almacenar información útil para el autómata reconocedor así como para reflejar su estado.

Las funcionalidades de captura de la señal de voz, preproceso de ésta y posterior invocación a las rutinas de reconocimiento se implementan en la clase principal de la aplicación (*AudioLoopC*). Por otro lado, esta clase ha sido diseñada para ser anidada fácilmente en entornos multihilo de forma que su incorporación a aplicaciones más complejas no acarree ningún tipo de ralentización en el funcionamiento normal de las mismas.

Una tercera clase (*RecIvory*) actúa como interfaz de enlace a las funciones del reconocedor disponibles a través de una librería externa.

A continuación se describe gráficamente el diseño modular del sistema:

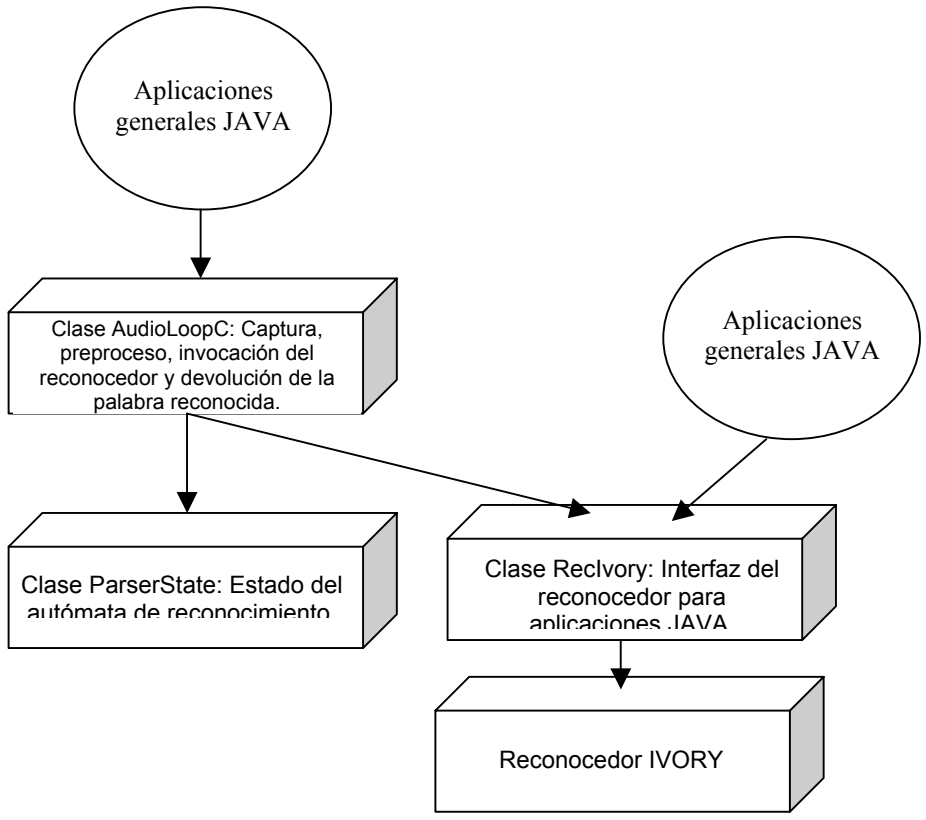


Figura 4.1.: Diagrama de bloques del sistema de adquisición y reconocimiento presentado.

4.2 La clase ParserState.

La clase ParserState se utiliza para mantener el estado del autómata que implementa la función de reconocimiento. Esta clase recoge un indicador de presencia de voz en la plantilla correspondiente a una determinada ventana mientras que otro refleja el número de silencios percibidos hasta el momento.

Si una ventana contiene voz, se calculará la probabilidad de emisión de su centroide en el estado en que se encuentre el reconocedor para cada uno de los modelos disponibles, determinando mediante la tabla de transiciones el nuevo estado al que conduce el centroide en cada uno de dichos modelos. De esta forma se va estimando la probabilidad que tiene cada modelo de corresponder a la secuencia de centroides recibida.

