Facultad
de
Ciencias

DESARROLLO DE UN SERVICIO DE RECONOCIMIENTO DE VOZ PARA SU INTEGRACIÓN EN IDBOX

(Development of a voice recognition service for its integration in IDbox)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Elsa Cerezo Fernández

Director: Marta Elena Zorrilla Pantaleón

Co-Director: Ángel Tezanos Ibáñez

Septiembre - 2018

Índice de contenido

Índice de figuras

Índice de tablas

Agradecimientos

En primer lugar, quiero agradecer a mi familia y amigos por todo el apoyo que me han brindado durante este largo camino, animándome y ayudándome en medida de lo posible a lidiar con los obstáculos que se interponían.

A CIC Consulting Informático, por darme la oportunidad de realizar este proyecto en su empresa. Concretamente, quisiera agradecer a Ángel Tezanos y a David Perera la ayuda que me han dedicado tanto en mi primer contacto con el mundo laboral, como en la realización de este Trabajo de Fin de Grado.

Por otro lado, dar las gracias a todos los docentes que me han formado por los conocimientos que me han aportado en cada una de mis etapas académicas con los que he conseguido llegar hasta aquí. En especial, a Marta Zorrilla por la energía, preocupación y dedicación que siempre ha trasmitido.

Por último, quisiera hacer una mención especial a una persona que me ha dado apoyo incondicional desde el primer día, muchas gracias por todo Álvaro.

Resumen

CIC dispone de un producto denominado IDbox que permite la monitorización de procesos industriales tales como la generación energética, las redes de distribución o las líneas de producción entre otros.

CIC quiere incorporar a su APP móvil en Android una nueva funcionalidad que facilite a los usuarios la realización de consultas en lenguaje natural mediante dictado por voz. Este TFG, por ello, tiene por objeto estudiar y analizar los distintos *frameworks* de traducción de voz a texto (*Speech-to-Text*) y de procesado de lenguaje natural (*information retrieval*) que operen sobre el sistema operativo Android, preferentemente gratuitos y sin necesidad de que el cliente móvil esté conectado a internet; seleccionar los *frameworks* que mejor satisfacen estos requisitos y desarrollar una prueba de concepto para mostrar su viabilidad.

Palabras clave: Procesamiento del Lenguaje Natural, procesamiento voz a texto, aplicación móvil, Android.

Abstract

CIC provides a product named IDbox that allows monitoring of industrial processes such as energy generation, distribution networks or production lines, among others.

CIC wants to incorporate a new feature to its mobile APP in Android that enables users to perform queries in natural language through voice dictation. This project, therefore, aims to study and analyze the different *Speech-to-Text* and Natural Language Processing frameworks that operate on the Android Operating System, preferably free and without being necessary internet connection, for the mobile client; select the frameworkds which best meet these requirements and develop a proof of concepts to check its viability.

Keywords: Natural Language Processing, *Speech-to-Text*, mobile app, Android.

Capítulo 1

Introducción

En este capítulo se explica el contexto y objetivo del presente proyecto de fin de carrera, así como definir su alcance.

1.1. Antecedentes y objetivo

Desde que en los 90 saliese al mercado el primer smartphone, estos han ido evolucionando tanto a nivel hardware como software, consiguiendo cada vez mejores prestaciones de rendimiento y usabilidad. El reconocimiento de voz es una de las últimas funcionalidades incorporadas en los dispositivos móviles que facilitan al usuario la realización de tareas de forma más ágil y directa.

CIC Consulting Informático es una empresa de consultoría informática de Cantabria que tiene un producto denominado IDbox que permite a sus clientes monitorizar el estado de una planta industrial. Su aplicación móvil no dispone de funcionalidades de reconocimiento de voz y quieren incorporarlas para que el usuario pueda consultar datos o solicitar la creación de una gráfica sin necesidad de navegar por la interfaz.

Este proyecto, por tanto, tiene como objeto estudiar las herramientas de reconocimiento de voz y procesamiento del lenguaje natural disponibles de forma gratuita, que no necesiten conexión a Internet y que puedan ser integradas en IDbox para su uso en teléfonos móviles con sistema operativo Android, seleccionar las que cumplan mejor los requisitos funcionales especificados para, posteriormente, implementar un prototipo que permita traducir la consulta realizada mediante voz por los usuarios a las siguientes acciones:

- Buscar un elemento del sistema.
- Mostrar el tiempo real de una señal.
- Mostrar el histórico de una señal.
- Mostrar la gráfica de una señal.

1.2. Alcance

El sistema de reconocimiento de voz especificado en este proyecto se puede descomponer en tres módulos:

1. El módulo de reconocimiento de voz y el paso del audio a texto (paso 1 en la figura 1.1).
2. El módulo de procesamiento del texto resultante del paso 1 (paso 2 en la figura 1.1).
3. El módulo encargado de resolver la acción que se ha obtenido mediante el procesamiento de la consulta del usuario (paso 3 en la figura 1.1).

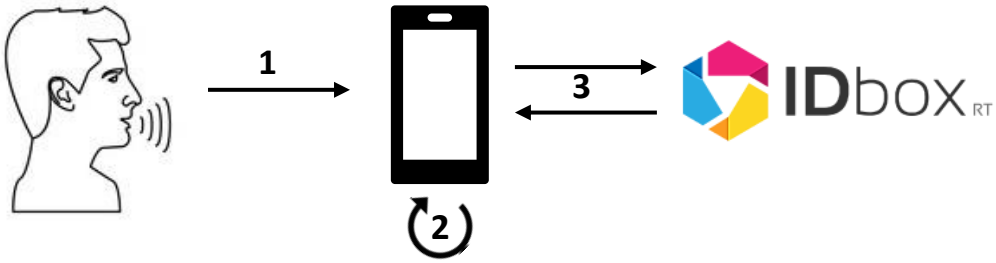


Figura 1.1. Esquema del sistema a alto nivel

El presente proyecto se centra en la búsqueda de las herramientas que realicen, por un lado, la trascripción *Speech-to-Text* (paso 1 de la figura 1.1) y, por otro, el procesamiento de consultas (paso 2 de la figura 1.1) y de su encapsulación en un servicio que pueda ser utilizado por un aplicativo externo.

Por lo tanto, el proyecto abarca el estudio y análisis de herramientas y la implementación de una prueba de concepto que muestre el grado de cumplimiento de los requisitos y limitaciones de integración que presenta. En ningún caso, el fin del proyecto será la integración en el producto IDbox, por ello, la implementación de las funciones que serán ejecutadas para responder a la consulta del usuario y el correspondiente acceso al servidor de IDbox, no son desarrollados.

Capítulo 2

Procesamiento del Lenguaje Natural (NLP)

Antes de explicar el proceso de desarrollo del sistema planteado, conviene explicar en qué consiste un sistema de Procesamiento del Lenguaje Natural (NLP).

El NLP es un ámbito dentro de la Inteligencia Artificial que lleva unos años en auge y tiene como objetivos principales hacer posible que las computadoras puedan entender el leguaje natural del usuario para que éste pueda realizar instrucciones, el procesamiento de textos para la recuperación de datos y la traducción automática [1][2].

Sin embargo, los algoritmos de NLP encuentran algunas dificultades a la hora de procesar la información que les llega del usuario:

> <<La ambigüedad del lenguaje natural tanto a nivel léxico (el significado de una palabra se debe deducir a partir del contexto oracional), como a nivel referencial (anáforas y catáforas), estructural (se requiere de la semántica para desambiguar la dependencia de los sintagmas preposicionales) y pragmático (el uso de la ironía puede cambiar por completo el sentido de la oración y, por lo tanto, toma un papel importante la interpretación del mensaje). En resumen, el problema principal para resolver estas ambigüedades es la traducción de lenguaje natural a una interpretación sin ambigüedad>> [3].
>
> Otro de los principales problemas dentro del NLP es la recepción imperfecta de datos, ya sea porque la acentuación difiere dependiendo de la lengua, por los regionalismos o por cualquier tipo de problema que pueda causar un incorrecto desarrollo del lenguaje, tanto hablado como escrito.

Entonces, para implementar un procesador de lenguaje natural, se debe definir un modelo de lenguaje, diferente por cada lenguaje existente (p.ej. el modelo de lenguaje para el español será diferente que para el inglés), compuesto por los niveles fonológico, morfológico, sintáctico, semántico y pragmático. De esta manera se especifica el lenguaje desde los fonemas y morfemas que componen las palabras, pasando por un etiquetado de la categoría gramatical de cada palabra, pudiendo así, posteriormente, procesar el lenguaje hasta conseguir el significado de cada oración, determinado por el contexto [4].

Para entender los términos usados en los siguientes apartados relacionados con el procesamiento del lenguaje, se definen a continuación:

- **Tokenización**: Es el proceso por el cuál un texto es dividido en partes más pequeñas llamadas *tokens*. En este contexto, el texto que será tokenizado será la consulta del usuario y los tokens las palabras en las que se divide [5].
- **Stemming**: En este proceso se halla el lexema/raíz de una forma flexionada [4].
- **Stop Word**: Este concepto engloba todas las palabras que carecen de significado.

Capítulo 3

Análisis de requisitos

Como se ha mencionado con anterioridad, CIC Consulting Informático es una empresa cuyo producto, IDBox, permite a sus clientes monitorizar el estado de una planta industrial.

Actualmente, el software almacena señales y documentos que los usuarios necesitan consultar, y para entender las funcionalidades que debe cumplir la prueba de concepto es necesario determinar qué son señales y qué son documentos:

> Las señales son parámetros de una planta industrial que se componen de un código identificador propio de la empresa y una descripción (p.ej. la señal 'Punto de rocío a 10 metros de altura' en CIC se identifica como CICA16). Además, éstas pueden ser representadas en gráficas, mostradas en tiempo real o incluso se pueden ver sus históricos.
>
> Por otro lado, los documentos son ficheros contenedores de información y se caracterizan por su nombre y su tipo. Esta última característica identifica el tipo de información que contienen y puede ser: de agrupaciones, diagramas, sinópticos, informes, alarmas y eventos, favoritos, mapas, gráficas de tendencia, de agrupados, de predicción, de dispersión, de correlación y de comparación, *dashboards*, señales y de contenido genérico.

Después de explicar ambos conceptos, se establece, a continuación, la definición de requisitos de la nueva funcionalidad que CIC pretende añadir a la aplicación móvil de su producto.

3.1. Requisitos funcionales

El objetivo de la prueba de concepto es que el sistema traduzca las consultas del usuario dictadas por voz, mostrando una cadena de texto que contenga el nombre de la función que se ejecuta en cada caso y los valores de los parámetros correspondientes (ver figura 3.1). Estos comandos han sido definidos de la siguiente manera:

Comando	Función
Buscar <type> <name>	Search (type, name)
Tiempo real <signal>	ShowRealTime (signal)
Histórico <signal> <number> <measure>	ShowHistoric (signal, number, measure)
Gráfica <signal> <number> <measure>	Show Graphic (signal, number, measure)

Tabla 3.1. Comandos y las funciones que accionan

- ***Search (type, name)***

  Es esta la función que responde al comando *Buscar <type> <name>*, siendo *type,* el tipo de documento que se quiere consultar, y *name,* el nombre del documento.

  El objetivo futuro de esta función es buscar y mostrar las coincidencias según el tipo y el nombre del documento especificados. En el caso de que se encuentre una sola coincidencia, se visualizará directamente.

