**UNIVERSIDAD POLITÉCNICA DE PUEBLA**

PROGRAMA ACADÉMICO DE

INGENIERÍA EN INFORMÁTICA

# Integración de un Dispositivo de Comunicación Bluetooth al Reconocedor de Voz Sphinx 4

*Raúl Corona Xolaltenco*

Reporte Técnico 31-12-09

COMITÉ EVALUADOR

C.Dr. Javier Velásquez Sandoval (*Asesor*)

Dr. Luis Alberto Morales Rosales (*Sinodal*)

M.C. Javier Caldera Miguel (*Sinodal*)

*PROFESOR(A) DE PROYECTO DE INVESTIGACIÓN I*

Dra. María Auxilio Medina Nieto

Juan C. Bonilla, Puebla

Octubre 2009

# Índice

# Capítulo 1. Planteamiento del Problema de Investigación

## 1.1. Introducción

Durante las últimas décadas se ha estudiado la posibilidad de desarrollar interfazs hombre computadora controlados por la voz para sustituir en ciertas aplicaciones a las interfazs tradicionales basadas en teclados, ratón o dispositivos similares [Villamil 2005]. El reconocimiento automático de voz y la informática son campos de investigación que tratan de introducirse a nuevas áreas para dar apoyo a las necesidades que tienen los ingenieros y usuarios quienes necesitan innovar la comunicación entre dispositivos electrónicos.

Comúnmente, cuando se realiza alguna interfaz para la comunicación entre un dispositivo electrónico y una computadora, se utilizan cables en la interacción. Por esa razón, este proyecto propone eliminar esos cables que algunas veces pueden ser estorbosos. Se puede utilizar tecnología como reconocedores de voz y dispositivos inalámbricos para poder tener una interacción entre dispositivos electrónicos y una computadora.

Los reconocedores de voz actuales manejan cada vez vocabularios más grandes que logran menores tasas de error en reconocimiento y menores tiempos de procesamiento. Por otro lado, los dispositivos electrónicos inalámbricos como bluetooth cada vez son más utilizados en la vida cotidiana. Esto es un punto a favor para el desarrollo del proyecto, pues la integración de tecnología de voz con tecnología inalámbrica podría ayudar a innovar la interfaz que se tiene entre una computadora y un dispositivo electrónico. A la fecha, no se cuenta con el dispositivo electrónico específico, pero se espera contar con uno a corto plazo para poder realizar pruebas necesarias.

El proyecto propone el uso del sistema sphinx 4 como reconocedor de voz al cual se le adaptará un dispositivo inalámbrico bluetooth para establecer y poder enviar órdenes a un dispositivo electrónico. De esta forma, el objetivo general del proyecto es el siguiente:

## 1.2. Objetivo general

Integrar al sistema reconocedor de voz Sphinx 4 un dispositivo de comunicación bluetooth

## 1.3. Objetivos específicos

- Usar los modelos acústicos del habla hispana que genera el modulo Training de Sphinx.
- Implementar los modelos acústicos generados para reconocer la voz e interpretar las palabras que se mencionan al hablar en un micrófono.
- Establecer comunicación entre el bluetooth y el reconocedor de voz para ejecutar las órdenes que serán enviadas al dispositivo electrónico.
- Desarrollar clases necesarias para la manipulación del mecanismo de comunicaciones inalámbrica "bluetooth" empleando el API JSR-82
- Definir un conjunto de comandos para desarrollar pruebas al reconocedor automático de voz
- Ejecutar comandos de voz en un dispositivo habilitado con el reconocedor de voz y con la interfaz bluetooth .

## 1.4. Justificación

Los sistemas informáticos día a día se introducen en diferentes áreas de investigación como la medicina, la electrónica o la robótica. Esta última es la razón de iniciar el proyecto que pretende apoyar por medio de un sistema de reconocimiento de voz el proceso de comunicación que existe entre un dispositivo electrónico como un robot y un ser humano. Generalmente, esta comunicación se realiza por medio de cables o controles remotos. Algunas veces los cables pueden ser innecesarios, en tanto, los controles remotos están limitados a efectuar tareas básicas como avanzar, retroceder, girar o parar.

Para explorar otro medio de interacción que permita la comunicación entre un robot y un ser humano, el reconocedor podrá ejecutar comandos de voz simples (avanzar, retroceder, girar, etc.) hasta aquellos que impliquen la combinación de los básicos.

El proyecto servirá como base para el desarrollo de nuevas herramientas que ayuden a personas con capacidades diferentes a realizar tareas específicas, por ejemplo, operar una silla de ruedas mediante la voz.

## 1.5. Cronograma de actividades

Para llevar a cabo el proyecto, se proponen las actividades de la Tabla 1 y 2.

**Tabla 1**. Cronograma de actividades para Proyecto de Investigación 1

| No. | Actividad | Enero | | | | Febrero | | | | Marzo | | | | Abril | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | S 1 | S 2 | S 3 | S 4 | S 1 | S 2 | S 3 | S 4 | S 1 | S 2 | S 3 | S 4 | S 1 | S 2 | S 3 | S 4 |
| 1 | Búsqueda de proyecto | | | | | | | | | | | | | | | | |
| 2 | Asignación asesor | | | | | | | | | | | | | | | | |
| 3 | Asignación proyecto | | | | | | | | | | | | | | | | |
| 4 | Revisión de literatura | | | | | | | | | | | | | | | | |
| 5 | Investigación Sphinx 4 | | | | | | | | | | | | | | | | |
| 6 | Investigación API Java Bluetooth | | | | | | | | | | | | | | | | |
| 7 | Configuración Servidor | | | | | | | | | | | | | | | | |
| 8 | Entrenando y decodificando Sphinx 4 | | | | | | | | | | | | | | | | |
| 9 | Primera revisión del protocolo de investigación | | | | | | | | | | | | | | | | |
| 10 | Marco teórico | | | | | | | | | | | | | | | | |
| 11 | Presentación de la propuesta de protocolo de investigación | | | | | | | | | | | | | | | | |
| 12 | Elaborar capitulo de diseño | | | | | | | | | | | | | | | | |

**Tabla 2**. Cronograma de actividades para Proyecto de Investigación 2

| No. | Actividad | Septiembre | | | | Octubre | | | | Noviembre | | | | Diciembre | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | S1 | S2 | S3 | S4 | S1 | S2 | S3 | S4 | S1 | S2 | S3 | S4 | S1 | S2 |
| 13 | Revisión de ejemplos API JSR-82 Bluecove | | | | | | | | | | | | | | |
| 14 | Análisis del Funcionamiento del Bluetooth | | | | | | | | | | | | | | |
| 15 | Definición de órdenes para el robot | | | | | | | | | | | | | | |
| 16 | Programación API Bluetooth con reconocedor de voz Sphinx 4 | | | | | | | | | | | | | | |
| 17 | Implementar el sistema reconocedor de voz con bluetooth al robot | | | | | | | | | | | | | | |
| 18 | Pruebas | | | | | | | | | | | | | | |
| 19 | Elaboración de reporte de investigación | | | | | | | | | | | | | | |

## 1.6. Alcances y limitaciones

Limitaciones

- Como reconocedor de voz se utiliza el CMU sphinx-4
- Para realizar la conexión bluetooth se utiliza el API Bluecove que implementa el estándar JSR-82

Alcances

- Se utilizarán sólo los diccionarios y la gramática habla hispana para reconocer la voz.
- Para probar la comunicación entre el robot y bluetooth, se utilizaran órdenes como avanzar, retroceder, izquierda, derecha o alto.

## 1.7. Recursos

Software

- Sistema Operativo  Windows XP Profesional Service Pack 2
- Interprete Active Perl  free 5.8.8.822
- Servidor Java Ant 1.6.0 o superior
- Microsoft Visual  C + + versión 6,
- Java Development Kit  de Java 2, Standard Edition 5.0 o superior
- IDE Eclipse SDK versión: 3.4.0
- API Java Bluetooth versión 1.1

Hardware

- Procesador 2.00 GHz
- Memoria RAM 1 GB
- USB Bluetooth PC 10 Metros

# Capítulo 2. Marco teórico

Dentro del capítulo 2 se describe los elementos que serán necesarios para desarrollar el proyecto, debido a que se utilizará hardware y software, se dará una explicación para identificar las herramientas que se ocuparán durante la construcción del proyecto. El hardware que se utilizará y que es independiente del procesador y de la memoria, será bluetooth. Éste será de ayuda para realizar la conectividad y hacer la interacción con otro dispositivo. Para el software, independientemente que la computadora cuente con el sistema operativo y los lenguajes de programación, será necesario utilizar un "framework" como reconocedor de voz y un "API" para establecer la comunicación.

## 2.1 CMU Sphinx

CMU Sphinx o también Sphinx es el término general que describe a un grupo de sistemas de reconocimiento de voz desarrollado en la Universidad Carnegie Mellon, es un reconocedor de voz gratuito, el código está patentado bajo licencia GPL (General Licence Public). Sphinx incluye una serie de reconocedores de voz que van desde Sphinx 2 hasta Sphinx 4 [Mitsubishi Electric Research 2009].

## 2.2 Sphinx 4

Sphinx 4 es un reconocedor de voz que está desarrollado en el lenguaje de programación Java, lo que hace que sea multiplataforma. Está basado en Modelos ocultos de Markov (HMM), para reconocer la voz y encontrar la secuencia de palabras o unidades que encaja con mayor probabilidad en la señal de voz de entrada. Sphinx 4 se creó a través de una colaboración conjunta entre la Universidad Carnegie Mellon, Sun Microsystems Laboratories, laboratorios de investigación de Mitsubishi Electric (Merl), y Hewlett Packard (HP), con contribuciones de la Universidad de California en Santa Cruz (UCSC) y el Instituto de Tecnología de Massachusetts (MIT).

## 2.3 Componentes internos del Sphinx 4

Los principales componentes del reconocedor Sphinx 4 que sirven para llevar a cabo el proceso de reconocimiento de voz se ilustran en la Figura 1, la cual muestra el núcleo.

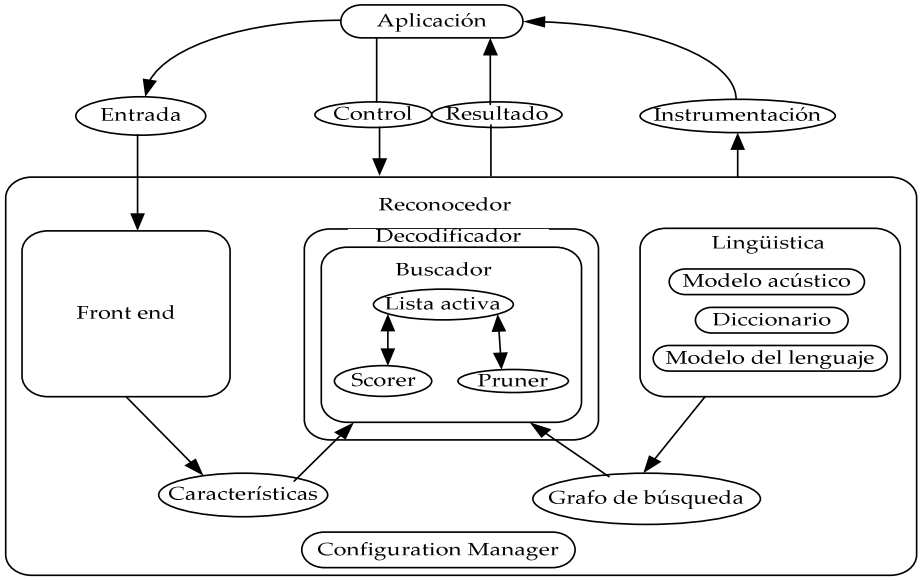


**Figura 1** Arquitectura del reconocedor Sphinx 4 [Mitsubishi Electric Research 2009].

### 2.3.1 Front-end

El propósito del módulo "front-end" es descomponer una señal de entrada de sonido en un conjunto de parámetros que la definen. Como se muestra en la Figura 2, el "front-end" consta de una o más cadenas paralelas de procesamiento de la señal llamadas procesadores de datos, que procesan simultáneamente diferentes tipos de parámetros o distintas señales de entrada. Así, se permite la creación de sistemas que pueden decodificar señales de distinta índole como audio y video a la vez.

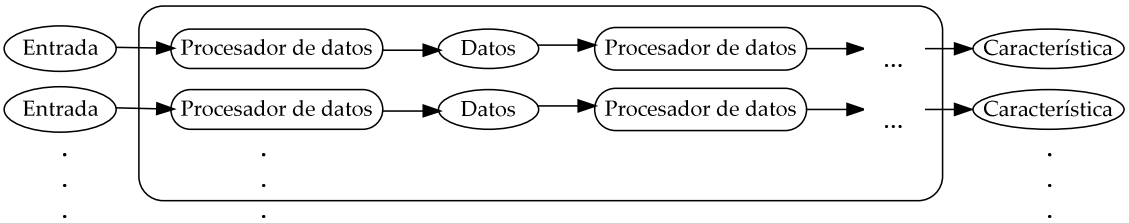


**Figura 2.** Front-end del reconocedor Sphinx 4

### 2.3.2 El decodificador

La principal función del módulo decodificador es usar las características obtenidas por el "front-end" junto con el grafo de búsqueda del bloque lingüístico para generar un resultado posible. Este módulo está formado por un buscador y otro código de soporte que simplifica el proceso de decodificado.

Para el funcionamiento, el decodificador ordena al buscador que reconozca una serie de características. En cada paso del proceso, el buscador crea un objeto resultado que contiene todos los caminos que han alcanzado un estado final no emitido.