Se deja transcurrir un lapso de siete ventanas sin voz, es decir, siete silencios antes considerar que se ha alcanzado el fin de palabra. Cuando se detecta esta condición se observan las probabilidades de los distintos modelos, considerando el de mayor probabilidad como el correspondiente a la palabra reconocida. Estas probabilidades también se almacenan en la estructura de manera que pueda llevarse a cabo su actualización con la recepción de nuevos centroides.

4.3 La clase AudioLoopC

La clase *AudioLoopC* constituye la clase principal de la aplicación. Esta clase implementa tanto las estructuras como los mecanismos necesarios para hacer posible la captura de la señal de voz procedente de un micrófono. Además se lleva a cabo una de preprocesamiento de la señal cuya misión consiste en descartar la información presente en ella que no resulta relevante para el reconocimiento de su contenido. Una vez aplicado este filtro, la señal es suministrada a las distintas funciones que constituyen el reconocedor para la extracción de sus características más significativas y la tarea de reconocimiento propiamente dicha. Por último, se procederá a la comunicación de la palabra reconocida, si este fuera el caso, a la aplicación general.

Esta clase tiene como atributos más importantes la línea por la que se recibirá la señal de voz, el tamaño de los buffers interno y externo, así como el contexto de trabajo actual. El buffer interno constituye la estructura donde la línea almacenará los datos capturados para pasar de ahí a otro buffer accesible desde la aplicación o buffer externo. Por esta razón los tamaños de ambos buffers deben ser similares.

El constructor de esta clase recibe como parámetro de entrada un objeto que define un formato de audio específico. Este formato establece propiedades como la técnica de codificación, el número de canales, la frecuencia de muestreo, el número de bits por muestra y canal, la frecuencia de construcción de trama, el tamaño en bytes de dichas tramas y la disposición de sus bytes. El constructor también recibe el tamaño de los buffers internos donde se almacenará la señal.

La tarea básica de este constructor es la apertura de una línea de entrada que permita a la aplicación recibir datos de audio. Comúnmente la obtención de una línea se realiza a partir de alguno de los manejadores instalados en el sistema. Si nuestra aplicación procediera de este modo debería, en un principio, detectar todos los manejadores disponibles en el sistema, comprobar los que pueden dar soporte a una línea con las características deseadas y elegir uno de entre ellos para obtenerla. Así pues, por razones de portabilidad y eficiencia, se ha optado por obtener la línea de entrada de forma directa a partir del controlador general de los recursos audio del sistema, evitando así el trato con manejadores específicos. Además, este controlador general considera como entrada estándar de audio al sistema la procedente del micrófono con lo que nos proporcionará de forma automática una línea con datos procedentes de este dispositivo.

Una vez obtenida la línea, se procede a su apertura indicando el formato de audio de sus datos y el tamaño de buffer solicitado. Esta operación dará lugar a que la línea tome posesión de los recursos del sistema necesarios y quede completamente operativa.

La clase *AudioLoopC* ha sido diseñada e implementada de forma que todo el proceso de captura, tratamiento y reconocimiento de la señal de voz se ejecute bajo un hilo de control. Esta característica supone una ventaja muy considerable a la hora de su integración en otra aplicación más compleja. De esta forma, la aplicación más general no tendrá que permanecer a la espera de recibir una señal de voz bloqueando mientras el resto de sus funcionalidades sino que una vez activado el hilo de captura y reconocimiento éste correrá de forma concurrente o en paralelo con el resto de actividades lanzadas por la aplicación.

La activación del citado hilo se lleva a cabo mediante la invocación del método `start()` de la clase *AudioLoopC*. Este método, antes de poner en funcionamiento el ciclo de captura y reconocimiento, activa, a su vez, la línea creada para la recepción de voz para que se pueda comenzar a recibir datos a través de ella.

Una vez activado el hilo de captura y reconocimiento y tras inicializar las estructuras de datos utilizadas para esta última tarea comenzará a recibirse la señal de voz dictada por el usuario. Esta señal será capturada, almacenándose en el buffer definido a tal efecto, y a continuación preprocesada antes de ser suministrada a las funciones del reconocedor.
El tamaño de los buffers interno y externo, a fin de contribuir a que no se pierda información, excede ampliamente del tamaño especificado para las ventanas IVORY. Por esta razón la primera tarea del preprocesamiento de la señal de voz consistirá en ir tomando fragmentos de la señal de voz con el tamaño adecuado.

