Centro de Investigación y de Estudios
Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

"Framework multiplataforma para reconocimiento de voz en aplicaciones open rich-client para dispositivos móviles"

Tesis que presenta

Irene Monserrat Torres Hernández

para obtener el Grado de

Maestro en Ciencias

en Computación

Directores de la Tesis:

Dr. Amilcar Meneses Viveros

M.C. Erika Hernandez Rubio

México, D.F. **Octubre, 2013**

Resumen

Actualmente más tecnologías convergen en los dispositivos móviles, lo que ha incrementado el desarrollo de aplicaciones para estos dispositivos. Uno de los obstáculos al que se enfrenta un desarrollador es la falta de estandarización para generar código que se ejecute en los diferentes sistemas operativos de los dispositivos móviles, por lo que no es posible el desarrollo de un software único compatible para las diferentes plataformas existentes.

El propósito de la presente tesis es el desarrollo de una capa de programación abierta, que permita realizar el reconocimiento de voz, para lograr una forma de interacción natural al usuario con el dispositivo móvil. El reconocimiento se valida por medio de la construcción de una aplicación que hace uso del reconocimiento de voz, a través del preentrenamiento de palabras definidas en un diccionario. La finalidad de esta es que su implementación sobre diferentes dispositivos móviles sea de forma transparente, sin la necesidad de particularizar el desarrollo a una sola plataforma.

Se propone la API de la capa y valida su funcionamiento con un ejemplo de reconocimiento. El módulo de reconocimiento será local y remoto al dispositivo donde se cargue la capa. Por medio de esta capa se logrará la reutilización de código en diversos dispositivos, lo que conlleva a un ahorro de tiempo de desarrollo y una amplitud de mercado a diferentes plataformas.

Abstract

At the present time more technologies converge into mobile devices, for this reason the mobile software development has increased. One of the main obstacles faced by a software developer is the lack of standardization to generate code that executes over different mobile platforms, in this moment it is difficult to create a compatible application with the different mobile platforms.

The purpose of the thesis is to develop an open programming layer, that enables speech recognition to let the user to interact in a natural way with the mobile device. The speech recognition is validated trough an application using speech recognition, this is defined by a small pre trained dictionary related to the user. This layer has to be implemented in transparent way over the different mobile platforms, without a particular software application for each platform.

In this work we proposed the API to interact with the framework layer and with an example implementation the API will be validated. The speech recognition will be performed in both local and remote way to the mobile device. Through this layer implementation we will achieve code reutilization on several mobile devices, this will carry out saving development time and reaching extended market for different mobile platforms.

Contents

Capítulo 1

Introducción

1.1. Motivación

El *Cómputo Móvil* se refiere a un amplio conjunto de operaciones computacionales que permiten al usuario acceder a información proveniente de dispositivos portables tales como laptops, PDAs, handhelds, celulares, reproductores de música, dispositivos de videojuego portátiles [1].

En los últimos años el uso de dispositivos móviles, especialmente el uso de *smartphones* [1] ha aumentado considerablemente dada la convergencia de tecnologías involucradas en el desarrollo de dichos dispositivos. Las capacidades actuales de los *smartphones* están impulsando el uso de estos como dispositivos de entrada a recursos como pantallas fijas, máquinas expendedoras y diversos electrodomésticos. La proliferación de uso de los *smartphones* les brinda un gran potencial para convertirlos en la principal interfaz física para las aplicaciones de cómputo ubicuo [3].

La movilidad inherente a estos dispositivos permite a los usuarios hacer uso de sistemas de cómputo en cualquier lugar y en cualquier momento, lo que ha dado pie a la creación de nuevos tipos de aplicaciones de software. Estos tipos de aplicaciones móviles proporcionan un rango de actividades más amplio que el que ofrecen las aplicaciones de escritorio, que permiten aprovechar la información sobre el entorno del usuario para proporcionar nuevas capacidades. Desde una perspectiva tecnológica, la movilidad da un giro a la computación global de estática, homogénea, con poderosas computadoras de escritorio a dispositivos móviles dinámicos, heterogéneos y con recursos limitados [4].

Con el cómputo móvil surge un nuevo paradigma de programación para la construcción de aplicaciones móviles, la particularidad de estas aplicaciones es que hacen uso de elementos que no se encuentran en una computadora de escritorio, lo que hace que trabajen en

[1]Se ha considerado al *smartphone* como un dispositivo móvil híbrido, es un combinación de una PDA y un teléfono móvil, que tiene el objetivo de crear comunicación móvil más eficaz y ser a la vez ser un dispositivo de información [2].

sistemas heterogéneos. Esta heterogeneidad se refiere tanto a software como a hardware. Por parte del hardware se deben tomar en cuenta las diferentes capacidades del dispositivo como la arquitectura del microprocesador, sensores, capacidad de almacenamiento. En el lado del software se debe considerar el sistema operativo usado en el dispositivo, su versión, la plataforma de desarrollo y la arquitectura usada en la construcción de la aplicación.

El desarrollo de aplicaciones móviles involucra el estudio del ambiente en el que el usuario hará uso de esta tecnología (lo que se conoce como contexto de uso) y como este puede tener un mayor impacto en la habilidad para interactuar con los dispositivos móviles o aplicaciones de una forma efectiva y satisfactoria [2].

Actualmente uno de los retos más importante al desarrollar una aplicación orientada a dispositivos móviles es su adaptabilidad. En el presente trabajo se trata el reconocimiento de voz debido a que cada dispositivo tiene sus propios mecanismos para realizar dicho procesamiento, lo que hace complicada la construcción de una aplicación única que funcione con diferentes dispositivos con distintas plataformas, por lo que actualmente tiene que realizarse un desarrollo específico incluso para cada diferente plataforma, es por ello que se toma este problema para proponer una solución que permita la construcción de una aplicación portable de forma transparente a diferentes tecnologías móviles.

1.2. Planteamiento del problema

El desarrollo de diferentes interfaces de interacción con el usuario es de gran importancia en el cómputo móvil, esto con la finalidad de aprovechar los múltiples sensores y características del dispositivo. Además debido a las limitaciones del dispositivo, como su tamaño, se deben buscar otras formas de interacción con el usuario. Una de las principales formas de interacción es la voz, la cual es la forma más natural por la cual el usuario puede interactuar con el dispositivo.

Por cuestiones de rendimiento y conectividad, lo ideal es que el procesamiento sea en el propio dispositivo. Debido a la diversidad de plataformas se busca que el desarrollador no realice una aplicación específica para cada dispositivo, sino que el desarrollo sea único, por ello se propone como solución un *framework*[2] que posea una capa abierta que permita trabajar con las diferentes plataformas. Por otro lado, al realizar procesamiento de voz en dispositivos móviles, se cuentan con distintas arquitecturas: en la nube, de forma distribuida y en el propio dispositivo. Se buscará la forma de desarrollar la arquitectura de procesamiento de voz integrado, con la finalidad de aprovechar los recursos del propio dispositivo móvil, de una forma abierta, la cual permita generar un desarrollo único multiplataforma, que trabaje de forma transparente principalmente sobre la plataforma iOS

[2]Un *framework* es un diseño reusable de partes o de un sistema de software completo, compuesto por un conjunto de clases y la forma en que las instancias de estas clases colaboran entre sí [5].

y Android.

Se busca la implementación de una plataforma abierta que permita, por medio de una interfaz web capturar la grabación de la voz, la que posteriormente serán procesadas dentro del dispositivo. El reconocimiento se llevará a cabo por medio del cálculo de la Transformada de Fourier, para obtener reconocimiento de voz independiente del usuario.

1.3. Objetivos

El objetivo general de esta tesis es desarrollar una capa de programación que permita generar soluciones abiertas, en particular tratar el reconocimiento de secuencias de voz.

Los objetivos específicos se enlistan a continuación:

1. Revisar la tecnología existente, para determinar cuál es el enfoque que se utilizará para poder implementar el software abierto, en particular se considerará fuertemente HTML5.
2. Analizar los requerimientos funcionales de la capa a desarrollar.
3. Definir la API (*Application Programming Interface*) que se implementará en los diferentes dispositivos.
4. Evaluar la interacción con cada una de las plataformas iOS y Android.
5. Desarrollar una aplicación basada en el *framework* y validarla con pruebas en múltiples plataformas.

1.4. Justificación

Con la presente tesis se busca generar un *framework* que permita la ejecución de aplicaciones móviles de reconocimiento de voz en distintas plataformas, sin la necesidad de realizar una codificación especifica para cada plataforma que haga uso de los propios recursos del dispositivo para el procesamiento.

Este tipo de desarrollos abiertos permite la reutilización de código, mejora el nivel de abstracción del desarrollo y logra un aprovechamiento de los recursos que proporciona el dispositivo.

1.5. Organización

La presente tesis se organiza en siete capítulos con la siguiente estructura, ver tabla 1.1:

Contenido	**Capítulo**
Marco Teórico	**Capítulo 2:** describe las principales características de las aplicaciones móviles, plataformas móviles y arquitecturas de desarrollo. Se trata el tema de la importancia del desarrollo de aplicaciones móviles multiplataforma. **Capítulo 3:** Estado del arte acerca del desarrollo de aplicaciones móviles y de reconocimiento de voz.
Contribuciones y Propuesta	**Capítulo 4:** se presenta la disertación de la propuesta para el desarrollo de la aplicación y la metodología seguida para la realización de los objetivos propuestos. **Capítulo 5:** presenta las etapas de análisis y desarrollo propuesto para la construcción del proyecto realizado. **Capítulo 6:** se muestran las pruebas realizadas con el proyecto.
Conclusiones	**Capítulo 7:** muestra los resultados y conclusiones obtenidas en el desarrollo del proyecto. De igual forma se mencionan los temas a discusión que han surgido en la construcción del proyecto y los trabajos relacionados que pueden obtenerse en un futuro.

Cuadro 1.1: Organización del documento.

Capítulo 2

Plataformas y aplicaciones móviles

En este capítulo se abordan las características inherentes a una aplicación móvil (sección 2.1), los diferentes tipos de plataforma existentes para el desarrollo de aplicaciones móviles, las cuales abarcan tanto sistemas operativos como entornos de desarrollo (sección 2.2). Se exponen las principales arquitecturas sobre las cuales se puede construir una aplicación móvil (sección 2.3). Finalmente se pone en evidencia la importancia de los *frameworks* multiplataforma móviles, su evolución y las ventajas que ofrecen al construir aplicaciones (sección 2.4).

2.1. Características de las aplicaciones móviles

El concepto de movilidad se refiere a la capacidad de moverse o de ser movido fácilmente [6]. En el contexto del cómputo móvil, la movilidad pertenece al uso y funcionalidad que los usuarios den a los poderosos dispositivos móviles, que ofrecen la habilidad de ejecutar un conjunto de aplicaciones y funciones, mientras son capaces de conectarse para, la obtención y entrega de datos a otros usuarios, aplicaciones y sistemas [6]. La movilidad se puede considerar desde dos perspectivas relacionadas:

a) el software móvil, el cual representa la funcionalidad computacional diseñada para migrar a través de dispositivos de hardware en tiempo de ejecución y ejecutarse en diferentes plataformas de hardware móviles;

b) los sistemas móviles son aplicaciones computacionales que incluyen elementos de software y hardware móvil.

Una aplicación móvil es un conjunto de instrucciones diseñadas específicamente para dispositivos móviles; con el fin de realizar una tarea o proveer una experiencia al usuario, la cual normalmente se encuentra estructurada como una aplicación multicapas, la cual consiste de una capa de usuario, de negocios y de acceso a datos [7]. El desarrollo de aplicaciones móviles presenta algunos requerimientos que no son comunes al desarrollo de aplicaciones tradicionales, algunos incluyen [8]:

1. **Interacción potencial con otras aplicaciones:** la mayoría de los dispositivos móviles tienen diversas aplicaciones preinstaladas, por lo que al desarrollar una aplicación se debe considerar la interacción entre ellas.

2. **Manejo de sensores:** los dispositivos móviles actuales incluyen sensores, como el acelerómetro, la pantalla táctil que responde a diversos gestos, GPS, micrófono y cámara, por lo que al crear una aplicación móvil se desea que se aprovechen al máximo los sensores que provee el dispositivo.

3. **Aplicaciones nativas e híbridas:** al desarrollar una aplicación híbrida que se conecte a un servidor remoto, se debe considerar el manejo de la conexión para permitir una administración de energía eficiente en el dispositivo. También se debe considerar la forma en la que se muestran los datos resultantes del procesamiento de la información, ya que deben adecuarse al tamaño de la pantalla.

4. **Familias de hardware y plataformas de software:** la mayoría de los dispositivos ejecuta código que está realizado a la medida de las propiedades de ese dispositivo, pero se deben soportar aplicaciones que se han codificado para otros dispositivos, ya sea que tengan diferentes sistemas operativos o una versión diferente de un mismo sistema. Esto representa un problema actualmente. Por ejemplo, un desarrollador de aplicaciones para Android, tiene que decidir si crea una aplicación única lo que puede llevar más tiempo, o a realizar múltiples versiones para que funcione con cada actualización del sistema operativo.

5. **Interfaces de usuario:** se busca que sean lo más intuitivas y fáciles de manipular por el usuario, aprovechando los recursos del dispositivo y teniendo en cuenta las limitaciones de tamaño, procesamiento y manejo de energía que el dispositivo pudiera tener.

6. **Consumo de energía:** al desarrollar una aplicación se debe tener en cuenta la duración de la batería del dispositivo. Las aplicaciones móviles pueden hacer uso excesivo de recursos, como conectarse a un servidor constantemente, lo que conlleva a la descarga de la batería en un tiempo menor.

Estas aplicaciones se caracterizan por arquitecturas de software personalizadas que son diseñadas para facilitar la movilidad [4].

2.2. Plataformas móviles

Al crear una aplicación móvil, el desarrollador debe seleccionar la o las plataformas donde ejecutará su aplicación. Hoy en día existe una diversidad de plataformas, entre las más comunes son Android, iOS, Windows Phone 8 y Symbian. *"Una plataforma móvil puede definirse como la combinación de un sistema operativo para un conjunto de dispositivos*

móviles compatibles con un conjunto relacionado de bibliotecas de desarrollo, APIs y herramientas de programación"[1].

La necesidad de sistemas operativos especializados para dispositivos móviles que permitan el desarrollo de aplicaciones ha aumentado de forma significativa, debido a la proliferación de estos dispositivos. Estas plataformas se han creado para que los desarrolladores y usuarios puedan contar con un entorno de desarrollo, que les permita crear y hacer uso de aplicaciones únicas y especializadas. Algunas de las plataformas más comerciales son: Android, iOs, Windows Phone 8 y Symbian. A continuación se describen algunas de las características de estas plataformas.

2.2.1. Android

Google lanzó Android en 2007, para dar un paso en los estándares abiertos para dispositivos móviles. Android es una plataforma de software libre con una licencia de código abierto basado en Linux para dispositivos móviles. Esta plataforma se conforma por el kernel de Linux, bibliotecas nativas, Android *run time* y un *framework* de aplicaciones. Los servicios del núcleo del kernel de Linux (incluyen los controladores del hardware, administración de memoria y procesos, seguridad, red y administración de energía) son manejados por el kernel de Linux 2.6. El kernel provee una capa de abstracción entre el hardware y el resto de la pila. Las bibliotecas se ejecutan sobre el kernel, Android incluye varias bibliotecas núcleo en C/C++ como libc y SSL, también incluye una biblioteca para la reproducción de archivos de audio y video, un administrador de superficie para la gestión de la visualización, bibliotecas gráficas que incluyen SGL y OpenGL para gráficos 2D y 3D. Soporte para bases de datos nativas con SQLite. SSL y Webkit para el navegador web integrado y la seguridad en Internet. En la Figura 2.1 se muestra cómo se encuentra distribuida la plataforma Android.

Las aplicaciones de Android son codificadas principalmente en java y compiladas dentro del formato ejecutable Dalvik, en byte code. Cada aplicación ejecuta sus propios procesos, con su propia instancia de Dalvik Virtual Machine. Dalvik ejecuta los archivos tipo DEX, los cuales son convertidos por el compilador en tiempo de ejecución de las clases estándar a archivos tipo JAR. Los archivos DEX son más compactos y eficientes que los archivos tipo class. Los desarrolladores tienen acceso total a todo el *framework* y APIs que las aplicaciones del núcleo utilizan y a las bibliotecas desarrolladas por Google. La arquitectura de Android está enfocada para simplificar la reutilización de código. El kit de desarrollo de Android (SDK) permite la creación de aplicaciones muy completas y funcionales. Se puede hacer uso de los sensores como el acelérometro, giroscopio, manipulación de gráficos en 3D, uso de GPS, así como colaboración entre aplicaciones como correo electrónico, mensajería, calendarios y servicios basados en localización. Las aplicaciones desarrolladas en versiones como 1.0,1.5, 1.6 y 2.0 pueden tener problemas al trabajar con diferentes versiones del sistema operativo. La apertura de la plataforma puede traer problemas de fragmentación.

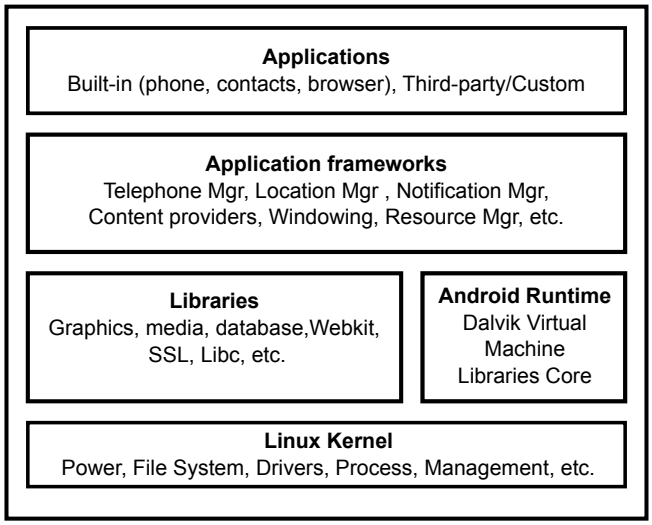


Figura 2.1: Arquitectura de sistema operativo Android.

2.2.2. iOS

Es el sistema operativo que se ejecuta en iPhone, iPod touch y en iPad. El sistema operativo administra el hardware de los dispositivos y provee las tecnologías requeridas para implementar aplicaciones nativas. El kit de desarrollo de software de iOS (SDK) contiene las herramientas e interfaces necesarias para desarrollar, instalar, ejecutar y probar las aplicaciones nativas. Las aplicaciones nativas son construidas usando los *frameworks* del sistema y el lenguaje Objective-C y se ejecutan directamente en iOS. En un nivel más alto, iOS actua como un intermediario entre el hardware y las aplicaciones que aparecen en pantalla. Las aplicaciones se comunican con el hardware a través de un conjunto de interfaces del sistema bien definidas que protegen la aplicación de cambios en el hardware. Esta abstracción facilita la codificación de aplicaciones que trabajarán de forma consistente en dispositivos con diferentes capacidades de hardware. La implementación de las tecnologías de iOS puede ser vista como un conjunto de capas, como se muestra en la Figura 2.2. En las capas más bajas del sistema se encuentran los servicios fundamentales y las tecnologías de las que depende la aplicación, en las capas de más alto nivel se encuentran los servicios y tecnologías más sofisticados [9].

La *capa cocoa touch* contiene el *framework* principal para la construcción de aplicaciones iOS. Esta capa define la infraestructura básica de la aplicación y el soporte para las principales tecnologías como multitarea, entrada basada en touch, notificaciones y diversos servicios de alto nivel del sistema. La *capa de medios* contiene las tecnologías de gráficos, audio y video orientadas a crear una mejor experiencia multimedia en el dispositivo móvil. La *capa de servicios principales* contiene los servicios del sistema fundamentales que todas las aplicaciones usan. La *capa núcleo del sistema operativo* contiene las características de bajo nivel sobre las cuales se construyen la mayoría de las aplicaciones. Se utilizan los *frameworks* de esta capa cuando se ven la seguridad y comunicación al utilizar un hardware

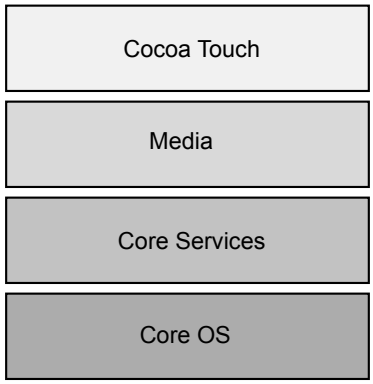


Figura 2.2: Arquitectura de sistema operativo iOS.

externo.

2.2.3. Windows Phone

El SDK de Windows Phone permite la construcción de aplicaciones usando una variedad de lenguajes y herramientas. Se pueden construir aplicaciones usando XAML *Extensible Application Markup Language* junto con cualquier lenguaje de programación, lo que permite dar mantenimiento a las aplicaciones existentes. Para proporcionar mayor flexibilidad y rendimiento, Windows Phone 8 introduce la posibilidad de usar C++ dentro de una aplicación en XAML y en los juegos desarrollados con Direct3D. En la figura 2.3 se muestra el conjunto de APIs que integran la API de Windows Phone.

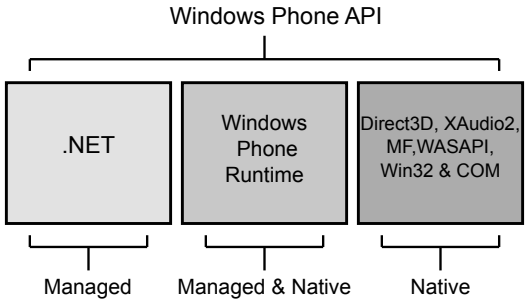


Figura 2.3: API de Windows Phone 8.

En la *API de .NET* se encuentran las clases y tipos del sistema y *namespaces* de Microsoft.Phone. La *API de ejecución de Windows Phone* es un subconjunto de un API nativa que se encuentra construida dentro del sistema operativo. Esta se encuentra implementada en C++ y proyectada dentro de C#, VB.NET y C++ lo que facilita su uso de forma natural en cualquiera de estos lenguajes. La *API Win32 y COM* permiten el acceso a características de bajo nivel de la plataforma. Esta incluye la API Windsock para trabajo en red de bajo nivel [10].

2.2.4. Symbian OS

Symbian es una compañía privada e independiente que desarrolla y proporciona mantenimiento al sistema operativo Symbian OS. Este era utilizado por algunos fabricantes de dispositivos móviles como Nokia, Ericsson, Sony Ericsson, Siemens y Samsung. Symbian se encuentra basado en el sistema operativo EPOC y fue principalmente usado en PDAs desarrolladas por Psion. Symbian OS está diseñado para soportar un rango amplio de voz y servicios de datos, como multimedia y sincronización de datos. Los servicios básicos son provistos por un *framework* de programación para los componentes del sistema operativo, los cuales incluyen APIs, controladores, archivos de sistema y una biblioteca estándar de C++. En la capa superior se encuentran los servicios básicos, como un conjunto de servicios de comunicación, servicios multimedia, conectividad a la PC y servicios genéricos del sistema operativo. El sistema operativo OS usa EPOC C++, un lenguaje puro orientado a objetos, como el lenguaje de programación con soporte para implementaciones de servicios del sistema y las interfaces de programación de aplicaciones.

2.2.5. Otras plataformas de desarrollo

Otras plataformas que pueden considerarse sólo de desarrollo, ya que no son un sistema operativo en sí son:

- *Java 2 Micro Edition*: es un subconjunto de la plataforma Java que provee de una colección de APIs certificadas para el desarrollo de aplicaciones en dispositivos como celulares, PDAs y receptores de televisión. Java ME se ejecuta sobre el kernel basado en la máquina virtual de Java, que permite un acceso razonable, pero no total del dispositivo. Java ME soporta desarrollo multiplataforma a través de un conjunto de configuraciones y perfiles:

  - Una *configuración* define las características mínimas y el conjunto de bibliotecas establecidos para una familia de dispositivos, es decir que tienen procesamiento y limitaciones de memoria similares, mismos requerimientos de interfaz de usuario y capacidades de conexión.
  - Un *perfil* involucra bibliotecas especializadas en las características únicas de un dispositivo en particular.