### 2.3.3 Lingüística

El módulo lingüística contiene elementos que sirven para poder identificar el tipo de lenguaje. Para el reconocimiento tiene sub módulos llamados modelo acústico, diccionario y un modelo de lenguaje, los cuales generan un grafo que usa el decodificador durante un proceso de búsqueda ocultando la complejidad para encontrar la respuesta más lógica.

#### 2.3.3.1 Modelo del lenguaje

El módulo modelo del lenguaje implementa gramáticas de grafos dirigidos y modelos de N-gramáticas. En una gramática de grafo dirigido, cada nodo representa una palabra individual y cada ciclo la probabilidad de que tenga lugar una transición entre palabras. La Figura 3 es un ejemplo de este tipo de gramática. Por otra parte, los modelos de n-gramáticas estiman las probabilidades de una transición basadas en las n-1 palabras anteriores. La Figura 4 muestra un ejemplo. El modelo del lenguaje limita la búsqueda para identificar lo que se habla en un contexto particular.

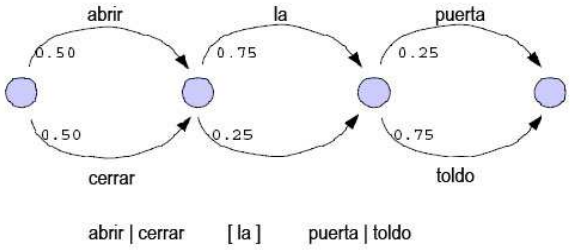

**Figura 3.** Modelo N-gramática [Mitsubishi Electric Research 2009]

### 2.3.3.2 Diccionario

El diccionario contiene la pronunciación de cada palabra incluida en el modelo del lenguaje. La pronunciación descompone cada palabra en una secuencia de sub-palabras encontradas en el modelo acústico. La interfaz del diccionario también soporta la clasificación de palabras, permite la pertenencia múltiple de palabras a grupos.

### 2.3.3.3 Modelo acústico

El modelo acústico ofrece una asociación de cada unidad del habla y un HMM, el cual puede ser etiquetado según las características entrantes que ofrece el "front-end", funciona como una base de datos donde cada modelo estadístico representa una sola unidad de expresión, como una palabra o fonema, y una vez que se analiza, se asocia la palabra más parecida. Esta asociación también puede tener en cuenta información sobre la posición de la palabra en un contexto. Por ejemplo, en el caso de tri-fonemas, el contexto representa los fonemas individuales situados a derecha e izquierda del fonema dado, y la posición de la palabra representa si el tri-fonema está en el principio, a la mitad o al final de la palabra, o bien, si es una palabra en sí mima.

Generalmente, la lingüística rompe cada palabra del vocabulario activo en una secuencia de unidades dependientes del contexto. Después, convierte las unidades y sus contextos al modelo acústico, recuperando los grafos HMM asociados a esas unidades.

## 2.4 Grafo de búsqueda

Aunque la lingüística puede ser implementada de diversas maneras y la topología de los espacios de búsqueda generados pueden variar mucho de una implementación a otra, este espacio está siempre representado por un grafo de búsqueda. En este grafo dirigido, cada nodo, llamado *estado de búsqueda*, representa un estado emitido o uno no emitido. Un estado emitido significa que se encontró la palabras más parecida que se ha dicho etiquetándola para su identificación, cuando no se encuentra se dice que no es emitido.

Los estados emitidos pueden ser etiquetados teniendo en cuenta las características acústicas mientras que los estados no emitidos generalmente son usados para presentar construcciones de alto nivel lingüístico y no serán puntuados directamente. La Figura 4 muestra cómo se hace la búsqueda de la palabra más probable.

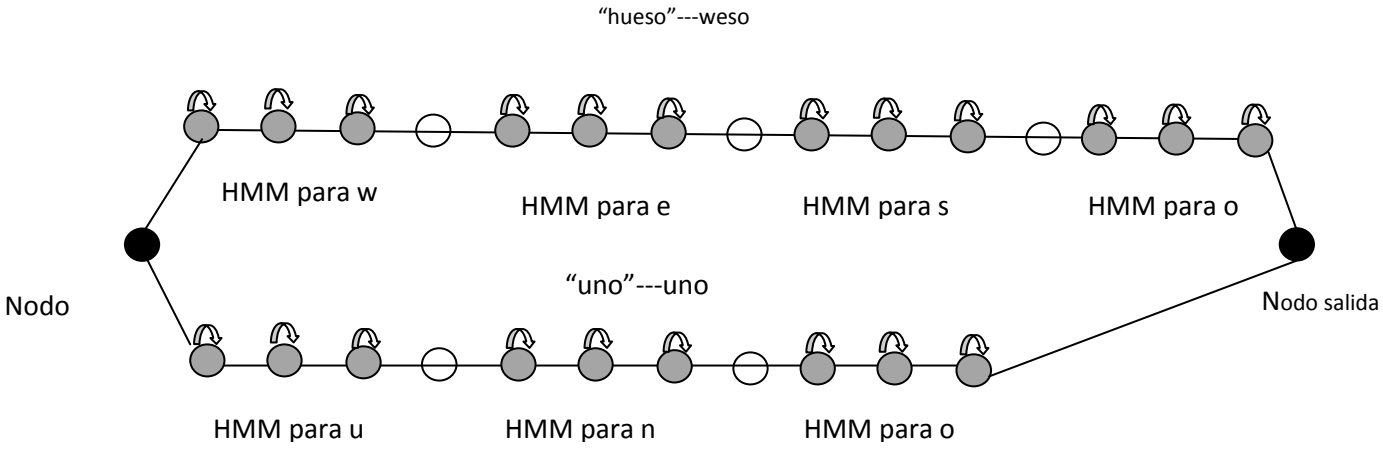


**Figura 4.** Ejemplo de búsqueda en un grafo HMM

En el ejemplo de la Figura 4, la probabilidad de que alguien diga "hueso" es mucho mayor de la que diga "uno". Entonces, la probabilidad de la transición desde el nodo central al primer nodo del HMM para W será del mayor, mientras que la probabilidad de la transición entre el nodo entrante y el nodo que representa el HMM para la U será menor. El camino a "hueso" tendrá, como consecuencia, una respuesta más alta.

## 2.5 Componentes para el entrenamiento

El modelo acústico necesita de modelos que reconozcan el idioma y su pronunciación para que el reconocedor pueda identificar lo se dice. Este modelo se genera a partir de un entrenamiento que se realiza con el módulo SphinxTrain, el cual se describe a continuación.

## 2.6 SphinxTrain

El módulo SphinxTrain (o entrenador) consta de un conjunto de programas, cada uno responsable de una tarea bien definida y un conjunto de scripts para determinar el orden en que se mandan a llamar los programas. El entrenador aprende de los parámetros de los modelos de las unidades de sonido mediante un conjunto de señales de muestra de voz. Este proceso se conoce como "formación de base de datos". Para formar esta base de datos, el entrenador requiere que se le indique las unidades de sonido que desea aprender, o la secuencia en la que se producen en cada discurso.

## 2.7 Bluetooth

El propósito de este capítulo es ofrecer una introducción a bluetooth, donde se explica brevemente sobre la tecnología así como las características principales con las que trabaja un dispositivo bluetooth.

Bluetooth se denomina al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, se basa en enlaces radio de corto alcance y de bajo coste, que faciliten la creación conexiones ad hoc entre terminales tanto móviles como estacionarios [André 2004]. Los dispositivos que lo implementan pueden comunicarse entre ellos cuando se encuentran dentro de un alcance. Las comunicaciones se realizan por radiofrecuencia de forma que los dispositivos no tienen por qué estar alineados, pueden incluso estar en habitaciones separadas si la potencia de transmisión lo permite.

Los dispositivos bluetooth se clasifican en "Clase 1", "Clase 2" o "Clase 3" según la potencia de trasmisión, todas las clases son compatibles con esta tecnología [Bluetooth SIG]. El alcance depende de la clase del dispositivo como se explica a continuación:

- Los radios de clase 3 suelen tener un alcance de entre uno y tres metros.
- Las radios de clase 2 son habituales de los dispositivos portátiles y tienen un alcance de diez metros.
- Las radios de clase 1 se utilizan principalmente en el sector industrial y logran un alcance de cien metros. Usan repetidores de frecuencia.

Las características principales de la tecnología bluetooth son su fiabilidad, bajo consumo y mínimo coste [Bluetooth SIG]. El núcleo del sistema consiste en un transmisor de radio, una banda base y una pila de protocolos. El sistema permite la conexión entre dispositivos y el intercambio de distintos tipos de datos. Para que un dispositivo pueda comenzar la comunicación, es necesario que se trabaje como un modelo cliente-servidor, donde no siempre un solo dispositivo puede ser servidor o cliente, sino que varía de acuerdo a la tarea que se realiza.

### 2.7.1 Arquitectura Bluetooth

La especificación tiene como objetivo permitir que los dispositivos bluetooth de diferentes fabricantes puedan trabajar el uno con el otro [André 2004]. Para esto, la especificación trabaja con la frecuencia de radio, además de que internamente cuenta con un sistema completo de pila de protocolos.

Para los desarrolladores de aplicaciones, el protocolo bluetooth se puede dividir en dos elementos principales: capas y perfiles [Hopkins 2003]. Las capas del protocolo bluetooth de la pila se muestran la Figura 5 y se describen a continuación:

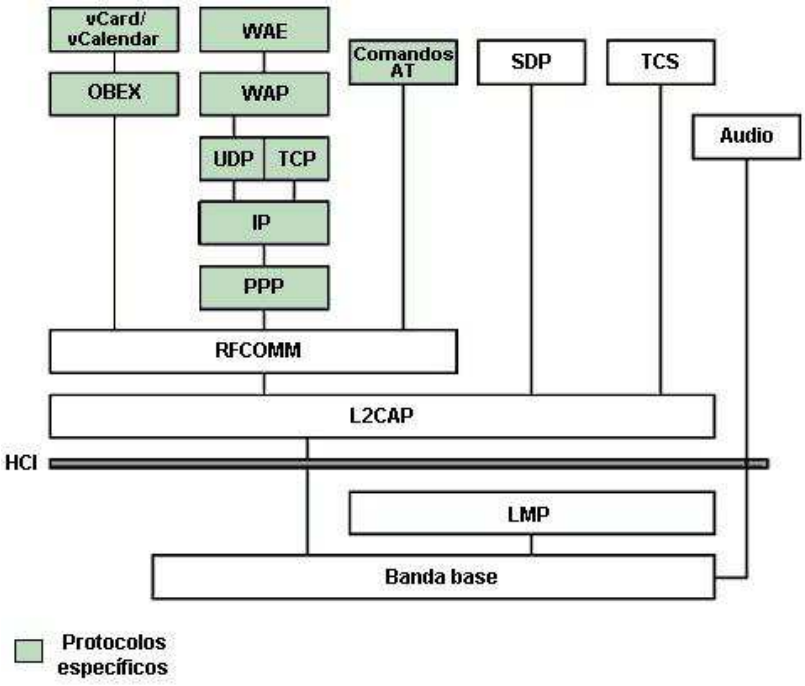


**Figura 5**. Pila de Protocolo Bluetooth

**1.-**La banda base es la parte baja de la pila que especifica o introduce los procedimientos de acceso de medios y capa física entre dispositivos [Bluetooth SIG]. Esta capa implementa el canal físico real, emplea una secuencia aleatoria de saltos a través de 79 frecuencias de radio diferentes. Los paquetes son enviados sobre el canal físico. La Banda Base controla la sincronización de las unidades y la secuencia de saltos, además, es la responsable de la información para el control de enlace a bajo nivel como el reconocimiento o el control de flujo.

**2.-**El Protocolo de Gestión de Enlace (*Link Manager Protocol* o LMP) es el responsable de la autentificación, encriptación, control y configuración del enlace. *LMP* también se encarga del manejo de los modos y consumo de potencia, configuración y el control de comunicaciones lógicas y enlaces lógicos, y el control de los enlaces físicos.

**3.-**La Interfaz del Controlador de Enlace *(Host Controller Interfaz* o HCI) brinda un método uniforme para acceder a los recursos de hardware y a todas las funciones. Contiene una interfaz de comando para el controlador banda base y la gestión de enlace para acceder al hardware.

**4.-**El Protocolo de Control y Adaptación de Enlace Lógico (*Logical Link Control and Adaptation Protocol* o L2CAP) , corresponde a la capa de enlace de datos. Ésta brinda servicios de datos orientados y no orientados a la conexión a capas superiores. *L2CAP* multiplexa los protocolos de capas superiores para enviar varios protocolos sobre un canal banda base. Para manipular paquetes de capas superiores más grandes que el máximo tamaño del paquete, L2CAP los segmenta en varios paquetes. La capa *L2CAP* del receptor reensambla los paquetes banda base en paquetes más grandes para la capa superior. La conexión permite el intercambio de información referente a la calidad de la conexión, además maneja grupos de tal manera que varios dispositivos pueden comunicarse entre sí.

**5.-**El Protocolo de Descubrimiento de Servicio (*Service Discovery Protocol* o SDP) define cómo actúa una aplicación de un cliente para descubrir servicios disponibles de servidores, además de proporcionar un método para determinar las características de dichos servicios.

**6.-**El protocolo comunicación por radio frecuencia (*Radio Frequency Communication* o RFCOMM) ofrece emulación de puertos seriales sobre el protocolo *L2CAP. RFCOMM* emula señales de control y datos *RS-232* sobre la banda base *Bluetooth.* Ofrece capacidades de transporte a servicios de capas superiores (por ejemplo OBject Excahnge u *OBEX*) que usan una línea serial como mecanismo de transporte.