Por otro lado, no toda la información recibida resulta de utilidad. En efecto, si la señal se ha recibido a través de dos canales (en estéreo), sólo uno de ellos contendrá datos de voz. La información contenida en el otro será una sucesión de ceros que nada aporta en la tarea de reconocimiento.

La siguiente fase del preprocesamiento de la señal consistirá, así, en descartar esta información no relevante. De forma que de cada 4 bytes recibidos sólo 2 serán de utilidad. De estos 16 bits, el primero indica el número del canal, con lo que los valores obtenidos se hallarán dentro del rango –16.384 a 16.383. Estos valores son, a continuación, normalizados para llevarlos al intervalo –1 a 1. El preprocesamiento de la señal tiene también en cuenta que las ventanas utilizadas en la metodología IVORY se mueven de forma solapada por la señal de voz, de manera que los 128 últimos valores de una ventana constituyen los primeros 128 de la ventana siguiente.

Las ventanas así construidas se encuentra ya en condiciones de ser suministradas a las funciones de reconocedor.

El contexto de trabajo actual se almacenará en el atributo `contexto` de la clase *AudioLoopC*, la cual ofrece también sendos métodos para su consulta y actualización denominados, respectivamente, `get_contexto()` y `set_contexto()`.

Cada vez que se produzca un cambio del contexto de trabajo en la aplicación donde se integre el sistema de captura y reconocimiento, ésta lo comunicará al módulo de captura actualizando el atributo `contexto` a través del método `set_contexto()`.

El módulo de captura, por su parte, consulta el contexto actual en cada lectura de la señal de voz y lo envía junto con los datos de dicha señal a las funciones del reconocedor que lo precisen. Normalmente, el usuario mantendrá invariable el contexto durante la pronunciación de una palabra. Un cambio en el mismo en estas circunstancias

simplemente invalidará el reconocimiento de la palabra ya que el tramo emitido después del cambio probablemente no corresponderá a ninguna de las palabras del nuevo vocabulario activo.

Una vez que se dispone de las ventanas que almacenan la señal de voz, el siguiente paso es iniciar la secuencia de llamadas a las funciones del reconocedor para su análisis e intento de reconocimiento.

Si se reconoce alguna palabra, ésta se comunicará a la aplicación principal para que se lleven a cabo las acciones oportunas.

4.4 La clase RecIvory

La clase *RecIvory* constituye el mecanismo de interconexión entre la aplicación de captura de la señal de voz y las funciones que implementan el reconocedor basado en la metodología IVORY.

Esta clase define un conjunto de métodos que se declaran como nativos para indicar que su implementación no se especifica dentro de la propia clase y, además, se ha escrito en un lenguaje diferente de JAVA.

Cada uno de estos métodos se corresponde con una de las funciones del reconocedor y constituye el enlace que hace posible la ejecución de su código compilado al tiempo que permite el envío y recepción de información a través de los parámetros de entrada /salida destinados a tal efecto.

Dado que esta clase actúa como interfaz de comunicación con el código objeto correspondiente a las funciones que conforman el reconocedor no se ha considerado necesaria la instanciación de la misma en objetos concretos por lo que todos sus métodos se han definido además como de clase o estáticos.

Este código objeto con el que se pretende enlazar debe ser, por otro lado, puesto a disposición de la clase. Este objetivo se alcanza cargando en memoria la librería dinámica resultante del proceso de compilación de las rutinas nativas. Este proceso de carga se lleva a cabo en la sección de inicialización de la clase.

Por último, la clase *RecIvory*, al proporcionar un interfaz a las funciones del reconocedor, ofrece un mecanismo que desvincula el acceso a las mismas de la

aplicación de captura de voz.. Esto hace posible que cualquier aplicación pueda tener acceso a ellas de forma independiente.

5. Integración de las funciones del reconocedor con la aplicación Java.

El Interfaz Nativo de Java (JNI) es el interfaz de programación de Java para código nativo que forma parte de la distribución JDK (JNI API Specification). La escritura de programas utilizando JNI asegura que el código resultante es completamente portable a todas las plataformas.