  Los desarrolladores deben proporcionar aplicaciones ligeramente diferentes para manejar las variaciones en los conjuntos de los JSR (Java Specification Request) y las implementaciones en los diferentes dispositivos. Este requerimiento resulta en diferentes versiones de archivos ejecutables para una sola aplicación, lo que se refiere como *fragmentación del dispositivo*, lo cual aumenta los costos de desarrollo y producción.

- *Adobe Flash Lite*: Flash Lite es una tecnología propietaria, es una plataforma popular para la programación de juegos y el desarrollo multimedia. La plataforma Flash Lite representa una buena opción para el desarrollo de aplicaciones gráficas para

teléfonos y PDAs. Su adopción se ha incrementado ya que los desarrolladores de aplicaciones Flash de escritorio pueden pasar a Flash Lite para dispositivos móviles de forma sencilla. El desarrollo acelerado es una de los principales beneficios de utilizar Flash Lite. Además de que es fácil de aprender y de migrar aplicaciones Flash a Flash Lite, ya que incluye un conjunto completo de herramientas y de forma adicional ofrece soporte multimedia (imagenes, video, sonido y animación). Flash Lite presenta un rendimiento gráfico pobre, en parte por el complejo procesamiento requerido para los gráficos vectoriales. Viene con un conjunto de herramientas completo pero este requiere pagar una licencia por su uso.

- *Python Mobile*: es un lenguaje ideal para la creación de prototipos ya que es fácil de aprender y de programar y es posible ahorrar considerable tiempo de desarrollo. Existen diferentes versiones de Python dependiendo del sistema operativo. En este caso se hace mención a PyS60 que corre en Symbian. Usualmente los scripts de Python son más cortos que el de los programas equivalentes de C, C++ y Java, por las siguientes razones [11]:

  - Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola sentencia.
  - El agrupamiento de las sentencias se realiza por medio de indentación, en vez de paréntesis iniciales y finales.
  - No es necesaria la declaración de variables o argumentos. La desventaja es que el interprete necesita ser instalado por separado con una aplicación o como un paquete separado. Debido al interprete, la velocidad de ejecución se decrementa, es por ello que Python no se recomienda para aplicaciones de alto desempeño como juegos avanzados. Otra desventaja, es que la mayoría de los sistemas operativos como iOS, Android y Windows Mobile no están habilitados para Python, lo que es una desventaja significativa en términos de adaptabilidad.

- *Qt*: es un *framework* multiplataforma que se ejecuta en computadoras tipo desktop con Windows, Linux y Mac. En los dispositivos móviles Qt se ejecuta en Symbian y MAEMO. Qt provee una biblioteca intuitiva de C++ con un conjunto de bloques de construcción de aplicaciones para el desarrollo. Qt va más alla de C++ en las áreas de comunicación entre objetos y flexibilidad para el desarrollo de GUI (*Graphical User Interface*) avanzados. Qt tiene la ventaja de ser compilado directamente dentro de los sistemas operativos por lo tanto no necesita una capa de traducción como JVM (*Java Virtual Machine*). Esto resuelve la velocidad y complejidad que lleva implícita la JVM [11].

2.3. Arquitecturas de Aplicaciones Móviles

Una arquitectura de software se define por un conjunto de decisiones principales de diseño P acerca de un sistema de software [4]. Se puede decir que la arquitectura identifica los componentes del problema, muestra las relaciones entre estos componentes y define la terminologia, reglas y restricciones en las relaciones y componentes.

El principal objetivo de la definición de una arquitectura es tratar de imponer orden en un caos [12]. Existen diferentes arquitecturas para la construcción de sistemas móviles, estas se distinguen por una serie de patrones. A continuación se muestran las principales arquitecturas de desarrollo para aplicaciones móviles.

2.3.1. Cliente-Servidor

La mayoría de las aplicaciones móviles se encuentran construidas bajo este esquema con uno o más dispositivos clientes que requieren información de un dispositivo servidor. El servidor responde con la información requerida. La arquitectura cliente-servidor utiliza capas y niveles y la comunicación se lleva a cabo entre estas capas y niveles. La creación de capas describe la división de procesos computacionales dentro del código de la aplicación en una sola máquina.

Las capas son módulos de código ubicados en diferentes directorios ya sea en el cliente o en el servidor. La capa de código que interactua de forma más cercana al usuario se conoce como la *capa de presentación*. La capa que contiene el código con las reglas de negocios de la aplicación se conoce como la *capa de negocios*. Finalmente la tercera capa que se encarga del manejo de la comunicación con la base de datos o con la fuente de datos se refiere como la capa *acceso a datos*.

Los niveles describen la división de los procesos de la aplicación en múltiples máquinas. La división en niveles involucra ubicar módulos de código en diferentes máquinas en un ambiente de servidores distribuidos. La capacidad de agregar más servidores es comúnmente conocido como escalamiento horizontal. La habilidad de agregar servidores más poderosos se conoce como escalamiento vertical [6].

2.3.2. Cliente

Los dispositivos móviles pueden trabajar como *Thin Client* o *Fat Client*. A continuación se describen las pricipales características de cada uno [6]:

a) *Thin-Client:* no cuentan con el código de la aplicación y su funcionalidad depende del servidor. Estas aplicaciones no dependen del sistema operativo del dispositivo móvil para su funcionamiento. Son más fáciles de mantener ya que no tienen el código de la aplicación o datos en sí. No hay necesidad de crear versiones de código específicas para cada dispositivo.

b) *Fat-Client:* estas aplicaciones cuentan con una o tres capas de código de la aplicación y pueden operar de forma independiente de un servidor por un periodo de tiempo. Este tipo de cliente es capaz de almacenar datos en una base de datos local hasta que se restablezca la conexión con el servidor y los datos puedan ser transferidos hacia el servidor. *Fat Clients* dependen para su funcionamiento del sistema operativo del dispositivo y la creación de versiones de la aplicación móvil para cada dispositivo puede ser costoso.

2.3.3. Rich-Client

Cuando se desarrolla una aplicación móvil, se puede escoger entre desarrollar una aplicación *thin client* basada en web o en una *rich client*. Las interfaces de usuario de las aplicaciones *rich client* pueden proveer alta capacidad de respuesta, son interactivas, enriquecen la experiencia de usuario para las aplicaciones que deben funcionar en modo autónomo, conectado, a veces conectado y en escenarios desconectados. Una aplicación de tipo *rich client* se estructura de forma multicapa, las cuales consisten de la capa de experiencia del usuario (presentación), negocios y de datos, como se muestra en la figura 2.4. Una aplicación de este tipo puede usar datos almacenados en un servidor remoto,

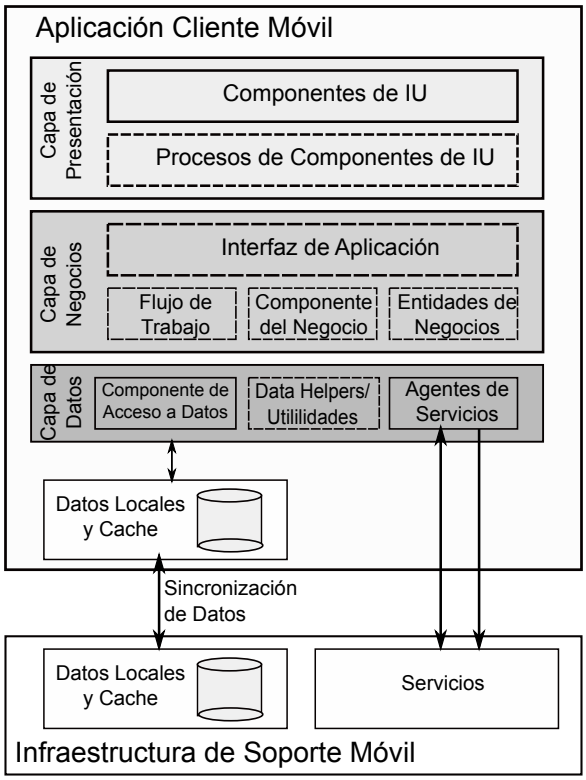


Figura 2.4: Arquitectura rich client.

locales o una combinación de ambos. Esta puede consumir servicios expuestos por otras

aplicaciones, incluyendo servicios Web. Se recomienda el desarrollo de una aplicación *rich client* si se tienen los siguientes casos [7]:

- La aplicación opere en escenarios desconectados u ocasionalmente conectados.
- La aplicación debe ser altamente interactiva y con respuesta rápida.
- La interfaz de usuario debe proporcionar alta funcionalidad pero no debe requerir de gráficos avanzados.
- La aplicación debe utilizar los recursos del dispositivo cliente.

2.3.4. Servidor Centralizado

Esta arquitectura se compone de una a tres capas de código implementadas de uno a tres niveles. Otras arquitecturas son consideradas, basadas en el número de niveles que los servidores implementan [6]:

a) *Arquitectura de un nivel:* las tres capas de código se encuentran en un sólo servidor. Esta arquitectura facilita la tarea de desarrollar e implementar la aplicación, pero exponen el acceso a datos a un acceso no autorizado.

b) *Arquitectura de dos niveles:* el servidor de base de datos se separa del servidor encargado de las capas de *presentación/aplicación*. Esto es conveniente porque permite tener un servidor especializado para el acceso a los datos pero es mucho más caro su mantenimiento y dificulta el escalamiento de la aplicación.

c) *Arquitectura de tres niveles:* las capas de acceso a datos, aplicación y presentación se encuentran separadas una de otra. Esta arquitectura permite servidores especializados, los cuales son escalables. Por otro lado el desarrollo de la aplicación se dificulta, el desarrollo y mantenimiento es más caro.

Para poder decidir qué arquitectura es más conveniente implementar es importante revisar los tipos de conexión del dispositivo, los cuales se mencionan a continuación:

- *Siempre Conectado:* este modo permite a los usuarios estar en constante conexión mientras se encuentra en movimiento.
- *Parcialmente Conectado:* el dispositivo móvil se encuentra fuera de conexión por largos periodos de tiempo.
- *Desconectado:* el dispositivo móvil nunca se conecta a sistemas finales, tal como algunos dispositivos de entretenimiento, especificamente videojuegos.

Los tipos de conexión afectan la forma en la que los dispositivos móviles pueden sincronizar datos con sistemas finales. La sincronización puede ocurrir de dos formas: continua o a través de métodos *guardar y reenviar*:

- *Continua:* cuando la conectividad es constante entre el cliente y el servidor, la sincronización de los datos entre el cliente y el servidor se realiza de manera constante.
- *Guardar y Reenviar:* cuando la conectividad entre el cliente y el servidor no puede ser garantizada. Una aplicación móvil cliente almacena datos en una base de datos local. Posteriormente, cuando la conexión ha sido restablecida, la aplicación móvil enviará datos desde el dispositivo hacia la base de datos del servidor.

2.3.5. Punto a Punto

Las arquitecturas llamadas punto a punto están diseñadas para compartir recursos computacionales como contenido, almacenamiento o ciclos de CPU, por directo intercambio, en vez de requerir un servidor centralizado de intermediario. Estas arquitecturas se adaptan a los fallos y acomodan las poblaciones de nodos itinerantes manteniendo la conectividad y el desempeño. Los sistemas punto a punto pueden ser definidos como *"...sistemas distribuidos conformados de nodos interconectados capaces de organizarse por sí mismos en topologías de red con el propósito de compartir recursos tales como contenido, ciclos de CPU, almacenamiento y ancho de banda, capaces de adaptarse a fallos y acomodar nodos itinerantes mientras mantienen una conectividad y desempeño aceptable, sin requerir de intermediación o soporte de un servidor global centralizado o autoridad"* [13]. Las principales características de la arquitectura son las siguientes [13]:

- Los nodos de una red punto a punto no dependen de un servidor central que coordine el intercambio de contenido y la operación de la red, estos participan de forma activa, independiente y unilateral realizando tareas tales como búsquedas por otros nodos, localización y obtención de contenido, direccionamiento de información y mensajes, conexión y desconexión de otros nodos vecinos, cifrado, introducción, descifrado y verificación del contenido.
- Habilidad de manejar la conectividad inestable y variable como una norma, adaptándose automáticamente a los fallos de conexiones de red y de nodos, así como a una población de nodos transitoria. Esta tolerancia a fallos sugiere la necesidad de una topología de red adaptable que cambiará a medida que los nodos entren o salgan y las conexiones de red fallen y se recuperen, con la finalidad de mantener su conectividad y rendimiento.

2.3.6. Cómputo en la nube

En un entorno de *cómputo en la nube*, el rol tradicional de proveedor de servicios se divide en dos partes: el proveedor de infraestructura el cual maneja las plataformas en la nube y el proveedor de servicios, que alquilan los recursos de unos o muchos proveedores de infraestructura para servir a los usuarios finales [14]. La definición de *nube* que provee el Instituto Nacional de Standards y Tecnología dice: *"el cómputo en la nube es un modelo para habilitar acceso en demanda a un conjunto compartido de recursos informáticos*

configurables (redes, servidores, almacenamiento, aplicaciones y servicios) que pueden ser provistos y liberados de forma rápida con el mínimo esfuerzo de gestión o interacción de proveedores de servicios" [14]. El computo en la nube se refiere tanto a las aplicaciones proporcionadas como servicio a través de Internet y al hardware y a los sistemas de software en los centros de datos que proveen estos servicios [15]. La arquitectura de un entorno de cómputo en la nube puede ser dividido en capas: la capa de hardware (centros de datos), la capa de infraestructura, la capa de plataforma y la capa de aplicación (ver Figura 2.5). Cada capa tiene una función específica como se describe a continuación [14][16]:

Figura 2.5: Modelo en capas de arquitectura de cómputo en la nube.

- *Capa de aplicación:* se conforma de las aplicaciones en la nube. A diferencia de las aplicaciones tradicionales, las aplicaciones en la nube pueden aprovechar la característica de escalamiento automático para lograr un mejor rendimiento, disponibilidad y menor costo de operación.

- *Capa de plataforma:* esta se encuentra construida sobre la capa de infraestructura, esta capa consiste del sistema operativo y *frameworks* de aplicaciones, ofrece un avanzado entorno integrado para construir, probar e implementar aplicaciones personalizadas. El principal propósito de esta plataforma es minimizar la carga de la implementación directa sobre los contenedores de la máquina virtual.

- *Capa de infraestructura:* esta crea un grupo de recursos computacionales y de almacenamiento, mediante la partición de los recursos, usando tecnologías de virtualización. La capa de infraestructura es un componente esencial en el cómputo en la nube, ya que muchas de las principales características, como la asignación dinámica de recursos son sólo disponibles a través de las tecnologías de virtualización.

- *Capa de hardware:* esta capa gestiona los recursos físicos como: servidores, routers, switches, energía y sistemas de enfriamiento. Esta capa se implementa en centros de datos. Un centro de datos se conforma por servidores que están organizados en racks e interconectados. Los problemas más comunes de esta capa incluyen la configuración del hardware, la tolerancia a fallos, la gestión del tráfico, recursos energéticos y refrigeración.

Las aplicaciones móviles pueden almacenar datos en la nube para resolver la restricción del limitado espacio de almacenamiento en el dispositivo (procesamiento multimedia), las aplicaciones usan el *cloud* para escalar de forma infinita y permanente el almacenamiento, los dispositivos móviles usan la nube como un punto de encuentro para compartir datos con otros dispositivos y para algunas aplicaciones que realizan procesos computacionales complejos en la nube [17]. *"El cómputo móvil en la nube, se refiere a la infraestructura donde se almacenan los datos y se realiza el procesamiento fuera del dispositivo móvil. Las aplicaciones móviles en la nube transportan la capacidad computacional y el almacenamiento de datos fuera de los dispositivos móviles hacia la nube, lo que crea aplicaciones y cómputo móvil no solo para usuarios de* smartphones *si no para un rango más amplio de usuarios de diferentes dispositivos móviles"* [16]. Existen diversas razones para considerar que el cómputo móvil en la nube puede ser usado para resolver algunos problemas del cómputo móvil [16]:

- *Ampliación de la vida de la bateria:* la bateria es una de las principales limitantes de los dispositivos móviles. El cómputo en la nube se propone con el objetivo de migrar procesos computacionales complejos de los dispositivos móviles hacia los servidores en la nube. Esta solución evita el consumo de energía en el dispositivo móvil.
- *Mejoramiento de la capacidad de almacenamiento y de procesamiento:* el cómputo móvil en la nuble permite a los usuarios almacenar y acceder a grandes volúmenes de datos en la nube a través de la red inalámbrica.
- *Mejoramiento de la confiabilidad:* el almacenamiento de los datos y la ejecución de aplicaciones en la nube mejora la confiabilidad porque los datos y las aplicaciones son almacenadas y respaldadas en diferentes computadoras. Esto reduce la pérdida de datos y el número de aplicaciones ejecutadas en el dispositivo.

2.3.7. Hub

La idea básica de una arquitectura *hub* es un servidor que enruta mensajes. En el *hub* hay oportunidades de realizar las siguientes acciones [12]:

- Enrutar el mensaje usando tipo de mensaje, origen, volumen del tráfico, volumen del tipo del mensaje, etc.
- Reformatear el mensaje.
- Enviar el mensaje a más de un destinatario.
- Agregar información al mensaje.
- Dividir el mensaje, enviar a diferentes partes a diferentes destinatarios.
- Realizar una verificación de seguridad adicional.
- Definir las reglas del flujo de trabajo.

- Monitorizar el flujo de los mensajes.

Un *hub* puede ser usado como puente entre diferentes redes o tecnologías tipo *middleware*. Existen dos tipos de *hubs*, los que manejan las interacciones de peticiones-respuesta y los *hubs* que se encargan de enrutar mensajes aplazados (ver figura 2.6). Los *hubs* para interacciones de petición-respuesta son los que tienen más problemas. Claramente estos deben ser rápidos y resistentes, ya que están en la ruta crítica del tiempo de respuesta. Los *hubs* para reformatear y enrutar mensajes diferidos son mucho más simples. Por definición no hay problema con los retardos de la aplicación con los mensajes diferidos y el procesamiento de los mensajes diferidos no puede usar un estado de sesión.

La ventaja de una arquitectura de tipo *hub* yace en la funcionalidad adicional que provee. Una alternativa al *hub* es crear una aplicación con la lógica que envía el mensaje. Por ejemplo, en vez de reconstruir el mensaje, el destino puede crear un mensaje con el formato correcto. En vez de realizar el enrutamiento en el mensaje, el destino puede determinar hacia donde debe dirigirse el mensaje. Pero esta solución no es la mejor si hay varias aplicaciones realizando peticiones de información, por ello la mejor opción es tener el ruteo de la información en un *hub*. Los *hubs* son útiles en algunos escenarios, como puente ente tecnologías de red. Otro caso es aplicable cuando los *hubs* permanecen como puente en aplicaciones donde estas no pueden adaptarse a las necesidades de formato y de enrutamiento. Las arquitecturas *hub* pueden ser vistas como menos rígidas, en los *hubs* pueden resolverse diferencias entre el destino y destinatario [12].

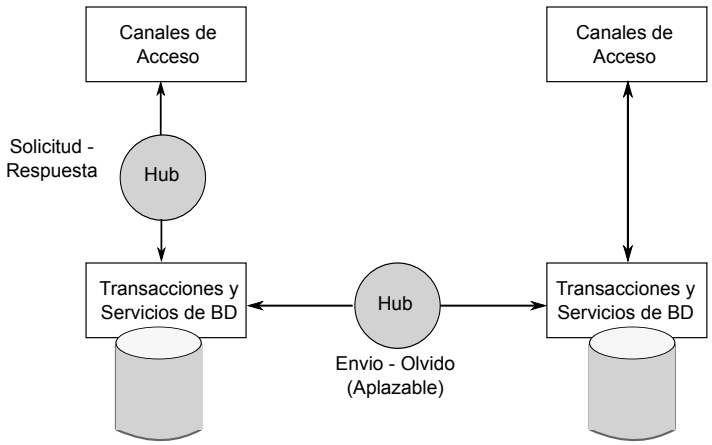


Figura 2.6: Arquitectura Hub.

2.3.8. Middleware

Otro enfoque para el desarrollo de aplicaciones móviles es por medio de la implementación de *Middlewares*. Un *"Middleware es una aplicación necesaria en la práctica para la construcción de aplicaciones distribuidas"* [12]. *"Software que proporciona un modelo*

de programación sobre bloques básicos arquitectónicos: procesos y paso de mensajes" [18]. Una característica importante de un *middleware* es que es capaz de operar sobre la red. Los siguientes elementos se consideran parte de un *middleware* (ver figura 2.7 [12]):

- *Enlace de comunicación:* habilita a las aplicaciones A y B el envió de datos recíproco a través de un enlace físico de comunicación, ya sea de área local o amplia.
- *Protocolos:* hay dos conjuntos de protocolos, un protocolo de red para llevar los datos a través de la red y el protocolo del *middleware* para manejar el dialogo entre las aplicaciones A y B. Los dos conjuntos de protocolos son complementarios y juntos proveen el grado requerido de confiabilidad y desempeño.
- *Interfaz de programación y el formato común de datos:* definen la forma en la que las aplicaciones se comunicarán con el *middleware*. El formato común de datos describe cómo se deben estructurar los datos para que ambas aplicaciones puedan entenderse y la interfaz de programación de aplicaciones define la forma en la que los datos son presentados al *middleware*. Estos cuatro elementos juntos son suficientes para asegurar la comunicación requerida entre las aplicaciones A y B.
- *Servidor de control de procesos:* se refiere a la forma en la que el sistema operativo, *middleware* y otro software administra la calendarización y ejecución de las aplicaciones, es crucial para el rendimiento, específicamente para la escalabilidad.
- *Servicio de directorio de nombres:* proporciona los medios para localizar los elementos de comunicación.
- *Seguridad:* se refiere a asegurar que la comunicación entre las aplicaciones A y B es lo suficientemente segura y cumple con los requerimientos, esto incluye cifrado, identificación fiable de los sistemas y otorgamiento de permisos para el acceso.
- *Administración de sistemas:* se refiere a la configuración, operación, manejo de errores y rendimiento del ambiente, este mantiene a todo el sistema operando de forma correcta.

La capa de *middleware* usa protocolos de paso de mensajes entre los procesos, con la finalidad de proveer abstracciones de alto nivel, tales como invocaciones remotas y eventos (ver figura 2.7). Un aspecto importante del *middleware* es que proporciona transparencia de ubicación e independencia de los detalles de los protocolos de comunicación, los sistemas operativos y el hardware de los dispositivos. Algunas formas de *middleware* permiten que los componentes separados se encuentren escritos en diferentes lenguajes de programación [18].

- *Transparencia frente a ubicación:* en RPC (*Remote Procedure Call*), el cliente que llama a un procedimiento remoto no puede distinguir si el procedimiento remoto se ejecuta en el mismo proceso o en un proceso diferente, posiblemente en otro dispositivo. El cliente tampoco necesita conocer la ubicación del servidor. De forma

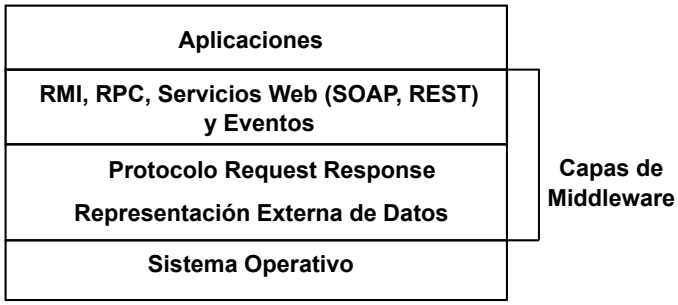


Figura 2.7: Capa de middleware.

análoga, en RMI (*Remote Method Invocation*) el objeto que realiza la invocación no puede definir si el objeto que invoca es local o no y tampoco requiere conocer su ubicación. Asimismo, en los programas distribuidos basados en eventos, los objetos que generan eventos y los objetos que reciben notificaciones de esos eventos tampoco necesitan estar al tanto de sus ubicaciones respectivas.