**7.-**El protocolo control de telefonía (*Telephony Control Protocols* o TCS) define la señalización de control de llamadas para el establecimiento y liberación de una conversación o llamada de datos entre unidades. Ofrece funcionalidad para intercambiar información de señalización no relacionada con el progreso de llamadas.

**8.-**La capa de audio sólo envía este tipo de datos. Las transmisiones pueden ser ejecutadas entre una o más unidades usando modelos diferentes. Los datos de audio no pasan a través de la capa *L2CAP,* pero sí directamente después de abrir un enlace y un establecimiento directo entre dos unidades.

**9.-**Comandos AT Control de telefonía. Bluetooth soporta un número de comandos *AT* para el control de telefonía a través de emulación de puerto serial (*RFCOMM*) como AT+CGMI y AT+CBC.

**10.-** Protocolo Punto-a-Punto (PPP) es un protocolo orientado a paquetes, usa un mecanismo serial para convertir un torrente de paquetes de datos en una corriente de datos seriales. Este protocolo corre sobre RFCOMM para lograr las conexiones ppp.

**11.-** Protocolos UDP/TCP – IP permiten a las unidades Bluetooth conectarse a *Internet* a través de otras unidades conectadas. Por lo tanto, una unidad puede actuar como un puente. La configuración *TCP/IP/PPP* está disponible como transporte para *WAP*.

**12.-** Protocolo de aplicación inalámbrica WAP.- Trabaja con una amplia variedad de tecnologías de red inalámbricas conectando dispositivos móviles a internet. Bluetooth puede ser usado como portador para ofrecer el transporte de datos entre el cliente *WAP* y su servidor de *WAP* adyacente. Además, las capacidades de red de bluetooth dan a un cliente *WAP* posibilidades únicas en cuanto a movilidad comparada con otros

portadores *WAP*. Un ejemplo es un almacén que transmite ofertas especiales a un cliente cuando éste entra en el rango de cobertura.

**13.-** Protocolo OBEX.- Es un protocolo opcional de nivel de aplicación diseñado para permitir a las unidades bluetooth soportar comunicación infrarroja.

Además de los protocolos, existen perfiles que se han definido por el Bluetooth SIG. Un perfil define formas de utilización del protocolo de la pila y las características que permiten un uso particular modelo [Timothy 2008]. En otras palabras, para que un dispositivo pueda utilizar la tecnología inalámbrica bluetooth, debe saber interpretar los perfiles que describen las distintas aplicaciones posibles. Estos perfiles son guías que indican los procedimientos por los dispositivos equipados con tecnología bluetooth. Al seguir las directrices proporcionadas en las especificaciones, los desarrolladores pueden crear aplicaciones compatibles con otros dispositivos que se ajusten a este estándar [Bluetooth SIG].

Cada perfil incluye como mínimo información sobre las siguientes cuestiones:

- Dependencia de otros perfiles
- Propuestas de formato de interfaz de usuario
- Características concretas de la pila de protocolos bluetooth utilizada por el perfil.

Para realizar su función, cada perfil se sirve de ciertas opciones y parámetros en cada capa de la pila. También se puede incluir un breve resumen de los servicios requeridos si resulta necesario

## 2.7.2 Red Bluetooth

Una red bluetooth es una red Personal Area Network (PAN) ya que por lo regular un dispositivo puede hacer comunicación en un rango no mayor a 10 metros, la red que se genera entre dispositivos bluetooth se llama piconet, la cual se compone de un maestro y uno o más esclavos como muestra la Figura 6. El dispositivo al iniciar automáticamente una conexión bluetooth se convierte en el maestro. Una piconet puede constar de un maestro y hasta siete esclavos activos. Los esclavos sólo podrán transmitir datos en tiempo de transmisión cuando lo conceda el dispositivo maestro; los esclavos

no podrán comunicarse directamente entre sí, toda comunicación debe estar orientada a través del maestro [André 2004].

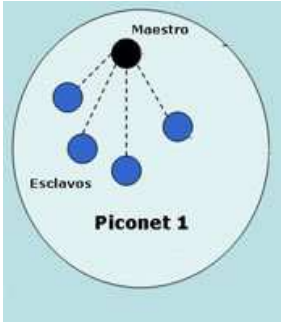


**Figura 6.** Ejemplo de una piconet

Al conectar dos piconets el resultado será un scatternet. La Figura 7 se puede describir como una estructura de piconets múltiples. Dado que la especificación bluetooth soporta tanto conexiones punto a punto como punto a multipunto, se pueden establecer y enlazar varias piconets en forma de scatternet. Las piconets pertenecientes a una misma scatternet no están coordinadas y los saltos de frecuencia suceden de forma independiente, es decir, todos los dispositivos que participan en la misma piconet se sincronizan con su correspondiente tiempo de reloj y patrón de saltos determinado. El resto de piconets utilizarán diferentes patrones de saltos y frecuencias de relojes distintas, lo que supone distintas velocidades de salto entre canales. Aunque no se permite la sincronización de diferentes piconets, los dispositivos pueden participar en diferentes piconets gracias a una multiplexación por división de tiempo (TDM). Esto permite a un dispositivo participar de forma secuencial en diferentes piconets, estando activo sólo una piconet cada vez [André 2004].

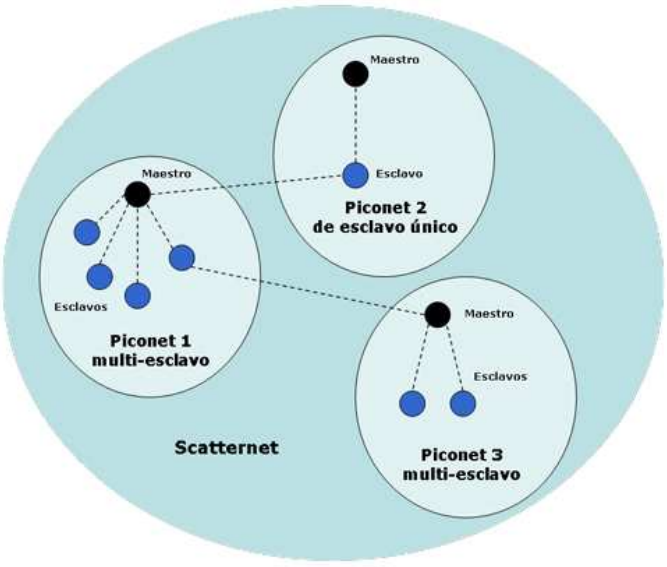


**Figura 7** Ejemplo de una scatternet

### 2.7.3 Bandas de frecuencia y organización de canales

Los dispositivos bluetooth funcionan en la banda de 2,4 GHz, una de las bandas de radio ISM (industrial, científica y médica) que no requieren licencia [Bluetooth SIG]. Los dispositivos usan un transmisor de salto de frecuencia para contrarrestar las interferencias y la pérdida de intensidad. Se definen dos modos de modulación: mn modo obligatorio, llamado *modo de transferencia básica*, que usa una modulación de frecuencia binaria para reducir al mínimo la complejidad del transmisor/receptor. La tasa de transferencia de símbolos de todas las secuencias de modulación es de 1 Ms/s. La velocidad de transmisión aérea total es de 1 Mbps con el modo de transferencia básica; 2 Mbps con la transferencia de datos mejorada; y 3 Mbps con la transferencia de datos mejorada. Para la transmisión bidireccional se emplea una técnica de dúplex por división de tiempo (TDD) en ambos modos. Esta especificación define los requisitos de una radio Bluetooth, tanto para el modo de transferencia básica como de transferencia mejorada de datos.

El sistema bluetooth funciona en la banda ISM de 2,4 GHz. Esta banda de frecuencias abarca 2400 - 2483,5 MHz. Los 79 canales RF se organizan por números de 0 a 78 con un espacio de 1 MHz entre ellos, empezando por 2402 GHz.

## 2.8 JSR-82

El JSR-82 es un estándar para la integración inalámbrica de dispositivos pequeños. Normaliza la especificación de un Java API que permiten a los dispositivos integrarse en un entorno Bluetooth [Sun Microsystems]. El objetivo de esta especificación es definir la arquitectura de las APIs necesarias para manipular dispositivos con bluetooth dentro de un entorno de desarrollo de aplicaciones. El estándar se divide en dos paquetes independientes: javax.bluetooth y javax.obex.

El JSR-82 es la base para poder crear APIs que puedan ser implementadas en Java. Dos APIs que pueden utilizarse son Bluecove y Marge, las cuales pueden ser utilizados para programar en Java 2, Standard Edition (J2SE).

### 2.8.1 javax.bluetooth

El paquete javax.bluetooth define clases e interfazs básicas para el descubrimiento de dispositivos y servicios; se encarga de la conexión y comunicación. La comunicación es de bajo nivel porque se realiza mediante flujos de datos o mediante la trasmisión de arreglos de bytes [Hopkins 2003].

Javax.bluetooth cuenta con 13 clases e interfazs para llevar a cabo la comunicación inalámbrica [Hopkins 2003]. La Tabla 3 y 4 contiene una breve descripción de la tarea que realiza cada clase.

**Tabla 3**. Interfazs del paquete javax.bluetooth [Sun Microsystems 2002].

| **Interfaz** | **Descripción** |
|---|---|
| DiscoveryListener | Permite hacer solicitudes a los dispositivos para descubrir los servicios. |
| L2CAPConnection | Representa una relación orientada al canal L2CAP. |
| L2CAPConnectionNotifier | Provee una notificación de conexión con L2CAP. |
| ServiceRecord | Describe características de un servicio para bluetooth. |
| DataElement | Define los distintos tipos de datos que un servicio puede atender. |

**Tabla 4**. Clases del paquete javax.bluetooth [Sun Microsystems 2002].

| **Clase** | **Descripción** |
|---|---|
| DeviceClass | Representa la clase de dispositivo (COD) de registro, tal como se define por la especificación. |
| DiscoveryAgent | Contiene métodos para realizar el descubrimiento de dispositivos y servicios. |
| LocalDevice | Representa el dispositivo local. |
| RemoteDevice | Representa un dispositivo remoto. |
| UUID | Define identificadores únicos universales. |
| BluetoothConnectionException | Se lanza cuando una conexión Bluetooth (L2CAP, RFCOMM, OBEX) no puede establecerse con éxito. |
| BluetoothStateException | Se lanza cuando se hace una solicitud al sistema de Bluetooth porque el sistema no puede apoyar en su estado actual. |
| ServiceRegistrationException | Se lanza cuando hay un fracaso para agregar un servicio de registro a la base de datos local de Service Discovery (SDDB) o modificar un servicio existente en el registro SDDB. |

En una comunicación con el paquete java.bluetooth**,** existe un dispositivo que ofrece un servicio también conocido como servidor y otros dispositivos que acceden a él (clientes).

Un cliente deberá realizar las siguientes tareas:

- *Búsqueda de dispositivos*. La aplicación realizará una búsqueda de los dispositivos a su alcance que estén en modo conectable.
- *Búsqueda de servicios*. La aplicación realizará una búsqueda de servicios por cada dispositivo.
- *Establecimiento de la conexión*. Una vez encontrado un dispositivo que ofrece el servicio deseado, se conecta a él.
- *Comunicación*. Ya establecida la conexión se puede leer y escribir en ella.

Un servidor deberá hacer las siguientes operaciones:

- Crear una conexión servidora
- Especificar los atributos de servicio
- Abrir las conexiones clientes

### 2.8.2 java.obex

El paquete java.obex permite manejar el protocolo de alto nivel OBEX. Este protocolo es muy similar al HTTP y es utilizado para el intercambio de archivos [Hopkins 2003]. OBEX no sólo es utilizado para hacer comunicación bluetooth, también es utilizado sobre otras tecnologías diferentes como internet.

OBEX consiste en el intercambio de mensajes entre el cliente y el servidor. Tales mensajes están formados por cabeceras y opcionalmente un cuerpo. En este protocolo el cliente envía comandos al servidor como CONNECT, PUT, GET, DELETE, SETPATH, DISCONNECT [Huang 2007].

Las 8 clases de javax.obex son necesarias para enviar los objetos entre los dispositivos, independientemente del mecanismo de transporte entre ellas. Estas clases se describen en la Tabla 5 y 6.

**Tabla 5.** Interfazs del paquete javax.obex [Sun Microsystems 2002].

| **Interfaz** | **Descripción** |
|---|---|
| Authenticator | Proporciona una manera de responder a la autenticación y respuesta de cabeceras. |
| ClientSession | Proporciona métodos para solicitudes. |
| HeaderSet | Define los métodos establecidos y obtiene los valores de las cabeceras. |
| Operation | Proporciona los medios para manipular unA OPERACIÓN PUT o GET. |
| SessionNotifier | Define una conexión del lado del servidor. |

**Tabla 6**. Clases del paquete javax.obex [Sun Microsystems 2002].

| **Clase** | **Descripción** |
|---|---|
| PasswordAuthentication | Combinación de nombre de usuario y contraseña. |
| ResponseCodes | Contiene la lista de códigos de respuesta válida de un servidor que envía un cliente. |
| ServerRequestHandler | Define un evento que responde a las peticiones hechas al servidor. |

# Capítulo 3. Diseño de investigación

## 3.1 Introducción

El capítulo de diseño describe los requerimientos del proyecto con un plan de acción que divide el trabajo de tal forma que facilite las tareas y así poder tener resultados positivos con mayor facilidad. Dado que el diseño de investigación es un plan de acción dentro de este capitulo se describe los requerimientos del sistema así como sus casos de uso y de prueba.