JNI permite que el código Java escrito para correr sobre la Maquina Virtual de Java (JVM) pueda operar con aplicaciones y librerías codificadas en otros lenguajes de programación como C, C++ ó ensamblador, existiendo además la posibilidad de anidar la propia JVM en estas aplicaciones nativas.

El procedimiento general de integración que permite la invocación de métodos nativos desde aplicaciones Java consta de una serie de etapas que se enumeran a continuación:

1. Creación de una clase Java con las declaraciones de los métodos nativos que se pretende invocar.
2. Compilación de dicha clase Java.
3. Generación del archivo cabecera para la implementación de los métodos nativos utilizando la herramienta *javah* con el modificador *–jni* para indicar la utilización de un interfaz a código nativo. Este fichero proporcionará la definición o prototipo formal de cada uno de los métodos nativos.
4. Codificación de los métodos nativos en el lenguaje de programación elegido.
5. Compilación del fichero cabecera y el fichero con la implementación de los métodos nativos para obtener una librería compartida.
6. Ejecución de la aplicación Java.

El proceso de integración del módulo codificado en JAVA con las funciones del reconocedor C se describe gráficamente en la figura 5.1:

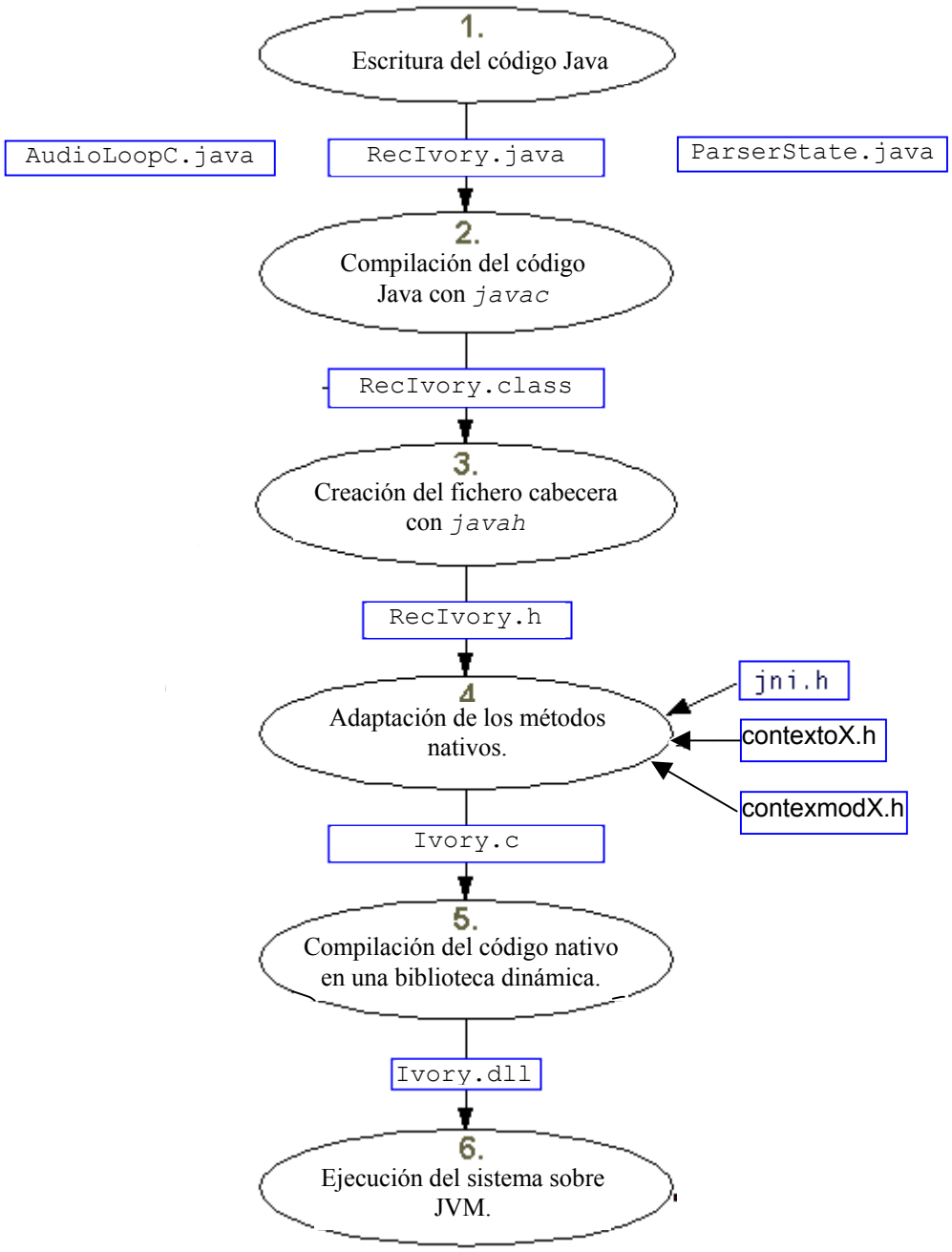


Figura 5.1.: Integración de las funciones C del reconocedor con JAVA.

La clase RecIvory contiene las declaraciones de todos los métodos nativos que actúan de interfaz para la ejecución e intercambio de información con las funciones del reconocedor así como la secuencia para la carga de la biblioteca resultante de la compilación de los métodos nativos.

Las clases JAVA codificadas son compiladas con la herramienta *javac* ofrecida por la distribución JDK (JAVA Development Kit) a tal efecto(JAVA API Specification). Como resultado se obtendrán los correspondientes archivos *ParserState.class*, *RecIvory.class* y *AudioLoopC.class* en la misma ubicación que sus respectivos archivos con el código fuente.