  Será necesario especificar uno de los dos parámetros para la correcta resolución de la consulta. Si no se reconoce ningún tipo actualmente definido por el sistema, ese *string* se añadirá al parámetro *name*, realizando una búsqueda por texto libre.

- ***ShowRealTime (signal)***

  Cuando la función *ShowRealTime* sea correctamente implementada se encargará de mostrar la gráfica de valores en tiempo real de la señal que se especifica con el parámetro *signal*, el cual recoge el nombre de la señal. Es esta la función que será activada para resolver el comando *Tiempo Real <signal>.*

- ***ShowHistoric (signal, number, measure)***

  Aunque actualmente las funciones retornan el nombre de la propia función, esta función deberá mostrar el histórico de valores de la señal *signal*, delimitado según las características *number* y *measure* especificadas. La primera indica la cantidad de registros (los últimos que han sido registrados), agrupados por la segunda, *measure*, el tipo de medida, pudiendo ser: horas, días, semanas, meses, años y valores. Esta función será la que se ejecute tras el procesamiento del comando *Histórico <signal> <number> <measure>.*

- ***ShowGraphic (signal, number, measure)***

  Esta función responderá al comando *Gráfica <signal> <number> <measure>* y la consulta deberá resolverse mostrando la gráfica de la señal que se especifica con el parámetro *signal*, según las características *number* y *measure*, las cuales ya se han explicado anteriormente.

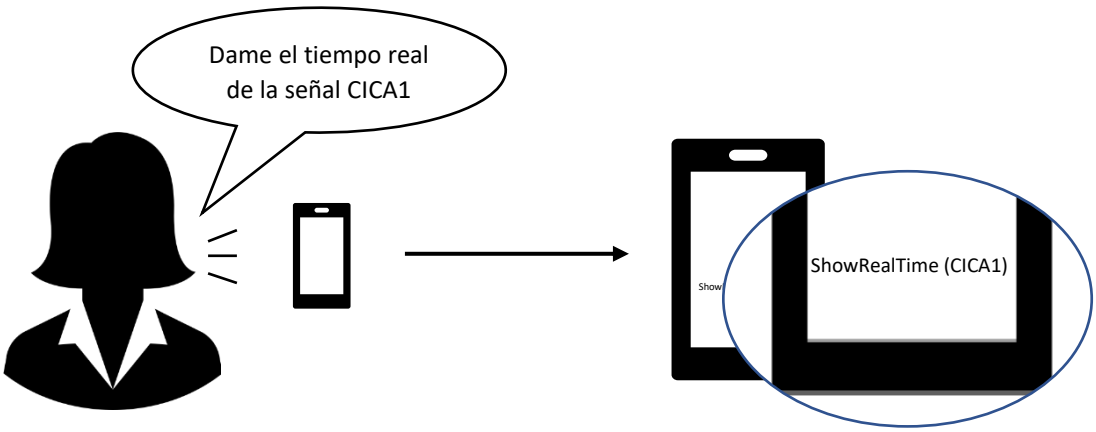


Figura 3.1. Ejemplo del funcionamiento del sistema con el comando Tiempo Real <signal>

3.2. Requisitos no funcionales

El sistema que se desarrolle también deberá atender a los siguientes requisitos:

- Implementación con Xamarin.Forms
  La aplicación del producto IDbox ha sido implementada con Xamarin.Forms y para facilitar la integración de la nueva funcionalidad se deberá desarrollar de la misma forma.
- Patrón MVVM
  Por la misma razón que el requisito anterior, se utilizará el patrón MVVM para la construcción de la prueba de concepto.
- Todo en el dispositivo
  Se requiere que tanto el reconocimiento de voz, como su transcripción a texto y el procesamiento del lenguaje se realicen en el propio dispositivo.
- Offline y sin costes
  Además, las herramientas seleccionadas deberán formar un sistema lo más independiente posible y de coste cero, por lo que es importante que pueda dar servicio *offline* y las herramientas sean de código abierto y/o con una licencia que permita la comercialización de estás de forma gratuita.

3.3. Casos de uso

Antes de empezar con la implementación se ha definido el caso de uso 'Consultar por comando de voz' que es el que se va a desarrollar, especificando el flujo de pasos que debe seguir cuando sea activado. A continuación, se encuentra esta definición:

Nombre:	**Consultar por comando de voz**
Descripción:	El usuario realiza una consulta por voz que puede ser una búsqueda de un documento o de una señal, o el tiempo real, el histórico o la gráfica de una señal.
Actores primarios:	Usuario de IDbox
Actores secundarios:	-
Evento de activación:	El caso de uso comienza cuando el usuario pulsa el botón "REC".
Precondiciones:	Previamente, el usuario debe haber dado a la aplicación permiso para grabar audios.
Flujo principal:	1. El usuario formula su consulta. 2. El sistema reconoce el lenguaje y lo transcribe a texto. 3. El usuario pulsa el botón "PROC". 3. El sistema procesa el texto. 3.1. El sistema separa el texto en palabras. 3.2. El sistema elimina del texto las palabras sin un significado determinante para la consulta. 3.3. El sistema analiza las palabras resultantes del paso 3.2 y determina que acción ejecutar. 4. Se muestra el resultado.

Postcondiciones:	-
Flujos alternativos:	2.a El audio recogido está vacío. 2.a.1. El sistema muestra el estado que presentaba la aplicación antes de iniciarse el caso de uso. 4.a. Se muestran diferentes resultados según la estructura del comando de la consulta realizada: 4.a.1. Si la estructura del comando es *Buscar <type> <name>*, se muestra el resultado de la función *Search (type, name).* 4.a.2. Si la estructura del comando es *Buscar <type>*, se muestra el resultado de la función *Search (type, null).* 4.a.3. Si la estructura del comando es *Buscar <name>*, se muestra el resultado de la función *Search (null, name).* 4.a.4. Si la estructura del comando es *Tiempo real <signal>*, se muestra el resultado de la función *ShowRealTime (signal).* 4.a.5. Si la estructura del comando es *Histórico <signal> <number> <measure>*, se muestra el resultado de la función *ShowHistoric (signal, number, measure).* 4.a.4. Si la estructura del comando es *Gráfica <signal> <number> <measure>*, se muestra el resultado de la función *ShowGraphic (signal, number, measure).* 4.b. Si sistema no reconoce ningún comando en el audio, el sistema muestra el mensaje “El sistema no dispone de ninguna acción que coincida con la estructura de su consulta”.

Tabla 3.2. Definición del caso de uso “Consultar por comando de voz”

Capítulo 4

Recursos utilizados y metodología de trabajo

En los siguientes apartados, se van a enumerar y describir las herramientas, librerías y tecnologías utilizadas para el desarrollo del sistema propuesto y, además, se explicará la metodología de desarrollo que se ha seguido durante la implementación de la prueba de concepto.

4.1. Tecnologías, librerías y herramientas utilizadas

En el presente apartado se describen las tecnologías, librerías y herramientas utilizadas durante el desarrollo del proyecto.

1. **C#**

Un lenguaje de programación orientado a objetos, derivado de C y C++ y con una sintaxis similar a la de Java. C# forma parte del Framework de Microsoft .NET que engloba una gran cantidad de lenguajes de programación. En este lenguaje se ha implementado la aplicación de la prueba de concepto [6].

2. **.NET**

.NET es una plataforma de desarrollo de código abierto, gratuita y multiplataforma que permite la implementación de diferentes tipos de aplicaciones [7]. Además, consta de diferentes tipos de implementaciones, de entre todas se destacan las siguientes: .NET Framework, compatible con sitios web, servicios, aplicaciones de escritorio, entre otros; y Xamarin, para ejecutar aplicaciones en todos los principales sistemas operativos móviles.

3. **Microsoft Visual Studio IDE**

El entorno de programación Visual Studio se ha elegido para la implementación en .NET, concretamente en Xamarin. Se ha utilizado la especificación .NET Standard para crear una biblioteca de clases y poder reutilizar código en la implementación de la aplicación entre distintas plataformas [8].

4. **Android**

Un sistema operativo basado en el núcleo de Linux y diseñado para dispositivos móviles táctiles. La estructura de Android se puede descomponer en cinco componentes principales: las aplicaciones que vienen por defecto, el marco de trabajo de las aplicaciones (*Application Framework*), librerías de C/C++, librerías Java que proveen la mayor parte de funcionalidades disponibles (*Android Runtime*) y el núcleo de Linux, que sirve como una capa de abstracción entre el hardware y el software del sistema [9] [10].

5. **Xamarin.Forms**

Xamarin.Forms [11] es un *toolkit* para el desarrollo en .NET de aplicaciones multiplataforma para Android, iOS y Windows implementadas con el lenguaje C# en Visual Studio.

6. **JSON**

Es un formato basado en un *subset* de JavaScript y utilizado para para el intercambio de datos ya que, además de ser fácil de leer y utilizar para los humanos y fácil de analizar y generar para las máquinas, utiliza convenciones similares a las utilizadas en el desarrollo con lenguajes de la familia C [12].

7. **Newtonsoft.json**

Es un paquete que nos permite aplicar las funciones de serialización y deserialización a distintos ficheros JSON [13].

8. **Librería Speech de Google integrada en Android**

Speech es una librería de Android que utiliza los servicios de Google de forma gratuita para realizar las funciones de reconocimiento de voz y transcripción de audio a texto [14].

9. **MagicDraw 19.0.**

Herramienta CASE que permite tanto la realización de modelos del estándar UML, como el desarrollo de código en distintos lenguajes. En este caso, se ha utilizado para diseñar la arquitectura del sistema creando diagramas bajo el estándar UML [15].

10. **NinjaMock**

Herramienta *on-line* utilizada para bosquejar la interfaz de la aplicación a desarrollar para realizar la prueba de concepto [16].

11. **Git**

Git es un software utilizado para el control de versiones, gratuito y de código abierto. A diferencia de otros sistemas de control de versiones centralizados como Apache Subversion (SVN), Git es un sistema distribuido en el que cada desarrollador tiene el histórico de versiones completo en su repositorio local [17].

12. **NUnit**

NUnit es un *framework* utilizado para la implementación de las pruebas unitarias de la aplicación. Es para todos los lenguajes y esta soportado por una amplia gama de plataformas .NET [18].

13. **MS Project**

Este producto es utilizado para la administración de proyectos, recursos y presupuestos, ayudando a realizar un seguimiento de los proyectos. En el presente proyecto se ha utilizado para realizar el diagrama de Gantt de sus tareas [19].

4.2. Metodología

El proyecto se ha llevado a cabo de manera individual, aunque se han realizado reuniones cada dos semanas con un programador sénior, que forma parte del equipo de desarrollo de la aplicación de IDbox, y junto al jefe de proyecto, que dirige al equipo de desarrollo. Ambos se han encargado de especificar los requisitos del proyecto y de dar indicaciones durante el desarrollo de éste.