- *Servicios Web:* sistemas de software diseñados para soportar un interacción interoperable máquina a máquina sobre una red.
- *Protocolos de comunicación:* los protocolos que dan soporte a las abstracciones del *middleware* son independientes de los protocolos de transporte subyacentes.
- *Hardware de los dispositivos:* se emplea el empaquetado y desempaquetado de mensajes. Estos ocultan las diferencias de la arquitectura en el hardware, como el ordenamiento de los bytes.
- *Sistemas operativos:* las abstracciones de mayor nivel que provee la capa de *middleware* son independientes de los sistemas operativos subyacentes.
- *Utilización de diversos lenguajes de programación:* diversos *middlewares* se diseñan para permitir que las aplicaciones distribuidas sean escritas en más de un lenguaje de programación. Esto se obtiene por medio de un lenguaje de definición de interfaz.

2.4. Frameworks y bibliotecas multiplataforma

Los dispositivos móviles se han convertido, más que en un accesorio, en una herramienta indispensable para la comunicación diaria. Debido a esto, ha aumentado la demanda para el desarrollo de aplicaciones móviles. Una de las problemáticas a las que se enfrenta este tipo de desarrollo, es la poca compatibilidad existente entre las diversas plataformas móviles. Por lo que se deben realizar desarrollos específicos para cada plataforma, lo que aumenta la carga de trabajo y no hay reutilización de código [19].

La fragmentación existente en los dispositivos representa un problema para los desarrolladores por la diversidad de sistemas operativos móviles y APIs. El acceso a los sensores

del dispositivo como el GPS y el acelerómetro puede estar restringido a ciertas aplicaciones o lenguajes de programación [20]. Esta fragmentación se debe a la constante competencia por el mercado de los fabricantes de dispositivos y de operadores de servicios. Cada compañía desea distinguirse ofreciendo un producto diferente y novedoso. Los fabricantes diseñan su propia versión (a menudo incompatible con las demás) sin esperar a que se desarrolle un estándar unificador [21]. Como se mencionan en [22] una aplicación es adaptable entre distintas clases de entornos si el esfuerzo requerido para transportar y adaptar la aplicación a una nueva clase de entorno es menor que el esfuerzo de rediseñarla. Existen dos niveles de adaptabilidad:

- *Adaptabilidad a nivel binario:* una aplicación puede transportarse como ejecutable a un nuevo entorno y ejecutarse ahí sin necesidad de cambios, ni de disponer del código fuente.

- *Adaptabilidad a nivel de código:* este tipo de adaptabilidad existe si el código fuente de la aplicación puede transportarse y adaptarse para su funcionamiento en un nuevo entorno con poco esfuerzo. Se requiere una nueva compilación y enlazado de la aplicación.

El objetivo ideal sería poder crear una sola aplicación usando un conjunto de APIs bien documentadas que se encuentren disponibles para todos los dispositivos móviles independientemente de la plataforma o red [20]. Es decir crear una aplicación adaptable entre distintas clases de entornos.

Desde el punto de partida del desarrollador de la aplicación, es muy caro tener versiones en diversas plataformas. En específico cuando existen varias versiones para cada dispositivo, por ello se tienen las siguientes opciones [8]:

- Desarrollar para una sola plataforma y en la medida de lo posible, un subconjunto común de las funciones disponibles a todas las variantes y versiones de esta plataforma, por ejemplo, el desarrollador debería tener un solo código base para una aplicación que debería correr en diferentes versiones de iPhone, iPad y posiblemente el iPod Touch. Este enfoque simplificaría el trabajo del desarrollador, pero la aplicación resultante no sería capaz de obtener ventaja de las diferentes características de cada dispositivo.

- Desarrollar aplicaciones nativas para cada plataforma y actualizaciones, lo que contrapone los costos de mantenimiento y desarrollo con la habilidad de optimizar la aplicación para cada plataforma.

- Desarrollo de aplicaciones web, esto minimiza la cantidad de código nativo para cada plataforma, pero aún es un tanto incierto si este enfoque satisface las necesidades del mercado, dada la falta de compatibilidad entre navegadores.

- La utilización de capas de abstracción que permitan desarrollar una aplicación sólo una vez en programas ejecutables nativos, pero que puedan ejecutarse en múltiples plataformas.

A partir de los 90's, empezaron a surgir los *frameworks* multiplataforma para las aplicaciones de escritorio, de esta forma se facilitó el desarrollo de aplicaciones en un sólo código que podía compilarse para cada plataforma (Mac o Windows). Para la mayoría de los sistemas operativos móviles como Symbian, iOS, Android, Windows Phone existe un lenguaje nativo de desarrollo, el cual es requerido para crear aplicaciones óptimas para esa plataforma. Mientras que es posible crear aplicaciones en otros lenguajes, normalmente hay inconvenientes o limitaciones en el desarrollo. Un ejemplo, es al desarrollar una aplicación en Java para la plataforma Symbian, sin embargo, diversas APIs (*Application Programming Interface*) que permiten el acceso diversas funcionalidades del dispositivo, no se encuentran disponibles.

Las características de los diferentes dispositivos móviles son muy similares, GPS, cámara, acceso a contactos, almacenamiento interno, pero para usar estas características se requieren APIs específicas en cada plataforma. La creación de nuevos *frameworks* para *smartphones* ha sido impulsada por la necesidad del desarrollo acelerado de aplicaciones como se observa en las aplicaciones para web. Existen tres técnicas específicas en el desarrollo de aplicaciones web que se han utilizado para los *frameworks* móviles [23]:

1. Diseño basado en etiquetas con (HTML/CSS).
2. Uso de URLs para identificar diseños de pantalla y el estado visual.
3. Incorporación de lenguajes dinámicos, como Javascript y Ruby.

Los nuevos *frameworks* multiplataforma aprovechan estas habilidades a través de un navegador web incorporado como mecanismo para la visualización de la interfaz de usuario de la aplicación. Esto se combina con una aplicación nativa que transforma las peticiones a URLs en la representación de la aplicación móvil que simula el entorno web en el contexto de una aplicación móvil desconectada.

En 1990, Microsoft y Apple crearon plataformas GUI con ventanas, entrada por medio del mouse, menús, etc. Los distribuidores de software necesitaban aplicaciones para ambas plataformas, por lo que los desarrolladores de software crearon bibliotecas y *frameworks* para abstraer las diferencias, haciendo fácil el desarrollo de una aplicación que pueda ejecutarse sobre estas plataformas. En los 2000's la mayoría de las aplicaciones se movieron hacia un entorno web y hacia diferencias sintácticas en navegadores. Los desarrolladores de software crearon bibliotecas multiplataforma y *frameworks*, como jQuery, Dojo y OpenLaszlo.

Antes de que los *frameworks* multiplataforma se crearan, muchos desarrolladores encontraron que insertando una interfaz de usuario web en una aplicación nativa era una manera práctica para desarrollar aplicaciones móviles multiplataforma de fácil mantenimiento. Cada plataforma móvil tiene un control de interfaz de navegador web que puede ser insertado en una aplicación como un checkbox o un botón. Mediante la colocación de

un control de navegador web en la aplicación que sea del tamaño de la pantalla, se consigue que toda la interfaz de usuario se implemente en HTML. Actualmente han surgido diversos *frameworks* multiplataforma. Estos *frameworks* caen en dos categorías: aquellos que crean una aplicación móvil nativa usando APIs multiplataforma y aquellos que usan HTML/CSS/Javascript que permiten crear interfaces multiplataforma que se ejecutan en un navegador web [23].

Algunas características de ambos tipos de *frameworks* son [20]:

- *Ligeros:* debido a las limitaciones de ancho de banda, se hace énfasis en el peso reducido del archivo correspondiente al *framework*.
- *Optimizados para dispositivos tipo touchscreen:* los dedos como dispositivos de entrada en vez del mouse proporcionan un conjunto adicional de retos para el diseño de interfaces de usuario. Los *frameworks* de desarrollo móvil ofrecen una interfaz de usuario estándar y el control de eventos específicos para las plataformas móviles.
- *Usan estándares de HTML5 y CSS3:* la mayoría de los dispositivos móviles tienen navegadores que soportan HTML5 y CSS3, por lo que los *frameworks* de desarrollo toman ventaja de las nuevas características disponibles en las especificaciones de W3C para una mejor experiencia al usuario.

Algunos ejemplos de estos *frameworks* multiplataforma son [20]:

- *Titanium:* provee un sólo SDK (*Software Development Kit*) para todos los sistemas operativos móviles. Usando una interfaz basada en JavaScript, se pueden almacenar las preferencias de usuario, guardar archivos de datos, o implementar una versión móvil de una cookie mediante SQL Lite o el sistema de archivos nativo de iOS o Android [24].
- *PhoneGap:* es un *framework* abierto y gratuito el cual permite crear aplicaciones móviles usando APIs web estandarizadas para diferentes plataformas móviles[25]. Las aplicaciones resultantes son híbridas, lo que significa que no son totalmente nativas (todo el diseño se lleva a cabo através del navegador web) pero no son totalmente basadas en web.
- *MoSync:* es una herramienta para el desarrollo acelerado de aplicaciones móviles en HTML5 y Javascript para iOS, Android y Windows Phone. Con esta herramienta se puede codificar en HTML5 y generar versiones para diferentes dispositivos [26].
- *Rhodes y RhoSync:* esta herramienta hace uso de Ruby para la lógica de negocios, es un tipo de *framework* de MVC (*Modelo Vista Controlador*) y aprovecha HTML, CSS y Javascript para la interfaz de usuario. El servidor opcional RhoSync admite la sincronización de datos de cliente-servidor. Con Rhodes se pueden construir aplicaciones para iOS, Android, Blackberry y Windows Mobile [23].

Adicionalmente a estos *frameworks* para el desarrollo de aplicaciones nativas, existen diversos *frameworks* para crear HTML,CSS y Javascript para aplicaciones móviles web. Muchos de estos *frameworks* son más un conjunto de estilo y elementos gráficos. Algunos de estos *frameworks* son [23]:

- *Sencha Touch:* un *framework* de JavaScript que permite construir aplicaciones móviles web en HTML5 con CSS3 y Javascript para iOs y Android. De igual forma este *framework* ofrece integración de datos con varias fuentes como AJAX, JSON y YQL [20].
- *JQuery Mobile:* un *framework* multiplataforma que provee herramientas para la construcción de interfaces dinámicas de tipo *touch* que se adaptarán a una amplia variedad de formas de dispositivos. Este incluye dos tipos de presentación (listas, paneles) y un amplio conjunto de controles de formulario y *widgets* de interfaz de usuario [20].
- *Dojo Toolkit:* es un *framework* adaptable usado para construir aplicaciones web [23].

Capítulo 3

Sistemas de reconocimiento de voz para dispositivos móviles

En este capítulo se muestra el surgimiento y evolución de las aplicaciones de reconocimiento de voz (sección 3.1), hasta llegar a aplicaciones de reconocimiento de voz para dispositivos móviles donde se muestran los principales ejemplos de aplicaciones existentes y la importancia de la voz como instrumento de interacción humano-máquina (sección 3.2). Se muestran los principales algoritmos para el procesamiento de voz (sección 3.3) y las principales arquitecturas para la construcción de aplicaciones de reconocimiento de voz en dispositivos móviles (sección 3.4).

3.1. Historia del desarrollo de sistemas de reconocimiento de voz

Las primeras investigaciones acerca del procesamiento de la voz iniciaron en 1948 cuando Ralph K. Potter, George A. Kopp y Harriet C. Green demostraron que los sonidos del habla tenían patrones característicos, esto por medio de espectrogramas de sonidos y sílabas tomados con el sonograma, un instrumento mecánico de análisis espectral. A la par, la computación fue evolucionando. En la década de los 60's salió la computadora IBM 1620, la cual tenía una capacidad de 4000 palabras de memoria. Siguiendo la Ley de Moore, en 1970 apareció la supercomputadora PDP-10 y la minicomputadora PDP-11 por Digital Equipment, con ello inicio la era del procesamiento de señales. En los laboratorios de AT&T, en Bolt Beranek y Neumann, en Laboratorio de Tecnología del Habla en Santa Barbara, los investigadores estudiaron los matices de voz por medio de codificación de predicción lineal. Este eficiente algoritmo permite calcular las resonancias asociadas al sonido del habla de una forma sencilla, ignorando el tono de voz. De esta forma se confirmó la propuesta realizada por Potter, Kopp y Green que sostenía que los sonidos del habla tienen una estructura predecible.

En 1965 J.W. Cooley and J.W. Tukey publicaron el trabajo “An algorithm for the

machine calculation of complex Fourier Series", a este algoritmo se le conoce como *Fast Fourier Transform* (FFT) [27], esta técnica redujo el tiempo computacional del cálculo de la Transformada Discreta de Fourier de $\Theta(n^2)$ a un tiempo de $\Theta(n\ lg\ n)$, es por ello que se ha utilizado principalmente en el procesamiento de señales. El análisis de Fourier permite expresar una señas como una suma ponderada de sinusoides desfasadas de diferentes frecuencias. Los pesos y las fases asociadas con las frecuencias caracterizan la señal en el dominio de la frecuencia [28].

En 1970 Fumitada Itakura y Shuzo Saito notaron que el espectro derivado de LPC (Codificación de Predicción Lineal) remarcaba las resonancias del tracto vocal y definieron la "Distancia Itakura" la cual, bajo los supuestos correctos, expresa la distancia entre dos espectros como información a medir [29].

Para el análisis del reconocimiento de voz, LPC puede ayudar a estimar la recurrencia de la vibración de las cuerdas vocales, la forma del tracto vocal y las frecuencias y anchos de banda de los polos y ceros espectrales (resonancias del tracto vocal). Proporciona un número de parámetros de voz (llamados coeficientes LPC), que se refieren a la configuración del tracto vocal y a cómo los sonidos son pronunciados. Estos coeficientes se pueden usar en filtros digitales como valores multiplicadores que generan una versión sintética de la señal original, que puede almacenarse como una plantilla para el reconocimiento de patrones [30].

Otra ramificación del reconocimiento de voz, es por medio de Modelos Ocultos de Markov (HMM), desarrollada por IBM. Esta tecnología fue presentada a la comunidad por IDA (Institute for Defense Analysis) en 1982, pero el área de investigación de IBM la había estado explorando desde 1970 en un amplio conjunto de aplicaciones del habla. La mayoría de las aplicaciones modernas de reconocimiento de voz utilizan esta tecnología.

Los Modelos Ocultos de Markov permiten modelar por fonema o palabra en vez de partes de expresiones. Los algoritmos de reconocimiento de HMM modelan la señal de voz en vez de la composición acústica, por lo que tienden a ser más robustos al ruido y a la distorsión. En las aplicaciones actuales, el costo asociado con los errores en reconocimiento es mucho más bajo y dado que las técnicas sofisticadas de supresión de ruido son muy costosas computacionalmente no son viables, por lo que es muy recomendable usar HMM para datos con ruido [29]. En 1973, se iniciaron las primeras investigaciones para el desarrollo de un teléfono móvil, en ese año Martín Cooper de Motorola mostró un teléfono compacto de 2.2 libras, esta tecnología fue convertida en un teléfono de auto (posteriormente en un teléfono de bolsillo), con lo que inicio la era móvil en los 80's. La infraestructura celular cambio de ser radio manejado por operadora, al servicio de marcado analógico y posteriormente al servicio digital durante los años 80's y 90's. A principios de los 80's se desarrollaron los primeros reconocedores de voz para teléfonos de automóvil, como el marcador para coche producido por Intersate Electronics y el Victory Dialer por AT&T [29].

En 1986, Rumelhart, Hinton y Williams, formalizaron un método para que una red neuronal aprendiera la asociación que existe entre los patrones de entrada de la misma y las clases correspondientes, utilizando más niveles de neuronas. Este nuevo método se le conoce como *Backpropagation* que es un tipo de aprendizaje supervisado, el cual emplea un ciclo de propagación-adaptación de dos fases [31].

A mediados de los 90's inicio la era celular digital. Con ello evolucionaron los teléfonos, redujeron su tamaño y aumentaron su capacidad de procesamiento y memoria. En el año 2002, se presentó el modelo Samsung A500, el cual posee un sistema de reconocimiento de voz independiente del usuario. Este permite al usuario presionar un botón y decir "llamar número" y después de una indicación verbal por el teléfono, se puede decir una serie de dígitos que representan el número telefónico. Este sistema no requiere entrenamiento y fue el primer sistema de reconocimiento de voz que utilizó HMM en un teléfono móvil.

Los desarrollos para sistemas móviles se han vuelto un mercado importante a cubrir, debido al desarrollo del cómputo móvil. La mayor ventaja que ofrecen las interfaces de reconocimiento de voz, es que permiten al usuario realizar otras actividades, sin tener que controlar el dispositivo móvil con sus manos, o permiten la interacción con personas con discapacidades visuales. Un ejemplo del reconocimiento de voz integrado, es la creación del *framework iSay-SMS*. Este *framework* permite convertir la voz en abreviaturas de texto personalizadas, las cuales se encuentran definidas en una base de datos interna en el teléfono, lo que permite la redacción de mensajes de texto con abreviaturas definidas previamente por el usuario [32]. Uno de los últimos desarrollos realizados, son *Voice Action* de Google y *Siri* para iPhone; estas aplicaciones habilitan el control del teléfono móvil por medio de la voz, donde se puede hacer llamadas, enviar mensajes de texto o emails, navegar por Internet y completar algunas tareas como abrir otras aplicaciones. Ambas aplicaciones requieren de conexión a Internet para realizar el procesamiento de la voz, por medio de un servidor [33].

Otra aplicación a mencionar es *WAMI (Web Accesible Multimodal Applications)* [34] desarrollada por el grupo Spoken Language Systems del MIT Computer Science and Artificial Intelligence Laboratory. Esta plataforma consiste en un conjunto de componentes que se ejecutan por medio de un servidor y un cliente que se ejecuta en el navegador del dispositivo cliente. Los componentes que se encargan de la parte del reconocimiento de voz que se encuentran en un servidor web sobre JavaEE (Java Enterprise Edition) y una bitácora de operaciones.

Se han desarrollados algunos frameworks para el procesamiento de reconocimiento de voz, para dispositivos móviles se tiene *OpenEars*, el cual permite realizar el reconocimiento de voz para la plataforma iOS. Este *framework* es capaz de realizar el reconocimiento en base a un diccionario de palabras definidas. La ventaja de este *framework* es que no utiliza la conectividad de la red para realizar el procesamiento, ya que lo realiza de manera local

en el dispositivo [35]. Otro *framework* multiplataforma para el reconocimiento de voz es Sphinx-4, un *framework* modularizado que incorpora patrones de diseño de sistemas existentes con suficiente flexibilidad. Este *framework* se encuentra codificado en Java y es ejecutado como un servicio, el cual se puede usar por medio de un API [36].

3.2. Aplicaciones de reconocimiento de voz para dispositivos móviles

Se han creado diversas aplicaciones que realizan reconocimiento de voz, que permiten la manipulación de los dispositivos de una forma más amigable al usuario. Como ejemplo tomado de la referencia [37] se mencionan:

- La *marcación por nombre*, esta función es muy útil, ya que realiza la función básica de un teléfono celular, llevar a cabo una llamada. Después de la activación de la función, el usuario dice el nombre de la persona que será llamada. Esto implica una acción inmediata. Para la marcación por nombre, se necesita un registro predefinido de nombres con sus números asociados (una base de datos de contactos).
- Aplicaciones de *orden y control*, donde el usuario ingresa la acción deseada, la cual es pasada a un intérprete y la acción es ejecutada.
- *Dictado*, su alcance es muy diferente, ya que esta funcionalidad no es básica para los teléfonos. Para uso general, los sistemas de reconocimiento de voz son más propensos a tener éxito en el entorno móvil, donde los usuarios se ven motivados a utilizar esta tecnología debido a lo complicado que resulta el manejo de los mecanismos de interfaz tradicionales. El dictado por reconocimiento de voz puede ser implementado en la nube, por medio de la transmisión de la voz sobre la red inalámbrica, o puede llevarse a cabo de forma parcial en el teléfono y en la nube.
- *Reconocimiento de palabras aisladas* es la capacidad de reconocer una sola palabra (orden). El reconocimiento de palabras aisladas es útil para el marcado por nombre y aplicaciones de mando y control, sin embargo este resulta muy artificial.
- La *identificación de palabras clave* permite una operación más amigable al usuario porque no es necesario que el usuario proporcione palabras aisladas. El usuario puede hablar frases de forma natural las cuales contienen las palabras clave y los comandos. El proceso de reconocimiento de voz separa las palabras clave útiles de la información que no es útil (clasificada como basura). El tamaño del vocabulario puede mantenerse restringido, de igual forma al reconocimiento de palabras aisladas, con más de 100.

La identificación de palabras clave es un tipo de reconocimiento de palabras conectadas. Se refiere a una técnica en la cual al usuario se le permite hablar varias palabras de una

forma hilada. El reconocimiento de voz conectado mejora la calidad de las interfaces de usuario. La dificultad es que las palabras son pronunciadas diferente cuando están hiladas que cuando se pronuncian de manera aislada. Por otra lado, la complejidad computacional y los requerimientos de memoria aumentan significativamente.

Existe otro enfoque que considera la diferencia entre sistemas de reconocimiento de voz y los cataloga como reconocimiento de voz *dependiente de la persona, independiente de la persona y adaptados por la persona* [37]. En un *sistema de reconocimiento de voz dependiente de la persona*, el usuario puede incluir cualquier nueva palabra en el diccionario con la finalidad de entrenar al sistema. De esta forma, el lenguaje y pronunciación del usuario se toman en cuenta de forma automática. El *reconocimiento de voz dependiente de la persona* es independiente de lenguajes, dialectos y pronunciaciones. Ejemplo de aplicaciones de reconocimiento de voz dependientes de la persona es la marcación por nombre. Los usuarios no desean entrenar al sistema, o olvidan las etiquetas que ya asignaron. Los sistemas independientes superan esas dificultades por el preentrenamiento realizado a modelos del lenguaje sobre grandes cantidades de datos de entrenamiento, en los cuales se encuentran predominantemente los modelos ocultos de Markov (HMM) [38].

Para resolver el problema de la diversidad de lenguajes, dialectos y modismos de los usuarios, se han realizado modificaciones en el diseño de algoritmos y en el preentrenamiento de los sistemas de reconocimiento de voz independientes de la persona basados en modelos ocultos de Markov (HMM). Para la obtención de un mejor rendimiento en un amplio rango de usuarios, se tienen bases de datos en varios lenguajes que contienen un amplio conjunto de muestras del habla tomadas de diferentes personas, estas son necesarias para el preentrenamiento de los modelos. La necesidad de entrenamiento en los sistemas dependientes de la persona impactan en la popularidad de su uso. Los usuarios no desean entrenar el sistema o pueden olvidar las directivas de voz introducidas.