Existe una herramienta especial (*javah*) que resulta de gran utilidad en la generación de prototipos de funciones que se correspondan con las declaraciones de métodos nativos realizadas en la aplicación Java (Schildt, 2001). Para generar este fichero de cabecera, se aplica la herramienta *javah* al fichero con extensión *.class*. Con esta herramienta se especificará la opción *–jni* para indicar que se pretende generar un fichero de cabecera con prototipos de funciones que corresponden a métodos nativos. Como resultado de ambas órdenes se obtendrá un fichero cabecera, de nombre *RecIvory.h*, donde figurará una línea con la cabecera para su implementación de cada uno de los métodos nativos declarados en la clase *RecIvory.java*.

Los métodos nativos C que implementan las funciones del reconocedor deben ser convenientemente adaptados para hacer posible su invocación desde aplicaciones JAVA así como el intercambio de información con las mismas. Las modificaciones introducidas se centran fundamentalmente en estas dos cuestiones.

Antes de abordar las operaciones de adaptación en los métodos nativos propiamente dichos, es importante indicar que deben incluirse dos archivos de cabecera fundamentales en él que contiene a los citados métodos. Estos dos archivos son los siguientes:

- *recIvory.h* Archivo de cabecera generado en el paso anterior y que contiene los prototipos que deben adoptarse para la implementación de los métodos nativos declarados en la clase JAVA.
- *jni.h* Este archivo de cabecera proporciona información que el código en lenguaje nativo necesita para interactuar con el sistema de ejecución de JAVA. Es automáticamente incluido en el archivo anterior.

Además deberán incluirse también los archivos con los libros de códigos y modelos de Markov obtenidos para las palabras correspondientes a los contextos de trabajo establecidos.

La declaración de los métodos nativos en el código para su implementación debe corresponderse con los prototipos generados en el archivo de cabecera y sus identificadores son el resultado de la concatenación de los siguientes componentes (JNI API Specification):

- El prefijo Java_
- El nombre del paquete al que pertenece la clase si ésta estuviera incluida dentro de alguno.
- Un símbolo de subrayado "_" como separador.
- El nombre completo de la clase.
- Un símbolo de subrayado "_" como separador.
- El nombre del método.