## 3.2 Requerimientos

Los requerimientos funcionales son los siguientes:

1. Definir un diccionario con las palabras que identificará el reconocedor de voz
2. Procesar la orden que se da en voz para obtener una variable
3. La variable obtenida será envida al dispositivo electrónico como una orden mediante comunicación bluetooth
4. Realizar búsqueda de dispositivos con el API Bluecove
5. Realizar búsqueda de servicios en dispositivos Bluetooth
6. Realizar comunicación entre dispositivos bluetooth
7. Mostrar el resultado de la orden que se envía a través del bluetooth atreves de un mensaje en pantalla.

**Requerimientos no funcionales**

1. *Eficiencia:* Debido a las características del hardware que se utilizará, la respuesta que deberá tener cuando se realice una orden deberá ser casi instantánea.
2. *Plataforma:* A pesar de que las herramientas de software pueden ser utilizadas en diferentes sistemas operativos, el proyecto se realizará en plataforma Windows
3. *Aspectos legales:* Las herramientas de desarrollo serán de acceso libre.

4. *Extensibilidad:* Se implementarán módulos de forma que la adición de otros requerimientos funcionales pueda llevarse a cabo con facilidad relativa.
5. *Mantenimiento:* El mantenimiento deberá realizarse periódicamente para prevenir posibles fallas en la integración del reconocedor de voz y bluetooth.
6. *Compatibilidad:* Solo será disponible para versiones Windows pero en algún momento podrá ser compatible con más de un sistema operativo.

## 3.3. Casos de Uso

Este apartado contiene tablas que describen cómo trabajará el sistema mediante la descripción de casos de uso (C.U.) y casos de prueba (C.P.).

**Tabla 7.** Caso de uso decir-orden.

| | |
|---|---|
| **Descripción:** | El usuario podrá decir palabras que se asocien a las órdenes que se hallan definido para que el reconocedor de voz las identifique y así generar la variable que servirá para ser trabajada en el API de Java. |
| **Datos Entrada:** | Palabras definidas como órdenes: avanza, adelante, izquierda, derecha, parar. |
| **Datos Salida:** | Se enviará una variable al API si se identificó la orden o se mostrará el mensaje "no se encuentra esta orden" en caso que no sea reconocida la orden. |

**Tabla 8.** Caso de uso implementar-orden.

| | |
|---|---|
| **Descripción:** | El caso de uso espera que el C.U. Decir-orden genere la variable de acuerdo a lo que dijo el usuario. Cuando haya enviado la variable, ésta se enviará a otro dispositivo electrónico mediante comunicación bluetooth. |
| **Datos Entrada:** | Variable que genera C.U. decir-orden |
| **Datos Salida** | Enviará una señal para que el dispositivo electrónico muestre un movimiento como respuesta. |

## 3.4. Casos de Pruebas

**Tabla 9.** Caso de prueba para el caso de uso decir-orden.

| | |
|---|---|
| **Datos Entrada:** | Orden "avanza" |
| **Datos Salida:** | Si el reconocedor identifica la orden, entonces se enviará una variable *avanzar* con valor de 1. Si no se identifica entonces mostrara un mensaje informando que no se reconoce la que se dice. |

**Tabla 10.** Caso de prueba para el caso de uso implementar-orden

| | |
|---|---|
| **Nombre:** | C.P.Implementar-orden para C.U.Implementar-orden |
| **Datos Entrada:** | Avanzar=1 |
| **Datos Salida** | Si avanzar es igual a 1, se enviará una señal vía bluetooth para que el dispositivo electrónico inicie un trabajo. Si no se logra la enviar la señal se mostrara un mensaje informando que lo intente nuevamente. |

# Capítulo 4. Implementación

## 4.1 Introducción

Cada tema de este capítulo describe las tareas implementadas que forman del funcionamiento del software …. PONER EL NOMBRE AQUÍ.  HERE I AM!!

## 4.1 Crear .jar con modelos acusticos

Para utilizar los modelos acústicos que se generar en el apéndice C y tenerlos como una librería mas, es necesario crear un archivo .jar y así poder manipularlos. Para crear este archivo se consideró como referencia el manual "How to Use Models from SphinxTrain in Sphinx-4"[1] que ofrece el grupo de trabajo cmusphinx, donde muestra los pasos siguientes:

**Paso1:** Crear  Directorio para el modelo

Se crea mi directorio llamado toy dentro de los subdirectorios de la caperta sphinx-4-1\models\acoustic como se muestra en la Figura 8.

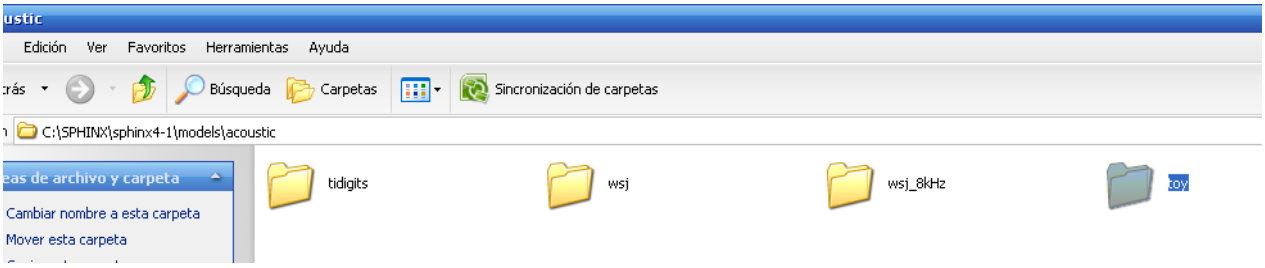


**Figura 8.** Crear directorio de prueba llamada toy

**Paso 2.** Copiar archivos de modelos acústicos a la caperta toy

En este paso se crea los subdirectorios dentro de la carpeta model\acoustic\toy\ con los archivos que se obtuvieron llamados 8g, H4.2500.mdef, h4.dict, filler.dict
Se crea el directorio cd_continuous_8gau aquí se depositaran los archivos que contiene el paquete 8g  como se ve en la figura 9.

[1] Sitio web para acceder al manual de referencia How to Use Models from SphinxTrainig in Sphinx-4 http://cmusphinx.sourceforge.net/sphinx4/doc/UsingSphinxTrainModels.html

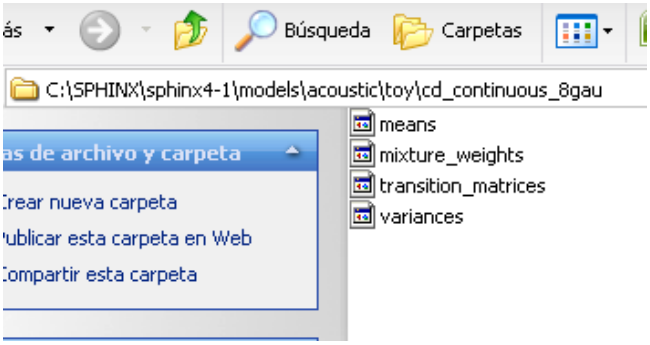


**Figura 9.** Ubicación de los archivos empaquetados en 8g

Posteriormente se crea una carpeta dict donde se ubican los archivos filler.dict y h4.dict. (Ver la Figura 10).

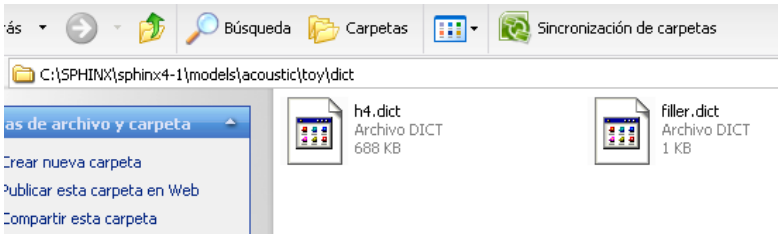


**Figura 10.** Ubicación de los archivos h4.dict y filler.dict

Después se crea un directorio llamado etc que contiene al archivo H4.2500.mdef. (Ver la Figura 11).

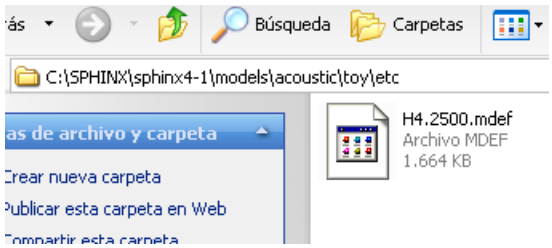


**Figura 11.** Ubicación de los archivos h4.2500.mdef

**Paso 3.** Crear archivo model.props

En este paso se crea un archivo llamado model.props y se ubica en la carpeta toy. Su contenido se muestra a continuación y la descripción de las propiedades se muestra la Tabla 11:

```
description = TOY acoustic models
modelClass = edu.cmu.sphinx.model.acoustic.TOY.Model
modelLoader=edu.cmu.sphinx.model.acoustic.TOY.ModelLoa
der
dataLocation = cd_continuous_8gau
modelDefinition = etc/ H4.2500.mdef

isBinary = true
featureType = 1s_c_d_dd
vectorLength = 39
sparseForm = false

numberFftPoints = 512
numberFilters = 40
gaussians = 8
minimumFrequency = 130
maximumFrequency = 6800
sampleRate = 16000
```

**Tabla 11.** Tabla de propiedades para crear el archivo model.props

| | |
|---|---|
| description | Define una descripción del modelo acústico |
| modelClass | Establece"edu.cmu.sphinx.model.acoustic.${VARAIBLE}.Model", donde esta directiva hace ferencia al Framework de Sphinx y VARIABLE es el nombre del .jar que se genera |
| modelLoader | Es similar a modelClass pero esta debe tener como directiva "edu.cmu.sphinx.model.acoustic.${VARIABALE}.ModelLoader" |
| *dataLocation* | Esto corresponde al nombre del directorio donde se encuentran ubicados los archivos de los modelos acústicos |
| *modelDefinition* | Aquí se debe poner la ubicación y el nombre del archivo que tiene los modelos acústicos |
| *isBinary* | Este atributo debe ser puesto en true en caso que los archivos de modelos acústicos sean binarios |
| *featureType* | Esta formación 1s_c_d_dd es colocada por características de compatibilidad con Sphinx-4 y sphinx-3 |
| *vectorLength* | Longitud del vector de características, que normalmente es 39 |
| *sparseForm* | Si las matrices de transición del modelo acústico está en forma dispersa se coloca True, es decir, omitir los ceros de los Estados no de transición. |
| *numberFftPoints* | Número de puntos de FFT usado cuando se crean las características de la formación |
| *numberFilter* | Número de filtros usados al crear características de la formación |
| *gaussians* | Número de gaussianas de los modelos generados |
| *minimumFrequency* | frecuencia minima de los filtros usado cuando se crean las características de la formación |
| *maximumFrequency* | frecuencia máxima de los filtros usado cuando se crean las características de la formación |
| *sampleRate* | Frecuencia de muestreo de los datos de entrenamiento |

Una vez editado el archivo model.props se debe ubicar en la carpeta toy como se ve en la Figura 12.

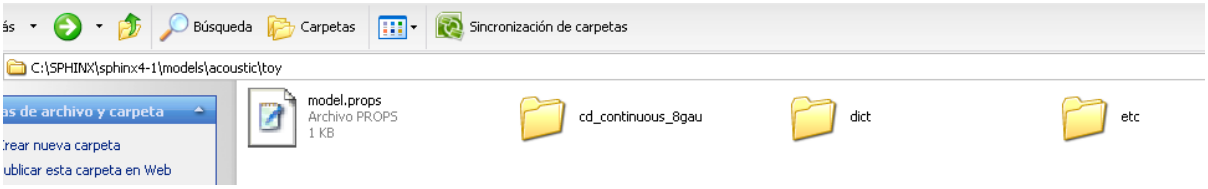


**Figura 12**.Ubicación del archivo model.props

**Paso 4.** Agregar propiedades al archivo build.xml

Se modifica el archivo build.xml donde dice el comentario "For generating the WSJ and TIDIGITS models" con estas líneas:

*<property name="toy_name" value="TOY"/>*
*<property name="toy_data_dir" value="models/acoustic/toy"/>*

```
    <!-- ************************************************************ -->
    <!-- *                                                          * -->
    <!-- * For generating the WSJ and TIDIGITS models.              * -->
    <!-- *                                                          * -->
    <!-- ************************************************************ -->

    <property name="wsj_name" value="WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz"/>
    <property name="wsj_data_dir" value="models/acoustic/wsj"/>

    <property name="wsj_8kHz_name" value="WSJ_8gau_13dCep_8kHz_31mel_200Hz_3500Hz"/>
    <property name="wsj_8kHz_data_dir" value="models/acoustic/wsj_8kHz"/>

    <property name="tidigits_name"
              value="TIDIGITS_8gau_13dCep_16k_40mel_130Hz_6800Hz"/>
    <property name="tidigits_data_dir" value="models/acoustic/tidigits"/>

    <property name="toy_name" value="TOY"/>
    <property name="toy_data_dir" value="models/acoustic/toy"/>
```

**Paso 5.** Crear modelo de clases en build.xml

Se modifican las líneas siguientes del archivo build.xml:

*<antcall target="create_my_model_classes">*
*  <param name="my_model_name" value="${toy_name}"/>*
*</antcall>*

```
    <!-- ************************************************************ -->
    <!-- *                                                          * -->
    <!-- * Create/Delete the acoustic model class files.            * -->
    <!-- *                                                          * -->
    <!-- ************************************************************ -->

    <target name="create_all_model_classes"
        description="Creates all the model class files.">
    <antcall target="create_my_model_classes">
        <param name="my_model_name" value="${wsj_name}"/>
        </antcall>
    <antcall target="create_my_model_classes">
        <param name="my_model_name" value="${wsj_8kHz_name}"/>
        </antcall>
    <antcall target="create_my_model_classes">
        <param name="my_model_name" value="${tidigits_name}"/>
        </antcall>

      <!-- add for me -->
      <antcall target="create_my_model_classes">
        <param name="my_model_name" value="${toy_name}"/>
      </antcall>

    </target>
```