Para hacer al sistema compatible con varios idiomas, dialectos, modismos del hablante, se han realizado grandes esfuerzos en el diseño de algoritmos y en el preentrenamiento en sistemas independientes del usuario basados en modelos ocultos de Markov (HMM). Esto con el fin de obtener un buen rendimiento sobre un amplio rango de variantes del usuario, por medio de bases de datos que contienen un gran conjunto de muestras del habla tomadas de diferentes hablantes en diferentes condiciones, las cuales son necesarias para el preentrenamiento. La combinación de los sistemas de reconocimiento independientes y dependientes del usuario aprovecha las ventajas de ambos sistemas: facilidad de uso debido al diccionario preentrenado y un alto rendimiento debido a los vocablos agregados por el usuario al sistema. Además los sistemas avanzados de reconocimiento de voz independiente del usuario soportan la adaptación de los modelos acústicos sobre la marcha. Los sistemas adaptados por el usuario maximizan la exactitud de los sistemas para el usuario y las variaciones del entorno, mientras que mantienen un bajo nivel de interacciones de usuario. La adaptación del sistema se lleva a cabo durante el uso del sistema, de una forma transparente al usuario.

Los dispositivos móviles se han comercializado ampliamente a nivel mundial y por lo general, son más accesibles por su precio que una computadora personal. Por esta razón se ha incrementado el desarrollo de aplicaciones móviles que permitan al usuario realizar la mayoría de las tareas que realizaban en una computadora personal. Al desarrollar una aplicación para dispositivos móviles se busca la facilidad de operación, debido a las limitantes que tienen los dispositivos, como son su tamaño pequeño, duración de batería y capacidad de procesamiento. Por ello, se busca explorar diferentes formas de interacción con el dispositivo, esto con la finalidad de aprovechar todas sus características. Anteriormente sólo se había limitado el desarrollo de aplicaciones para explotar el uso de la pantalla táctil y por medio del teclado de los dispositivos móviles, pero estas por lo regular son pequeñas, por lo que llega a ser incómodo su uso. Además existen situaciones, en que operar estos dispositivos de forma manual resulta poco práctico, por la naturaleza de estos dispositivos, que se encuentran en constante movimiento.

3.3. Principales técnicas de reconocimiento de voz

Para la realización del procesamiento del reconocimiento de voz, se pueden emplear diferentes técnicas, entre las más utilizadas se muestran a continuación.

3.3.1. Redes neuronales

El estudio de las redes neuronales artificiales fue inspirado por la neurobiología. Una red neuronal artificial consiste en un número potencialmente alto de elementos simples de procesamiento (llamados unidades, nodos o neuronas), los cuales influyen en el comportamiento de unos con otros, a través de una red de pesos excitadores o inhibidores. Cada unidad simple calcula una suma ponderada no lineal de su entrada y emite el resultado sobre sus conexiones de salida hacia otras unidades. Un conjunto de entrenamiento consiste en patrones de valores que se asignan a las unidades de entrada o salida. Como los patrones se presentan en el conjunto de entrenamiento, una regla de aprendizaje modifica los pesos de cada uno de los nodos, así la red aprende gradualmente del conjunto de entrenamiento. Este paradigma puede implementarse en diferentes formas, de modo que diferentes tipos de redes pueden aprender a calcular funciones implícitas de vectores de entrada o de salida, realizar cálculos sobre clusters de datos de entrada, generar representaciones compactas de datos o proporcionar una memoria de contenido direccionable y realizar la finalización de un patrón [31].

Las redes neuronales son usadas usualmente para realizar reconocimiento de patrones estáticos, esto es, para mapear de manera estática entradas complejas a salidas simples, tal como una clasificación N-aria de patrones de entrada. Más allá, la forma más común de entrenar una red neuronal para esta tarea es por medio de un procedimiento llamado *backpropagation*, por lo que los pesos de la red se modifican en proporción a su contri-

bución al error observado en las actividades de unidades de salida (relativo a las salidas observadas). El reconocimiento de voz ha sido otro campo de pruebas para la aplicación de redes neuronales. Los investigadores lograron excelentes resultados en tareas como discriminación de voz, reconocimiento de fonemas y reconocimiento de dígitos hablados [31].

3.3.2. Codificación predictiva lineal

La finalidad del método LPC (*Linear Predictive Coding*) es encontrar un conjunto de vectores representativos denominados en un conjunto de vectores código, que forma una matriz que a su vez conforma lo que se denomina *libro código*. La predicción lineal modela la zona vocal humana como una respuesta al impulso infinito, que produzca la señal de voz. Basa su hipótesis en el hecho de que la muestra $s[n]$ de una señal en el dominio del tiempo puede ser determinada a partir de las n muestras anteriores, si conocemos el peso por el cual cada una de ellas está afectada por todas las muestras anteriores $s[n-1], s[n-2], s[n-M]$ [30].

$$s[n] \approx \hat{s}[n] = -\sum_{i=1}^{M} a_i s[n-i].$$

Donde $s[n]$ es la señal muestreada y a_i, $i = 1, 2, ..., M$ son los predictores o coeficientes LPC. Un pequeño número de coeficientes LPC $a_1, a_2, ..., a_M$ pueden ser usados para representar eficientemente una señal $s[n]$[3, 6]. Para el análisis del reconocimiento de voz, LPC puede ayudar a estimar la recurrencia de la vibración de las cuerdas vocales, la forma del tracto vocal, las frecuencias y ancho de banda de los polos espectrales y las resonancias de los tractos vocales. Pero este principalmente provee un pequeño número de parámetros del discurso (coeficientes LPC) que se encuentran relacionados a la configuración del tracto vocal y al sonido emitido. Estos coeficientes se usan en filtros digitales como valores multiplicadores para generar una versión sintética de la señal original, o que puede ser almacenada como un patrón de reconocimiento [30].

3.3.3. Análisis por Transformada Rápida de Fourier

El desarrollo de métodos numéricos ha permitido un avance significativo en el análisis y síntesis del sonido y particularmente de la voz humana. Existe la posibilidad de adquirir valores numéricos de la onda acústica a intervalos discretos, suficientemente próximos en el tiempo, que permitan analizar y recomponer una señal. Una vez adquiridas estás muestras pueden almacenarse en memoria para posteriormente ser procesadas. La cantidad de cualquier magnitud a lo largo del tiempo se identifica perfectamente por sus componentes frecuenciales. Es decir cualquier fenómeno cuantificado por un determinado parámetro puede interpretarse como una superposición de periodicidades de distintas frecuencias, que se encuentran en el tiempo. Esto da pie al desarrollo de la herramienta matemática denominada Transformada de Fourier [39].

En términos matemáticos esta transformada es un operador (F) que aplicado a una función temporal $g(t)$ la convierte en otra función de la frecuencia $G(f)$ que aporta la misma información que la primera:

$$F[g(t)] = G(f)$$

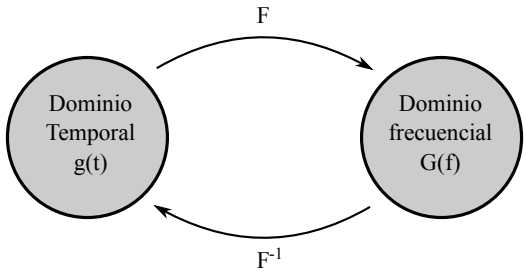


Figura 3.1: Transformada de Fourier.

La función $G(f)$ es la que proporciona la información correspondiente a las periodicidades de la función $g(t)$. Se pude interpretar la transformada de Fourier como un camino para pasar del dominio del tiempo al dominio de la frecuencia. Desde el dominio frecuencial, ciertos fenómenos se explican más fácilmente y tienen un tratamiento matemático más ágil [39] (ver figura 3.1).

Algunas aplicaciones de la transformada de Fourier [40]:

- Filtrado: tomar la transformada de Fourier de una función es equivalente a su representación como la suma de funciones sinusoidales. Al eliminar los componentes indeseables de alta o baja frecuencia (es decir, eliminando algunas funciones seno) y tomando una transformada inversa de Fourier para llevar de vuelta al dominio del tiempo, se puede filtrar una imagen o sonido para eliminar el ruido.

- Compresión de imagenes: para suavizar, el filtrado de imágenes contiene menos información que la original, mientras que se conserva una apariencia similar. Por medio de la eliminación de coeficientes de las funciones seno que contribuyen relativamente poco a la imagen, se puede reducir el tamaño de la imagen a un bajo costo y con fidelidad de la imagen.

- Convolución y deconvolución: la transformada de Fourier puede computar de forma eficiente la convolución de dos secuencias. Una convolución es el producto par de elementos de dos diferentes secuencias, tal como una multiplicación de dos $n-$variables polinomiales f y g o de la comparación de dos caracteres de una cadena.

3.3.4. Modelos ocultos de Markov

El núcleo de los sistemas de reconocimiento de voz consiste en un conjunto de modelos estadísticos que representan los diferentes sonidos de un lenguaje para que este sea reconocido. Dado que el discurso tiene una estructura temporal y puede ser codificado como una secuencia de vectores espectrales que abarcan la gama de frecuencias, los modelos ocultos de Markov (HMM) proveen un marco natural para la construcción de tales modelos [41].

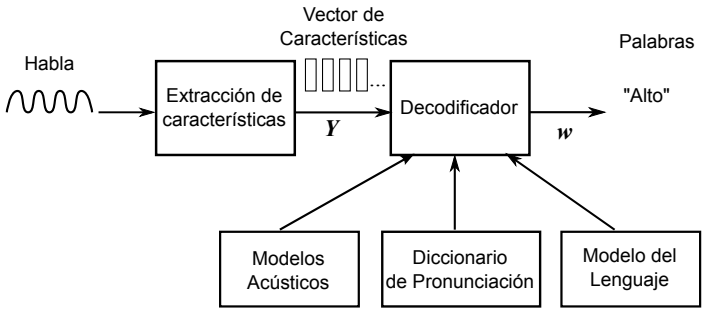


Figura 3.2: Arquitectura de un reconocedor de voz basado en HMM.

Los principales componentes de un sistema de reconocimiento de voz continuo de vocabulario amplio se muestran en la figura 3.2. La forma de onda de la entrada del audio desde un microfono se convierte en una secuencia de vectores acústicos de tamaño fijo $\boldsymbol{Y_{1:T} = y_1, ..., y_T}$ en un proceso llamado extracción de características. A continuación el decodificador intenta encontrar la secuencia de palabras $\boldsymbol{w_{1:L} = w_1, ..., w_L}$ que es más probable que se haya generado $\boldsymbol{Y}$, por ejemplo el decodificador trata de encontrar

$$\hat{\boldsymbol{w}} = \boldsymbol{arg_w max\{P(w|Y)\}.} \tag{3.1}$$

Sin embargo, dado que $\boldsymbol{P(w|Y)}$ es difícil de ser modelado directamente, la regla de Bayes es usada para transformar 3.1 en un problema equivalente para encontrar:

$$\hat{\boldsymbol{w}} = \boldsymbol{arg_w max\{p(Y|w)P(w)\}.} \tag{3.2}$$

La semejanza $\boldsymbol{p(Y|w)}$ es determinada por un modelo acústico y el anterior P(w) es determinado por el modelo del lenguaje. La unidad básica del sonido representada por el modelo acústico es el fonema. Por ejemplo, la palabra "bat" se compone de tres fonemas /b/ /ae/ /t/. Cerca de 40 fonemas son requeridos para el idioma ingles. Para cualquier $\boldsymbol{w}$, el correspondiente modelo acústico es sintetizado por la concatenación de fonemas modelo para hacer palabras como se definen por un diccionario de pronunciación. Los parámetros de estos fonemas modelo son estimados por datos de entrenamiento que constan de formas de onda del habla y sus transcripciones ortográficas. El modelo del lenguaje es típicamente un modelo de N-gramas en el cual la probabilidad de cada palabra es condicionada sólo a su $\boldsymbol{N-1}$ predecesor. Los parámetros N-gramas son estimados por el conteo de $\boldsymbol{N-}$tuplas en el cuerpo del texto correspondiente. El decodificador funciona mediante la búsqueda a través de todas las posibles secuencias de palabras por medio de

la poda para remover hipótesis inverosímiles manteniendo de esta forma una búsqueda manejable. Cuando el fin del enunciado se alcanza, la salida es la secuencia de palabras más probable. Alternativamente, los decodificadores modernos pueden generar matrices que contienen una representación compacta de las hipótesis más probables.

3.4. Arquitecturas de procesamiento de voz

El cómputo móvil impone un cambio y un fuerte reto en el área de Interacción Humano-Computadora (HCI) [1]. Debido a que las aplicaciones y servicios son usados en movimiento, la interacción entre el usuario y el dispositivo móvil y entre el usuario y la aplicación final debe ser lo suficientemente simple, conveniente, intuitiva, flexible y eficiente. Además, la interfaz de usuario tiene que ser diseñada para ahorrar energía lo más posible. Las interfaces de usuario de los dispositivos móviles deben contar con más de un método de ingreso de datos, como son el teclado tipo *touch*, voz y otros tipos de retroalimentación con el usuario, como el uso de LEDs, sonido, vibración del dispositivo o por medio de imágenes y texto. Se han desarrollado interfaces multimodales, que combinan más de un modo de lenguaje corporal como entrada de datos, ya sean gestos corporales, expresiones faciales y movimientos de los ojos, para facilitar la comunicación efectiva entre el usuario y la computadora [1]. Los elementos clave en el contexto de HCI son los siguientes [1]:

- *Diseño de interfaz de usuario en un dispositivo móvil:* los factores intrínsecos a las redes inalámbricas móviles y a los dispositivos móviles afectarán el diseño de la interfaz de usuario para una aplicación móvil. Ejemplos de dichos factores son el tamaño, la profundidad del color, la resolución de la pantalla, la duración de la batería, el ancho de banda de la conexión inalámbrica, la confiabilidad de la conexión, el procesador, la capacidad de almacenamiento, etc.

- *Diseño coherente de la interfaz de usuario a través de múltiples dispositivos móviles:* debido a la falta de estandarización en la capa de presentación de entrada y salida en los dispositivos móviles, una interfaz de usuario tiene que ser diseñada para cada tipo de dispositivo en específico. La disposición de los componentes y la lógica de funcionamiento de la interfaz debe ser coherente para permitir una navegación sencilla y operar de forma intuitiva.

- *Diseño de la interfaz de usuario adaptable:* la aplicación y el sistema operativo del dispositivo móvil deben ser capaces de optimizar dinámicamente la interfaz de usuario del dispositivo móvil, con base en la configuración del hardware del dispositivo móvil y en el contexto de aplicación en ejecución.

Una de las principales formas de interacción es el reconocimiento de voz, debido a que la voz es la forma de interacción natural para HCI. Como se menciona en [42] y [43] se han

[1]Se define como el estudio de la relación (interacción) entre las personas y los sistemas de cómputo y aplicaciones [2].

desarrollado tres principales arquitecturas para el reconocimiento de voz en dispositivos móviles:

1. *Procesamiento de voz integrado:* todo el proceso de reconocimiento automático de voz (RAV), que incluye extracción de características y búsquedas, se realizan de forma local en el dispositivo móvil (ver figura 3.3).

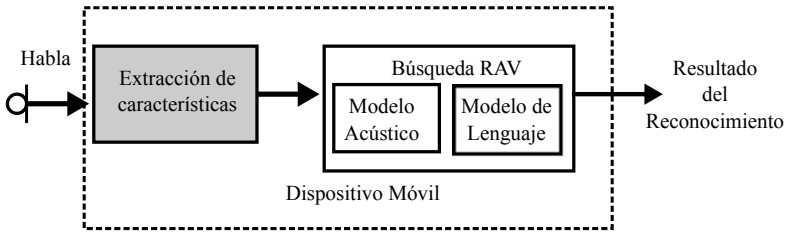


Figura 3.3: Procesamiento de voz en el dispositivo móvil.

2. *Procesamiento de voz en la nube:* la extracción de características y la búsqueda se realizan en un servidor remoto. Los dispositivos móviles se encargan de decodificar la señal antes de trasmitirla por medio de la red al servidor (ver figura 3.4).

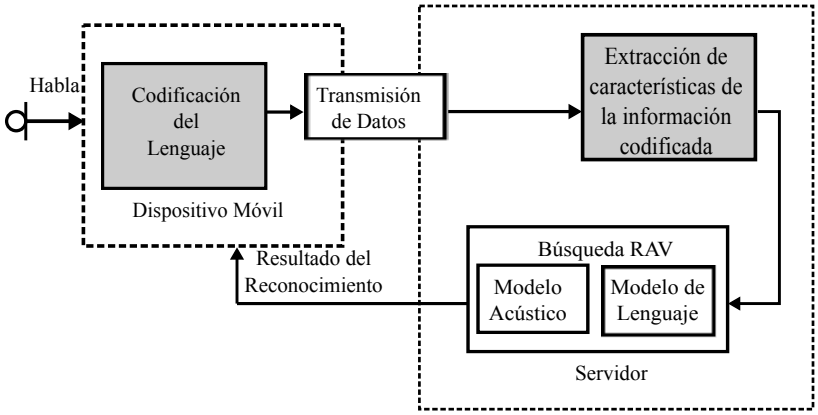


Figura 3.4: Procesamiento de voz en la nube.

3. *Procesamiento de voz distribuido:* representa una arquitectura cliente-servidor, donde el procesamiento del reconocimiento de voz se comparte entre el dispositivo y un servidor remoto. La extracción de características de la voz se realiza en el dispositivo y en el servidor se realiza la búsqueda de patrones (ver figura 3.5).

En [44] se propone una nueva arquitectura de *Procesamiento de voz compartido con adaptación basada en el usuario*. Esta combina las ventajas más importantes de las arquitecturas basadas en servidor, tratando de minimizar las desventajas de estas. La mayoría de los subprocesos de reconocimiento de voz se encuentran en el dispositivo móvil, por lo que el dispositivo puede llevar a cabo el reconocimiento de voz aunque no cuente con

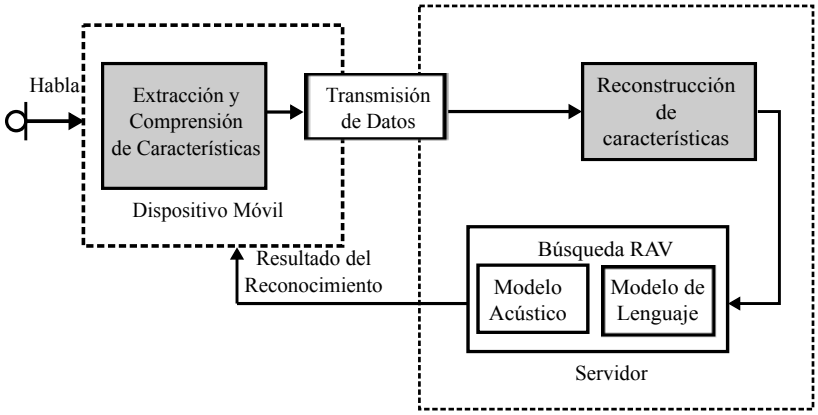


Figura 3.5: Procesamiento de voz distribuido.

conexión a Internet. Sin embargo, cuando cuente con conexión disponible, el dispositivo móvil, enviará al servidor las características de la voz recolectadas en forma de metadatos, con la información del usuario y del dispositivo, como los niveles de ruido, el canal de información y las salidas decodificadas. Esto habilita al servidor para evaluar el desempeño del rendimiento del reconocimiento por cada usuario de forma independiente (ver figura 3.6).

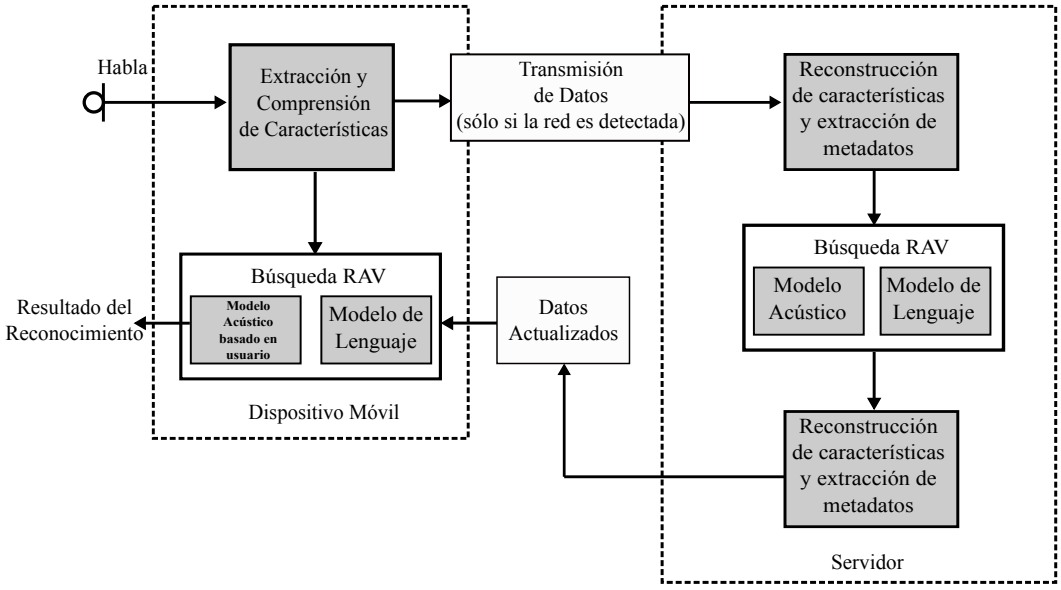


Figura 3.6: Arquitectura del sistema cuando el reconocimiento de voz se realiza en el dispositivo móvil de forma local, cuando este detecta una conexión disponible, las actualizaciones periódicas para la adaptación y mejora del rendimiento del reconocedor en el dispositivo móvil se reciben desde el servidor. Estos cambios se basan en la acústica de cada usuario y en meta-datos.

Capítulo 4

Propuesta de solución

El principal objetivo del presente trabajo de tesis es la construcción de un *framework* de reconocimiento de voz multiplataforma que pueda ser implementado de forma transparente en las principales plataformas de desarrollo, que son iOS y Android, el resultado esperado es una capa de software que permitirá la construcción de aplicaciones de tipo rich-client[1], por lo que el reconocimiento de voz se llevará a cabo usando las capacidades que ofrecen los dispositivos móviles.

El siguiente capítulo se encuentra dividido en las siguientes secciones, en la sección 4.1 se muestra la arquitectura propuesta para el esquema de solución general para la construcción del *framework*, en la sección 4.2 se listan las principales funcionalidades con las que contará el *framework*, la sección 4.3 muestra los componentes principales de la arquitectura del *framework*, en la sección 4.4 se mencionan los requisitos funcionales y no funcionales sobre los que operará el *framework*, en la sección 4.5 se muestran los casos de uso contemplados, en la sección 4.6 se describe el diagrama de clases propuesto para la construcción del *framework*, la sección 4.7 se describe la arquitectura propuesta para el desarrollo y finalmente en la sección 4.8 se describe la API de desarrollo propuesta.

4.1. Solución general

Al analizar las diferentes arquitecturas de desarrollo en el capítulo 2, se muestran las diferentes características de estas para el desarrollo de aplicaciones móviles. Se ha encontrado que uno de los principales problemas para el desarrollo de aplicaciones abiertas multiplataforma, es lograr la compatibilidad entre las plataformas móviles heterogéneas.

La arquitectura basada en *middleware* puede resolver los requerimientos para el desarrollo de aplicaciones móviles a través de la implementación de una aplicación móvil tipo rich-client. La arquitectura *middleware* permite la construcción de una solución transparente multiplataforma a través de una capa que maneje las diferencias entre estas plata-

[1]Se usará el término rich-client ya que no hay tradución literal

formas.

Por lo que se propone en este trabajo la construcción de una aplicación de tipo rich-client, con una interfaz de usuario creada sobre HTML5, con una capa que se encargue del soporte multiplataforma y a su vez proporcione un API de implementación sencilla al usuario, que sea capaz de realizar reconocimiento de palabras aisladas, dependiente de un hablante. En la figura 4.1 se muestra la propuesta de solución de forma general.