  Gráficamente, puede representarse de la siguiente forma:

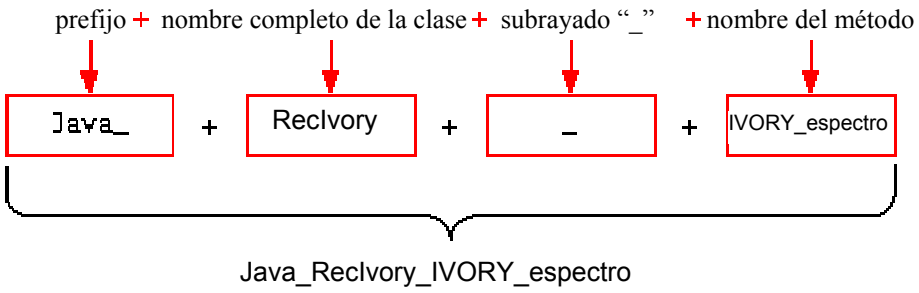


Figura 5.2.: Obtención del identificador de un método nativo.

El interfaz JNI especifica un conjunto de conversiones entre los tipos de datos de Java y los tipos de datos utilizados en los lenguajes nativos. El uso de los tipos de datos de Java en métodos nativos será necesario cuando se pretenda llevar a cabo alguna de las siguientes acciones:

- Acceder, en un método nativo, a los parámetros suministrados desde una aplicación Java.
- Crear objetos Java en el método nativo
- Devolver algún resultado o condición de terminación desde el método nativo.
- Acceso a valores de tipos primitivos Java en los métodos nativos.

Desde un método nativo se puede acceder directamente a valores expresados en tipos primitivos Java tales como valores enteros, lógicos, en coma flotante, etc... Por ejemplo, los tipos *boolean* y *float* de Java son convertidos, respectivamente, a los tipos *jboolean* y *jfloat* del lenguaje nativo.

Los objetos Java se suministran a los métodos nativos mediante referencias a ellos. Todas las referencias a objetos Java se declaran del tipo *jobject*. Por conveniencia y para evitar errores de programación, el interfaz JNI implementa un conjunto de tipos de referencias estructurado en subclases que derivan de *jobject*.

JNI proporciona funciones que pueden ser utilizadas por los métodos nativos para acceder y manipular las variables miembros o atributos, tanto de instancia como de clase, de los objetos Java. Para acceder y manipular dichos atributos se obtiene el identificador del atributo en cuestión a partir de la clase a la que pertenece, de su nombre y un indicador de su tipo. A continuación pueden utilizarse las correspondientes funciones para obtener o modificar el valor contenido por el atributo cuyo identificador se ha obtenido.

El acceso y capacidad de manipulación, desde un método nativo, de los elementos de un array suministrado por la aplicación Java resulta fundamental para la integración de la aplicación de captura de voz desarrollada con las funciones C del reconocedor. Estos módulos necesitan intercambiar gran cantidad de información consistente en colecciones de datos que se enviarán y recibirán contenidas en los correspondientes arrays.

JNI utiliza los tipos de datos *j<TipoPrimitivo>array* para declarar referencias a arrays Java que contienen elementos de algún tipo primitivo, como valores enteros, en coma flotante, lógicos, etc. Desde el código nativo C no se puede tener acceso directo a los elementos de arrays así declarados, siendo necesario, para ello, el uso de funciones expresamente proporcionadas por JNI (JNI API Specification). Mediante otro tipo de funciones proporcionadas por JNI se podrá liberar la memoria asignada al array.

JNI permite obtener un puntero a **todos** los elementos almacenados en el array, proporcionando a tal efecto un conjunto de funciones cada una correspondiente a uno de los posibles tipos de datos primitivos. Una vez obtenido este puntero, podrán utilizarse las operaciones convencionales del lenguaje C para el acceso y manipulación de los elementos del array.

Cuando un método nativo obtiene acceso al array suministrado desde una aplicación Java, JNI permite especificar la forma en que dicho array va a ser accedido por el

método nativo en cuestión. De esta forma, el citado método podrá acceder a una copia del array mantenido por la aplicación Java, o bien, podrá referenciar directamente a los elementos del mismo. Los requisitos en velocidad de cómputo que exige un procesamiento ágil de la señal de voz para poder ofrecer al usuario una respuesta en tiempo real, obligan a que cualquier intercambio de información entre la aplicación de captura de voz y las funciones de reconocimiento se realice siempre a través de referencias a los datos.