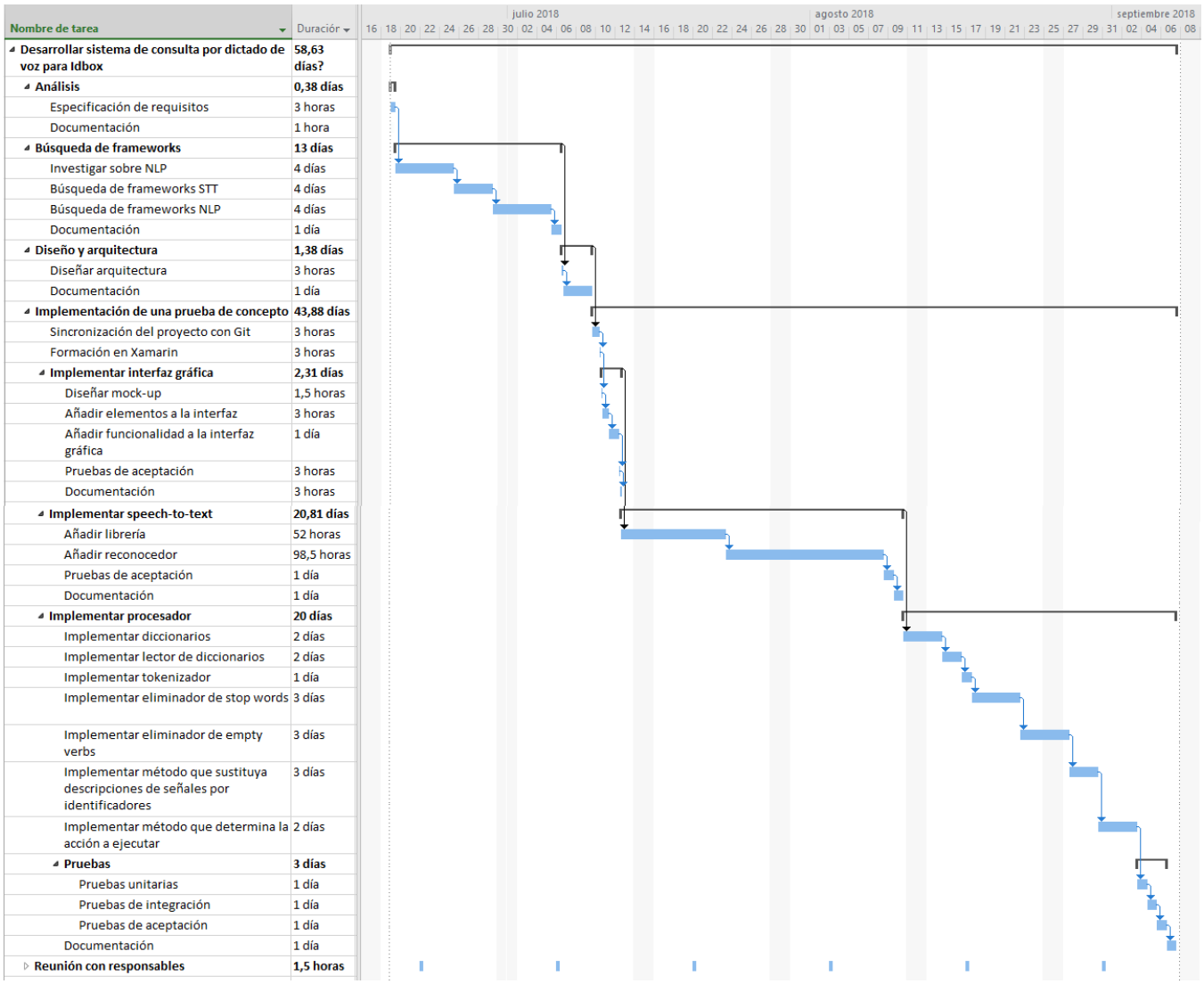
Durante el desarrollo del proyecto se ha decidido seguir una metodología iterativa e incremental en su desarrollo, dado que con esta técnica se cubre los defectos del modelo en cascada.

El modelo en cascada es un procedimiento de desarrollo que sigue una secuencia lineal cuyo principal inconveniente es que, si los errores son cometidos en las primeras etapas de la vida del proyecto, se arrastran hasta la última etapa obligando a realizar nuevamente todos los pasos del modelo para corregirlos.

Por el contrario, con la metodología escogida se realizan las distintas etapas por cada módulo a desarrollar y, si en algún caso es necesario repetir alguno de los pasos solo se tendría que añadir una nueva iteración sobre el módulo correspondiente.

En el diagrama de Gantt que se muestra en la figura 4.1, se ha representado el tiempo total que ha llevado cada tarea. Sin embargo, hay algunas partes que no se corresponden totalmente con lo representado:

- Las pruebas y la documentación de cada bloque de tareas aparecen con una cierta duración y en el diagrama da la impresión de que se han realizado al finalizar el resto de las tareas de las que dependen, pero no ha sido así. Tanto las pruebas, como la documentación, se han llevado a cabo a medida que se efectuaban las otras tareas.
- Aunque las reuniones produzcan solapamientos con otras tareas, solo han ocupado quince minutos el día que se realizaba cada una de ellas, por lo que han permitido realizar las otras tareas.
- Las tareas “Implementar interfaz gráfica”, “Implementar speech-to-text” e “Implementar procesador” se han realizado de forma iterativa durante su desarrollo. Se han hecho pruebas periódicamente (como ya se ha indicado) en cada una de las iteraciones para corregir los fallos que se han mostrado con los test.



Nombre de tarea	Duración
Desarrollar sistema de consulta por dictado de voz para Idbox	**58,63 días?**
Análisis	**0,38 días**
Especificación de requisitos	3 horas
Documentación	1 hora
Búsqueda de frameworks	**13 días**
Investigar sobre NLP	4 días
Búsqueda de frameworks STT	4 días
Búsqueda de frameworks NLP	4 días
Documentación	1 día
Diseño y arquitectura	**1,38 días**
Diseñar arquitectura	3 horas
Documentación	1 día
Implementación de una prueba de concepto	**43,88 días**
Sincronización del proyecto con Git	3 horas
Formación en Xamarin	3 horas
Implementar interfaz gráfica	**2,31 días**
Diseñar mock-up	1,5 horas
Añadir elementos a la interfaz	3 horas
Añadir funcionalidad a la interfaz gráfica	1 día
Pruebas de aceptación	3 horas
Documentación	3 horas
Implementar speech-to-text	**20,81 días**
Añadir librería	52 horas
Añadir reconocedor	98,5 horas
Pruebas de aceptación	1 día
Documentación	1 día
Implementar procesador	**20 días**
Implementar diccionarios	2 días
Implementar lector de diccionarios	2 días
Implementar tokenizador	1 día
Implementar eliminador de stop words	3 días
Implementar eliminador de empty verbs	3 días
Implementar método que sustituya descripciones de señales por identificadores	3 días
Implementar método que determina la acción a ejecutar	2 días
Pruebas	**3 días**
Pruebas unitarias	1 día
Pruebas de integración	1 día
Pruebas de aceptación	1 día
Documentación	1 día
Reunión con responsables	**1,5 horas**

Figura 4.1. Diagrama de Gantt del proyecto

Capítulo 5

Búsqueda y selección de herramientas

En este capítulo se recoge la búsqueda y selección de las herramientas que se encargarán de la trascripción Speech-to-Text y del procesamiento de la consulta.

Primeramente, se ha realizado una fase de investigación donde se ha realizado la búsqueda de las herramientas que podrían encajar con los requisitos especificados y, después, se ha realizado la selección. Este proceso se muestra en el siguiente apartado:

5.1. Búsqueda y selección de herramientas: *Speech-to-Text*

Para el módulo dedicado a la transcripción Speech-to-Text, donde se transcribe el audio de la consulta realizada por el usuario a texto, se han encontrado librerías con distintas características y que sirven para implementar el módulo.

La búsqueda se ha realizado en Google utilizando las siguientes palabras clave: *Speech-to-Text, Speech Recognizer,* Xamarin, C#, Android.

En la tabla 5.1., se recogen las características que deben cumplir las herramientas para satisfacer los requisitos de la aplicación. También se incluye si las herramientas permiten personalizar el modelo, esto significa que permiten añadir nuevas palabras al dominio del vocabulario que puede ser reconocido. En un principio no se contempló esta característica, sin embargo, después de la primera prueba que se hizo con una de las librerías, se comprobó que era una característica imprescindible.

Herramientas	**Características**				
	Gratuita	**Offline**	**Lenguaje Español**	**Android**	**Personalizar modelo**
IBM Watson Speech-to-text	No	No	Sí	Sí	Sí
Bing Speech API	No	No	Sí	Sí	No
Google Cloud Speech-to-text	No	No	Sí	Sí	Sí
Speech de Google en Android	Sí	No	Sí	Sí	No
API.AI	No	No	Si	Si	No
Amazon Transcribe	No	No	Sí	Sí	Sí
CMUSphinx	Sí	Sí	Sí	Sí	Sí

Tabla 5.1. Herramientas encontradas dedicadas al Speech-to-Text

A continuación, se describen las herramientas encontradas:

1. **IBM Watson Speech-to-Text**
   Este servicio de pago y dependiente de internet, que proporciona acceso a las capacidades de IBM Watson Speech Recognition para realizar el reconocimiento de audio y su transcripción a texto. Además, da la opción de entrenar el modelo de lenguaje para extender el diccionario de términos que pueden ser reconocidos [20] [21].
2. **Bing Speech API**
   API con la que Microsoft nos proporciona reconocimiento de voz y transcripción de voz a texto en tiempo real. Sus principales inconvenientes: necesita conexión a internet y es de pago [22].
3. **Google Cloud Speech-to-Text**
   Este servicio de Google Cloud no es gratuito y tampoco funciona offline, pero además del reconocimiento de voz y la trascripción del audio a texto, da opción a añadir *phrase hints* de varias palabras o palabras sueltas, entre otras opciones, que ayudan al reconocimiento de palabras, aumentando el dominio de términos que pueden ser reconocidos [23].
4. **Speech de Google en Android**
   Speech es una librería de Android que utiliza el servicio básico de Google Speech-to-Text, sólo proporciona la funcionalidad de reconocimiento de voz y su transcripción a texto. Por otro lado, a diferencia de Google Cloud Speech-to-Text, es gratuita [14].
5. **API.AI**
   API.AI se trata de una librería más focalizada en los *chatbots*. Dispone de un plugin para C# y, aunque ofrece servicios gratuitos, son limitados. Por ello, para su uso comercial debería pagarse la edición *Enterprise* [24]. Por otra parte, tampoco da la posibilidad de añadir nuevo vocabulario al reconocedor.
6. **Amazon Transcribe**
   Esta librería de Amazon es de pago y tampoco funciona offline, sin embargo, da opción a añadir un diccionario personalizado con palabras propias del dominio de la aplicación [25].
7. **CMUSphinx**
   Esta librería es de software libre, no necesita conexión a internet y da la opción de añadir palabras al modelo [26]. A continuación, se detallan las versiones este software que más interesan:
   - *pocketsphinx*, programada en C++ y con una última actualización hace ocho meses. Según su documentación sirve tanto para dispositivos móviles como para el escritorio.
   - *pocketsphinx-android*, programada en Java y con una última actualización hace unos meses. Según su documentación sirve solo para Android.

   Aunque es gratuita, los modelos que utiliza están cubiertos bajo la licencia GNL.

El grado de precisión en la respuesta de cada herramienta no ha sido especificado, dado que algunas herramientas permiten que cada usuario pueda entrenar al reconocedor de tal forma que disminuya el índice de error y otras no son tan utilizadas y no se encuentra información sobre su índice de *accuracy*. Sin embargo, se puede recoger de distintas fuentes que las herramientas con mayor madurez [27] y mejores valoraciones [28] [29] [30] son: Google Cloud Speech-to-Text, IBM Watson Speech-to-Text, Bing Speech de Microsoft y Amazon Transcribe.