**Paso 6.** Eliminar modelo de clases en build.xml

Se modifican las líneas siguientes del archivo build.xml en la ubicación de delete_all_model_classes

```
<target name="delete_all_model_classes"
        description="Deletes the WSJ and TIDIGITS model classes.">
<antcall target="delete_my_model_classes">
    <param name="my_model_name" value="${wsj_name}"/>
    </antcall>
<antcall target="delete_my_model_classes">
    <param name="my_model_name" value="${wsj_8kHz_name}"/>
    </antcall>
<antcall target="delete_my_model_classes">
    <param name="my_model_name" value="${tidigits_name}"/>
    </antcall>
</target>

<target name="delete_my_model_classes"
        description="Deletes the class files of a model.">
<delete dir="${model_class_dir}/${my_model_name}/"/>

  <!-- add for me -->
 <antcall target="delete_my_model_classes">
   <param name="my_model_name" value="${toy_name}"/>
 </antcall>

</target>
```

**Paso 7.** Crear modelo JAR en build.xml

Se modifican las líneas siguientes del archivo build.xml agregando el código siguiente en la ubicación create_all_models

<antcall target="create_my_model">

*<param name="my_model_data_dir" value="${toy_data_dir}"/>*
*<param name="my_model_name" value="${toy_name}"/>*
*</antcall>*

```
    <!-- ************************************************************ -->
    <!-- *                                                          * -->
    <!-- * Builds the TIDIGITS and WSJ acoustic model files.        * -->
    <!-- *                                                          * -->
    <!-- ************************************************************ -->
    <target name="create_all_models"
        description="Creates the WSJ and TIDIGITS acoustic model JARS.">
        <antcall target="create_my_model">
        <param name="my_model_data_dir" value="${wsj_data_dir}"/>
        <param name="my_model_name" value="${wsj_name}"/>
        </antcall>
        <antcall target="create_my_model">
        <param name="my_model_data_dir" value="${wsj_8kHz_data_dir}"/>
        <param name="my_model_name" value="${wsj_8kHz_name}"/>
        </antcall>
    <antcall target="create_my_model">
        <param name="my_model_data_dir" value="${tidigits_data_dir}"/>
        <param name="my_model_name" value="${tidigits_name}"/>
        </antcall>

     <!-- add for me -->
     <antcall target="create_my_model">
      <param name="my_model_data_dir" value="${toy_data_dir}"/>
      <param name="my_model_name" value="${toy_name}"/>
    </antcall>

    </target>
```

**Paso 8.** Reconstruir archivo .jar

Se escribe ant en el nivel del directorio donde se encuentra el archivo build.xml como se muestra en la Figura 13.

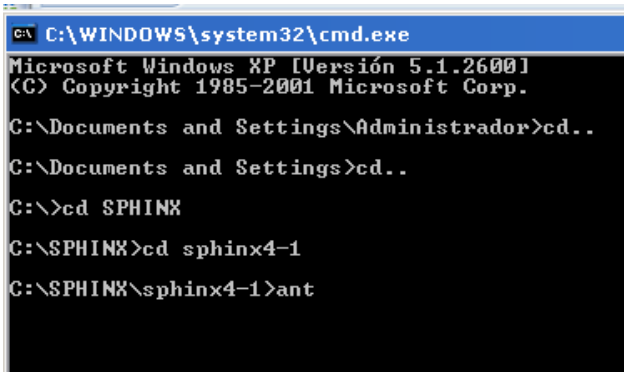


**Figura 13**. Escribir comando ant en carpeta aphinx4-1

El resultado después de teclear ant se muestra en la Figura14.

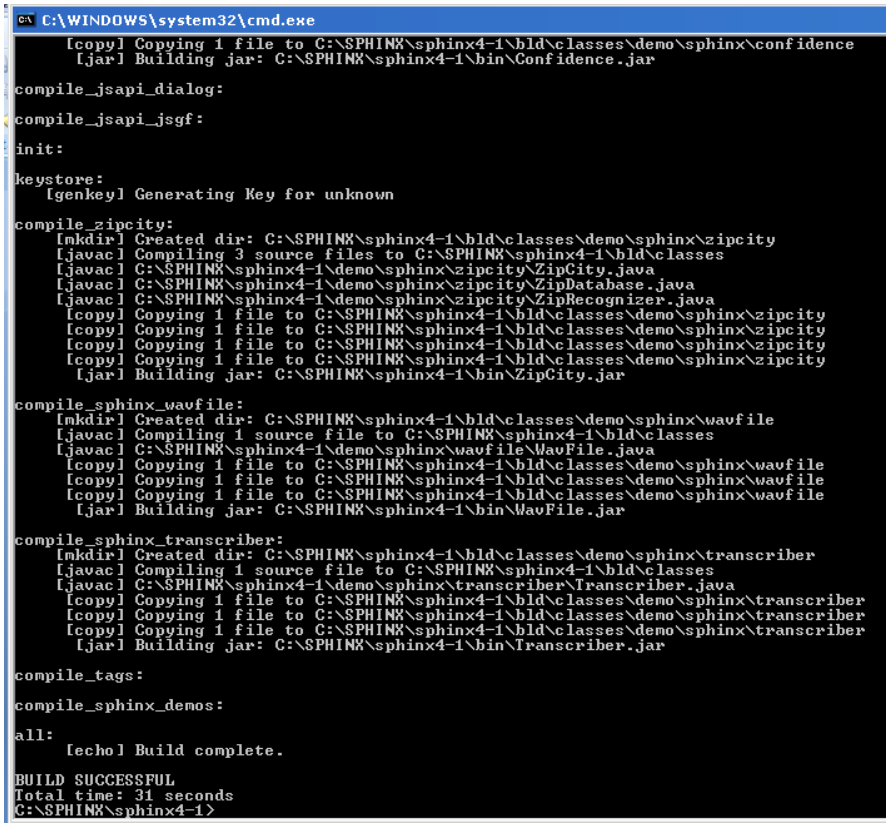


**Figura 14.** Finalizando el proceso de comando ant

Cuando se llega a este punto, se genero un archivo .jar dentro del directorio lib (como se ve la Figura 15). Éste se asociará a la programación que se realizará en Java.

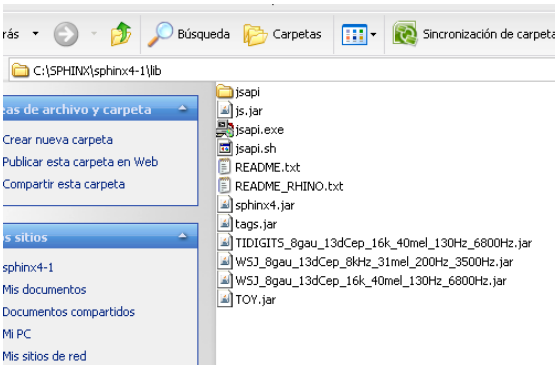


**Figura 15**. Archivo jar generado

Para garantizar que el archivo .jar es correcto, se ejecuta el comando java –jar TOY.jar y el resultado será el mostrado en la Figura 16.

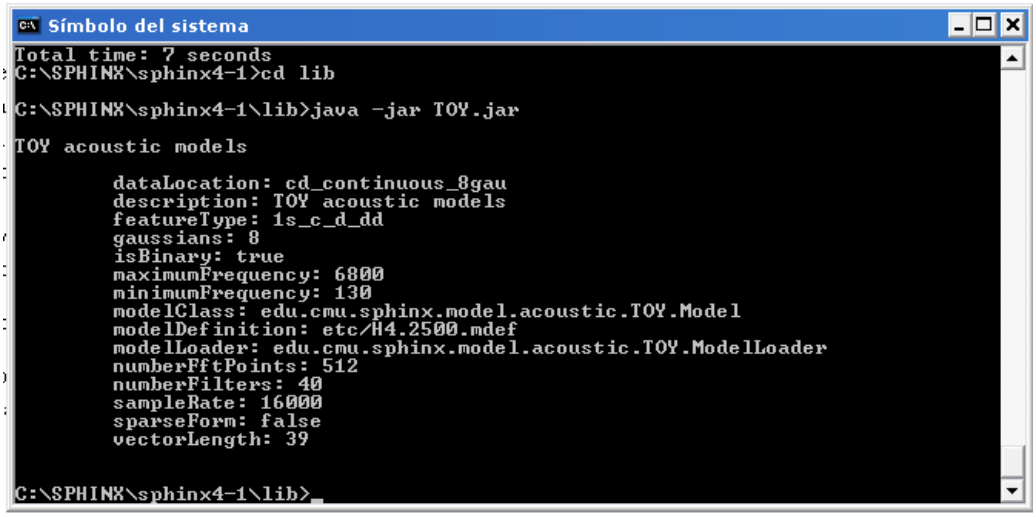


**Figura 16**. Ejecución del archivo .jar obtenido

## 4.2.- Módulo de reconocimiento de voz

El modulo de reconocimiento de voz es construido con base al ejemplo que trae el reconocedor de voz, el ejemplo llamado hellodigits reconoce números en ingles y muestra las palabras correspondientes en pantalla. Las modificaciones se realizan en los archivos *digits.gram, hellodigits.config.xml , hellodigits.Manisfest y HelloDigits.java* para que este pueda funcionar en el idioma español. Los siguientes párrafos muestran la configuración y cambio necesario para que el reconocedor funcione en el idioma español.

Iniciando el cambio de la configuración del archivo digits.gram sólo será necesario cambiar las líneas que están en inglés al español como se ve en las siguientes líneas de código

Anterior:

```
public <numbers> = (oh | zero | one | two | three | four | five | six
| seven | eight | nine) * ;
```

Modificación:

```
public <numbers> = (uno | dos | tres | cuatro | cinco | seis | siete |
ocho | nueve | diez | once) * ;
```

En el archivo hellodigits.config.xml se deberán cambiar varias líneas de configuración por lo que es necesario cambiar las líneas que se encuentran en negritas, este cambio servirá hacer la referencia correcta a las clases que contiene el archivo .jar:

```
  <!-- ********************************************************* -->
   <!-- The Dictionary configuration -->
   <!-- ********************************************************* -->

   <component name="dictionary"
type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
<property name="dictionaryPath"
        value="resource:/edu.cmu.sphinx.model.acoustic.TIDIGITS_8gau_13dCep_1
6k_40mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/TIDIGITS_8gau_
13dCep_16k_40mel_130Hz_6800Hz/dictionary"/>

<property name="fillerPath"
        value="resource:/edu.cmu.sphinx.model.acoustic.TIDIGITS_8gau_13dCep_1
6k_40mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/TIDIGITS_8gau_
13dCep_16k_40mel_130Hz_6800Hz/fillerdict"/>
      <property name="addSilEndingPronunciation" value="false"/>
      <property name="wordReplacement" value="&lt;sil&gt;"/>
      <property name="allowMissingWords" value="false"/>
      <property name="unitManager" value="unitManager"/>
   </component>
```

Cambiar por:

*"resource:/edu.cmu.sphinx.model.acoustic.H4.Model!/edu/cmu/sphinx/model/acoustic/H4/dict/h4.dict"*

*value="resource:/edu.cmu.sphinx.model.acoustic.H4.Model!/edu/cmu/sphinx/model/acoustic/H4/dict/filler.dict"*

En las líneas:

```
<!-- ********************************************************** -->
<!-- The acoustic model configuration                           -->
<!-- ********************************************************** -->
<component name="tidigits"

type="edu.cmu.sphinx.model.acoustic.TIDIGITS_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model">
    <property name="loader" value="sphinx3Loader"/>
    <property name="unitManager" value="unitManager"/>
</component>

<component name="sphinx3Loader"

type="edu.cmu.sphinx.model.acoustic.TIDIGITS_8gau_13dCep_16k_40mel_130Hz_6800Hz.ModelLoader">
    <property name="logMath" value="logMath"/>
    <property name="unitManager" value="unitManager"/>
</component>
```

Cambiar por:

```
"edu.cmu.sphinx.model.acoustic.H4.Model"
"edu.cmu.sphinx.model.acoustic.H4.ModelLoader"
```

En las líneas:

```
<!-- ********************************************************** -->
<!-- The frontend pipelines                                     -->
<!-- ********************************************************** -->
```

**`<component name="featureExtraction"`**
**`type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor"/>`**

y cambiarlo por :

```
<component name="featureExtraction"
type="edu.cmu.sphinx.frontend.feature.S3FeatureExtractor"/>
```

Cuando se haya cambiado estas líneas se deberá volver a teclear el comando ant para regenerar el archivo .jar y ver los resultados. Al finalizar la ejecución del comando ant, se realiza la prueba escribiendo el comando java –jar bin/Hellodigits.jar para ver el resultado que muestra en pantalla como se ve en la Figura 17.