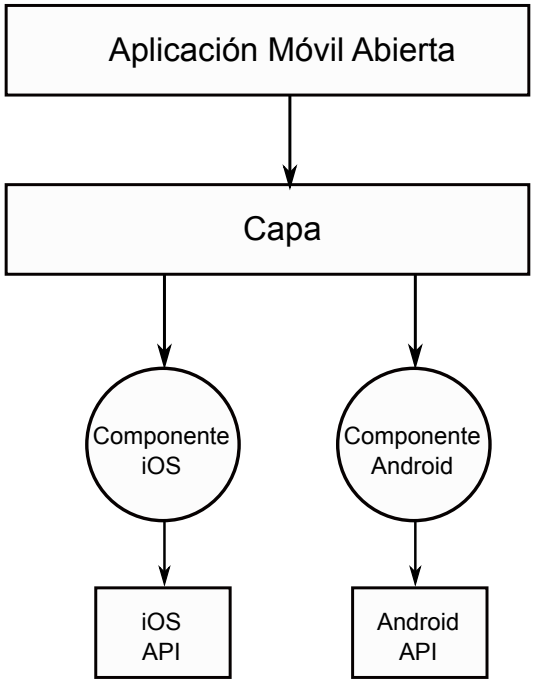


Figura 4.1: Esquema de solución general.

4.2. Funcionalidad del framework

Un *framework* puede reducir considerablemente los costos del desarrollo de una aplicación ya que permite la reutilización de código [5]. El propósito del desarrollo de esta capa de abstracción es ofrecer una solución multiplataforma, para el procesamiento de reconocimiento de voz, que permita hacer uso de las capacidades de los *smartphones*, los cuales cuentan cada vez con mayor poder de cómputo. Entre las principales funcionalidades que proporcionará el desarrollo de este *framework*, se pueden mencionar las siguientes:

- Crear una interfaz de usuario que facilite la comunicación con el dispositivo, ya que la voz es la forma más natural del ser humano para comunicarse.
- Realizar procesamiento de voz para la identificación de palabras definidas en un diccionario.

- Reducir el tiempo de desarrollo en las diferentes plataformas móviles.
- Obtener un alto nivel de abstracción, para que las diferencias en el procesamiento de reconocimiento de voz en cada plataforma se manejen de forma independiente.
- Crear un API reducida y clara, que permita una implementación del *framework* de forma transparente en las diferentes plataformas de desarrollo.

El *framework* será incluido en una aplicación híbrida, la cual incluye código nativo de la plataforma y código en html, en el dispositivo móvil, por medio de una biblioteca de JavaScript, la cual será compatible para las plataformas iOS y Android[2]. Al incluirse en la aplicación llevará a cabo el procesamiento de reconocimiento de voz y mostrará en la aplicación el resultado del procesamiento interpretado como texto.

4.3. Arquitectura general del sistema

Es importante señalar los elementos que intervendrán, como actores principales y las relaciones que se establecen entre ellos, en la implementación realizada se prueba la funcionalidad del presente *framework*, ver figura 4.2. Estos se muestran a continuación:

Figura 4.2: Arquitectura General del Sistema.

- Usuario. Es la persona que se encargará de proporcionar la muestra de audio, en este caso una grabación breve de su voz, esto por medio de una interfaz de tipo híbrida construida por medio de código nativo y html5.
- Dispositivo móvil. En este caso se refiere a un *smartphone* o *tablet*, en el que se hará uso del micrófono, almacenamiento y poder de cómputo ya que en este se realizará todo el procesamiento para el reconocimiento de voz. Mostrará el resultado del procesamiento como un mensaje en la interfaz híbrida de captura de la voz.

[2]No se contempló WindowsPhone para este *framework*.

- Red. En este caso es la red inalámbrica o la red celular que se encuentre disponible. Se utilizará para hacer uso de un servidor remoto en donde se realizará el reconocimiento de voz.

4.4. Especificación de requisitos

A continuación se muestran los requisitos necesarios funcionales y no funcionales sobre los cuales operará el *framework* y su implementación.

4.4.1. Requisitos Funcionales

- Manejo de archivos de sonido para procesamiento de reconocimiento de voz.
- API que permita el uso del *framework* de una forma clara y simple.
- Ejecuciones de forma transparente en iOS y Android.
- El resultado del procesamiento debe mostrarse por medio de un mensaje en la interfaz de usuario.
- El tiempo de respuesta esperado es máximo de 3 a 5 segundos.

4.4.2. Requisitos No funcionales

- Dispositivo móvil con sistema operativo iOS o Android con versión mayor a la 2.2.
- Comunicación con servidor remoto para intercambio de servicios utilizando la API de google.

4.5. Casos de uso

Los casos de uso permiten obtener una visión general de las funciones principales que realizará el *framework*. En este apartado se muestran los casos de uso contemplados en el diseño y construcción del *framework*, como se observa en la figura 4.3.

Se muestra en el cuadro 4.1 la descripción general del caso de uso. A continuación se muestra una descripción detallada de cada caso de uso:

- *Ingresa Voz*, aquí se realiza la captura del habla, el usuario ingresa su voz por medio del micrófono del dispositivo móvil, el cuál se habilita hasta que el usuario termina la grabación. Esta grabación se almacena en el dispositivo como un archivo de sonido. En la plataforma iOS se almacena como un archivo tipo WAV, mientras que en Android dada la configuración y manejo de archivos propia de la plataforma se almacena como un archivo tipo 3gp.

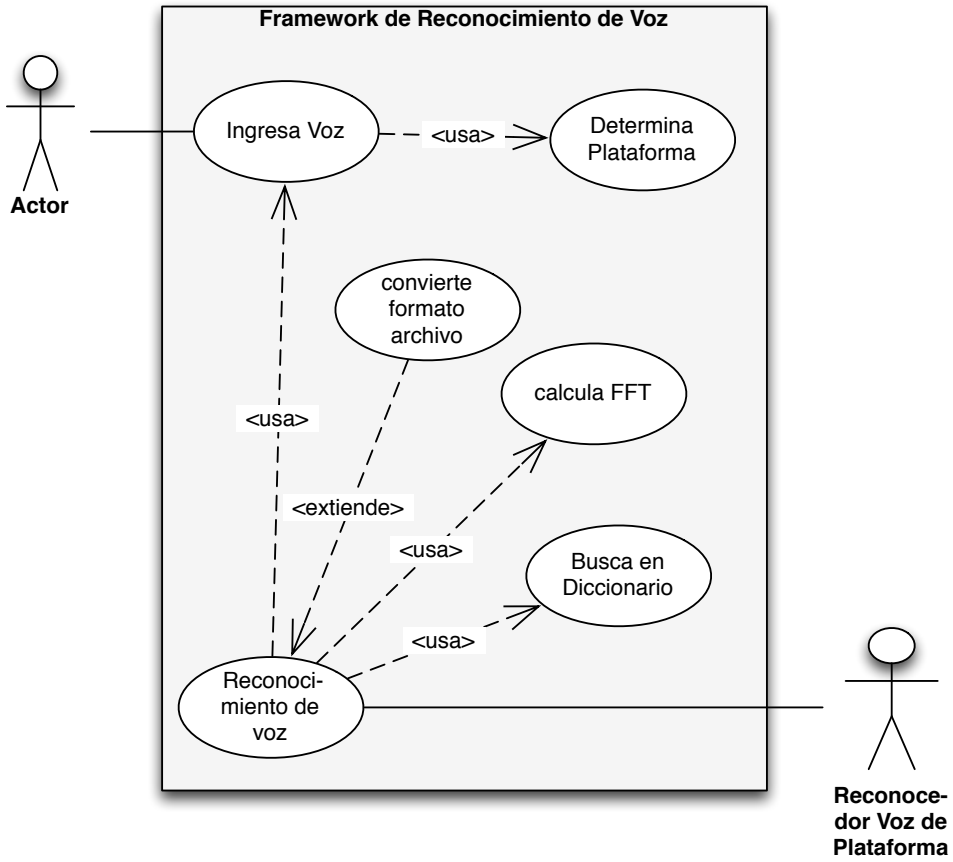


Figura 4.3: Casos de uso para el framework de reconocimiento de voz en dispositivo móvil.

- *Determina Plataforma*, esta acción es determinada por la interfaz de usuario, se encarga de definir bajo que plataforma se encuentra el usuario, este proceso se lleva a cabo una vez que la aplicación ha iniciado su ejecución, esto para determinar el sitio en el cual se guardara el archivo de sonido.
- *Reconocimiento de voz*, una vez grabado el archivo, la API se encarga de ejecutar el procedimiento de reconocimiento de voz específico para cada plataforma. El cual recibe el nombre del archivo de sonido que procesará.
- *Convierte formato de archivo*, esta acción se realiza en caso de que la plataforma de ejecución sea Android, ya que se requiere que los archivos de voz grabados se encuentren en formato WAV.
- *Cálculo de FFT*, se lleva a cabo el cálculo de FFT (Fast Fourier Transform) sobre el vector correspondiente al archivo de sonido y la normalización de estos valores.
- *Busca en diccionario*, se encarga de buscar el resultado obtenido en el diccionario que se ha predefinido para el *framework*.

Nombre	Procesamiento de reconocimiento de voz.
Descripción	Permite realizar el procesamiento del reconocimiento de voz.
Actores	Usuario y Reconocedor de voz desarrollado para cada plataforma.
Flujo Normal	1. El usuario ingresa la voz al sistema. 2. Se determina la plataforma móvil sobre la cual se esta ejecutando la aplicación. 3. El reconocedor de voz de la plataforma realiza el procesamiento de la voz del usuario. 4. Realiza el cálculo de FFT sobre a voz ingresada. 5. Busca el resultado en el diccionario predefinido para el reconocedor. 6. Determina el resultado.
Flujo Alternativo	3.1. Si la plataforma de ejecución es Android, se realiza una conversión de formato del archivo que contiene la voz del usuario.

Cuadro 4.1: Descripción de casos de uso de reconocimiento de voz.

En primer lugar se realiza la captura del habla (*Ingresa Voz*), usuario ingresa la secuencia de voz, esta se graba en el dispositivo móvil como un archivo tipo WAV. Posteriormente se realizará el procesamiento del habla ingresado, se realiza la extracción de características y el procesamiento, una vez finalizado, se envía al usuario el resultado del reconocimiento de voz por medio de un mensaje a la interfaz de usuario, para que pueda ser visualizado. La secuencia de acciones puede observarse en el siguiente diagrama de la figura 4.4.

4.6. Diagrama de clases

El diagrama de clases representa la estructura y el funcionamiento de cada parte que constituye el *framework*, así como las relaciones que se establecen entre cada uno de los elementos que lo conforman. En la presente sección se muestran las clases que se han considerado en la construcción del *framework*, ver figura 4.5.

- *IniApp*: se encarga de cargar la página construida en HTML5 que funcionará como interfaz de usuario. Al ser esta una aplicación híbrida, esta clase se instanciará en cada una de las plataformas en las cuales se implementará el *framework*.
  Atributos:

  - **htmlPage:** nombre de la página realizada en HTML5 que cargará la aplicación.

  Métodos:

  - **starApp:** este método carga la página HTML previamente definida en el dispositivo móvil.

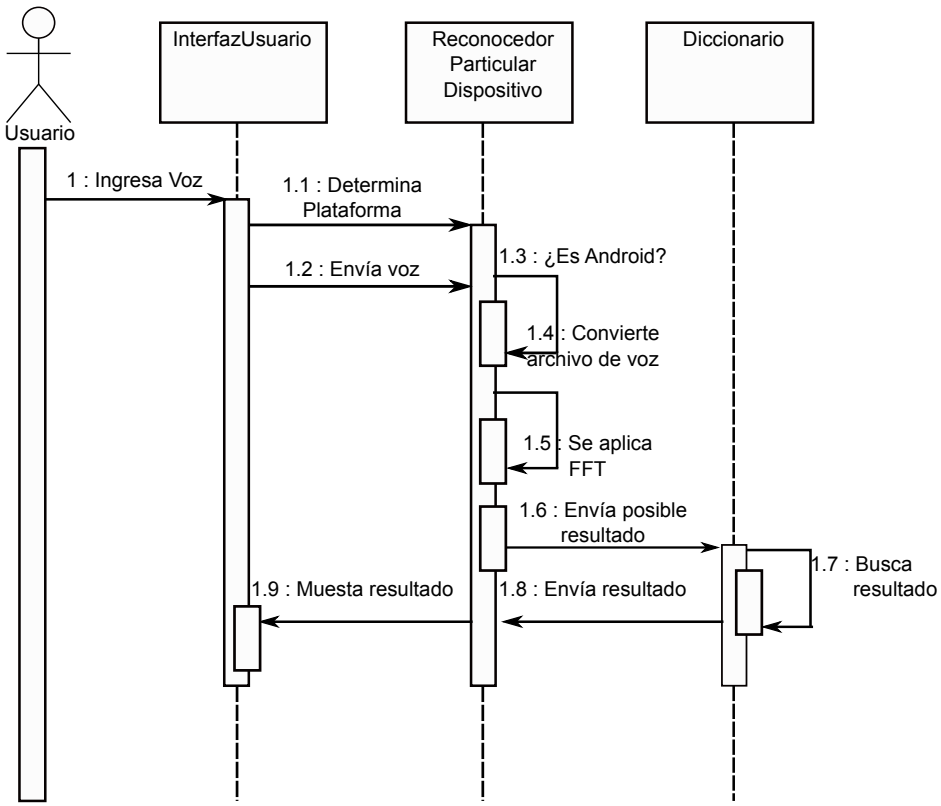


Figura 4.4: Diagrama de secuencia del framework.

- *RecordAudioDevice*: se encarga de la identificación de la plataforma de ejecución de la aplicación, contiene las funciones correspondientes a grabar, parar y reproducir el archivo y los llamados a las funciones de la API que permiten realizar el reconocimiento de voz.
  Atributos:

  - **audioFile:** contiene el identificador del archivo que se almacenará en el dispositivo móvil donde se guarda temporalmente la grabación de la voz del usuario.

  Métodos:

  - **setPlataform:** determina la plataforma de ejecución de la aplicación.
  - **callgetAudio:** manda a llamar la función de la API que se encarga de la grabación de la voz del usuario. Define el lugar de almacenamiento del archivo.
  - **stop:** permite parar la grabación del archivo de voz.
  - **play:** reproduce el audio recién grabado.
  - **callprocessAudio:** manda a llamar la función de la API que se encarga del procesamiento del reconocimiento de voz.

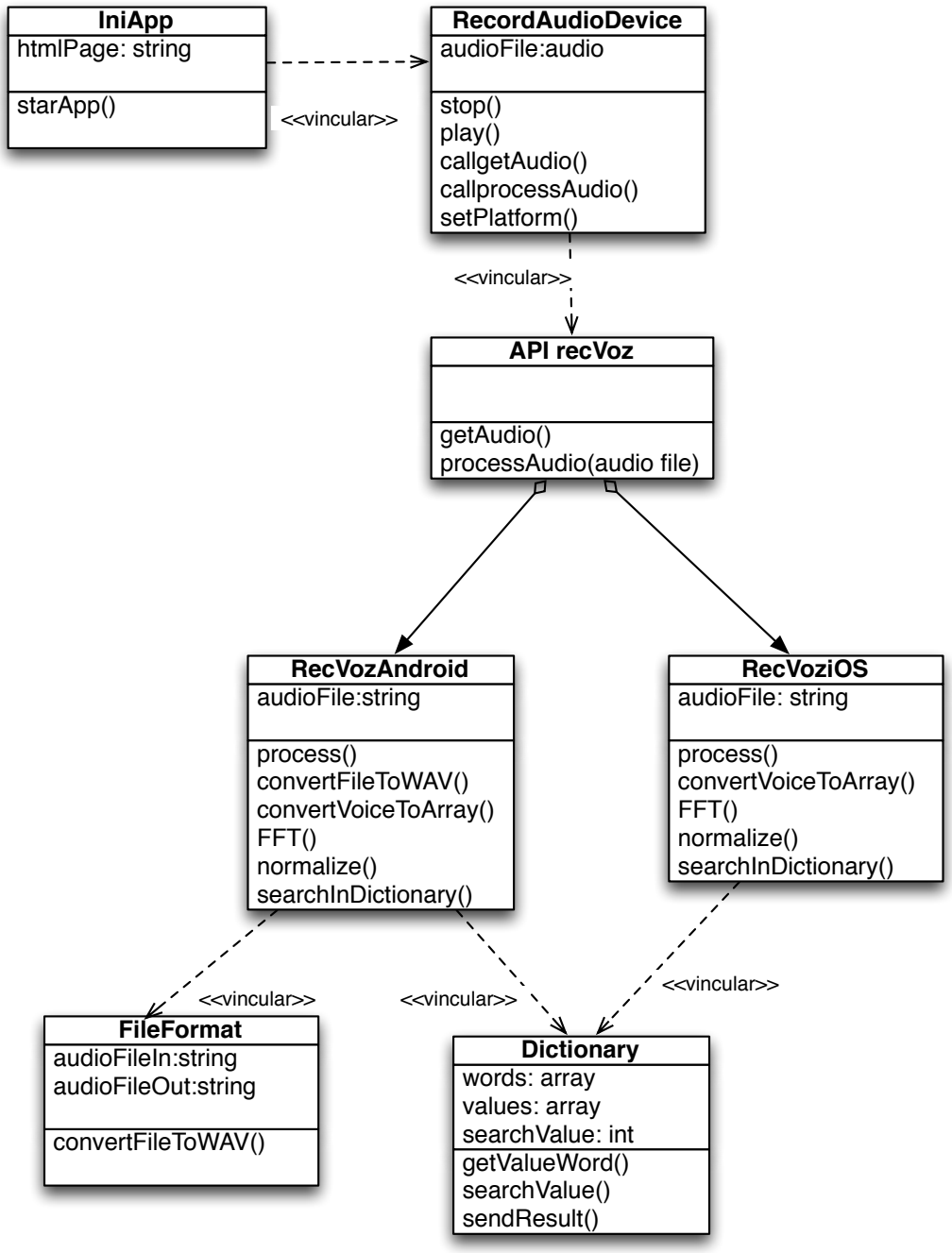


Figura 4.5: Diagrama de clases que integran el framework.

- *API recVoz*: definición de la API que permitirá interactuar con las funciones de obtención de audio y de reconocimiento de voz multiplataforma.
  Métodos:

  - **getAudio:** método que habilita el microfono para la captura del audio y de-

termina donde se almacenará el archivo de audio.

- **processAudio:** método que se encarga de ejecutar el reconocedor de voz correspondiente a la plataforma y de mostrar el mensaje del procesamiento.

- *RecVozAndroid:* lleva a cabo el reconocimiento de voz para Android.
  Atributos:

  - **audioFile:** nombre del archivo que será procesado para el reconocimiento de voz.

  Métodos:

  - **process:** realiza el procesamiento de reconocimiento de voz.
  - **convertFileWAV:** convierte el formato del archivo que será procesado de 3gp a WAV.
  - **convertVoiceToArray:** convierte en un arreglo numérico el archivo de audio.
  - **FFT:** calcula (Fast Fourier Transform) sobre el vector de audio.
  - **normalize:** realiza la normalización de los valores del vector.
  - **searchInDictionary:** muestra si encontró el resultado en el diccionario.

- *RecVoziOS:* lleva a cabo el reconocimiento de voz para iOS.
  Atributos:

  - **audioFile:** nombre del archivo que será procesado para el reconocimiento de voz.

  Métodos:

  - **process:** realiza el procesamiento de reconocimiento de voz.
  - **convertVoiceToArray:** convierte en un arreglo numérico el archivo de audio.
  - **FFT:** calcula (Fast Fourier Transform) sobre el vector de audio.
  - **normalize:** realiza la normalización de los valores del vector.
  - **searchInDictionary:** muestra si encontró el resultado en el diccionario.

- *FileFormat:* convierte el archivo de 3gp a WAV.
  Atributos:

  - **audioFileIn:** nombre del archivo al que se le cambiará el formato.
  - **audioFileOut:** nombre de archivo de salida.

  Métodos:

- **convertFileToWAV:** función que se encarga de convertir el archivo al formato tipo WAV.

- *Dictionary:* contiene el conjunto de palabras que serán reconocidas por el *framework*. **Atributos:**

  - **words:** conjunto de palabras que pueden ser reconocidas por el *framework*.
  - **values:** valores correspondientes a cada palabra.
  - **searchValue:** valor a buscar en el diccionario.

  Métodos:

  - **getValueWord:** obtiene el valor que buscará dentro del diccionario.
  - **searchValue:** realiza la búsqueda del valor en el diccionario y determina la palabra con la que coincide.
  - **sendResult:** envía el resultado al reconocedor.

4.7. Arquitectura de desarrollo

La arquitectura de desarrollo propuesta para la construcción de la aplicación final, se encuentra divida en capas como se observa en la figura 4.6.

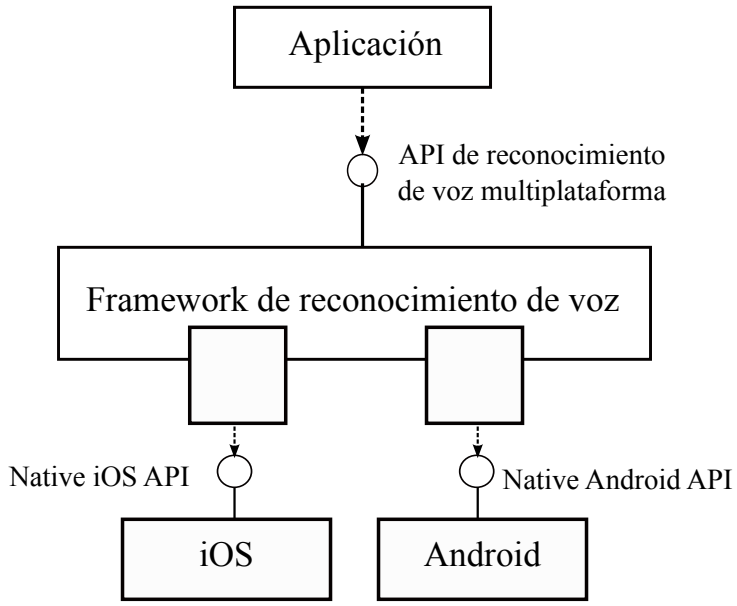


Figura 4.6: Arquitectura del framework de reconocimiento de voz en dispositivo.

En la capa superior o de presentación se encuentra la aplicación construida en HTML5 con código nativo, con una interfaz construida para la captura de la voz. En esta interfaz

el usuario será capaz de dictar al dispositivo palabras, esta muestra será almacenada en el dispositivo de forma temporal en un archivo de tipo WAV, el cual será interpretado por el reconocedor.

La segunda capa se refiere al *framework* abierto, en el que por medio de un API será posible realizar su implementación en iOS y Android. Este *framework* encapsulará las diferencias existentes entre plataformas. Por lo que el usuario podrá hacer uso de la API definida para la realización del reconocimiento de voz de forma transparente.

La tercera capa se refiere al sistema operativo y bibliotecas que son utilizadas para llevar a cabo el proceso de reconocimiento de voz en cada plataforma. Esta parte no será accesible al usuario de manera directa, sólo podrá hacer uso de las funciones por medio de la API, el cual determinará bajo que plataforma se esta trabajando.

4.8. Definición de la API

La API determina los métodos disponibles para los programadores por los cuales pueden interactuar con las funcionalidades del *framework*. Un API puede ser vista como una abstracción sobre la funcionalidad y la implementación interna de cada componente que pertenece a un sistema de software [45], es un conducto que habilita datos desde una aplicación o servicio [46].