Primeramente, se encontraron otras librerías para el reconocimiento de voz, como SFSpeechRecognizer [31], que fueron descartadas de forma inmediata ya que solo servían para implementaciones en iOS. Por ello, no aparecen recogidas en la tabla.

De todas las librerías encontradas, para Android, las primeras en ser descartadas fueron IBM Watson Speech-to-Text, Google Cloud Speech-to-Text, Bing Speech API, API.AI y Amazon Transcribe por requerir algún tipo de coste por su uso, ya que forma parte de los requisitos especificados en el proyecto. Además, con API.AI no se podrían añadir palabras al vocabulario del reconocedor. Por lo tanto, las posibilidades se redujeron a Speech de Google integrada en Android y CMUSphinx.

Cómo se ha dicho anteriormente, la librería Speech de Google integrada en Android nos ofrece la funcionalidad básica de Google Cloud Speech-to-Text, por esta misma razón, aunque requiere conexión a internet, esta librería tiene más madurez y mejores valoraciones respecto CMUSphinx. Además, CMUSphinx es una librería implementada en C++ y requiere la utilización de un *port* para poder utilizarse en C#, así que utilizar la librería Speech evitaría cargar a la aplicación con más software.

Sin embargo, al integrar y probar el *Speech Recognizer* de la librería Speech de Google en la aplicación, se detectó que el reconocedor no interpretaba correctamente los códigos internos que se utilizan para la identificación de señales (p.ej. “CICA1”) porque no son palabras definidas en el modelo de lenguaje utilizado por el reconocedor.

Debido a esto, se planteó una primera solución que trataba de etiquetar el *string* reconocido por el sistema de reconocimiento de voz, al escuchar el nombre del identificador, con el propio nombre del identificador. Esto es que, si al decir “CICA1”, el sistema de reconocimiento obtuviera “C Y C A1”, se sustituiría este último *string* por el identificador. Sin embargo, esta solución no fue válida porque:

1. Después de hacer varias pruebas, para el identificador “CICA1” se ha obtenido como resultado que el sistema de reconocimiento de voz no detecta el identificador y, por lo tanto, lo reconoce cada vez con una palabra distinta. Concluyendo que es casi imposible abarcar todas las posibilidades por las que un identificador puede ser reconocido.
2. Además, si el dominio de identificadores fuera pequeño, podría ser posible esta solución a pesar de la carga de trabajo que llevaría, sin embargo, no es escalable. No es un procedimiento que deba y/o pueda ser llevado bajo un domino de, por ejemplo, un millar de identificadores.

Por ello, tras descartar esta primera solución, se concluyó que la librería a utilizar debería permitir la personalización del modelo, pudiendo así añadir al vocabulario los identificadores de señales.

Entonces, de entre todas las librerías encontradas, se eligió CMUSphinx como mejor opción, dado que funciona sobre Android, es gratuita, no necesita conexión a Internet y permite añadir palabras al modelo. Sin embargo, después de trabajar con ella e integrarla, no llegó a funcionar. Se dedicaron tres semanas a la integración de la librería y a la implementación del reconocedor, pero debido a la falta de documentación y a que la comunidad que usa esta librería es pequeña, no se halló solución, por lo que se descartó dejándola como línea de futuro.

Finalmente, se retomó la librería *Speech* integrada en Android, para la implementación de la prueba de concepto ya que, en ese momento, no se halló mejor opción de acuerdo con los requisitos. Y, aunque no reconoce los identificadores de las señales, actualmente se ha conseguido que las señales se puedan identificar por sus descripciones.

Para finalizar con el apartado dedicado a la búsqueda de la herramienta del módulo de reconocimiento de voz, se concluye que a pesar de no haber conseguido hacer funcionar la librería CMUSphinx, es la opción que más se ajusta a los requisitos especificados. Sin embargo, IBM Watson Speech-to-Text, Google Cloud Speech-to-Text y Amazon Transcribe son otras librerías que pueden servir como solución en un futuro si la empresa se replantea utilizar software que sea de pago o que necesite conexión a internet, porque a excepción de esas características, también permiten la adición de nuevas palabras al vocabulario del reconocedor y, además, cuentan con buenas valoraciones en el ámbito del reconocimiento de voz.

5.2. Búsqueda y selección de herramientas: NLP

En relación con el módulo dedicado al procesado de la consulta, que analiza y trata el texto recibido del módulo Speech-to-Text y lo traduce a una de las acciones definidas en el apartado 3.1, se hizo una búsqueda en Google con las siguientes palabras clave: NLP, Spanish, free, offline C# y .NET y se han encontrado las siguientes herramientas que se recogen en la tabla 5.2.

Las características recogidas en la tabla son dos de los principales requisitos del proyecto respecto al *framework* que se seleccione para la implementación del módulo, y otras características o, mejor dicho, funcionalidades que deben incorporar las herramientas. A continuación, se describe de forma breve para qué sirven cada una de ellas:

- Tokenizer: Se encarga de dividir un texto en una secuencia de tokens, que suelen coincidir con una división por palabras.
- Tagger: Se encarga de etiquetar cada token con su categoría gramatical.
- Entity Recognizer: Se encarga de identificar secuencias de palabras que son nombres de cosas, como nombres de personas, de empresas, de ubicaciones, etc.
- Lemmatizer: Encuentra el lexema base de cada palabra.

Herramientas	Características						
	Gratuita	**Offline**	**Español**	**Tokenizer**	**Tagger**	**Entity Recognizer**	**Lemmatizer**
Stanford CoreNLP	No	Si	Si	Si	Si	Si	Si
Apache OpenNLP	Si	Si	Si	Si	Si	Si	Si
Spacy API	Si	Si	Si	Si	Si	Si	Si
GATE	Si	Si	No	Si*	Si	Si	Si*
LingPipe	No	Si	No	Si	Si	Si	Si

Tabla 5.2. Herramientas encontradas dedicadas al NLP

Ahora, se describirán cada una de las herramientas encontradas:

1. **Stanford CoreNLP**
   Stanford CoreNLP es una librería implementada en Java que proporciona un conjunto de herramientas dedicadas al procesamiento del lenguaje. Es una librería de software libre para desarrolladores, sin embargo, está protegida mediante la licencia GNL lo que no permite su comercialización, aunque tienen una licencia especial para poder ser usada como parte de productos comercializables, pero ésta sí es de pago [32].
2. **Apache OpenNLP**
   Esta librería, también implementada en Java, tiene las herramientas básicas de Stanford CoreNLP. Por el contrario, Apache OpenNLP está definida bajo la licencia Apache que si permite comercialización y no obliga al usuario a redistribuir el código derivado de la librería [33].
3. **spaCy API**
   En este caso, se muestra una librería *open-source* implementada en Python. Está orientada a producción de software para la construcción de aplicaciones siendo capaz de procesar un gran volumen de texto y capaz de analizar los sentimientos que se han expresado en él [34].
4. **GATE**
   Es una herramienta *open-source* que utiliza servicios como ANNIE o JAPE [35], desarrollados por la misma organización que GATE, que dotan a GATE de funcionalidades como *Entity Recognizer* y *Matcher* [36]. Sin embargo, estas funcionalidades han sido diseñadas para uso con lengua inglesa. Además, GATE dispone de *plugins* que utilizan otras librerías como Stanford CoreNLP, que están bajo la licencia GNL, por lo que no están disponibles en ámbito comercial. Por lo tanto, se ha indicado en la tabla mediante un asterisco las funcionalidades que puede obtener de diferentes *plugins*.
5. **LingPipe**
   Es un *toolkit* para procesamiento de textos utilizando lingüística computacional. Esta librería está enfocada a tareas como el reconocimiento de entidades (nombres de personas, organizaciones y localizaciones), clasificación de resultados de Twitter en categorías y como corrector ortográfico [37]. Sin embargo, no dispone de modelos

de lenguaje para procesar textos en lengua española [38]. Además, para uso comercial y para que el software producido pueda ser privado, las licencias que se ofrecen son todas de pago [39].

En primer lugar, de las cinco librerías encontradas fueron Stanford CoreNLP y LingPipe las primeras en ser descartadas, ya que no cumplían el requisito de ser gratuitas. Seguidamente, se descartó la herramienta GATE por no haber disponibles modelos de lenguaje para procesado de texto en lengua española y utilizar *plugins* de herramientas que no pueden ser comercializadas.

Por consiguiente, de la criba anterior resultaron las siguientes opciones: spaCy y Apache OpenNLP. Al ser ambas implementadas en diferentes lenguajes a C#, se buscaron puertos o paquetes que dieran la opción de utilizarlas para la implementación de nuestra aplicación. Durante esta búsqueda, no se encontró ninguna forma en la que se pudiera utilizar spaCy en la implementación de nuestra aplicación, ya que, como se ha dicho, está desarrollada en Python.

Entonces, después de excluir a spaCy de entre las librerías potencialmente válidas, se realizó la misma búsqueda anterior con la librería Apache OpenNLP. En este caso sí que se encontraron dos formas para poder utilizar la librería en la aplicación:

- La primera forma para poder utilizar la librería en .NET es mediante el NuGet OpenNLP.NET. Para probarlo, se creó un proyecto .NET Framework que referenciaba a un proyecto .NET Standard para simular la estructura de nuestra aplicación. Sin embargo, al instalar el NuGet en el proyecto .NET Standard (de la misma forma que se haría en la aplicación) dio errores de referencia e indicaba que el paquete no era completamente compatible con el tipo .NET Standard ya que había sido declarado para .NET Framework. Por lo tanto, esta forma de utilizar la librería quedó descartada.
- La segunda forma en la que poder implementar el procesador de consultas con la librería OpenNLP, es utilizando un *port* que traduzca esta librería Java a C#. De esta manera, se encontró la librería SharpNLP [40], con licencia LGNU que permite utilizarla en productos comerciales. Sin embargo, no procesa texto en lengua española. Se encontraron otros *ports*, pero comparten la misma característica. Por lo tanto, tampoco se puede utilizar OpenNLP de esta forma.

Después de este proceso de investigación, actualmente no se han encontrado otras formas en las que utilizar esta librería en .NET Standard, con lenguaje C# y que disponga de procesado de lengua española.

Como consecuencia y con objeto de cumplir con los requisitos se optó por implementar el procesador de consultas de tal forma que no tenga dependencia a internet, se implemente en lenguaje C# en .NET Standard y sin ningún coste. El diseño y desarrollo del procesador se recogen en los capítulos 6 y 7 respectivamente.

En resumen, por todo lo expuesto anteriormente, se seleccionó la librería Speech de Google integrada en Android para el desarrollo del *Speech Recognizer*, dejando como línea de futuro su sustitución por la librería CMU Sphinx o, en el caso de que se modifiquen los requisitos, otra de las recomendadas; y para el desarrollo del procesador, dado que no se ha hallado ninguna librería que cumpliera los requisitos, se implementó de forma ad-hoc para el sistema objetivo.

Capítulo 6

Arquitectura y diseño

Previamente a implementar el sistema, se ha realizado el diseño que se seguirá en su construcción, con el objetivo de cumplir los requisitos especificados. En este capítulo se explicará el diseño arquitectónico y detallado del sistema.

6.1. Diseño de arquitectura

La arquitectura del software sigue un modelo de tres capas, concretamente se ha estructurado según el patrón *Model-View-ViewModel* (MVVM) (ver figura 6.1.), correspondiendo la capa de presentación con la capa *View*, la de datos con la *Model* y la de negocio con la *ViewModel*.