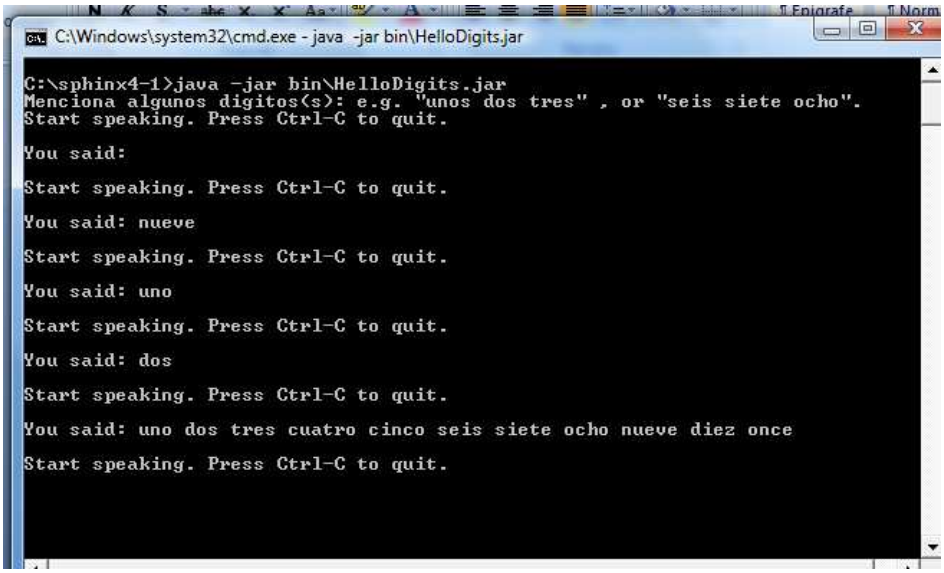


**Figura 17.** Ejemplo reconocimiento de voz con muero del 1 al 11.

## 4.3.- Anatomía de conexión bluetooth

Este tema explica la forma de usar el API Bluecove pues debido a que el API corresponde a una especificación abierta, éste puede usarse como más convenga, para establecer una comunicación entre dispositivos bluetooth. El "Bluetooth SIG"(Special Interest Group de ACM) dice que la anatomía de una aplicación Bluetooth está dividida en cuatro partes para realizar la comunicación , éstas son [Bluetooth SIG]:

- Inicialización de la pila
- Descubrimiento de dispositivos y servicios
- Manejo del dispositivo
- Comunicación

En los siguientes subtemas se muestra cómo se implementa la anatomía en Bluecove. El API contiene una interfaz DiscoveryListener que tiene métodos que sirven para establecer una comunicación entre los dispositivos blueetooth como: `deviceDiscovered`, `inquiryCompleted`, `serviceSearchCompleted` y `servicesDiscovered`.

## 4.4.- Módulo búsqueda de dispositivos bluetooth

Para realizar la búsqueda de los dispositivos que se encuentran en un área no mayor a 10 metros, esta módulo utiliza bluetooth clase 2. Se requiere utilizar la interfaz DiscoveryListener e implementar los métodos deviceDiscovered, inquiryCompleted.

El método deviceDiscovered se llama cuando un dispositivo es encontrado durante una búsqueda con la conexión bluetooth activa. Su implementación es:

```
public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {

try {
      dispodescubiertos.addElement(btDevice);
     System.out.println("Dispositivo  " +
     btDevice.getBluetoothAddress() + "-"+
     btDevice.getFriendlyName(false));

    } catch (IOException cantGetDeviceName) {}
}
```

Los parámetros para el método son:
btDevice - Sirve para enviar el objeto que representa el dispositivo encontrado
cod- Sirve para saber los tipo de servicio así como la clase a la que pertenece el bluetooth

El vector "dispodescubiertos" sirve para guardar todos los dispositivos encontrados y en su momento poder utilizarlos; éste es un atributo de la clase descubrirdispositivos definido como sigue:

```
public static final Vector dispodescubiertos = new Vector();
```

Después de que el `deviceDiscovered` ha finalizado la búsqueda de los dispositivos, se informa con el método sincronizado `inquiryCompleted` al objeto `busquedaeventocompletada`, la implementación del método `inquiryCompleted` se muestra como sigue:

```
//finalizando la búsqueda de dispositivoc
```

```
public void inquiryCompleted(int discType) {
      System.out.println("Examinacion de dispositivos completada!");
                synchronized(busquedaeventocompletada){
                    busquedaeventocompletada.notifyAll();
                }
            }


//inicia la busqueda de los dispositivos aplicando la interfaz
DiscoveryListener

synchronized(busquedaeventocompletada) {
   boolean started =
   LocalDevice.getLocalDevice().getDiscoveryAgent().startInquiry(Disco
   veryAgent.GIAC, listener);

   if (started)
    {
      System.out.println("Espere a que la examinacion de dispositivos
      se complete...");
      busquedaeventocompletada.wait();
      System.out.println(dispodescubiertos.size() +  " dispositivo(s)
      encontrados");
    }
 }
```

La Figura 17 es una posible salida de la ejecución del módulo buscar dispositivos.

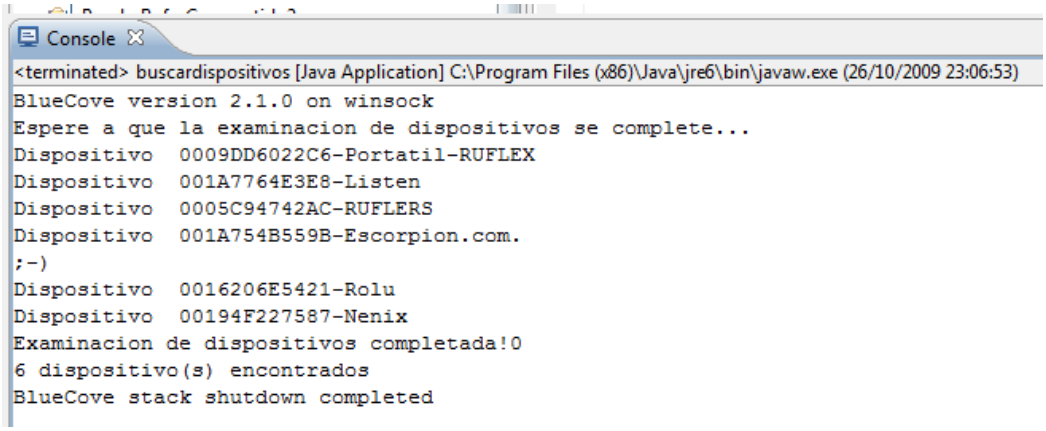


**Figura 17.** Resultado de la ejecución módulo buscar dispositivos.

## 4.5.- Módulo búsqueda de servicios en bluetooth

Para poder desarrollar este modulo es necesario utilizar el código del tema anterior pues como se menciona en la anatomía de los dispositivos, después de haber encontrado los dispositivos será necesario conocer los servicios del bluetooth, para lograr esto nuevamente se utiliza la interfaz DiscoveryListener pero ahora llamando a sus métodos `servicesDiscovered y serviceSearchCompleted.`

Para conocer los servicios que ofrece cada dispositivo será necesario recorrer el vector *dispodescubiertos* que se tiene en el modulo búsqueda dispositivos y aplicar el método `searchServices` de la clase `DiscoveryAgent` accediendo a esta con la clase `LocalDevice` y su método *getLocalDevice,* para que el método *searchservices* sea llamado deberá asociarse los parámetros siguientes:

`attrIDs`-Indica los atributos de los servicios que se especifican en el UUID (identificadores únicos universales) searchUuidSet, algunos atributos por default son: Servicio registro de titulo (0x0000), listado de clases (0x0001), registro de estados(0x0002), Id del servicio (0x0003) y lista de protocolos (0x0004).

`searchUuidSet`-Indica el conjunto de UUID que están siendo buscadas, algunos UUID pueden ser: solicitud perfil OBEX(0x1105 ) ,trasferencia de archivos utilizando OBEX( 0x1106).

`btDevice`-Indica el dispositivo Bluetooth remoto para buscar sus servicios

`listener`- Es el objeto que recibirá los eventos cuando se descubren los servicios

Vea el siguiente código para conocer como es aplicado el método `searchServices` con sus respectivos atributos:

```
for(Enumeration en=buscardispositivos.dispodescubiertos.elements();
en.hasMoreElements();)
{
  RemoteDevice btDevice=(RemoteDevice)en.nextElement();
  synchronized(serviceSearchCompletedEvent)
    {
      System.out.println("Buscando servicios disponibles en: " +
      btDevice.getBluetoothAddress() + " " +
      btDevice.getFriendlyName(false));

      LocalDevice.getLocalDevice().getDiscoveryAgent().searchServices(
      attrIDs,searchUuidSet,btDevice,listener);
      serviceSearchCompletedEvent.wait();
    }
}
```

Como pueden ver en el código anterior se muestra la forma de solicitar los servicios que soporta cada uno de los dispositivos encontrados, esta tarea se encuentra dentro de un **`synchronized`** , esta palabra reservada lo que hace es evitar que todos los métodos

searchServices que se lancen tengan acceso al mismo recurso, de manera que cuando se llamen los métodos cada uno espere a que finalice su proceso, si no estuviera presente esta palabra un método puede estropear lo que hace el otro.

Hasta el momento lo que se ha explicado es la forma de cómo solicitar los servicios, pero quien hace la petición de los servicios es listener, y como se decía, este es un objeto de tipo DiscoveryListener, entonces por cada solicitud de servicios esta interfaz es utilizada para aplicar sus métodos servicesDiscovered y serviceSearchCompleted, vea el siguiente código para ver el contenido de cada uno de los métodos:

```
public void servicesDiscovered(int transID, ServiceRecord[]
servRecord)
{
   for(inti=0;i<servRecord.length;i++)
    {
       String url =
       servRecord[i].getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOE
       NCRYPT, true);

       if(url!=null)
       {
        serviceFound.add(url);
       }
       DataElement serviceName =
      servRecord[i].getAttributeValue(0x1105);

      if(serviceName!=null)
      {
            System.out.println("servicio " + serviceName.getValue() +
            " encontrado " + url);
      }else{
            System.out.println("servicio encontrado " + url);
      }
    }
 }//fin servicesDiscovered


public void serviceSearchCompleted(int transID, int respCode)
 {
   System.out.println("Busqueda de servicio competada!\n\n");
    synchronized(serviceSearchCompletedEvent){
                 serviceSearchCompletedEvent.notifyAll();
                }
 }//fin servicesSearchCompleted
```

El método servicesDiscovered es llamado para solicitar la búsqueda de los servicios, este método tiene parámetros necesarios para su funcionamiento:

transID - Identificador de la operación búsqueda de servicios de cada dispositivo.

`servRecord` – Contiene la lista de los servicios encontrados durante la busqueda

El método regresa en `servRecord` todos los servicios encontrados, para mostrar cada uno de los servicios encontrados es necesario recorrer el arreglo, como se ve en el código mandamos a mostrar la url de conexión con el método `getConnectionURL` al mismo tiempo que lo agregamos en el vector *`serviceFound`* para un uso futuro.

Por último cuando termina la búsqueda de de los servicios para el dispositivo la interface `DiscoveryListener` finaliza la búsqueda con el método `serviceSearchCompleted`, este método también contiene parámetros que deben ser asignados de lo contrario mandaria error de compilación.

`transID`- Indica el ID de transacción a la solicitud que inició el servicio de búsqueda
`respCode`- Corresponde al código de respuesta que indica el estado de la transacción.

Como pueden ver también utilizamos la palabra sereservada **`synchronized`** esta sirve para indicarle al **`synchronized`** que se utilizo cuando se recorría en vector `dispodescubiertos`, que se ha finalizado la búsqueda de servicios en el dispositivo actual y que puede seguir con el siguiente dispositivo. Vea el resultado de la ejecución del código en la Figura 18.