Algunas de las funciones del reconocedor reciben como argumento una matriz con datos en coma flotante que almacena los valores que constituyen las plantillas anteriores y posteriores a la correspondiente a la ventana procesada.
La declaración y tratamiento de esta matriz en la aplicación Java no presenta mayores dificultades, sin embargo, no ocurre lo mismo en el lado nativo (JNI API Specification).

El procedimiento de acceso a los elementos de la matriz en el código nativo viene determinado por la consideración de ésta como un array de elementos que a su vez son colecciones de datos. Esta consideración de la matriz es soportada por JNI mediante el tipo general *jobjectArray* que permite declarar arrays que contengan referencias a objetos en vez de datos de algún tipo primitivo(JNI API Specification]). Así pues, las funciones del reconocedor que precisan recibir la citada matriz de plantillas, lo harán a través del correspondiente parámetro declarado de tipo *jobjectArray*.

JNI no permite obtener un puntero al conjunto completo de referencias almacenado en un array de objetos, como en el caso de los tipos primitivos, sino que debe obtenerse acceso a cada una de ellas de forma individual.

Los requisitos en materia de seguridad impuestos por Java implican severas restricciones que prohíben, por ejemplo, el uso de apuntadores, o comúnmente punteros, que podrían constituir una vía de acceso a zonas del sistema fuera del control de la JVM**,** con lo que éste vería su integridad seriamente comprometida.

Con el fin de mantener las ventajas que, en el entorno de desarrollo de aplicaciones C, supone la utilización de punteros en el interfaz de una biblioteca de funciones y, al mismo tiempo, hacer posible la invocación de las mismas desde una aplicación escrita en un lenguaje como Java, se ha diseñado un mecanismo de simulación de punteros que aprovecha las características de la transmisión de información descrita para los arrays.

Así pues, cuando un método nativo acepta, como argumento, una referencia a un objeto de tipo array, está recibiendo un apuntador a sus elementos con el que poder acceder a

ellos para consultarlos o modificarlos según convenga. Esta es la filosofía de utilización de los punteros tradicionales C.

El mecanismo diseñado se basa en la conclusión de que para suministrar un apuntador a un método nativo de forma que éste pueda, a través de él, devolver algún tipo de resultado, el elemento referenciado deberá formar parte de un array.

Una vez adaptados los métodos nativos para hacer posible su correcta invocación desde las aplicaciones JAVA, se procede a su compilación para obtener una biblioteca con el correspondiente código objeto. Como resultado del proceso de compilación se obtendrá una biblioteca compartida (*shared library*) en sistemas como Solaris o una biblioteca de enlaces dinámicos (*dynamic link library* o *DLL*) si se trata de un sistema bajo Windows.
Si bien el sistema presentado en esta memoria ha sido diseñado para ser adoptado por aplicaciones más complejas resulta perfectamente posible realizar su ejecución de forma autónoma. Para ello, bastará, sencillamente, con dotar del método *main()* a la clase *AudioLoopC*. Este método creará una instancia de la propia clase *AudioLoopC* indicando las propiedades del formato de audio de la señal humana de voz. Una vez instanciado este objeto bastará con activar el hilo de control encargado de capturar y preprocesar la señal de voz para suministrarla a continuación a las funciones del reconocedor.

6. Integración del sistema de adquisición y reconocimiento de voz en una aplicación.

Para mostrar la facilidad y viabilidad de la integración del sistema presentado en una aplicación más compleja, se presenta a continuación el conjunto de acciones necesario para completar dicha integración.

El primer paso consistirá en añadir las clases correspondientes al sistema de adquisición y reconocimiento al conjunto de clases que conformen las aplicación general.
En la clase principal de la aplicación general se declarará e instanciará un objeto de la clase *AudioLoopC* suministrando los datos del formato de audio requerido para la señal de voz. Así mismo, se activará el hilo de control que pondrá en funcionamiento el sistema de captura de reconocimiento.

La aplicación deberá proporcionar al módulo de reconocimiento de voz el contexto de trabajo en que se encuentre el usuario en cada instante. Este dato determinará el conjunto de palabras, o vocabulario activo, que puede reconocerse en cada situación del manejo de la aplicación. Esta información se suministrará a través del método que la clase *AudioLoopC* define con tal propósito.