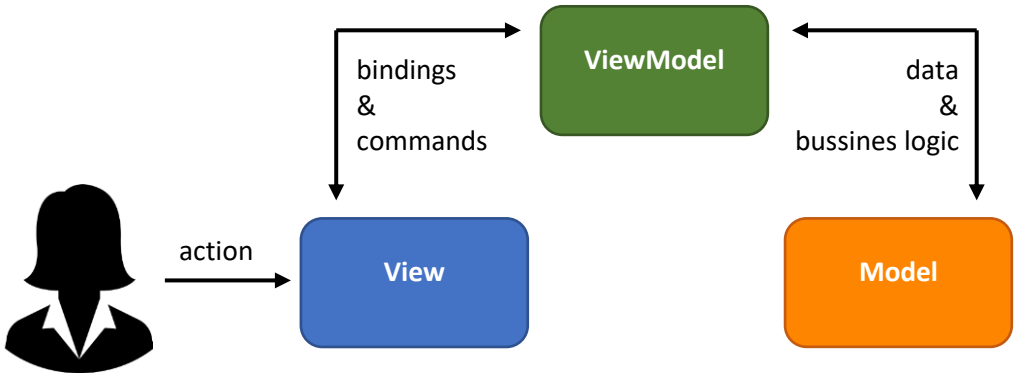


Figura 6.1. Patrón Model-View-ViewModel

El uso del patrón MVVM permite separar la lógica de negocio y de presentación desde su interfaz de usuario (IU), de esta forma, se pueden evitar numerosos problemas de desarrollo y facilita los procesos de pruebas, mantenimiento y evolución del software. Además, este patrón puede ayudar de forma significativa a la reutilización de código y permite a los desarrolladores y diseñadores de IU colaborar más fácilmente en el desarrollo de una aplicación [41] [42] [43].

Aunque puede parecer igual al patrón *Model-View-Presenter*(MVP), se diferencian en que, al realizar los cambios en la capa de presentación, el patrón *MVP* define a la capa *Presenter* como la responsable de modificar directamente *View*, en cambio, el patrón *MVVM* consiste en que desde la capa *ViewModel* se notifique mediante eventos los cambios que debe realizar la *View* [44]. A continuación, se muestra esta diferenciación de forma gráfica (ver figura 6.2.):

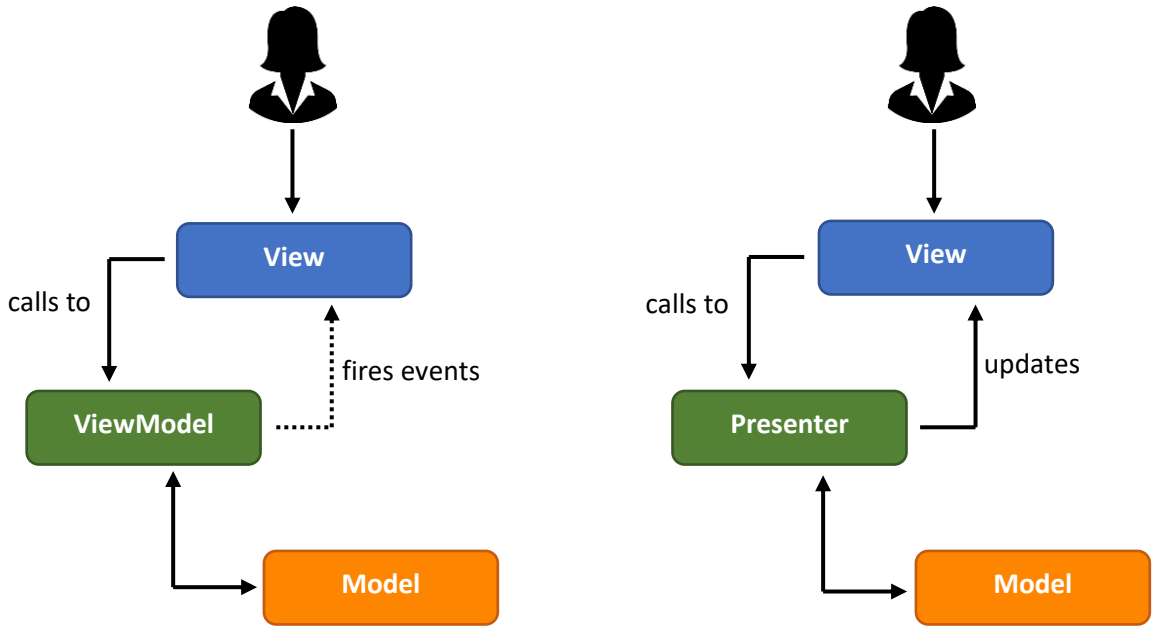


Figura 6.2. Comparación entre el patrón Model-View-ViewModel y el Model-View-Presenter

A continuación, siguiendo la estructura del patrón utilizado, aparecen representados los componentes que forman la arquitectura del sistema. La conjunción de estos componentes con las interfaces necesarias para proveer los puntos de interacción entre ellos conforma el diagrama que modela la arquitectura del sistema a nivel de componentes (figura 6.3.).

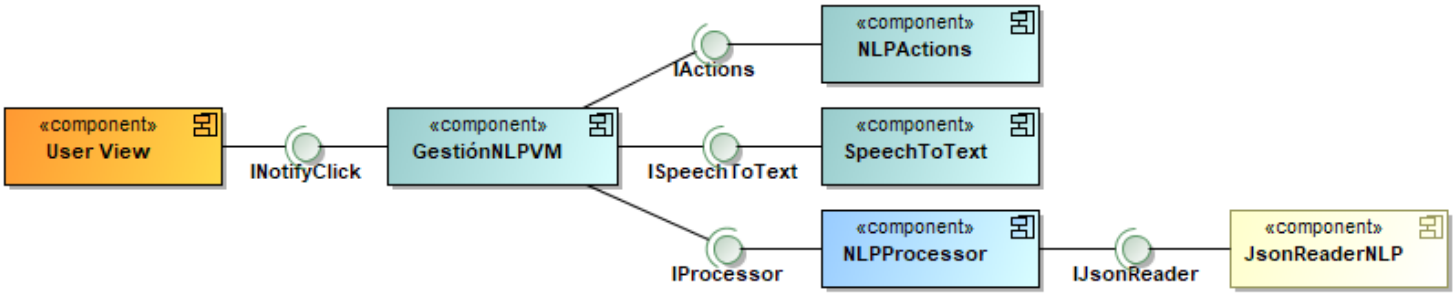


Figura 6.3. Diagrama de componentes

En el diagrama de la figura 6.3, en la capa *View,* se puede encontrar al componente denominado *User View* que encapsulará la definición de los componentes que conforman la interfaz gráfica de la aplicación que será mostrada al usuario para que éste interactúe con ella.

Así mismo, se han determinado hasta cuatro componentes correspondientes a la capa *ViewModel*, siguiendo la división de acuerdo con el patrón utilizado y con el objetivo de satisfacer los requisitos implementando un sistema lo más modularizado posible.

De acuerdo con el patrón MVVM, el componente *GestiónNLPVM* proporciona al componente *User View* los métodos definidos en la interfaz *INotifyClick* necesarios para establecer una conexión entre la vista y los componentes propios de la lógica de negocio: NLPActions, SpeechToText y NLPProcessor. De este modo, éstos tres últimos componentes se encargan de proveer a *GetiónNLPVM* de las interfaces que implementan: *IActions*, *ISpeechToText* e *IProcessor*, respectivamente.

Además, el componente *JsonReaderNLP* proporciona la interfaz *IJsonReader*, requerida por el componente *NLPProcessor* para acceder a los datos necesarios para el procesamiento.

Después de explicar cómo se conectan entre sí los distintos componentes con las interfaces, a continuación, se exponen las interfaces mencionadas:

En primer lugar, se muestra la interfaz INotifyClick (ver figura 6.4.), la cual no está literalmente implementada por el componente *GestiónNLPVM*. Esta interfaz se ha especificado con el objetivo de definir la conexión entre el componente *UserView* y *GestiónNLPVM.* Como se ha dicho anteriormente, en el patrón utilizado la capa *View* y *ViewModel* interactúa mediante eventos y notificaciones.

Figura 6.4. Interfaz INotifyClick

Esta interacción entre el *front-end* y el *back-end* de la aplicación se realiza cuando el usuario pulse el botón de grabar de la interfaz, que activa el método `ButtonGrabClicked(...)` implementado en el componente *GestiónNLPVM*. Este método se encarga de iniciar la función que realiza el *speech-to-text*, recoger la transcripción del audio y de notificar el cambio que debe realizar la *View* en el campo de texto, dónde debe mostrar el texto transcrito.

De esta misma forma interactúan los componentes cuando el usuario pulsa el botón de grabar. Sin embargo, en este caso el método que se activa es `ButtonProcClicked(...)`. Éste recoge el texto del campo de texto de la interfaz e inicia la acción de procesamiento, y cuando ésta termina, notifica a la *View* que debe mostrar el resultado del procesamiento en contenedor destinado a la muestra de resultados.

Por otro lado, se muestra la interfaz IAction (ver figura 6.5.) de la capa ViewModel. Esta interfaz provee al componente GestionNLPVM de los métodos que son el objetivo final del sistema. El componente GestiónNLPVM ejecuta una de las funciones de la interfaz según el resultado recibido del procesamiento. Estas funciones y su funcionamiento (tanto el actual, como el que se espera en un futuro) se ha definido en el capítulo 3 relacionado con los requisitos del sistema.

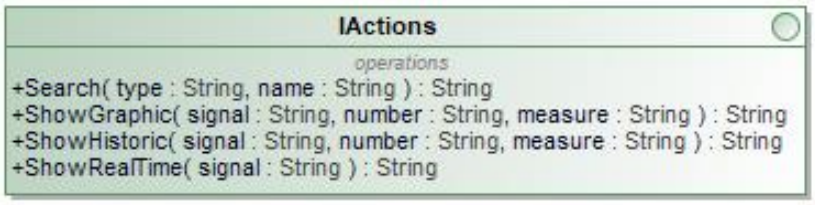


Figura 6.5. Interfaz IActions

La siguiente interfaz se trata de ISpeechToTex (ver figura 6.6) y es la que define el método encargado de realizar la tarea de transcripción *speech-to-text*. El funcionamiento de este proceso se explica en el apartado dedicado a su desarrollo (apartado 7.3.).

Figura 6.6. Interfaz ISpeechToText

También se encuentra IJsonReader (ver figura 6.7.), la interfaz implementada por el componente de la capa *Model*, *JsonReaderNLP*. Los métodos de esta interfaz se encargar de leer los distintos diccionarios de datos que se necesitan para el procesamiento porque, tal y como se ha comentado en el apartado dedicado a la búsqueda y selección de herramientas (ver capítulo 5), nuestro procesador no va a utilizar ninguna herramienta externa y, por ello, se han utilizado los siguientes diccionarios:

- Diccionario de stop words
  En este diccionario encontraremos todas las palabras, como determinantes, artículos, pronombres y preposiciones, que carecen de significado propio. Este diccionario se utilizará para eliminar todo este tipo de palabras de la consulta del usuario.
- Diccionario de verbos sin significado o empty verbs
  El contenido de este diccionario se utiliza para la localización y eliminación de verbos que no son significativos para la consulta.
- Diccionario de tipos de documentos
  Aquí se recogen los distintos tipos de documentos que soporta la aplicación, los cuales ya fueron nombrados en el capítulo de análisis (ver capítulo 3).
- Diccionario de señales
  El diccionario de señales guarda en su interior las señales propias de cada empresa a la que la que CIC da el servicio de IDbox. Cada una de las señales del diccionario se compone del identificador y la descripción de la señal (ver capítulo 3).