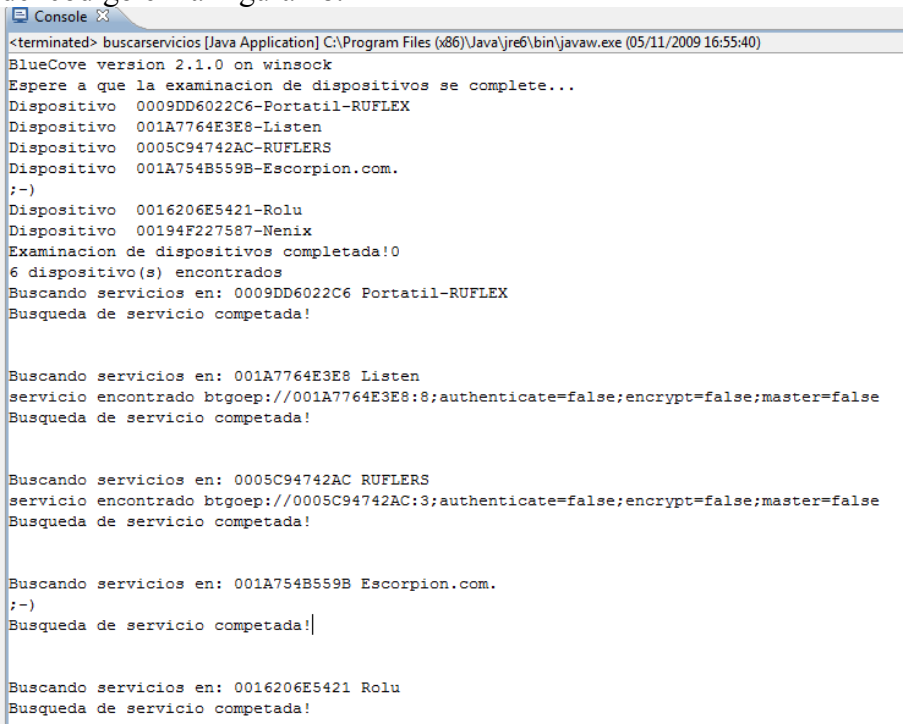


**Figura 18.** Resultado de la ejecución módulo buscar servicios.

## 4.5.- Integrar módulos reconocimiento de voz y bluetooth

## Referencias


| | |
|---|---|
| [André 2004] | André N. 2004. J2ME Bluetooth Programming. Master's Thesis. University of Bergen Klingsheim Department of Informatics |
| [Bluetooth SIG] | Bluetooth SIG, Inc. All rights reserved. The Bluetooth Special Interest Group.Fecha Consulta:25 Marzo 2009. http://spanish.bluetooth.com/Bluetooth/Technology/Basics.htm |
| [Sun Microsystems 2009] | Copyright 1995-2009 Sun Microsystems.2009. JSR 82: Java APIs for Bluetooth. Fecha de consulta: 2 de Abril 2009. http://jcp.org/en/jsr/detail?id=82 |
| [Sun Microsystems 2002] | Copyright 2001-2002 Sun Microsystems. 2002. Java APIs for Bluetooth Wireless Technology (JSR-82). William Cannon Drive West,Austin, TX |
| [Gehrmann 2004] | Gehrmann C, Persson J, Smeets B .2004. *Bluetooth Security*. Artech House. Boston,London |
| [Harte 2004] | Harte L .2004. *Introduction to Bluetooth: Technology, Market, Operation, Profiles, & Services.* ALTHOS |
| [Hopkins 2003] | Hopkins B, Antony R. 2003. *Bluetooth for Java*. Apress. U.S.A |
| | |
| [Huang 2007] | Huang A. S, Rudolph L. 2007. *Bluetooth Essentials for Programmers*. Cambridge University Press. New York, U.S.A |
| [Muller 2000] | Muller N J. 2000. *Bluetooth Demystified*. McGraw-Hill. U.S. A |
| [Pérez 2006] | Pérez E. P. H. 2006 Construcción de un reconocedor de voz utilizando sphinx y el corpus DIMEx100. Tesis de ingeniería. Universidad Nacional Autónoma de México, Facultad de Ingeniería en Computación. |
| [Mitsubishi Electric Research 2009] | Portions Copyright 2002-2008 Mitsubishi Electric Research Laboratories. 2009. The CMU Sphinx Group Open Source Speech Recognition Engines. Fecha de consulta: 15 Enero 2009. *http://* cmusphinx.sourceforge.net/sphinx4/ |
| [Bluetooth Demystified 2003] | Rodríguez C. O.D, Maya C. R. A , implementación de una red inalámbrica Bluetooth, Tesis Ingeniero electrónico, Universidad del Valle Facultad de ingeniería, Programa de ingeniería electrónica Santiago de Cali,2003 |
| [Timothy 2008] | Timothy J. Thompson, Paul J. Kline, and C Bala Kumar.2008 . *Bluetooth Application Programming with the Java APIs Essentials Edition*. Morgan Kaufmann; Essentials Ed edition. Burlington U.S.A |

| | |
|---|---|
| [Vikas 2002] | Vikas G, Dr. K. V. K. K. Prasad, Avnish D, Deepesh J. 2002. *WAP, Bluetooth, and 3G Programming: Cracking the Code*. Hungry Minds, Inc. New York, NY |
| [Villamil 2005] | Villamil I. H. 2005. Aplicaciones en reconocimiento de voz utilizando htk. Tesis de Doctorado. TG 0446. Universidad Javeriana, Facultad de Ingeniería Electrónica. |
| [Walker 2004] | Walker W, Lamere P, Kwok P, Raj B, Singh R, Gouvea E, Wolf P, Woelfel J. 2004.Sphinx-4: a flexible open source framework for speech recognition. SMLI TR-2004-139.Sun Microsystems. |


# Apéndice A. Instalación de software

A) Java Development Kit (JDK)

Descargar del portal de Sun Microsystems, Inc el software JDK del enlace https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewFilteredProducts-SingleVariationTypeFilter). Cuando abre el navegador, se mostrará una pantalla como la Figura 1, seleccionar ahí la opción Java SE Development Kit 6u15.


**Figura 1**. Portal descargas Sun Microsystems

Una vez que se descargue el paquete, para instalarlo es necesario hacer doble 'click'. En cada pantalla de configuración presionar la opción "siguiente". Este software requiere definir una variable de entorno, lo cual se explica en los pasos siguientes:

**Paso 1:** Ir a inicio - > Mi PC y presionar el botón derecho del ratón para elegir "Propiedades" (ver la Figura 4).


***Figura 4**. Buscando el icono Mi PC*


***Figura 5**. Elección propiedades de Mi Pc*

**Paso 2:** Después de elegir la opción "Propiedades", aparecerá una ventana como la de la Figura 6.

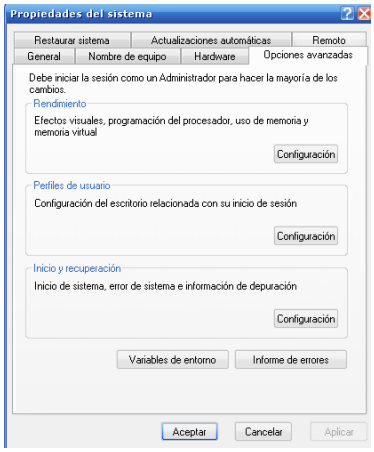


***Figura 6**. Propiedades del sistema*

**Paso 3:** Seleccionar la pestaña "Opciones avanzadas" de la Figura 6, posteriormente el botón "Variable de entorno". Se mostrará el cuadro de la Figura 7.

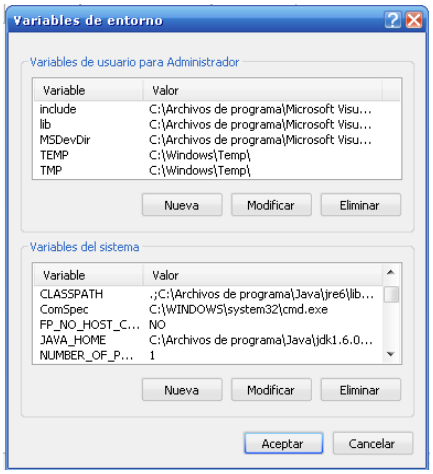


***Figura 7**. Ventana variable de entorno*

**Paso 4:** En la ventana de variables de entorno, configurar la variable "JAVA_HOME" y "PATH". La variable JAVA_HOME por lo general se configura automáticamente. Si no es el caso, declararla como variable del sistema al presionar el botón "nueva". Entonces se mostrará el formulario de la Figura 8.

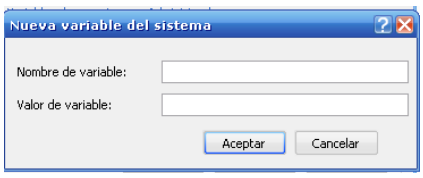


**Figura 8**. Formulario para crear nueva variable de entorno.

En el campo “Nombre de variable” se escribe JAVA_HOME, después en el campo “Valor de la variable” se pone la ruta donde fue instalado el JDK, en este caso, es “C:\Archivos de programa\Java\jdk1.6.0_10”. La configuración se muestra en la Figura 9.

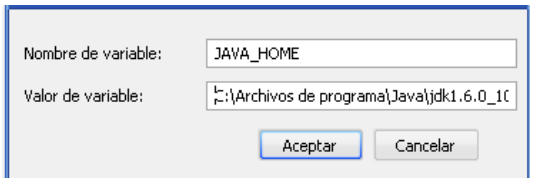


**Figura 9**. Configuración variable de entorno JAVA_HOME.

B) Java ant

La descarga del servidor java ant se realiza desde http://ant.apache.org/bindownload.cgi. Cuando se abre el portal se muestra una página como la Figura 2, aquí se selecciona de la parte inferior un enlace llamado apache-ant-1.7.1-bin.zip. Este archivo está empaquetado y no trae un instalador, por lo que es necesario desempaquetar en alguna unidad de almacenamiento de la computadora, por ejemplo en la unidad c: como se muestra en la Figura 10.

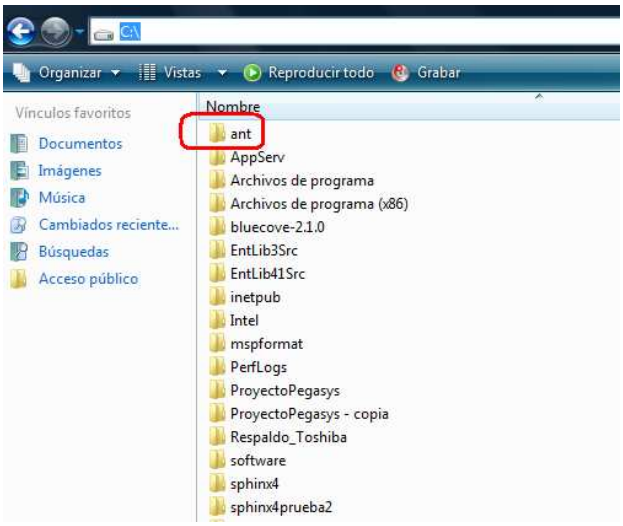


**Figura 10**. Ubicación del servidor java ant en la unidad c:\

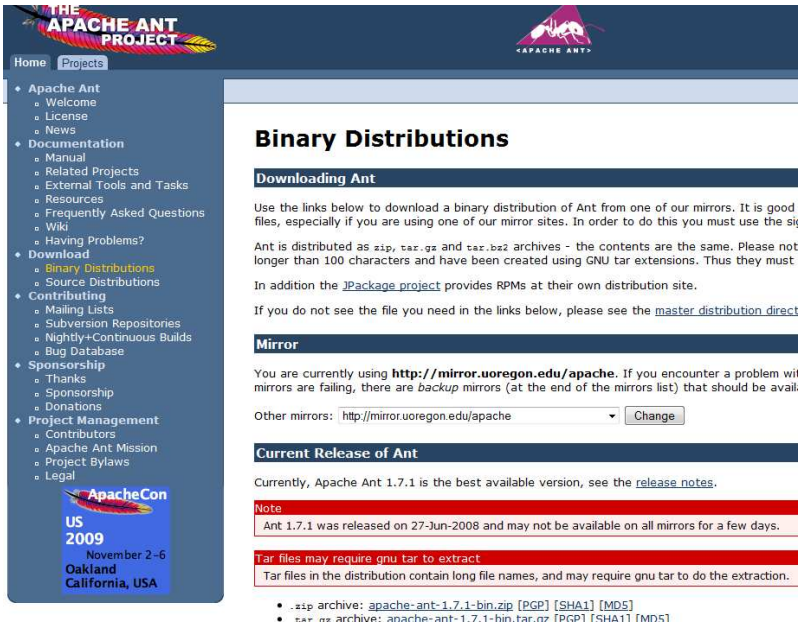


**Figura 11**. Portal del proyecto java ant

Posteriormente se debe definir la variable ANT_HOME. En el formulario, en el campo "Nombre de variable" se asigna el nombre de ANT_HOME y en el campo "valor de variable" se especifica la ruta donde se encuentra la carpeta que contiene los archivos de la herramienta ANT, en este caso, en C:\ant. La configuración se muestra en la Figura 12.

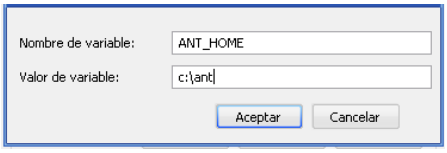


**Figura 12**. Configuración variable de entorno ANT_HOME.

Al final de la variable PATH de Windows (si no está deberá crearse como en el caso de las variables anteriores), agregar la siguiente línea *%PATH%;%ANT_HOME%\bin* como se muestra en la Figura 13.

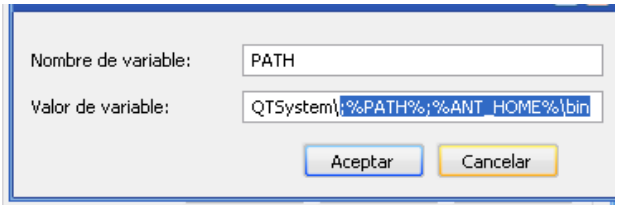


**Figura 13**. Configuración de la variable de entorno PATH con referencias a otros programas.

Para verificar que la configuración de las variables es correcta, desde el símbolo del sistema escribir el comando *ant* y deberá desplegarse el siguiente mensaje:

*Buildfile:build xml does not exist!*

*Build failed*

Si la configuración es incorrecta, aparecerá el mensaje:

*no se reconoce como un comando interno o externo,*

*Programa o archivo por lotes ejecutable*

## Apéndice B. Descarga de paquetería

Los paquetes de desarrollo de reconocimiento de voz y javabluetooth se acceden desde http://sourceforge.net/projects/cmusphinx/files/sphinx4/ y http://sourceforge.net/projects/bluecove/files/ respectivamente. Cuando el portal del reconocedor de voz inicie, mostrará la Figura 1, de ahí seleccionar 1.0 beta3->sphinx4-1.0beta3-src.zip y enseguida comenzará la descarga.

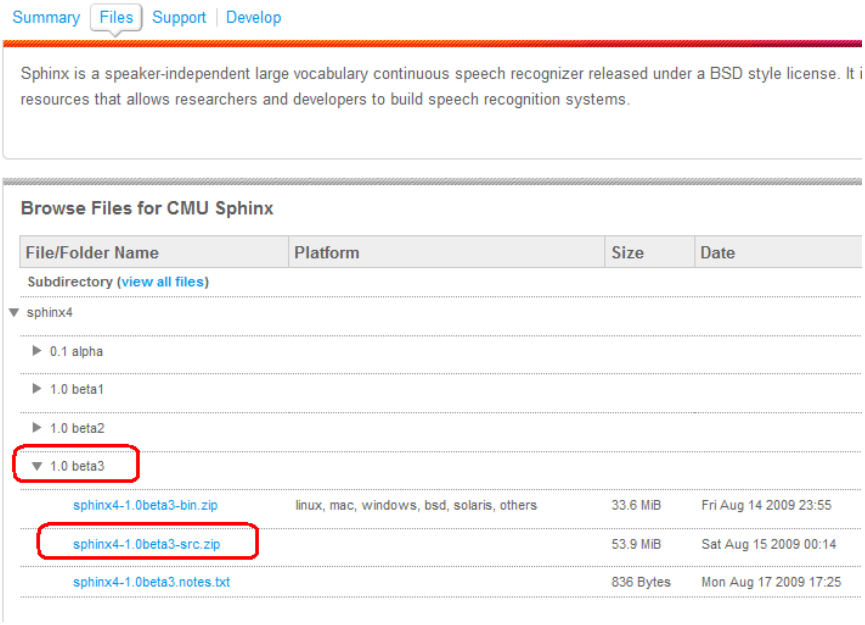


**Figura 1.** Portal de descarga del reconocedor de voz cmusphinx 4

Para la descarga del API de Java se mostrará una página como la de la Figura 2, se elige el enlace con nombre bluecove-2.1.0.jar para dar inicio a la descarga.