Una de las principales razones para pensar en la implementación de un API es que los usuarios finales usan una amplia diversidad de dispositivos conectados, ya sea a redes sociales y a varias formas de mensajería para acceder a información y servicios que ellos necesitan. Un API hace relativamente fácil el escalamiento a un alto número de implementaciones en un corto periodo de tiempo. Un ejemplo común es cuando se requiere construir una aplicación móvil. La primera aplicación es creada de forma rápida, en respuesta a la necesidad y codificada para que pueda ejecutarse en la plataforma móvil más popular, pero cuando surge la necesidad de crear una segunda aplicación, existe el riesgo de duplicar código para la plataforma especificada. A partir de la necesidad de crear aplicaciones móviles que pueden ejecutarse en diferentes plataformas, debe considerarse construcción de un API. Un API puede ayudar a los desarrolladores a dar soporte a más dispositivos con diferentes plataformas.

Un API puede brindar el máximo nivel de flexibilidad proveyendo a los contenidos del cómo y el cuándo, en los términos definidos y con un mejor control, cumpliendo con las necesidades del usuario [47]. Las mejores APIs son aquellas que son cuidadosamente diseñadas, fáciles de aprender, lógicamente estructuradas y consistentes, de esta forma los desarrolladores pueden dilucidar como solventar algunas dudas en su entendimiento e implementación con relativo éxito. Las APIs exitosas comienzan con una mínima funcionalidad, a lo que se añaden más funciones con el paso del tiempo, en base a la re-

troalimentación recibida, es recomendable empezar con el nivel mínimo de funcionalidad de esta forma se permite una evolución más limpia y rápida del producto [45].

Un reto importante en el diseño de APIs es que no hay métricas generales para el desarrollo de APIs de sistemas distribuidos, existen algunas métricas pero sólo en un enfoque teórico [48], lo que dificulta un diseño genérico que se adecue a todas las necesidades, especialmente hablando de sistemas móviles. El principal objetivo del diseño de un API es que debe ser fácil de usar, ampliamente adoptada y productiva. La API es acerca de la comunicación existente entre el programador que escribe la API y el programador que hace la implementación contra esa API. Un aspecto importante de un API es que debe ser consistente, para que puede verificarse fácilmente [47].

El creciente uso de JavaScript en sistemas web, ha permitido el desarrollo de APIs para sistemas móviles, pero es recomendable el uso de bibliotecas compactas ya que existe un alto consumo energético en el dispositivo al hacer uso de estas bibliotecas [49]. Para lograr la implementación del *framework* de reconocimiento de voz, en los diferentes sistemas operativos móviles contemplados (iOS y Android), se ha definido un API genérica desarrollada en JavaScript, que permitirá la manipulación e implementación del *framework*. Las funciones principales se muestran a continuación 4.2:

Función	**Descripción**
getAudio()	Esta función permite habilitar el micrófono del dispositivo móvil para iniciar la captura de voz, asigna el nombre del archivo, el cual se almacenará temporalmente en el dispositivo, para que la voz sea procesada.
stopRecord()	Esta función detiene la captura del archivo de audio. Almacena el audio en la ruta y nombre indicado al iniciar la captura del audio.
string processAudio(file)	Esta función recibe el nombre del archivo a procesar, se encarga de la evaluación y procesamiento del archivo de audio, para la realización del reconocimiento de voz en el dispositivo, de esta forma enviará como resultado una cadena con la sílaba identificada.

Cuadro 4.2: Definición de la API propuesta.

Capítulo 5

Desarrollo

Este capítulo aborda el proceso de construcción del *framework* de reconocimiento de voz multiplataforma. En las siguientes secciones se detalla la construcción de las capas que integran este software. En la sección 5.1 se describe la tecnología utilizada para la construcción del *framework*, se hace un análisis acerca del software utilizado para la construcción de aplicaciones multiplataforma. En la sección 5.2 se describe la capa de presentación, es decir la interfaz de usuario tipo web que se propone como capa multiplataforma, la cual puede ser integrada, para este caso en las plataformas iOS y Android, a una aplicación de tipo híbrida. La sección 5.3 describe la construcción de la capa de procesamiento del reconocimiento de voz y la forma de interacción con la capa de presentación y de la integración de esta capa con las plataformas iOS y Android.

5.1. Descripción de la tecnología utilizada

Como se mencionó en el capítulo 2 de esta tesis, existen diversas tecnologías que permiten el desarrollo de aplicaciones multiplataforma. Para la selección del *framework* a utilizar se analizaron las similitudes generales entre las diferentes plataformas móviles, una de ellas es el poder incorporar navegadores en las aplicaciones. Esto significa el hacer uso de una pantalla de la aplicación móvil como un browser que muestre una página HTML. Estos navegadores incrustados en la aplicación son llamados comúnmente como *WebViews*. Esto permite definir una de las pantallas de la aplicación como *WebViews*. Técnicamente, en este caso las plataformas usadas (iOS y Android), tienen soporte para poder ejecutar módulos nativos de JavaScript en el *webview*. Esto es, programáticamente, se puede lograr que todas las plataformas permiten hacer llamados a código nativo en Java, C y Objective C y viceversa [50].

Se puede escribir código que expongan módulos nativos a través del *webview* de la plataforma. Un ejemplo es crear código JavaScript que se ejecute en el *WebViews* e invoque el código nativo para obtener las coordenadas del GPS, este es el principio fundamental de funcionamiento del *framework* PhoneGap [50], el cual es utilizado en el presente trabajo de tesis.

5.1.1. Uso de HTML5 y CSS3

La web móvil es la plataforma más fácil de aprender, barata, estandarizada, disponible y de fácil distribución [51]. HTML5 y CSS3 están emergiendo como las principales tecnologías web. Estos elementos hacen las aplicaciones web más interactivas y ricas en características. HTML5 no sólo ha adicionado nuevos marcadores para un soporte multimedia más robusto, sino también ha agregado características como web workers para procesamiento en segundo plano, suporte de base de datos y soporte fuera de conexión.

CSS3 es el nuevo estándar para una interfaz de usuario transparente y más rica. Con el apoyo de CSS3 un sitio puede competir contra los sitios basados en flash. Un sitio se puede transformar fácilmente en un sitio móvil con un cambio de un archivo CSS. Los navegadores móviles son los primeros en adoptar los estándares del W3C. Esto significa que los dispositivos móviles son la plataforma adecuada para las aplicaciones HTML5/CSS3 [50].

5.1.2. Arquitectura de PhoneGap

PhoneGap es un *framework* para aplicaciones sobre HMTL5, se ha propuesto por Apache Software Foundation (ASF) bajo el nombre de Apache Cordova para desarrollar aplicaciones rich-client a través de tecnologías web. Esto significa que los desarrolladores puedan crear aplicaciones móviles con sus conocimientos sobre HTML, JavaScript y CSS.

Las aplicaciones que se desarrollan usando PhoneGap son aplicaciones híbridas. Estas aplicaciones no son creadas totalmente sobre HTML/JavaScript, pero tampoco son totalmente nativas. Parte de la aplicación, principalmente la interfaz de usuario, la lógica y la comunicación con el servidor esta basado sobre HTML/JavaScript. Otra parte de la aplicación que comunica y controla el dispositivo esta basado en el lenguaje nativo de la plataforma. PhoneGap provee un puente desde el JavaScript hacia la plataforma nativa, el cual permite que la API de JavaScript tenga acceso y control del dispositivo móvil (*smartphone* o *tablet*) [50].

PhoneGap proporciona la API de JavaScript que permite tener acceso a las capacidades del dispositivo como cámara, GPS, información del dispositivo, etc. El *framework* de PhoneGap es esencialmente una biblioteca de JavaScript que permite que las aplicaciones de HTML/JavaScript tengan acceso a las capacidades del dispositivo. El *framework* de PhoneGap también posee un componente nativo, el cual trabaja de forma independiente y hace el trabajo en el dispositivo. La figura 5.1 muestra la arquitectura del *framework*. Una aplicación construida usando PhoneGap tiene dos partes principales [50]:

1. La parte de lógica de negocios, desarrollada en JavaScript, la cual se encarga de la conexión entre la interfaz de usuario y la funcionalidad que por general se refleja en los datos de la aplicación.

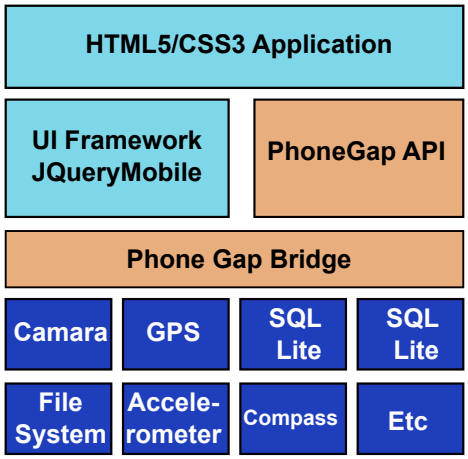


Figura 5.1: Arquitectura de aplicación creada con PhoneGap.

2. La parte de JavaScript que tiene acceso y tiene control del dispositivo (teléfono o tableta).

En cualquier tecnología, es común la práctica de reutilizar una característica que ya se encuentre disponible y ya se ha probado o bien incluir una nueva funcionalidad a un sistema [50]. Como se mencionó PhoneGap permite acceder a las diferentes capacidades del dispositivo como son cámara, GPS, acelerómetro, entre otras. Sin embargo, PhoneGap no tiene todas las APIs nativas para su uso en aplicaciones de JavaScript. Se requiere hacer algo más especializado, se puede utilizar el modelo de *plug-in* nativo de PhoneGap para ampliar las capacidades del núcleo de la API de PhoneGap [52].

Los plug-in's nativos en PhoneGap no son como los plugins de un navegador web, estos proveen una forma de conectarse al código nativo que ha sido agregado para que una aplicación que usa el *framework* PhoneGap pueda utilizarlo. Los plugins nativos de PhoneGap permiten crear una nueva funcionalidad personalizada en código nativo, con esto se puede usar este código por medio del puente nativo con el que cuenta PhoneGap para JavaScript. Esto significa que se puede exponer cualquier biblioteca nativa o *framework* para el uso de aplicaciones creadas con PhoneGap basadas en JavaScript [52].

Un plug-in de PhoneGap es una extensión de las funciones de PhoneGap. Este accede a una parte de la funcionalidad en el dispositivo. La funcionalidad tipo plugin sólo es capaz de acceder a las funciones nativas del dispositivo o bien pueden proporcionar la funcionalidad para acceder a servicios en la nube. Un plug-in de PhoneGap consiste de al menos dos archivos [50]:

- Archivo JavaScript.
- Archivo de funcionalidad en el lenguaje nativo.

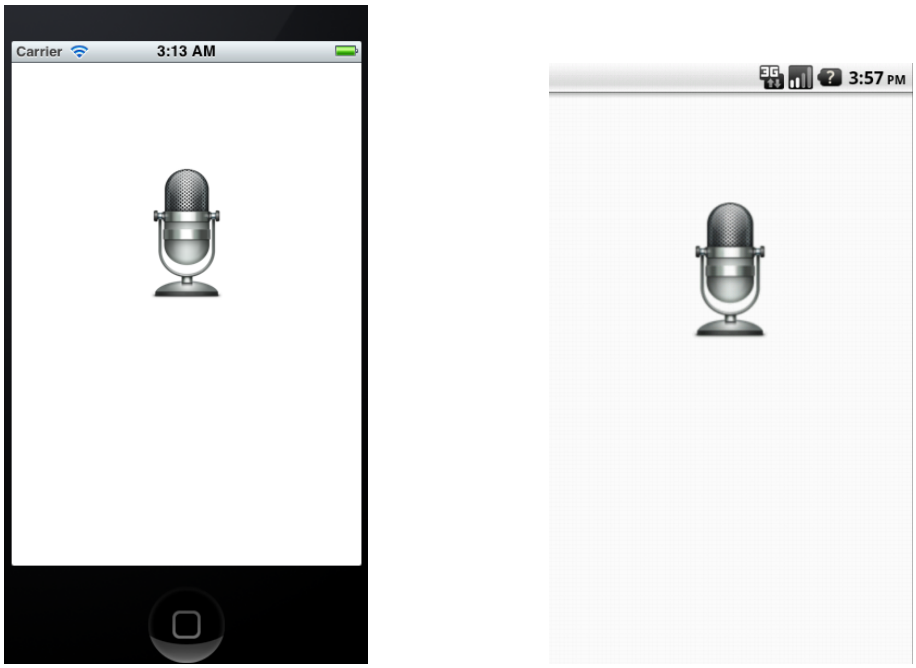
El archivo de JavaScript del plug-in es la interfaz entre la aplicación de PhoneGap y el plug-in de PhoneGap. Las funcionalidades de los plug-in's sólo son accesibles por medio

de funciones de JavaScript contenidas en el archivo de JavaScript.

El archivo nativo se usa por el *framework* de PhoneGap para interactuar con el dispositivo para acceder a las funciones nativas. Como usuario de un plug-in, se necesita agregar proyecto nativo en la estructura del proyecto [50].

5.2. Capa de presentación

La creación de una biblioteca de Javascript facilita el uso e implementación de la API a los programadores. Por medio del uso de HTML5 se puede crear una página web,que incorpore la biblioteca en JavaScript y que sirva de interfaz de usuario para la captura de voz y para la visualización del resultado del procesamiento. El uso de una interfaz web, permite que el procedimiento de la captura de la voz y la visualización del resultado, se realicen de la misma forma para el usuario en diversas plataformas (en particular iOS y Android), ver figura 5.2.



(a) Interfaz de usuario para aplicación iOS.

(b) Interfaz de usuario para aplicación Android.

Figura 5.2: Interfaz de usuario en HTML5

5.2.1. Captura de la voz

Como se estableció en la sección 4.8 la biblioteca de JavaScript, uno de las métodos que la integran es `getAudio()`, implicitamente este método reconoce la plataforma en la que se esta ejecutando el usuario y asigna el componente para la captura de la voz que se usará en la interfaz web ver figura 5.3. PhoneGap ofrece un plug-in que permite realizar la captura de la voz de forma multiplaforma. Para la plataforma iOS se utiliza el componente *Media* que ofrece PhoneGap, este permite habilitar el micrófono de dispositivo móvil para capturar y grabar sonido. Este componente en iOS permite guardar los archivos de sonido en formato WAV[1], el cual será necesario para su manipulación en el procesamiento de reconocimiento de la voz. En Android no se utilizó la API *Media* dado que el formato de

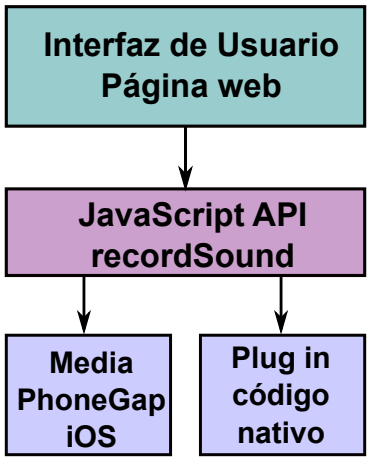


Figura 5.3: Función recordSound define el plug in a utilizarse para grabar sonido.

grabación lo asigna como 3gp, que es el que por defecto utiliza la API de PhoneGap para Android. Per ello se utilizó un método nativo que general el archivo WAV. La API *Media* permite grabar y reproducir audio en el dispositivo. Se pueden reproducir archivos de audio cargados localmente en el dispositivo, o reproducir archivos de audio recuperados de Internet [53].

El segundo método que interviene en la captura del audio, es el método de la API `stopRecord()`, este método permite detener la grabación del archivo de sonido y almacena el archivo de sonido en la ruta definida para cada plataforma. Para le ejecución de este método ya ha identificado la plataforma en la cual se esta ejecutando la aplicación, internamente el método `stopRecord()` define si se ejecuta el método `stopRecord()` que corresponde al API *Media* de PhoneGap, en el caso de iOS, o bien si se ejecuta el Plug-in nativo `stop` que se ha definido para Android.

[1]Waveform Audio Format, formato introducido por Microsoft e IBM en 1991. Este formato permite digitalizar el sonido de forma fiel a la original ya que es un formato sin pérdida, no compromete la calidad del audio.

5.2.2. Uso de API Media de PhoneGap en iOS

En esta sección se muestra la implementación de la API propuesta, usando el objeto media de PhoneGap. En el ejemplo en particular se muestra un botón con una imagen correspondiente a un micrófono, este permite interactuar con el plug-in, para el caso de iOS se inicializa el objeto *Media*. En la inicializacion del objeto, ver ejemplo 5.1, se define la ruta general en la cual se almacenaran los archivos de audio. El objeto se asigna a una variable de JavaScript, para poder utilizar sus métodos, como `startRecord()` o `stop()`. El método `startRecord()` es el que inicia la grabación del sonido y el método `stop()` detiene la grabación y almacena el archivo en la ruta definida en la inicialización. Los archivos de audio grabados se guardan en formato WAV, en este formato de audio los datos se guardan sin compresión de datos, a 44100 Hz y 16 bits.

Listado 5.1: Inicialización de objeto media

```
var media = new Media(src, mediaSuccess, [mediaError], [mediaStatus]);
```

Para inicializar el objeto Media se requieren los siguientes parámetros:
`src`: Ruta donde se almacenará el archivo de audio.
`mediaSuccess`: función a llamar después de que el objeto media ha finalizado la reproducción/grabación o parado la grabación.
`mediaError`: función a llamar si hay un error.
`mediaStatus`: función a llamar para indicar los cambios de estado.

5.2.3. Uso de Plug-in de PhoneGap con código nativo en Android

La implementación de la API *Media* de PhoneGap en Android, se lleva a cabo de forma similar que en iOS. Sin embargo varía la ruta donde se asigna los archivos de audio. El uso del objeto *Media* en Android presentó algunas variantes de comportamiento, una de estas, es que el objeto *Media* no permite almacenar el audio como WAV, como se analizó para el diseño de la API. La importancia del archivo de formato WAV, se debe a que no se encuentra comprimido, por lo que la perdida de información es casi nula. Esto se contrapone a los requerimientos definidos para la API por tal razón lo que se optó por el desarrollo de una clase nativa, que permita grabar el audio como formato WAV desde Android, las funciones de esta clase se mandan a llamar desde una función de JavaScript por medio de un Plug-in de PhoneGap. Para que un código nativo puede ser accedido por medio de un Plug-in en JavaScript este debe heredar de la clase `Plugin` que ofrece el *framework* PhoneGap para Android, ver figura 5.4.

La clase nativa que hemos creado par de grabación de la voz se llama `RecordSound`. En esta clase se debe implementar el método `execute`, el cuál es heredado de la clase `Plugin` de PhoneGap. Este método cuenta con los siguientes argumentos [50]:

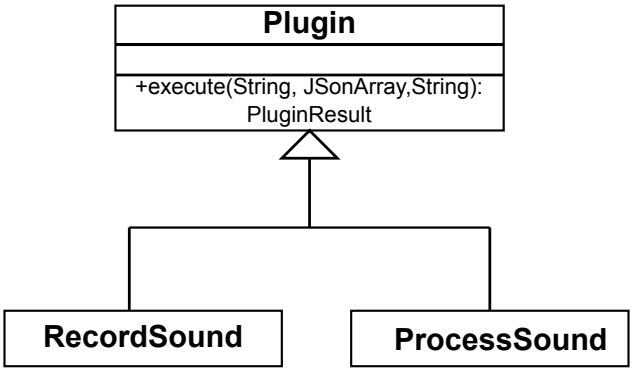


Figura 5.4: Diagrama de clases de herencia de objeto Plugin de PhoneGap.

1. `Action:` Es el nombre de la acción a ejecutarse. Por ejemplo, en este caso se envían los valores "record" y "stop", para iniciar o parar la captura del sonido.

2. `Data:` Es el conjunto de datos pasados desde JavaScript al plug-in. Este el conjunto de datos pasados desde la aplicación web con JavaScript hacia el código nativo. Por ejemplo, el nombre de un archivo.

3. `CallbackId:` Este es usado cuando se devuelve el llamado a la función JavaScript.

Listado 5.2: función execute heredada de la clase Plugin.

```
public PluginResult execute(String action, JSONArray data, String callbackId){
}
```

El tipo de retorno de este método es `PluginResult`. El PluginResult toma un estado de una enumeración y otro argumento que representa la causa o más información acerca de la operación. Por ejemplo:

Listado 5.3: función execute heredada de la clase Plugin.

```
new PluginResult(Status.OK);
```

En este caso, el método `execute` puede recibir las acciones `record` y `stop`.

1. `record`: se usa para crear una instancia de `AudioRecord`. La clase `AudioRecord` instancia por record administra los recursos de audio para las aplicaciones de Java para grabar audio desde el hardware de entrada de audio de la plataforma.

Listado 5.4: Inicialización de objeto AudioRecord.

```
recorder= new AudioRecord(MediaRecorder.AudioSource.MIC,
                          44100,
                          AudioFormat.CHANNEL_IN_MONO,
                          AudioFormat.ENCODING_PCM_16BIT,
                          buffer.length);
```

Para la inicialización del método se asigna la fuente de sonido para grabar el audio, en este caso se utiliza el micrófono por lo que se asignó la propiedad `MediaRecorder.AudioSource.MIC`, la frecuencia de muestreo en Hertz (44100,22050 o 11025), la configuración del audio (mono o stereo), el formato de codificación del audio y la longitud del buffer en bytes [54]. Una vez que se ha creado el objeto, este inicia su búffer de audio asociado que será llenado con los nuevos datos de audio. El tamaño de este buffer se especifica durante la construcción del objeto y determina que la duración que un `AudioRecord` puede grabar antes de que los datos que se encuentran en camino no hayan sido leídos todavía. Los datos deben ser leídos desde el hardware de audio en partes de tamaño inferior que el tamaño total de la memoria intermedia [55]. La acción `record` también ejecuta el método `recorder.startRecording()` el cuál inicia la captura del audio y comienza la escritura del audio capturado en un archivo temporal.

2. `stop()`: se encarga de terminar la grabación del audio por el medio del método `recorder.stop()` el cuál termina la captura del audio y finaliza la copia al archivo temporal, este archivo sirve para obtener la información del audio y realizar la conversión a un archivo de tipo WAV, el cuál será utilizado para realizar el procesamiento de reconocimiento de voz. Finalmente la acción regresa mediante el plug-in, un mensaje en un objeto tipo `JSONObject`, el nombre del archivo que ha sido grabado hacia la aplicación de HTML.

5.3. Capa de procesamiento

La construcción de está capa incluye una fusión de diversos elementos tecnológicos. El plug-in de PhoneGap, la biblioteca El elemento tecnológico principal es el plug-in de PhoneGap. Este plugin que permitiera la asignación del archivo de sonido, capturado en la interfaz web, para que sea procesado por algún módulo del dispositivo o por un servicio remoto. El método de la API process audio, interactúa con el plug-in por lo que define quien es el encargado de realizar el procesamiento de audio.

Por ello se creó una función de JavaScript dentro de API llamada **processAudio** que recibe el nombre del archivo a procesar. Este plug-in manda a llamar al procedimiento de reconocimiento de acuerdo a la plataforma del dispositivo móvil. El segundo elemento opcional es la inclusión de una biblioteca de lenguaje C llamada **Libsndfile**, en particular se probó con libsfile, pero existen diferentes bibliotecas de audio, en este caso el llamado del plug-in, dependera si el procesamiento es realizado por el dispositivo móvil, que permite leer y escribir archivos que contienen sonido (como archivos de MS Windows WAV y el formato Apple/SGI AIFF), a través de un API bien definida. El código fuente de esta biblioteca se encuentra bajo la licencia pública general de GNU. Esta biblioteca esta diseñada para manipular los datos de tipo little-endian (archivos WAV) y los big-endian como los (AIFF)[56]. La API de la biblioteca se compone de funciones que permiten convertir los archivos de sonido WAV en arreglos de tipo numérico para que puedan ser

manipuladas por el tercer elemento, que es el método nativo para cada plataforma, en el cuál se aplican los algoritmos necesarios para realizar el procesamiento de voz y obtener un resultado como tipo string, de acuerdo a un diccionario definido de palabras.