Es importante destacar que la ejecución de este hilo de control, por haberse implementado como tal, no interfiere de forma alguna con el resto de funcionalidades de la aplicación. Permitiendo a partir de su ejecución que la aplicación pueda ser controlada tanto por los dispositivos habituales (teclado, ratón,…) como por la voz del usuario. En ningún caso la aplicación quedará bloqueada a la espera de recibir indicaciones verbales del usuario, permitiéndose su normal funcionamiento en ausencia de éstas.

Cuando el autómata de reconocimiento obtiene un modelo ganador y da por reconocida la palabra a la que pertenece, devuelve el código de ésta al modulo JAVA de captura. Desde este módulo se comunicará el resultado del proceso de reconocimiento a la aplicación general.

7. Trabajos Futuros.

La incorporación de configuración dinámica del vocabulario o vocabularios a reconocer supondría una característica de gran interés para el sistema de captura y reconocimiento presentado en esta memoria.

Añadir nuevas palabras que puedan ser reconocidas conllevaría, en el estado actual del sistema, la recopilación de nuevas realizaciones de voz para ser incorporadas a la base de datos de palabras. Así mismo, sería necesario generar los correspondientes libros de códigos y modelos de Markov y por último recompilar la biblioteca de funciones del reconocedor con esta nueva información.

Una posible solución consistiría en mantener una base de datos de libros de códigos y modelos de Markov correspondientes a un número considerable de palabras. De esta forma el usuario o aplicación obtendría de ella los códigos y modelos de las palabras que pueden ser reconocidas. Este enfoque no resuelve el problema en caso de que la información relativa a la nueva palabra a reconocer no figure en la base de datos y se requeriría el mismo trabajo descrito antes.

Un paso más adelante consistiría en contar con la posibilidad de generar automáticamente, a partir de una o pocas realizaciones de una palabra, un conjunto significativo de muestras que reflejaran variaciones en el tono, timbre, modulación, acento, etc… Esta posibilidad puede quizás obtenerse mediante técnicas de programación genética y deja abierta una puerta a la automatización y autonomía completas del sistema.

8. Bibliografía.

DIAZ MARTIN, JC. GARCIA ZAPATA, JL. RODRIGUEZ GARCIA, JM. ALVAREZ SALGADO, JF. ESPADA BUENO, P. GÓMEZ VILDA, P. (2001): Scalable Distributed Voice Recognition via Cluster Computing. Actas del Simposio de Informática y Telecomunicaciones SIT2001, pp. 163-171

GÓMEZ, P., MARTÍNEZ, R., ALVAREZ, A. AND RODELLAR,V (2000) El Proyecto IVORY: Reconocimiento de Voz Robusto al Ruido. Investigación y Ciencia, pp. 38-39

GÓMEZ, P., MARTÍNEZ, R., ALVAREZ, A., NIETO; V. AND RODELLAR,V (1999): A Hybrid Signal Enhancement Method for Robust Speech Recognition. Proc. of the Workshop on Robust Methods for Speech Recognition in Adverse Conditions, pp. 203-206.

GÓMEZ, P., MARTÍNEZ, R., ALVAREZ, A., NIETO; V. AND RODELLAR,V (1999): A Noise-Robust Speech Processing and Recognition Development System. European Multimedia, Microprocessor Systems and Electronic Commerce EMMSEC'98 pp. 803-810.

GÓMEZ, P., MARTÍNEZ, R., ALVAREZ, A., NIETO; V. AND RODELLAR,V (1999): A DSP-based modular architecture for Noise Cancellation and Speech Recognition. 1998 IEEE Int. Symposium on Circuits and Systems, ISCAS'98, pp. V/178-181.

RABINER, L., JUANG, B.H. (1999). Fundamentals of Speech Recognition.Prentice-Hall, New Jersey.

SCHILDT, H. (2001) : *JAVA 2 Manual de Referencia.* McGraw-Hill

9. Webgrafía:

JAVA 2 Platform, Standard Edition, v 1.3.1. API Specification. Disponible on-line en http://java.sun.com/j2se/1.3/docs/api/index.html.

Java Native Interface Specification: Disponible en on-line en http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/jniTOC.doc.html.