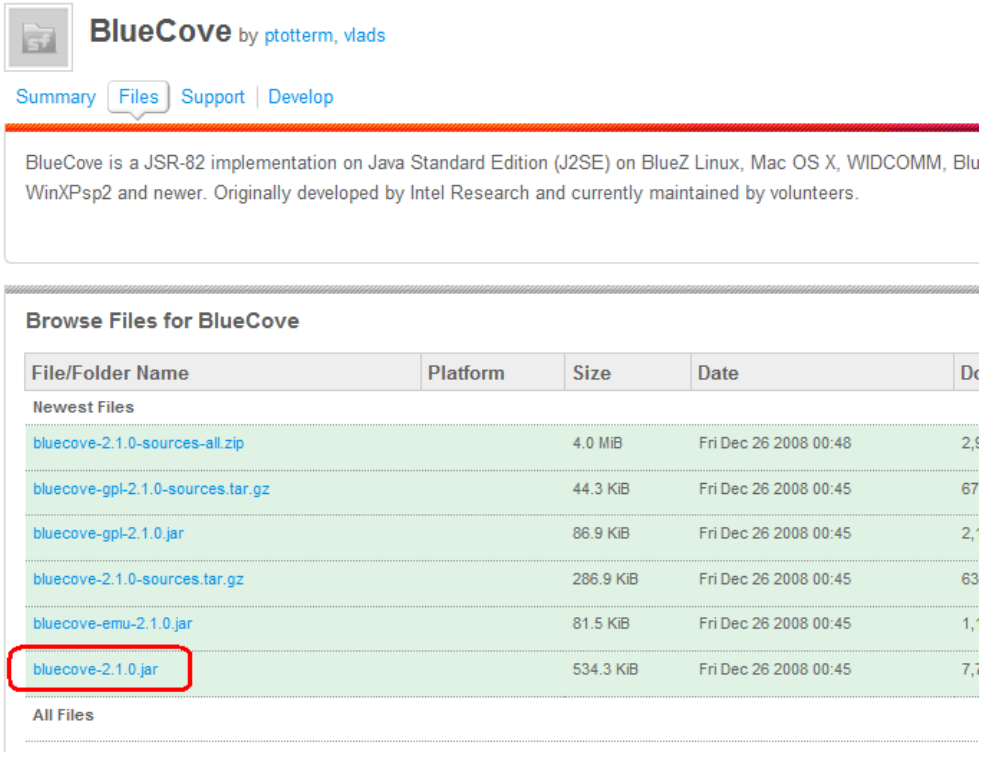


**Figura 2**. Portal de descarga del API Javabluetooth

## Apéndice C. Obtención de modelos acústicos

Para obtener los modelos acústicos de prueba, se usa la base de datos an4 y el paquete de entrenamiento SphinxTrain. La base de datos se puede conseguir desde la página principal de CMU PHINX o bien desde http://www.speech.cs.cmu.edu/databases/an4. El paquete de entrenamiento se puede obtener de http://cmusphinx.org/download/nightly/SphinxTrain.nightly.tar.gz. Una vez descargados los paquetes, se descomprimen ubicándolos en el directorio que más convenga, en este caso se utilizará C:/ dentro del directorio SPHINX. Posteriormente se compilan los archivos de entrenamiento escritos en el lenguaje de programación Microsoft Visual C++. Dentro de la carpeta SphinxTrain se encuentra un archivo llamado SphinxTrain.dsw (vea Figura 1), este archivo tendrá que cargarse con el visual de C++.

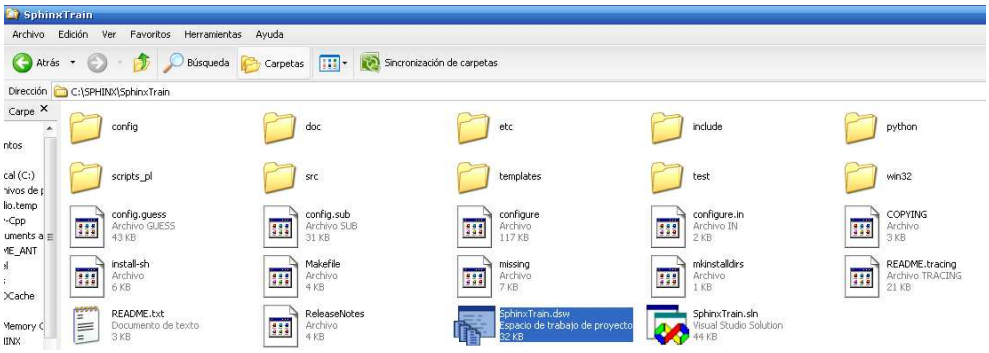


**Figura 1**.Ubicacion del archivo SphinxTrain.dsw

Cuando se halla cargado el archivo SphinxTrain.dsw se dará inicio a la compilación. Para esto es necesario ir a la barra de menús en la opción Bulid->Batch Build, inmediatamente saldrá una nueva ventana donde se eligen todas las opciones (vea Figura 2). Después se presiona el botón llamado “Rebuild All”. El tiempo de espera para finalizar la compilación depende del equipo que se utilize.

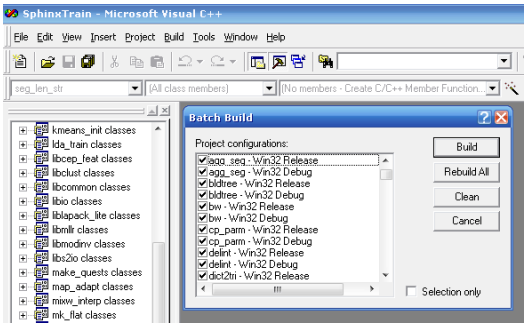


**Figura 2**. Elección de todos los archivos para iniciar la compilación.

Después de compilar, se crean nuevos directorios y archivos ubicados en C:/SPHINX/SphinxTrain. Los directorios llamados Debug, lib, bin y los archivos llamados SphinxTrain.ncb y SphinxTrain.opt serán utilizados para dar inicio al entrenamiento. Por esa razón, estos archivos deberán copiarse a la carpeta donde se encuentra la base de datos en el directorio an4. Para ello se utilizan los scripts que trae el paquete SphinxTrain que pueden ser ejecutarlos desde el símbolo del sistema, aunque esto requiere que se tenga instalado el ActivePerl. Para ejecutar los scripts es necesario acceder al símbolo del sistema y ubicarse en la carpeta SphinxTrain (vea Figura 3).

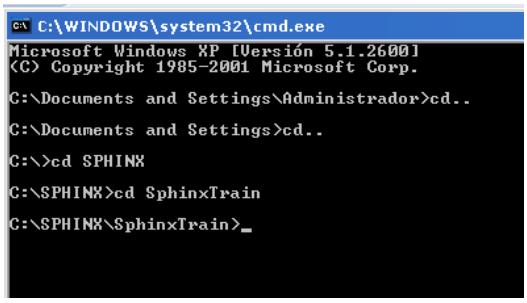


**Figura 3**.Ubicar el promt dentro de la carpeta SphinxTrain.

Cuando se situé en SphinxTrain se debe escribir el comando perl scripts_pl\setup_tutorial.pl an4, éste realizará la copia de los archivos obtenidos de la compilación SphinxTrain hacia la carpeta an4 que es donde se encuentra la base de datos de audio. Después de la copia, ir a la carpeta an4 para seguir con la fase de entrenamiento, vea la Figura 4.

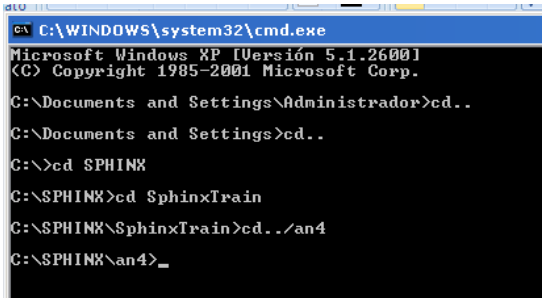


**Figura 4**.Pasar promt a la carpeta an4

El sistema no trabaja directamente con señales acústicas. Las señales primero se transforman en una secuencia de vectores. Para realizar esta transformación, en el

directorio an4 se escribe el comando perl scripts_pl/make_feats.pl -ctl etc/an4_train.fileids, vea la Figura 5.

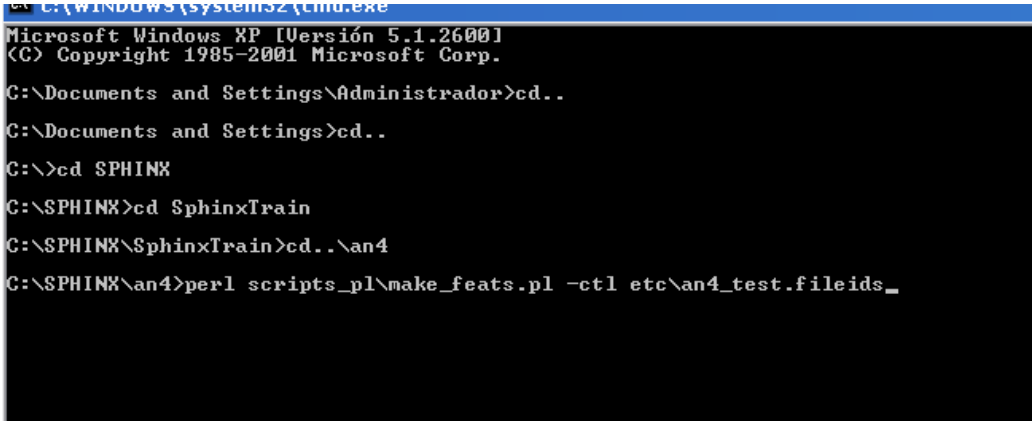


**Figura 5**. Ejecución del comando perl scripts_pl/make_feats.pl -ctl etc/an4_train.fileids

El comando de la Figura 5 procesará para cada expresión de entrenamiento una secuencia de 13 vectores dimensionales. Los archivos que se generan en el directorio an4->wav durante el proceso contienen los archivos de audio. Ahora se puede comenzar a entrenar el sistema.

Para comenzar el entrenamiento, an4 trae consigo archivos como diccionario de la lengua, archivo de fonemas, archivo de gramática, señales acústicas, entre algunos otros. Se inicia el entrenamiento al escribir el comando perl scripts_pl/RunAll.pl, el cual comenzará a ejecutar todos los scripts que se encuentran en la carpeta scripts_pl, (ver la Figura 6).

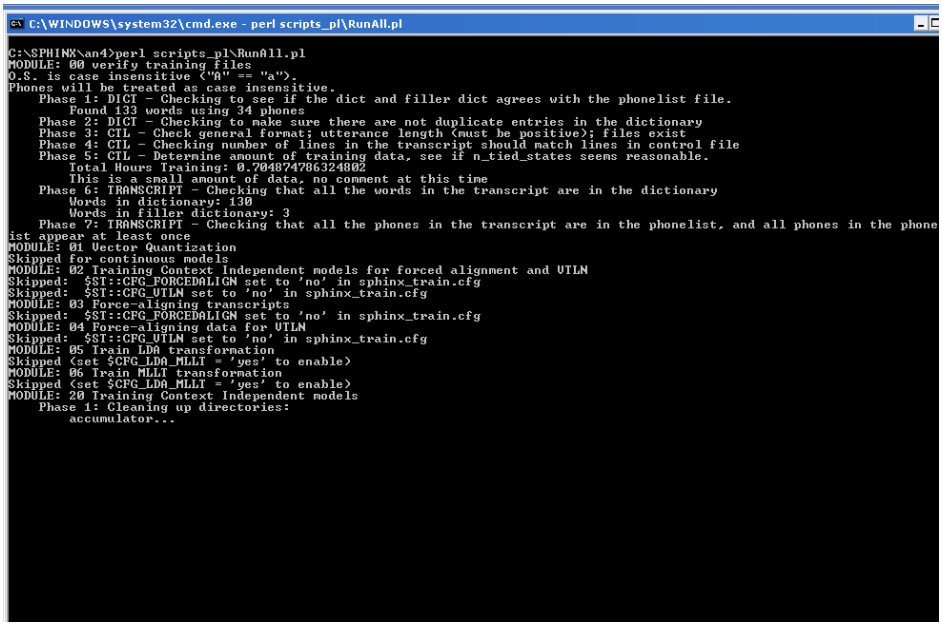


**Figura 6.** Inicio del entrenamiento con modelos acústicos para la lengua inglesa.

Cuando haya terminado el comando, se inicia una decodificación. Para realizar una prueba, ejecute el comando perl scripts_pl/make_feats.pl -ctl etc/an4_test.fileids que relacionará los archivos de entrenamiento con el archivo de decodificación. Este proceso tarda aproximadamente 5 minutos. Una vez que termine, se verifica que se realizó el entrenamiento al hablar por el micrófono.