5.3.1. Java Native Interface

Para lograr la integración de la biblioteca **Libsndfile** en lenguaje C en Android se hizo uso de Java Native Interface (JNI), el cual es una parte del estándar de Java que permite a los desarrolladores escribir métodos nativos en otros lenguajes, tales como C y C++ y llamar estos métodos desde código en Java. Esto es muy útil cuando se requiere usar características especificas de la plataforma o aprovechar el hardware de la plataforma, por ejemplo para obtener mayor velocidad en computación numérica, aprovechando el conjunto de instrucciones, o por medio del uso de la API de OpenGL para el manejo de gráficos. Los tipos de operaciones que son mejores candidatas para el uso de NDK son las de uso intensivo de CPU, que no requieren asignar demasiada memoria, como el procesamiento de señales, simulaciones físicas, etc.

Android usa JNI para acceder a implementaciones de código en C y C++ y el NDK (Android Native Development Kit) de Android es una extensión del SDK de Android que soporta el uso de código de C y C++ en aplicaciones de Android. JNI tiene ciertas convenciones que permiten la llamada a los métodos de otros lenguajes. Los cambios en el lado de los métodos nativos (principalmente en las bibliotecas de C o C++) son mayores que los cambios requeridos en el código de Java. Cuando la máquina virtual (VM) de Dalvik, invoca a una función implementada en C o en C++ pasa dos parámetros especiales [54]:

- Un apuntador a el objeto **JNIEnv**, el cual es una clase de manejador para el hilo en la máquina virtual que llama al método nativo.
- Un apuntedor a el objeto **jobject**, el cual hace referencia a la clase que llama al método nativo.

Estos parámetros son pasados de forma transparente por el código de Java. Esto es, que no aparecen en la firma del método declarado en el llamado del código de Java.

Cuando un método nativo es llamado, este se ejecuta en el mismo proceso y en el mismo hilo como el código de Java lo manda a llamar. Este puede asignar memoria desde la pila de Java para aprovechar el *garbage collector*, o fuera de la pila de Java para eludir la administración de memoria de Java. Las variables del código de C o C++ ubicadas en el stack tienen la misma semántica como en los ejecutables nativos en estos lenguajes. Estas están ubicadas desde el espacio del stack para el proceso en el que se están ejecutando [54].

Antes de que los métodos nativos puedan ser utilizados dentro de la clase de Java, la biblioteca que contiene los métodos nativos por medio del llamado **System.loadLibrary**. Los métodos nativos que pueden accederse por una clase son declarados usando la palabra clave **native**.

El *Android Native Development Kit (NDK)* es herramienta adicional del SDK de Android, para compilar el código nativo. Si se usa el NDK para crear código nativo, la aplicación se encuentra empaquetada dentro de un archivo tipo *.apk* que se ejecuta dentro de la máquina virtual del dispositivo. La estructura fundamental de la aplicación Android no cambia [54].

Para la incorporación de la biblioteca **Libsndfile** en Android se debe seguir la siguiente configuración:

1. Crear un directorio llamado *jni* dentro del proyecto.
2. El código nativo, los archivos .c y .h, se agregan a este directorio.
3. Se crea un archivo llamado *Android.mk* que es el archivo de configuración para que el las funciones del código nativo puedan utilizarse en Android.
4. Se ejecuta el comando **ndk/ndk-buid** dentro de la carpeta *jni*, para que compile los archivos en código nativo. Este paso genera un archivo de tipo *.so*.

El archivo *Android.mk* describe la fuente para la construcción del sistema. En realidad es un fragmento de un archivo tipo *make* que se ejecuta por el **GNU build system** cuando se construye la aplicación. En este caso el archivo *Android.mk* posee la siguiente estructura:

```
# Define la ruta local y regresa el directorio del proyecto
LOCAL_PATH:=$(call my-dir)
# Limpia las variables... hace un clean and build include $(CLEAR_VARS)
# Identifica la biblioteca de lenguaje C para manipulación de WAV.
LOCAL_MODULE:= sndfile
LOCAL_SRC_FILES:= $(GSM610_FILES) $(G72x_FILES) $(COMMON_FILES) $(SPECIFIC_FILES)
include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)
# Se define la biblioteca de métodos de procesamiento de voz.
LOCAL_MODULE := LeeWav
LOCAL_CFLAGS := -Werror
LOCAL_SRC_FILES := LeeWav.c
LOCAL_SHARED_LIBRARIES := sndfile
LOCAL_C_INCLUDES := $(LOCAL_PATH)/libsndfile/src
LOCAL_LDLIBS := -llog
include $(BUILD_SHARED_LIBRARY)
```

En este archivo se incluye la biblioteca **sndfile**, que permite la manipulación de archivos tipo WAV y la biblioteca **LeeWav** en donde se encuentran los métodos necesarios para el procesamiento del reconocimiento de voz. Una vez que se ha agregado el archivo *Android.mk* y los archivos nativos, se ejecuta el comando **ndk/ndk-buid** en el directorio

contenedor del proyecto para poder construir las bibliotecas. Si la construcción es exitosa, las bibliotecas compartidas serán copiadas dentro del directorio raíz de la aplicación y se añaden a la construcción del proyecto.

Una vez compiladas las bibliotecas pueden integrarse a una clase de java por medio de una la instrucción `System.loadLibrary()`.

Listado 5.5: función execute heredada de la clase Plugin.

```
static {
        System.loadLibrary ("sndfile");
        System.loadLibrary ("LeeWav");
}
```

5.3.2. Análisis de señal de voz por medio de Transformada Rápida de Fourier

La transformada discreta de Fourier toma como entrada $\boldsymbol{n}$ números complejos $\boldsymbol{h_k}$, $\boldsymbol{0 \leq k \leq n-1}$ correspondientes a los puntos equidistantes de una serie de tiempo y las salidas son $\boldsymbol{n}$ números complejos $\boldsymbol{H_k}$, $\boldsymbol{0 \leq k \leq n-1}$, cada uno describiendo una función sinusoidal de frecuencia dada. La transformada discreta de Fourier puede definirse por [40]:

$$\boldsymbol{H_m = \sum_{k=0}^{n-1} h_k e^{-2\pi ikm/n}}$$

y la transformada inversa de Fourier se define por:

$$\boldsymbol{h_m = \sum_{k=0}^{n-1} h_k e^{2\pi ikm/n}}$$

La salida de la transformada discreta de Fourier consiste de $\boldsymbol{n}$ números, cada uno es calculado usando una formula de $\boldsymbol{n}$ números, por lo que el tiempo computacional del cálculo es de $\boldsymbol{O(n^2)}$. La transformada rápida de Fourier *FFT* es un algoritmo que calcula la transformada discreta de Fourier en tiempo $\boldsymbol{O(n\ log\ n)}$. Es uno de los algoritmos conocidos más importantes, ya que impulsó el procesamiento moderno de señales. Este algoritmo opera por la descomposición en $\boldsymbol{N}$ puntos de una señal en dominio del tiempo en señales de dominio de tiempo $\boldsymbol{N}$ cada una compuesta por un único punto. El segundo paso es el cálculo de $\boldsymbol{N}$ espectros de frecuencia correspondientes a estas señales de dominio de tiempo $\boldsymbol{N}$. Por último, los $\boldsymbol{N}$ espectros se sintetizan en un espectro de frecuencia única [57]. El algoritmo *FFT* asume que $\boldsymbol{n}$ es potencia de 2, si no es el caso, se llena el vector con ceros hasta obtener una longitud de $\boldsymbol{n = 2^k}$, ver el algoritmo 1[28].

La descomposición del dominio del tiempo se lleva a cabo generalmente por un algoritmo de ordenación inversa de bits, ver algoritmo 2. Esto implica cambiar el orden de las $\boldsymbol{N}$ muestras de dominio de tiempo contando en binario con los bits reordenados de

Algorithm 1 Fast Fourier Transform Iterativo

```
1: BIT-REVERSE-COPY(a,A)
2: n ← length[a] ▷ n is a power of 2.
3: for s ← 1 to lg n
4:     do m ← 2^s
5:       ω_m ← e^(2π/m)
6:       for k ← o to n-1 by m
7:           do ω ← 1
8:             for j ← to m/2-1
9:               do t ← ωA[k+j+m/2]
10:                 u ← A[k+j]
11:                A[k+j] ← u+t
12:                A[k+j+m/2] ← u-t
13:                ω ← ωω_m
```

Algorithm 2 BIT-REVERSE-COPY(a,A)

```
n ← length[a]
for k ← to n-1
  do A[rev(k)] ← a_k
```

izquierda a derecha [57].

En la notación compleja, los dominios de el tiempo y la frecuencia contienen cada uno una señal compuesta de $\boldsymbol{n}$ puntos complejos. Cada uno de estos puntos complejos se compone de dos números, la parte real y la parte imaginaria. Un ejemplo de ello, es cuando se toma la muestra $X[24]$, esta se compone por la combinación del número real $Re[42]$ y el número imaginario $ImX[42]$. Cuando dos variables complejas se multiplican, los cuatro componentes individuales deben ser combinados para formar los dos componentes del producto [57].

En la implementación del algoritmo de *FFT* para la capa de reconocimiento de voz, se calculó el número de puntos a evaluar $\boldsymbol{n}$ en base a la frecuencia de muestreo *sample rate*[2],que en este caso es de 44100Hz, este valor se multiplico este valor por 2 y de este resultado se calculó el valor más próximo a potencia de 2.

Para la ejecución de la *FFT*, se debe descomponer el archivo de audio grabado en un arreglo de tipo `double`. Este proceso se realiza por medio de la función `f_read_double` incluida en la biblioteca *LibSndFile*. En este caso, los valores contenidos en el arreglo puede observarse en la figura 5.5.

Una vez que obtienen los datos en arreglo de tipo `double`, se aplica el procedimiento correspondiente a la Fast Fourier Transform. Al resultado obtenido por medio de *FFT*,

[2]Número de muestras por unidad de tiempo tomadas de una señal continua, para hacerlas una señal discreta.

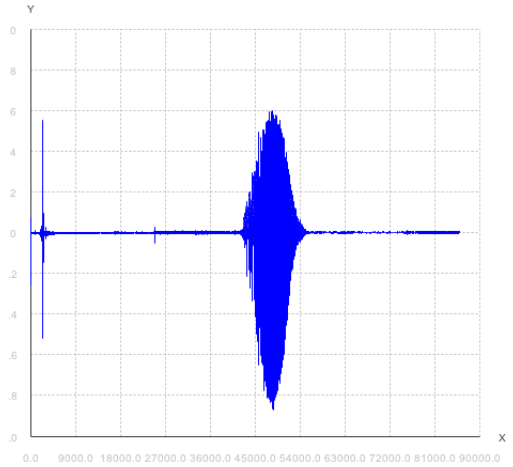


Figura 5.5: Contenido de archivo de audio.

se calcula la magnitud correspondiente al arreglo, es decir el valor absoluto del vector obtenido por medio de la transformada de Fourier. Esto con la finalidad de obtener la magnitud total de cada palabra ingresada.

$$\boldsymbol{Magnitud} = \sqrt{(\boldsymbol{real}(\boldsymbol{X})^2 + \boldsymbol{imag}(\boldsymbol{X})^2)}$$

Al obtener la magnitud, se observa el siguiente resultado de la figura 5.6. Este valor se analizó para determinar su correspondencia con un diccionario con rangos de frecuencia asociados.

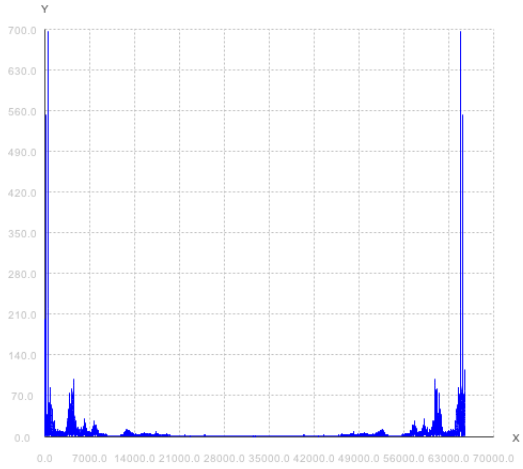


Figura 5.6: Magnitud calculada por palabra.

En este caso para realizar el procesamiento de voz se ha definido un pequeño diccionario correspondiente a las vocales *a, e* y las palabras bisílabas *cafe, hola y adios*.

Para poder determinar un rango asociado para cada palabra, se analizaron aproximadamente 30 muestras por cada palabra. De cada muestra se calculó la magnitud y finalmente se obtuvo un promedio de la magnitud de cada palabra.

En la siguiente tabla 5.1, se muestra parte de las magnitudes calculadas para la determinación de los rangos, que serán asignados por cada palabra, con la finalidad de construir un diccionario para su identificación.

Cuadro 5.1: Magnitudes obtenidas en análisis de archivos de audio capturados.

Palabra	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	Prom.
Letra "a"	41.69	43.65	43.91	89.57	43.24	43.096	42.39	41.88	44.00	47.32	48.07
Letra "e"	375.49	944.55	583.03	40.82	119.13	42.12	168.14	187.34	100.60	656.77	321.80
Hola	1354.71	1043.99	1531.56	1712.33	1130.71	1045.78	889.52	1487.22	1296.03	1174.92	1266.68
Adios	412.67	404.44	91.94	338.21	492.37	475.20	379.74	381.23	425.94	443.74	384.55
café	264.55	275.61	236.17	247.25	267.89	313.44	399.92	222.78	229.23	218.57	267.54
Abrir	739.48	838.84	784.56	1042.09	851.58	699.60	1115.21	209.38	486.19	527.78	729.47

Una vez obtenido el promedio de las magnitudes, se crea un diccionario, el cual tiene asociado a cada palabra un rango de magnitudes, con las cuales una palabra puede ser identificada, ver tabla 5.2.

Cuadro 5.2: Diccionario de palabras con magnitudes asociadas.

Palabra	Rango
Letra "a"	35-47
Letra "e"	240-295
Hola	1200-1400
Adios	350-570
café	180-235
Abrir	700-1000

Al calcular la magnitud de cada palabra, se obtiene la frecuencia de magnitud más alta, está se compara con el rango definido en el diccionario y se obtiene la palabra ingresada.

Capítulo 6

Pruebas Realizadas

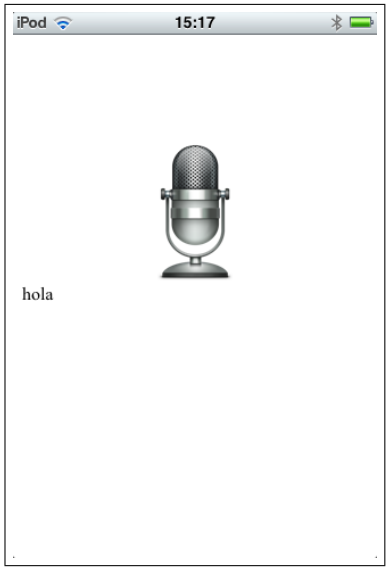
En esta sección se muestran las pruebas realizadas con las implementaciones de ejemplo que se hicieron sobre el *framework*. La primera sección muestra la usabilidad de la API. Se implementó en las diferentes plataformas de prueba (iOS y Android) y se válido el tiempo de respuesta (sección 6.1). La segunda sección (sección 6.2) muestra la implementación del reconocimiento de voz para Android, en este caso ejecutando el reconocimiento a través de JNI con la biblioteca en C y ejecutando el reconocimiento desde Java. En la (sección 6.3) se comparó el rendimiento de la API con respecto al API de Google para reconocimiento de voz.

6.1. Implementación de la API

Para probar la usuabilidad de la API de reconocimiento de voz, se crearon aplicaciones híbridas para las plataformas de prueba (iOS y Android). En este caso, se usó la versión del *framework* PhoneGap 2.1, para poder realizar el vínculo de la aplicación nativa con la interfaz creada en HTML5. La implementación de la API se llevó acabo por medio de la creación de una página web llamada "recordTest", en la cuál se implementan los archivos de JavaScript que conforman la API:

- *Plugin.js:* en este archivo se definen las variables globales que son utilizadas para definir los plug-ins de PhoneGap con las aplicaciones nativas.
- *CrossPlatformSR:* este es el archivo que cuenta con los métodos *getAudio*, *stopRecod()* y *processAudio(fileName)* en cuales se utilizan los plug-ins que son creados por medio de PhoneGap.

En este caso la integración de la API a las plataformas móviles de estudio, se llevó a cabo de forma transparente, la interfaz de usuario es una misma interfaz web y el procesamiento se lleva a cabo por medio de una capa de programación en lenguaje c, la cual puede ejecutarse en ambas plataformas. En las figuras figuras 6.1 se muestra el resultado del reconocimiento de voz al ingresar la palabra "Hola" por medio del micrófono del dispositivo móvil.



(a) Resultado obtenido en aplicación iOS.

(b) Resultado obtenido en aplicación Android.

Figura 6.1: Resultado de procesamiento de voz.

Las aplicación para iOS se ejecutó en un ipod touch segunda generación con sistema operativo iOS 4.2.1, con un procesador CPU de tipo ARM 11 a 532 Mhz. La aplicación para Android se ejecutó en un *smartphone* Xperia J con un sistema operativo Android 4.0.4 Ice Cream Sandwich, con un procesador Procesador Qualcomm MSM7227A a 1GHz y en otro *smartphone* Samsung Galaxy ACE con sistema operativo Android 2.3 Gingerbread.

Las pruebas realizadas se realizaron por medio de la captura de la voz, esto es el botón, representado por un micrófono, que se observa al centro de la pantalla, habilita la captura de la voz, en este caso debido a que el procesamiento es dependiente del usuario, las pruebas fueron realizadas por un mismo usuario. Una vez que empieza la captura, aparecerá el botón de stop, este se muestra en la figura 6.2, al presionar este botón se termina la captura del audio y comienza el procesamiento del audio capturado, el cual es interpretado por el reconocedor desarrollado para la plataforma y regresa la palabra asociada al audio como un mensaje de texto hacia la página web.

Las diferencias notorias en cuanto a ambas aplicaciones se encuentran relacionadas al rendimiento en cada una de las plataformas. Las pruebas realizadas corresponden al tiempo que toma la aplicación en llevar a cabo el procesamiento de voz, en base al número de muestras tomadas para la ejecución de la FFT, ver 6.1. Esta diferencia en el tiempo de ejecución se debe al overhead existente al hacer el llamado al código nativo en lenguaje

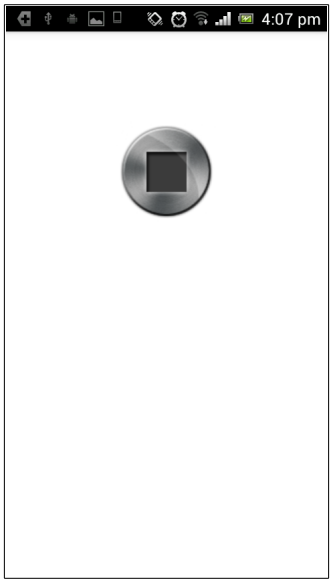


Figura 6.2: Interfaz de usuario web función stop.

C en la plataforma Java. Ya que en iOS no existe este overhead al hacer el llamado de forma directa a las funciones de la biblioteca estática de lenguaje C, el procesamiento es más rápido.

Cuadro 6.1: Tiempo de ejecución de procesamiento de FFT con 65536 muestras.

Plataforma	Tiempo de ejecución
iOS	menor a 1 seg.
Android	3 seg.

Otra de las diferencias en cuanto al funcionamiento de las aplicaciones, es que en la plataforma iOS si se logra obtener la grabación por medio del plug-in nativo de PhoneGap (Media) y este permite grabar el archivo como tipo wav. En cambio en la plataforma Android, el plug-in Media de PhoneGap graba el archivo como tipo 3gp [58], el cuál es la extensión por defecto en la que Android maneja los archivos de sonido, por lo que se tuvo que hacer uso de un plug-in nativo, que se encargará de dar el formato wav al archivo de audio ingresado por el usuario.

6.2. Ejecución JNI comparada con función en java

En este apartado se muestran las pruebas realizadas sobres dos aplicaciones Android, a simple vista son iguales, pero el procesamiento de voz se lleva a cabo de diferente forma, ver figura 6.3. En la primera aplicación se lleva a cabo por medio de una clase en lenguaje C, que es llamada por medio de JNI, en la segunda, se realiza por medio de una clase en Java para Android, ambas aplicaciones son vinculadas por medio de un plug-in de

PhoneGap. En esta sección se muestran las diferencias encontradas en cuanto a tiempo

(a) Resultado de reconocimiento de voz ejecutado en Java.

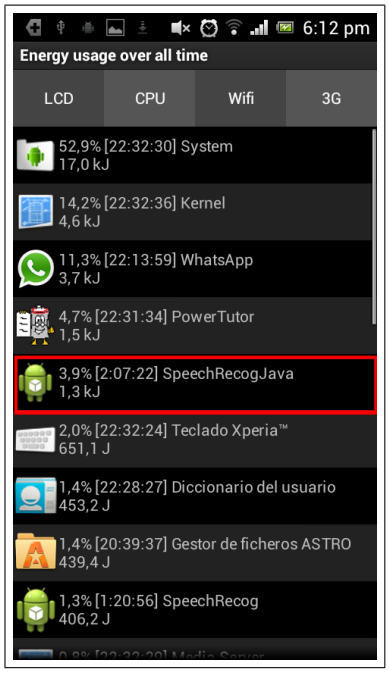
(b) Resultado de reconocimiento de voz ejecutado por JNI.

Figura 6.3: Implementación de la API en Java con diferencia en procesamiento de voz.

de ejecución de la aplicación y consumo energético, esto sirve de pauta para determinar la mejor opción al diseñar y codificar una aplicación. Para determinar el consumo energético de estas aplicaciones híbridas se uso la aplicación PowerTutor [1].

Un punto a favor del diseño y creación de un API reducida es que es ligera y no se conforma de demasiados archivos tipo JavaScript, lo que conlleva a un ahorro en el consumo energético, se ha determinado que JavaScript es el componente de una página web que más consume energía del dispositivo móvil [49]. En el análisis realizado a ambas aplicaciones el tiempo de respuesta fue similar en ambas aplicaciones, pero se presentó mejor rendimiento en el reconocimiento desarrollado en Java. El consumo energético si varia de forma considerable, ver figuras 6.4. Se observa que la aplicación llamada *SpeechRecogJava*, la cual tiene los métodos correspondientes el procesamiento de voz en la plataforma Java, consume aproximadamente 1.3 kJoules, lo cual representa un alto consumo de energía comparado con la aplicación *SpeechRecog*, la cuál realiza el procesamiento por medio de la biblioteca en lenguaje con JNI, la cual consume aproximadamente 406.3 Joules.

[1]PowerTutor es una aplicación para *smartphones* Android que muestra el consumo energético por los principales componentes del sistema como el CPU, red, pantalla, GPS y las diferentes aplicaciones que se encuentren en ejecución. La aplicación permite, a los desarrolladores de aplicaciones móviles, observar el impacto de los cambios de diseño de la aplicación en la duración de la batería. Este proyecto fue



(a) Resultado de consumo de energía con procesamiento de voz por medio de funciones en Java.

(b) Resultado de consumo de energía con procesamiento de voz por medio de funciones en C con JNI.

Figura 6.4: Consumo de energía de aplicación con diferencia en procesamiento de voz.

6.3. Procesamiento de voz en servidor remoto

Una de las limitaciones importantes de los dispositivos móviles son la batería, la capacidad de almacenamiento y en algunos casos para los dispositivos de gama media es el rendimiento del procesador. Estás restricciones pueden ser mejoradas pues el *cómputo offloading*: el envío de procesos de cómputo pesado hacia poderosos servidores y el recibir los resultados de estos procesamientos desde estos servidores. Cada vez se crea software más complejo, el cual se ejecuta en los dispositivos móviles, por ejemplo procesamiento de video y reconocimiento de voz [60].