Figura 6.7. Interfaz IJsonReader

Y, por último, se encuentra la interfaz IProcessor (ver figura 6.8.) que es implementada por el componente *NLPProcessor* y que contiene todos los métodos que contribuyen al procesamiento de la consulta del usuario. A continuación, se listan los métodos y sus funciones:

- `LoadResources (...)`
  Este método crea una dependencia entre el componente *NLPProcessor* e *JsonReaderNLP*, ya que el primero realiza llamadas a las funciones de la interfaz IJsonReader, implementada por el segundo. Este método se encarga de cargar los datos de los diccionarios anteriores en distintas estructuras de datos para, posteriormente ser utilizadas en el procesamiento.

- `Tokenizer (...)`
  Este método, como su nombre indica, se encarga de Tokenizar el texto recogido de la transcripción *speech-to-text.*
- `RemoveVerbs (...)`
  En este paso, se eliminan de la lista que se recibe como parámetro los verbos que no aporten ningún significado. Para ellos se utilizan los EmptyVerbs leídos del diccionario correspondiente.
- `RemoveStopWords (...)`
  Con esta función se eliminan las *stop words* de la lista de palabras que se recibe como parámetro. Para ello son leídos los datos del diccionario de *stop words* necesarios para realizar el filtrado de palabras.
- `AdaptListToAction (...)`
  Este método se encarga de adaptar la lista de palabras resultante del filtrado de los anteriores métodos que se recibe como parámetro, a la estructura que es requerida para identificar la función que se debe ejecutar. Para ello, si se identifica la descripción de alguna señal del sistema, mediante la lectura del diccionario de señales, ésta se sustituye por su identificador.
- `ChooseAction (...)`
  Este es el último método que se realiza en el procesado, aquí se concreta qué función se debe ejecutar y con qué parámetros se ejecutará. Tanto el *string* que identifique la función, como los parámetros, se retornarán en una lista al componente *GestionNLPVM* para que accione el método adecuado de la interfaz IActions. Además, en este paso se utiliza el diccionario de tipos de documentos para identificar si la consulta contiene alguno.

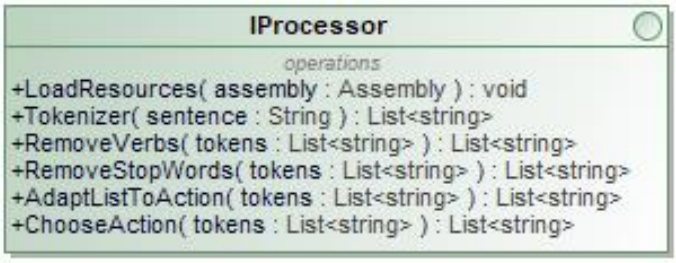


Figura 6.8. Interfaz IProcessor

Como la aplicación se ha desarrollado con el paradigma de reutilizar el máximo código posible en la implementación de esta aplicación para distintas plataformas, se ha utilizado una librería de clases .NET Standard donde se recogen todas las clases, interfaces y recursos que no sean específicos de cada plataforma.

6.2. Diseño detallado

En este apartado se muestra el diagrama de despliegue (ver figura 6.9.) de nuestro sistema. Como se puede observar, el sistema implementado se ejecuta en dispositivos móviles con sistema operativo Android.

Como se especificaba en los requisitos no funcionales, el procesamiento de la consulta debe realizarse en el propio dispositivo. Pero, la transcripción Speech-to-Text actualmente utiliza la librería Speech de Google integrada en Android, que utiliza el server de Google para la transcripción.

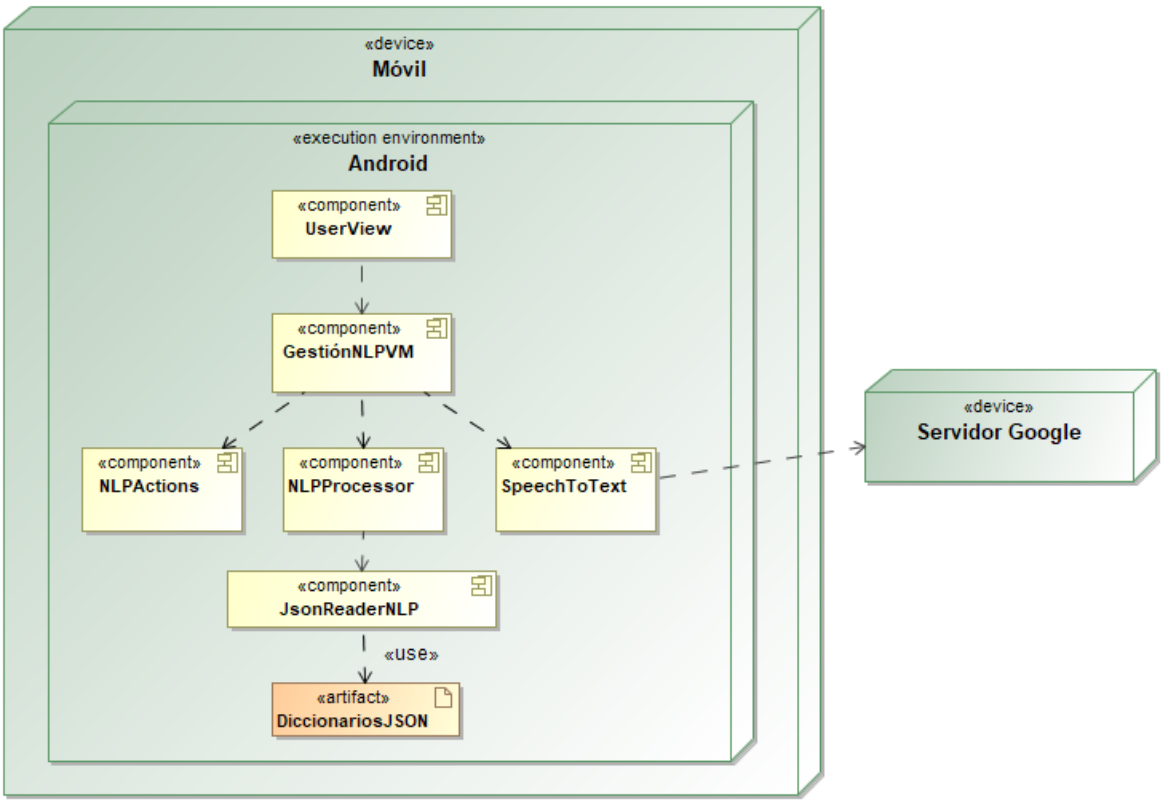


Figura 6.9. Diagrama de despliegue

Capítulo 7

Desarrollo de cada módulo

Después de exponer en qué consiste el Procesamiento del Lenguaje Natural, las herramientas que se van a utilizar, los requisitos y la arquitectura del sistema en este apartado, se explicará el proceso seguido para la implementación de cada uno de los tres módulos que se pueden distinguir en el sistema: la interfaz gráfica, el reconocimiento de voz a texto y el procesamiento de la consulta.

Antes de comenzar, se muestra la estructura de la solución en la figura 7.1. en la que se puede ver un proyecto NLP y otro proyecto NLP.Android. El primero es la librería de clases .NET Standard y es dónde se incluye todo aquello que se considere común para distintas plataformas, todo el código reutilizable; el segundo se trata del proyecto donde se implementa el código de la aplicación específico para la plataforma Android.

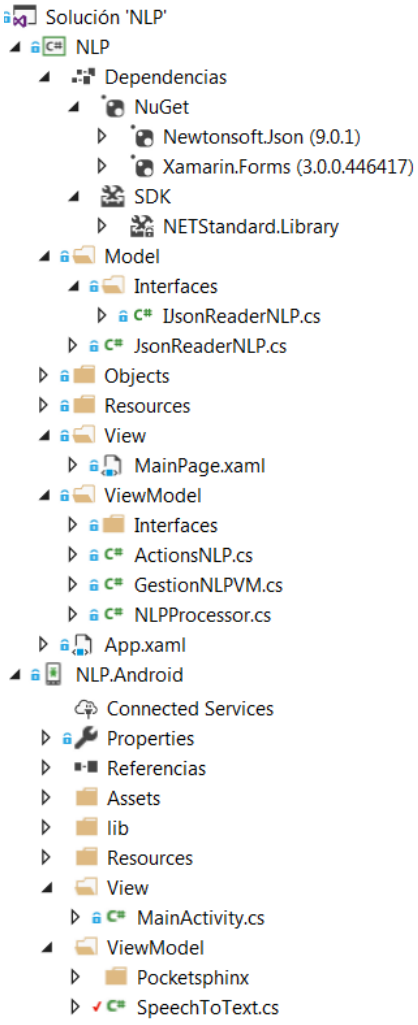


Figura 7.1. Estructura de la solución

7.1. Gestión de la configuración

Durante la realización del proyecto se ha observado la importancia de la gestión de la configuración en cualquier entidad dedicada a desarrollar software. Capacita al desarrollador de una mejor organización y mantenimiento, permitiendo controlar los cambios y las distintas versiones que se van produciendo en el producto, pudiendo volver a versiones anteriores para volver a usarlas o incluso para revertir cambios, facilitando el desarrollo simultáneo del software. En nuestro caso se ha utilizado Git para esta función y GitLab como repositorio, ya que la empresa está en proceso de cambio de la herramienta Subversion a la herramienta Git. Además, en Visual Studio se ha podido realizar directamente las preparaciones de los cambios y la inserción de éstos en el repositorio remoto (ver figura 7.2.).

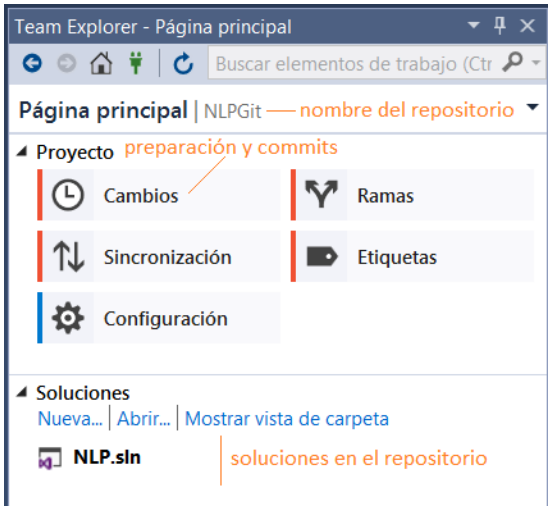


Figura 7.2. Gestión del repositorio Git desde Visual Studio

7.2. Interfaz gráfica

Antes de comenzar con la implementación de la aplicación se realizó un *mock-up* de la pantalla principal de la prueba de concepto. Además, se ha diseñado de tal forma que haya un paso intermedio entre la captura del audio de la consulta y su transcripción a texto y el procesamiento de éste, para que se pueda observar la consulta realizada y el resultado obtenido.