Offloading es una solución para aumentar las capacidades de los sistemas móviles, por medio de la migración hacia equipos con mayores recursos (servidores). Esto es diferente de la arquitectura tradicional cliente-servidor, donde un *thin client* siempre migra el cómputo hacia un servidor. La principal diferencia es que el cómputo *Offloading* migra los programas a servidores fuera del entorno informático inmediato a los usuarios, la migración de procesos se envían a un *grid* y se produce de una computadora a otra dentro de un mismo entorno informático [60]. El *cómputo offloading* permite ahorrar energía y

desarrollado por alumnos del Doctorado de la Universidad de Michigan [59].

mejorar el rendimiento en los sistemas móviles. Sin embargo, esto depende de distintos parámetros como el ancho de banda y las tasas de datos que se intercambian a través de la red. *Offloading* requiere el acceso a servidores por un pequeño periodo de tiempo a través de redes, alámbricas e inalámbricas. Estos servidores pueden usar virtualización para proveer los servicios remotos, de esta formas diferentes programas y sus datos pueden aislarse y protegerse [60].

La compañía Google ha definido *Web Speech API Specification*, la cual contiene un API en JavaScript que permite incorporar el reconocimiento y síntesis de voz en páginas web. Permite a los desarrolladores utilizar secuencias de comandos para generar la salida de texto a voz y reconocimiento de voz para utilizar como entrada para dictado continuo y control. La API de javaScript perite que las páginas web puedan controlar la activación y sincronización del reconocimiento de voz, para mejorar los resultados [61]. La arquitectura del *Web Speech* se basa en procesamiento de voz en la nube, la API permite a los usuarios grabar el audio desde el micrófono, el audio se envía a través de una solicitud POST HTTPS al servicio web de reconocimiento de voz. El resultado puede procesarse dentro de una aplicación que se ejecute en el navegador Chrome [62].

La implementación del *Web Speech API* se realizó sólo para la plataforma Android, inicialmente se trató de realizar por medio de una aplicación híbrida, en una página creada en HTML5, la cuál se cargará por medio de un *WebView*, el inconveniente encontrado fue la compatibilidad existente con el *WebView* de Android ya que actualmente la API sólo es compatible con el navegador Chrome 25 o superior [61] y aun no se habilitado el soporte de la API para otros navegadores como Safari, Firefox e Internet Explorer.

Se optó por llevar a cabo la implementación de API por medio del llamado al servicio web de forma directa, el audio grabado se envió de forma directa al servicio web, de esta forma los resultados son procesados sin las restricciones del *sandbox* del browser. Por lo que se creo una clase que abre una conexión HTTPS y sube los archivos de audio a través de una petición POST y finalmente recupera el resultado del reconocimiento por medio de plug-in de PhoneGap.

Las ventajas que ofrece el *Web Speech API* es que permite la realización del reconocimiento de voz automático continuo, es decir reconoce frases completas, el tiempo disponible de captura es de 1 minuto, es decir se reconocerá el total de palabras que se puedan ingresar en ese periodo de tiempo [61], ver figura 6.5.

En este caso las limitantes encontradas al implementar el *Web Speech API* son:

- La API aún no es compatible con diferentes browsers por lo que no fue posible integrarlo a una página web que se cargara mediante el *webview* de Android.
- La incompatibilidad de formatos de grabación de audio, la API de *Web Speech* sólo

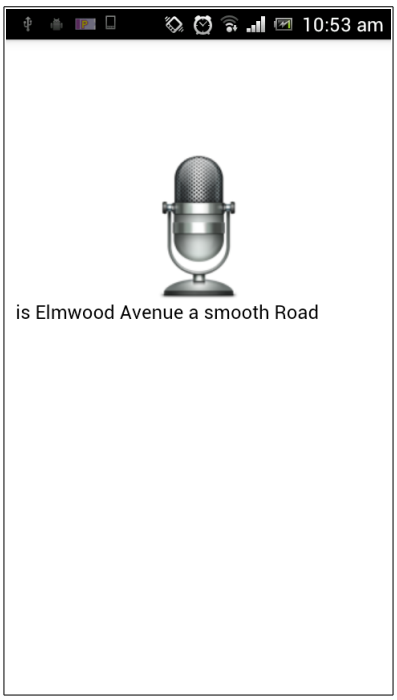


Figura 6.5: Uso de Web Speech API en aplicación Android.

acepta archivos de audio de tipo FLAC [2]. En Android de forma nativa no existe un encodificador de este tipo de formatos de audio, por lo que es necesario ejecutar un método que realice la conversión a este tipo de formato.

- El resultado del procesamiento dependerá de la calidad del servicio de la red.
- El consumo energético de la aplicación depende de la calidad del servicio de la red.

Para este caso se hicieron dos tipos de prueba, primero se ejecutó la aplicación llamada *SpeechRecog*, en la cual se implementa *Web Speech API* con el dispositivo conectado a la red inalámbrica, en este caso se obtuvo un consumo energético de 2.6 Joules, por medio de la aplicación PowerTutor, ver figura 6.6. El tiempo de respuesta depende de la calidad del servicio de la red inalámbrica como tiempo menor se determino aproximadamente de 2 seg y como máximo de 4 seg, pero hubo solicitudes de reconocimiento de voz sin respuesta. El segundo caso de prueba, se realizó por medio de la ejecución de la aplicación *SpeechRecog* utilizando la conexión 3G del dispositivo móvil, de igual forma que en la conexión inalámbrica, el tiempo de respuesta depende de la calidad del servicio de la red 3G. Al hacer uso de esta red, se detectó por medio de la aplicación PowerTutor que el consumo energético se elevó considerablemente a 30.1 Joules, ver figura 6.7.

[2]Free Lossless Audio Codec

Figura 6.6: Consumo energético de aplicación obtenido por medio de conexión inalámbrica.

Figura 6.7: Consumo energético de aplicación obtenido por medio de conexión 3G.

Capítulo 7

Resultados y Conclusiones

La necesidad de solventar el desarrollo multiplataforma ha impulsado el desarrollo de diversos *frameworks* como son PhoneGap [25], el cual permite la habilitación de los sensores del dispositivo para diversas aplicaciones existentes. Se ha desarrollado el *Web Speech API* de Google [61] que permite habilitar el uso del y micrófono del dispositivo sobre HTML5 y realizar el procesamiento de voz a través de un servidor remoto. Esta API se ha desarrollado por Google Inc., pero dado a que se encuentra en desarrollo sólo es soportada por sólo por el navegador Chrome versión 25 y superior.

Para solventar las diferencias existentes entre plataformas se hizo uso del *framework* PhoneGap, el cuál permite crear un puente entre las aplicaciones de HTML5 y el código nativo por medio de JavaScript. En este caso el desarrollo de las aplicaciones nativas se hizo con uso de código nativo de la plataforma y por medio del uso de bibliotecas estáticas en lenguaje C.

7.1. Resultados

Como resultado se ha obtenido un API compacta y flexible en JavaScript que permite su implementación de forma transparente en las plataformas iOS y Android. La implementación de esta API se debe llevar a cabo por medio de una página web construida en HTML5, la cual se mostrará en el *webview* correspondiente a la plataforma de ejecución. Esto permite una interfaz homogénea, que actúa de igual forma en ambas plataformas.

Se ha obtenido una capa de procesamiento,construida en lenguaje C, que contiene una biblioteca de funciones, las cuales pueden adaptarse a ambas plataformas, esta capa es la encargada de realizar el reconocimiento de voz en cada plataforma móvil. Para la construcción de la capa de reconocimiento de voz se hizo uso de una biblioteca en lenguaje C llamada *libsndfile*, la cual permite manipular archivos de audio. La integración de la biblioteca al código nativo se llevó a cabo en Android por medio de JNI y en iOS. Esta integración se hizo de forma directa en iOS, esto se ve reflejado en el tiempo de ejecución de la aplicación, ya que por medio de JNI, se debe crear una clase que vincule los llamados

a funciones en lenguaje C con Java, mientras que iOS se hace uso directo de las funciones que posee la API de *libsndfile*.

Se realizaron pruebas que permitan obtener el consumo energético de la aplicación, especificamente en la plataforma Android, donde se comparó el rendimiento de la aplicación realizando el procesamiento de voz en el dispositivo, por medio de la implementación de la biblioteca construida en lenguaje C y por medio del procesamiento del reconocimiento construido directamente en la plataforma de Java para Android.

Se comparó el rendimiento de la API propuesta con respecto al *Web Speech API* de Google. En este caso se intentó llamar al *Web Speech API* desde a misma página web desarrollada para la aplicación de prueba, pero no fue posible, ya que el *Web Speech API* cuenta con una compatibilidad limitada con otros navegadores. En base a las pruebas realizadas, se logró determinar el consumo energético promedio de la aplicación usando *Web Speech API*, usando la red por medio de Wifi y la red 3G.

7.2. Conclusiones

El principal problema en el desarrollo de aplicaciones móviles abiertas multiplataforma es resolver la compatibilidad entre aplicaciones móviles heterogéneas. Es común que para la construcción de una aplicación móvil se codifique de forma específica para las diferentes plataformas, el problema que los desarrolladores de software para dispositivos móviles tienen que enfrentar es resolver la portabilidad de la aplicación. En el análisis realizado en [51] se determina que la mejor solución para construir una aplicación abierta es sobre una arquitectura tipo *middleware*, implementando una aplicación cliente tipo rich-client.

En este caso de estudio, la aplicación rich client trabaja bajo un ambiente híbrido donde el elemento multiplataforma principal es la capa de interfaz de usuario, esta es una página web construida sobre HTML5 que implementa la API desarrollada. La aplicación nativa maneja las diferencias de los componentes de interfaz y selecciona el reconocedor de voz para cada plataforma, el cual comparte la arquitectura principal ya que ambos utilizan una biblioteca de funciones programadas en lenguaje C.
En la solución propuesta la comunicación entre el componente web y el reconocedor de voz de la plataforma se lleva a cabo por medio del *framework* PhoneGap, por medio de un componente llamado Plug-in. Esto elemento es implementado a través de la API de JavaScript, el cuál hace el llamado a los elementos nativos en lenguaje c y realiza el reconocimiento de voz. De esta forma las diferencias de los sistemas operativos móviles se encapsulan al usuario.

Una de las dificultades encontradas es la incompatibilidad de formatos de audio, en este caso la API se trabajo sobre el formato de audio tipo wav, por que es el que ofrece

mejor calidad de sonido, pero en Android no se puede grabar de forma directa sobre este formato, por lo que se tuvo que realizar un procedimiento adicional que permitiera hacer la conversión a dicho archivo de audio.

El *Web Speech API* definida por Google es una muy buena opción para realizar procesamiento de voz en la nube y cómputo *Offloading*, de esta forma se cuenta con un procesamiento de voz más robusto, ya que realiza procesamiento de voz continuo independiente del usuario, el inconveniente de esta API es que sólo se puede implementar de manera directa sobre páginas HTML5 que se ejecuten sobre el navegador Chrome versión 25 en adelante, por lo que no hay soporte aún para otros navegadores, como Safari, FireFox e Internet Explorer, menos aún para los navegadores nativos de las plataformas móviles o conocidos como *webviews*.

Si se desea implementar la API en una aplicación móvil, como en este caso, puede realizarse por medio de un llamado al servicio web, por medio de un método POST dentro de un método nativo. El otro inconveniente encontrado en esta API, es el manejo de formato de audio, esta API sólo acepta archivos tipo FLAC y no cuenta con sobrecarga de tipos de audio, por lo que se tiene que realizar un proceso de conversión de tipo de formato, antes de que sea enviado al servidor, lo que conlleva a un mayor consumo energético en el dispositivo y mayor tiempo de respuesta. En cuanto al tiempo de respuesta del *Web Speech API*, depende de la calidad de servicio de la red a la que se encuentre conectado el dispositivo, se encontro mejor rendimiento conectado a una red tipo Wifi, que conectado a servicio 3G, ya que si la señal tiene poca intensidad puede que la respuesta tarde hasta 4 seg. o bien no se reciba ninguna. El consumo energético detectado es mayor cuando se encuentra conectado a la red 3G.

El beneficio de desarrollar una API reducida, se ve reflejado en el consumo energético de la aplicación, ya que los archivos JavaScript son los que más consumen energía en una página web. De igual forma en base al tiempo de respuesta se determinó que la mejor solución para una aplicación en tiempo real, que incluya manejo de bibliotecas estáticas en lenguaje C, iOS presenta un mejor rendimiento que Android. En Android esto puede mejorarse, no haciendo uso de estas bibliotecas por medio de JNI, pero el rendimiento energético se ve elevado en un alto porcentaje, en comparación a al consumo energético mostrado haciendo uso de funciones estáticas por medio de JNI.

Se propone generar una única interfaz web que permita interactuar con servicios internos, es decir generar código nativo a la plataforma y con servicios web, para ello la solución es crear un API general que permita llamar a los servicios de una forma estandarizada en cualquier aplicación rich-client, ver figura 7.1.

Se determinó que una aplicación tipo rich client puede analizar los recursos disponibles y decidir el tipo de respuesta.
En este análisis el tipo de respuesta sugiere el llamado a código nativo en lenguaje c es el

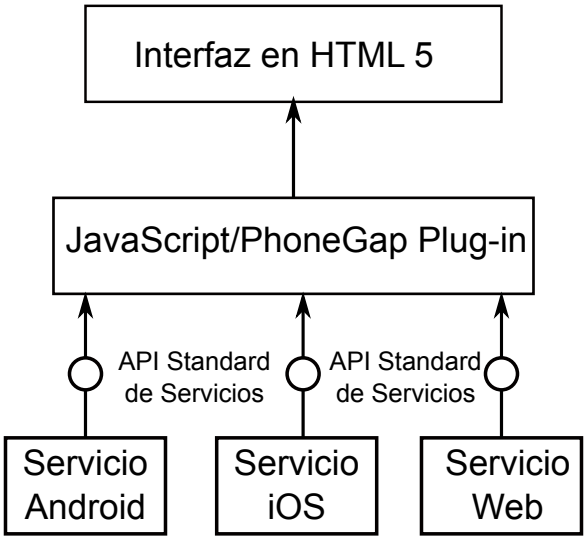


Figura 7.1: Solución general propuesta. API estándar para servicios locales y servicios web.

más apropiado, por su ahorro en consumo energético. El generar un API reducida permite realizar de manera adecuada el procesamiento de voz en diferentes plataformas.

Con todo lo anterior se determinó que si se puede tener un modulo abierto para móviles, se pueden encapsular las diferencias existentes entre plataformas por medio de una aplicación web creada en HTML5 con JavaScript.

7.3. Trabajo a futuro

En este apartado se muestran las principales mejoras que pueden realizarse a la capa de procesamiento de voz:

- buscar q ciertos servicios tengan sobrecarga de formatos
- proponer apis de servicios, patrones.
- Ampliar el diccionario de palabras reconocidas.
- Implementar el reconocimiento de voz por medio de la Transformada de Wavelet, este método se ha identificado por ser más eficiente que la transformada rápida de Fourier.
- Realizar una sobrecarga en la función de procesamiento de la API para que soporte diferentes formatos de sonido.
- Mejorar el procesamiento de voz para que sea independiente de la persona.

- Estandarizar los servicios, generar un API pública que permita llamar a los servicios de manera general.

Bibliografía

[1] Phei Zheng and Lionel Ni. *Smart Phone & Next Generation Mobile Computing*. The Morgan Kaufmann Series in Networking, 2006.

[2] S. Love. *Understanding Mobile Human-Computer Interaction*. Information Systems Series. Elsevier Science, 2005.

[3] R. Ballagas, J. Borchers, M. Rohs, and J.G. Sheridan. The smart phone: a ubiquitous input device. *Pervasive Computing, IEEE*, 5(1):70 – 77, jan.-march 2006.

[4] Nenad Medvidovic and George Edwards. Software architecture and mobility: A roadmap. *Journal of Systems and Software*, 83(6):885 – 898, 2010. Software Architecture and Mobility.

[5] Don Roberts, Ralph Johnson, et al. Evolving frameworks: A pattern language for developing object-oriented frameworks. *Pattern languages of program design*, 3:471–486, 1996.

[6] V. Lee, H. Schneider, and R. Schell. *Mobile applications: architecture, design, and development*. HP Professional Series. Pearson Education, 2004.

[7] M.P.P. Team. *Microsoft® Application Architecture Guide*. Microsoft Press, 2009.

[8] Anthony I. Wasserman. Software engineering issues for mobile application development. *FoSER 2010*, 2010.

[9] Apple developer. `http://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf`, 2012.

[10] Windows phone dev center. `http://msdn.microsoft.com/en-US/library/windowsphone/develop/ff626516(v=vs.105).aspx`, 2013.

[11] Birgitta König-Ries. Challenges in mobile application development. *it-Information Technology*, 51(2):69–71, 2009.

[12] C. Britton and P. Bye. *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*. Pearson Education, 2004.

[13] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, December 2004.

[14] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 2010. 10.1007/s13174-010-0007-6.

[15] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[16] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, pages n/a–n/a, 2011.

[17] Patrick Stuedi, Iqbal Mohomed, and Doug Terry. Wherestore: location-based data storage for mobile devices interacting with the cloud. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, MCS '10, pages 1:1–1:8, New York, NY, USA, 2010. ACM.

[18] George F. Coulouris, Jean Dollimore, and Tim Kindberg. *Sistemas Distribuidos Conceptos y Diseño*. Pearson Addison Wesley, 2007.

[19] Javier López and Cesar E. Castañeda. Middleware independent of the platform for definition and implementation of generic services in mobile devices. *Computing Congress(CCC), 2011 6th Colombian*, pages pp. 1–6, 2011.

[20] P. Smutny. Mobile development tools and cross-platform solutions. In *Carpathian Control Conference (ICCC), 2012 13th International*, pages 653 –656, may 2012.

[21] JM Noguera, RJ Segura, CJ Ogáyar, and R Joan-Arinyo. Middleware para desarrollo multiplataforma de gráficos 3d en dispositivos móviles.

[22] James Mooney. Developing portable software. *Information Technology*, pages 55–84, 2004.

[23] S. Allen, V. Graupera, and L. Lundrigan. *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution*. Apresspod Series. Apress, 2010.

[24] Titanium sdk. `http://www.appcelerator.com/platform/titanium-sdk/`, febrero 2013.

[25] Phonegap. `http://phonegap.com/`, octubre 2012.

[26] Mosync. `http://www.mosync.com/documentation/manualpages/mosync-reload`, febrero 2013.

[27] S.W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[28] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.

[29] Jordan Cohen. Embedded speech recognition applications in mobile phones: Status, trends, and challenges. *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages pp. 5352–5355, 2008.

[30] D. O'Shaughnessy. Linear predictive coding. *Potentials, IEEE*, 7(1):29 –32, feb. 1988.

[31] Joe Tebelskis. *Speech recognition using neural networks*. PhD thesis, Carnegie Mellon University, 1995.

[32] Lai Mei L. and Umi Kalsom Y. Hands-free messaging application (isay- sms): A proposed framework. *CSSR*, pages 1126–1131, 2010.

[33] Sanja Primorac and Mladen Russo. Android application for sending sms messages with speech recognition interface. *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 1763 – 1767, 2012.

[34] Alexander Gruenstein, Ian McGraw, and Ibrahim Badr. The wami toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *Proceedings of the 10th international conference on Multimodal interfaces*, ICMI '08, pages 141–148, New York, NY, USA, 2008. ACM.

[35] Openears. `http://www.politepix.com/openears/`, junio 2013.

[36] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: a flexible open source framework for speech recognition. Technical report, Mountain View, CA, USA, 2004.

[37] Zheng-Hua Tan, Børge Lindberg, Imre Varga, and Imre Kiss. Speech recognition in mobile phones. In *Automatic Speech Recognition on Mobile Devices and over Communication Networks*, Advances in Pattern Recognition, pages pp. 301–325. Springer London, 2008.

[38] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, vol. 77(no. 2):pp. 257 –286, feb 1989.

[39] JOSÉ MART. Fft como herramienta de análisis en fonética. 1988.

[40] Steven S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[41] J.F. Gales and S. Young. *The Application of Hidden Markov Models in Speech Recognition*. Now Publishers, 2008.

[42] Dmitry Zaykovskiy. Survey of the speech recognition techniques for mobile devices. *SPECOM*, pages 88–93, 2006.

[43] Alexander Schmitt, Dmitry Zaykovskiy, and Minker Wolfgang. Speech recognition for mobile devices. *International Journal of Speech Technology*, 11(2):63–72, 2009.

[44] A. Kumar, S. Horrigan, M. Kam, F. Metze, and Canny J. Rethinking speech recognition on mobile devices. *IUI4DR ACM*, pages 1–6, 2011.

[45] J. Tulach. *Practical API Design: Confessions of a Java Framework Architect*. Definitive Guide Series. Springer-Verlag New York Incorporated, 2008.

[46] M.D. Hawker. *Developer's Guide to Social Programming: Building Social Context Using Facebook, Google Friend Connect, and the Twitter API, The*. Developer's Library. Pearson Education, 2010.

[47] D. Jacobson, D. Woods, and G. Brail. *APIs: A Strategy Guide*. Oreilly and Associate Series. O'Reilly Media, 2011.

[48] S. Sarkar, G.M. Rama, and A.C. Kak. Api-based and information-theoretic metrics for measuring the quality of software modularization. *Software Engineering, IEEE Transactions on*, 33(1):14–32, 2007.

[49] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 41–50, New York, NY, USA, 2012. ACM.

[50] R. Ghatol and Y. Patel. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Apresspod Series. Apress, 2012.

[51] Irene Monserrat Torres Hernandez, Amilcar Meneses Viveros, and Erika Hernandez Rubio. Analysis for the design of open applications on mobile devices. In *Electronics, Communications and Computing (CONIELECOMP), 2013 International Conference on*, pages 126–131, 2013.

[52] Phonegapplugin. `http://www.adobe.com/devnet/html5/articles/extending-phonegap-with-native-plugins-for-ios.html`, febrero 2013.

[53] T. Myer. *Beginning PhoneGap*. Programmer to programmer. Wiley, 2011.

[54] Z. Mednieks, L. Dornin, G.B. Meike, and M. Nakamura. *Programming Android*. O'Reilly Media, 2011.

[55] Androiddeveloper. http://developer.android.com/reference/android/media/AudioRecord.html, mayo 2013.

[56] libsndfile. http://www.mega-nerd.com/libsndfile/, marzo 2013.

[57] Steven W. Smith. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, San Diego, CA, USA, 1997.

[58] Androiddeveloperformats. http://developer.android.com/guide/appendix/media-formats.html, febrero 2013.

[59] Powertutor. http://powertutor.org/, agosto 2013.

[60] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, and Bharat Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[61] Apispeech. https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html, agosto 2013.

[62] Julius Adorf. Web speech api. 2013.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

Departamento de Computación

Después de haber efectuado la revisión del trabajo de tesis titulado

"Framework multiplataforma para reconocimiento de voz en aplicaciones open rich-client para dispositivos móviles".

Realizado por la alumna Irene Monserrat Torres Hernández , bajo la dirección del Dr. Amilcar Meneses Viveros y M. en C. Erika Hernández Rubio.

1 El documento final de tesis cumple con los requisitos para el Programa de Maestría en Ciencias en Computación.

2 Solicitamos que la fecha de Examen de Grado se lleve a cabo el día 23 de octubre de 2013 a las 12:00 hrs.

3 Firman la presente el día 10 de octubre del año 2013:

Dr. Amilcar Meneses Viveros

M. en C. Erika Hernández Rubio

Dra. Sonia Guadalupe Mendoza Chapa

Dr. Sergio V. Chapa Vergara

Av. Instituto Politécnico Nacional # 2508 Col. San Pedro Zacatenco México, D.F. C.P. 07360
Tel.: 5747-3800 Fax: 5747-7002