En el diseño se recoge tanto la pantalla inicial como los cambios que presenta mientras se interactúa con la interfaz. A continuación, se exponen los bocetos realizados:

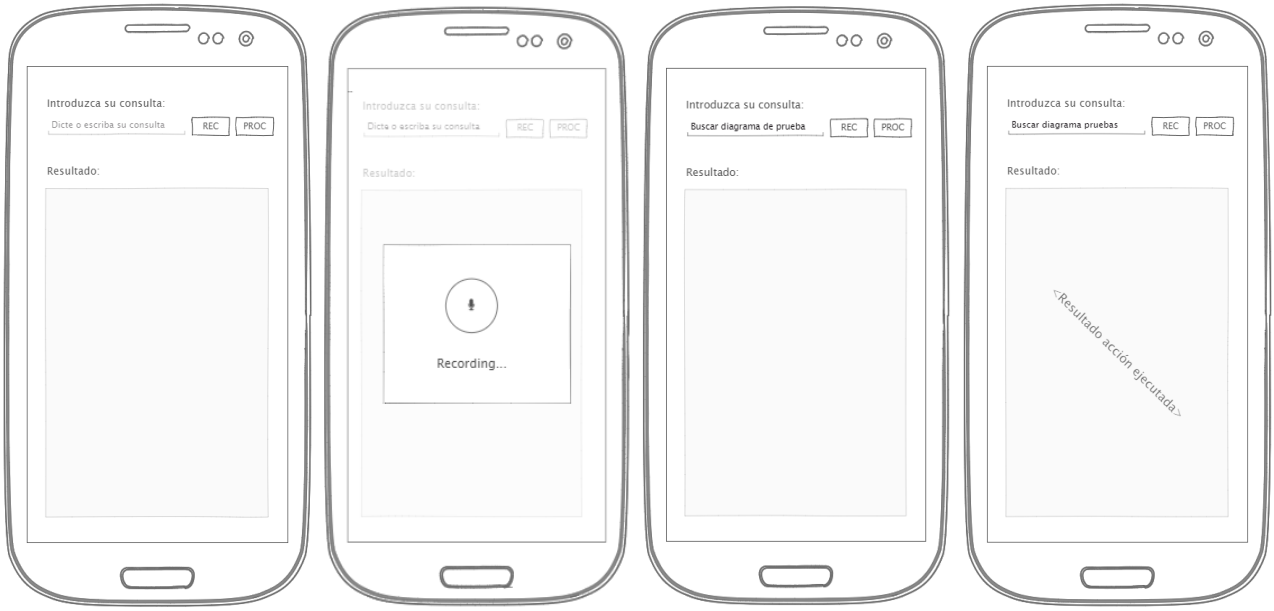


Figura 7.3. Mockups de la aplicación

Todo este diseño se ha implementado programáticamente y se ha recogido en la clase `MainPage.xaml.cs`. La interfaz se ha desarrollado de forma iterativa añadiendo, primeramente, los componentes gráficos de la UI (contenedores, botones, etiquetas, campos de texto…) y, posteriormente, la parte funcional con la que se activan las funciones de la aplicación mediante la interacción con la interfaz.

A continuación, se muestra la interfaz implementada y los estados que presenta dependiendo del estado de la consulta del usuario:

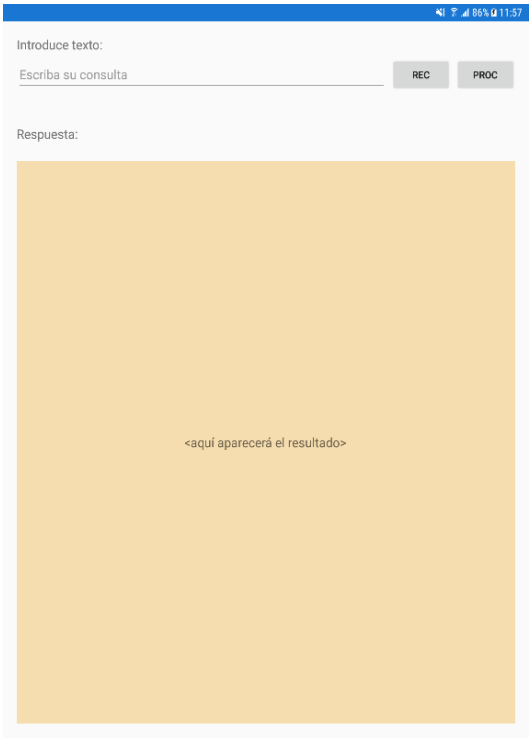


Figura 7.4. Página principal de la aplicación

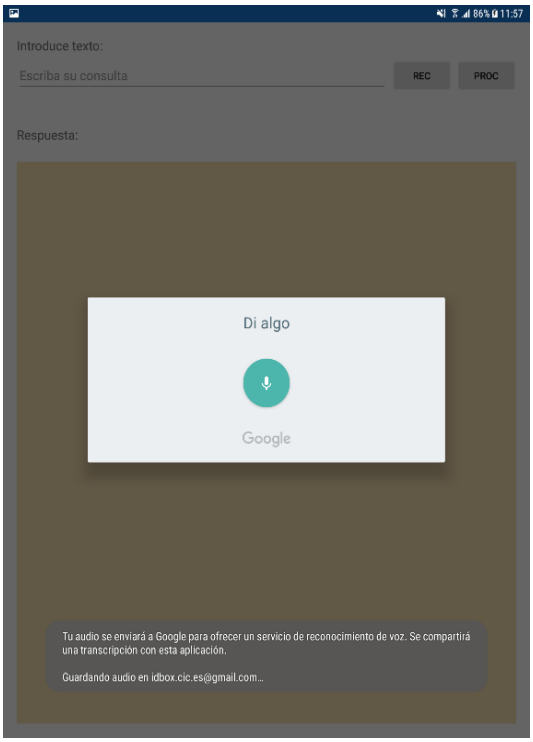


Figura 7.5. Pop-up de la tarea de grabación

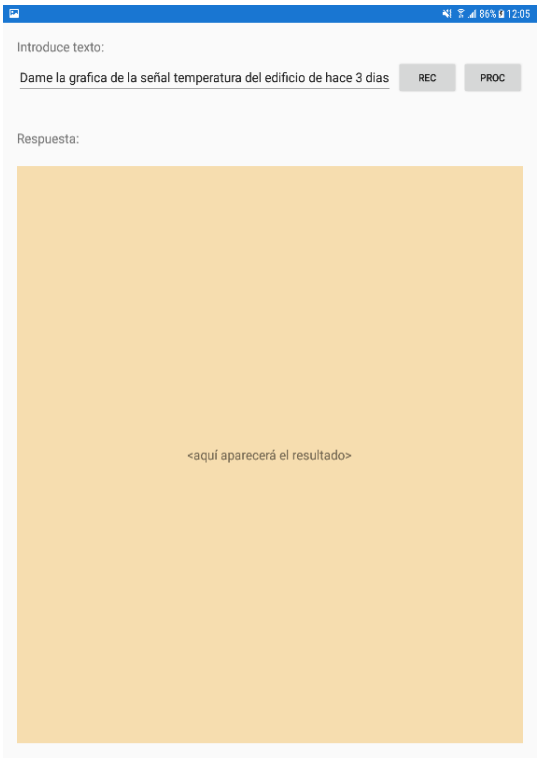


Figura 7.6. Página principal después de grabar la consulta del usuario

Figura 7.7. Página principal después de procesar la consulta del usuario

Como se puede observar en la figura 7.4, la aplicación consta de una pantalla principal dividida en dos partes: la parte superior, donde se realiza la petición, y la parte inferior, donde se muestra la respuesta.

La parte superior contiene un campo de texto donde aparecerá la consulta del usuario en forma de texto, ya sea porque el usuario la haya introducido a mano o porque se haya realizado la transcripción de audio a texto tras grabar la consulta dictada del usuario; un botón de grabar, el cual activa la tarea de Speech-to-Text que graba al usuario y muestra el audio recogido en forma de texto en el campo de texto; y un botón de procesamiento, que procesa el texto del campo de texto y devuelve un resultado.

La parte inferior se compone de un campo donde se muestra el resultado del procesamiento.

En el diseño de la aplicación se ha tenido en cuenta el uso de colores que favorezcan a la interpretación de la marca del producto porque se trata de una prueba de concepto. Por otra parte, se ha diseñado con el objetivo de ser intuitiva, clara y fácil. Por esa razón, se han añadido etiquetas descriptivas de cada parte para separar la parte donde se realiza la consulta y la parte donde se muestra el resultado.

Al iniciar la aplicación, será la clase `MainActivity.cs` la encargada de cargar la interfaz con el diseño establecido en la clase `MainPage.xaml.cs`.

La parte funcional de la aplicación se realiza entre las clases `MainPage.xaml.cs` y la clase *GestiónNLPVM* como ya se explicó en el apartado dedicado al diseño al describir la interfaz INotifyClick.

7.3. Speech-to-Text

El módulo Speech-to-Text se encarga de grabar la consulta del usuario dictada por voz y convertirla en texto. Para que pueda realizarse esta funcionalidad se ha implementado el componente *SpeechToText*, el cual implementa la interfaz ISpeechToText y con ello al único método definido en ella, `StartListening() : Task<String>`.

El método `StartListening`, que ha sido implementado mediante el uso de la librería Speech de Android, realiza una petición para realizar el reconocimiento de voz y, una vez aceptada, la aplicación muestra un *pop-up* que marca el inicio de la tarea de grabación. Aunque el método se haya implementado en una clase diferente a la `MainActivity.cs`, el *pop-up* que emerge se visualiza en la interfaz, ya que en la implementación del método se ha utilizado el contexto de la actividad principal. Y, por último, cuando la tarea de reconocimiento termina, se retorna el resultado de la transcripción.

Y ahora la pregunta es, ¿cómo sabe la aplicación que cuando se pulse el botón de grabar se debe iniciar esta tarea? Como se ha dicho anteriormente, la vista y la capa de negocio interactúan mediante la interfaz INotifyClick que es implementada por la clase *GestiónNLPVM*. Ésta última es la que sirve de conexión entre la vista y lógica de negocio propiamente dicha, por lo tanto, en la implementación de su método `ButtonGrabClicked` realizará una llamada al método `StartListening`.

7.4. Procesamiento de la consulta

Este módulo es el que se encarga de procesar la consulta del usuario, determinar que función tiene que ser ejecutada y ejecutarla. Para desarrollar este módulo se han utilizado los siguientes componentes:

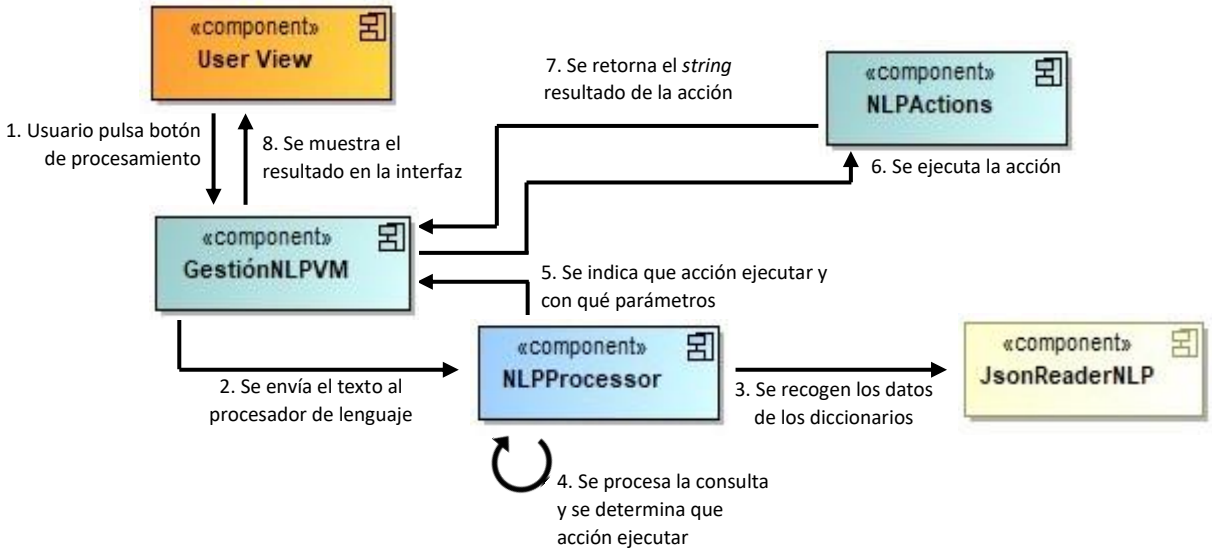


Figura 7.8. Interacción entre los componentes que conforman el módulo dedicado al procesamiento de la consulta

El componente *GestiónNLPVM* sirve como conexión entre la lógica y la interfaz, sin embargo, también se encarga de accionar la función de *NLPActions* que haya sido indicada por el resultado del procesamiento de la consulta realizado por NLPProcessor.

A continuación, se explica la implementación realizada para el componente NLPProcessor, ya que la interacción entre los componentes *UserView* y *GestiónNLPVM*, como ya se ha explicado en otros apartados (capítulo 6), y la interacción entre el componente *GestiónNLPVM* y el componente *NLPActions* se realiza con llamadas del primero a los métodos triviales (estos métodos solo retornan una cadena de texto) del segundo.

El componente *NLPProcessor* implementa los métodos principales que se pueden identificar en la figura 7.9. Como se puede observar, entre estos métodos se encuentra `ProcessQuery`, el cual no se había nombrado anteriormente. Esto es porque este método no es propio de la interfaz IProcessor. Su implementación consiste en llamar a los otros métodos que aparecen y el objetivo de su definición es hacer más fácil la llamada a los métodos de este componente desde *GestiónNLPVM*. El procesamiento se realiza con el orden de llamadas de la figura 7.10.

```
public void LoadResources(Assembly assembly)...
public List<string> ProcessQuery(string sentence, Assembly assembly)...
public List<string> Tokenizer(string sentence)...
public List<string> RemoveStopWords(List<string> tokens)...
public List<string> RemoveVerbs(List<string> tokens)...
public List<string> AdaptListToAction(List<string> tokens)...
public List<string> ChooseAction(List<string> tokens)...
```

Figura 7.9. Métodos del componente NLPProcessor

```
public List<string> ProcessQuery(string sentence, Assembly assembly)
{
    List<string> result = new List<string>();

    LoadResources(assembly);
    result = Tokenizer(sentence);
    result = RemoveVerbs(result);
    result = RemoveStopWords(result);
    result = AdaptListToAction(result);
    result = ChooseAction(result);
    return result;
}
```

Figura 7.10. Orden de llamada de los métodos para realizar el procesamiento

Por otro lado, están los métodos que se definieron en la interfaz IProcessor. El primer método que se ejecuta durante el procesado es `LoadResources` que, como ya se ha explicado, se encarga hacer llamadas al componente *JsonReaderNLP*, el cual lee los datos de los diccionarios. Sin embargo, todavía no se ha mencionado como se guardan estos datos leídos.

Los datos a leer son: las señales de cada empresa, los *empty verbs* definidos, las *stop words* y los tipos de documentos. Todos ellos se utilizan como parte del procesado para acabar determinando la acción que se debe ejecutar respecto a la consulta que realiza el usuario. Por lo tanto, de acuerdo con su uso se han decidido utilizar unas estructuras u otras:

- **HashSet<T>**

  Este conjunto se ha utilizado para guardar las *stop words,* por lo que se utiliza como HashSet<string>. Se ha utilizado esta estructura porque para hacer el filtrado de

stop words no hace falta recorrerla secuencialmente. Los métodos utilizados sobre esta estructura son `Contains` y `Add`, cuyas complejidades son O(1) [45] [46].

- **LinkedList<T>**

  Por otro lado, el resto de las estructuras utilizadas (para *empty verbs*, señales y tipos de documentos) han sido del tipo LinkedList<T>, siendo T el tipo EmptyVerb, Signal o Doctype en los distintos casos. Esta elección se ha tomado porque son estructuras que se van a recorrer secuencialmente ya que hay que acceder a cada uno de sus elementos, por lo que la complejidad será O(n) en cualquiera de los casos y aunque la estructura sea ordenada. Por otro lado, entre la List<T> y la LinkedList<T>, se ha descartado la List<T> dado que está basada en array y puede llegar a tener una complejidad de O(n). En cambio, la inserción de datos en LinkedList<T> es siempre de O(1) [47].

El segundo método que se ejecuta es `Tokenizer`. Este método se encarga de normalizar el texto recibido poniéndolo en minúsculas y sin signos de acentuación y, posteriormente, tokeniza el texto recibido.

Siguiendo el orden de procesamiento, el siguiente método que se acciona es `RemoveVerbs`, el cual se encarga de eliminar los verbos, que se encuentran entre los tokens de la consulta, que no nos aportan ningún significado durante el procesado.

Para poder localizar y eliminar los *empty verbs*, se recorren tanto la lista de éstos, como la lista de los tokens de la consulta, para encontrar los tokens cuyo inicio coincidan con algún lexema de la lista de *empty verbs*. Cuando coinciden con alguno de los lexemas se comprueba, por cada desinencia del lexema seleccionado, si el token coincide con la cadena 'lexema + desinencia'. Si hay alguna coincidencia, quiere decir que el token es un *empty verb* y tiene que ser eliminado.

Para que este tipo de filtrado de verbos fuese eficaz, se han estudiado las distintas formas en las que se puede realizar la consulta y, después, se han buscado sinónimos de las formas verbales encontradas y se han incorporado al diccionario.

Después de eliminar los verbos no significativos para la consulta, se realiza la eliminación de *stop words* con el método `RemoveStopWords`. Además, hay una parte de este método que contempla el caso especial de la siguiente consulta:

"Buscar la señal de temperatura edificio"

En otros casos, la palabra 'señal' no se considera importante porque lo que interesa es la descripción o el identificador de la señal. Sin embargo, en la anterior consulta sí es una palabra significativa, ya que se refiere al tipo de documento que el usuario ha requerido.

Los dos últimos métodos que quedan por realizar son `AdaptListToAction` y `ChooseAction`. El primero se encarga de adaptar la lista de tokens resultante de los anteriores métodos explicados para que pueda de terminarse en el segundo, qué función hay que ejecutar y cuáles son los valores de sus parámetros.

En esta adaptación de la lista, se sustituye la descripción de la señal por su identificador. Esto se realiza recorriendo secuencialmente la lista de señales y, por cada señal, se extrae su descripción y se iguala a la forma de la lista de tokens (sin *stop words*, en minúsculas, sin signos de

acentuación y tokenizada). Después, se comprueba si la cadena formada por la concatenación de la lista de tokens de la consulta contiene la cadena formada por la concatenación de los tokens de la descripción de la señal. Si se obtiene alguna coincidencia, la descripción es sustituida por el identificador. Si no, la cadena formada por tokens que corresponden con la descripción de la señal, que en ese caso no estaría registrada, se pasaría como el parámetro 'señal'. Por ejemplo, para explicar el caso anterior:

- Si tenemos la consulta: *"Quiero el tiempo real de la señal temperatura del agua"*
- Se obtiene la lista de tokens: *tiempo*, *real*, *temperatura*, *agua*.
- Si en el diccionario de señales, no hay ninguna señal con una descripción que sin *stop words,* ni signos de acentuación*,* en minúsculas y tokenizada coincida con "temperatura agua", la función final al ejecutarse resultaría así:

  "ShowRealTime (temperatura agua)"

- Sin embargo, si se obtiene una coincidencia de la descripción con, por ejemplo, la señal [Id: CICA43, Descripción: temperatura del agua], el resultado de la función ejecutada sería el siguiente:

  "ShowRealTime (CICA43)"

Finalmente, después de realizar los pasos anteriores, el componente *GestionNLPVM* se encarga de ejecutar la acción determinada en el procesado y de notificar a la *View* el resultado que debe mostrar, esto es el comando a ejecutar.

Capítulo 8

Pruebas y validación

En este capítulo, se presenta el plan de pruebas seguido para comprobar el correcto funcionamiento de sistema.

Aunque las pruebas se muestran por orden de ejecución, los planes se han definido siguiendo el orden que se indica a continuación: Pruebas de Aceptación, Pruebas de Integración y Pruebas Unitarias.

Además, las pruebas que aparecen se han realizado periódicamente durante el desarrollo del proyecto.

8.1. Pruebas unitarias

Las pruebas unitarias se han desarrollado de tal forma que se verifique el correcto funcionamiento de cada uno de los métodos de la interfaz que implementan los componentes *NLPProcessor* y *JsonReaderNLP*. Los métodos del componente *ActionsNLP* se consideran triviales por lo que se comprobará su correcto funcionamiento a la hora de integrarlo con otros componentes.

Por otra parte, el correcto funcionamiento de los componentes *GestionNLPVM, SpeechToText* y *User View* son probados directamente en las pruebas de aceptación.

- **Pruebas unitarias del componente JsonReaderNLP**

Como se ha dicho con anterioridad, este componente implementa la interfaz IJsonReader, por lo tanto, consta de los métodos que se muestran en la tabla 8.1.

Nombre método	Retorno	Parámetros
ReadStopWords	HashSet<string>	stopwordsJson : StreamReader
ReadEmptyVerbs	LinkedList<EmptyVerb>	emptyverbsJson : StreamReader
ReadSignals	LinkedList<Signals>	signalsJson : StreamReader
ReadDocTypes	LinkedList<DocType>	doctypesJson : StreamReader

Tabla 8.1. Métodos de la interfaz IJsonReader

Para la realización de pruebas se requerirán cuatro archivos JSON que simulen el contenido de cada diccionario. Por lo tanto, a continuación, se listarán las pruebas unitarias definidas para los

métodos dados, se mostrará el recurso utilizado y se explicará el resultado que se espera obtener.

PU01.- En esta prueba se comprobará el correcto funcionamiento del método `ReadStopWords` al cuál se le pasa un *StreamReader* para leer el siguiente diccionario de *stop words* de la figura 8.1.

Tras ser este método ejecutado se espera obtener un `HashSet<string>` que contenga el valor del campo Word de cada instancia de stop Word leída del JSON. En el caso de que el *StreamReader* sea nulo se retornará el conjunto vacío.

```
{
    "stopwords":[{
            "type":"articulo",
            "word":"el"
        },
        {
            "type":"articulo",
            "word":"los"
        },
        {
            "type":"articulo",
            "word":"la"
        },
        {
            "type":"articulo",
            "word":"las"
        }]
}
```

Resultado esperado `HashSet<string>`:

- el
- los
- la
- las

Figura 8.1. Json de stop words para la prueba PU01

PU02.- En esta prueba se comprobará el correcto funcionamiento del método `ReadEmptyVerbs` al cuál se le pasa un *StreamReader* para leer el siguiente diccionario de *empty verbs* de la figura 8.2.

Tras ser este método ejecutado se espera obtener una `LinkedList<EmptyVerb>` que recoja los *empty verbs,* sin que se repitan los lexemas y con su lista de desinencias. En el caso de que el *StreamReader* sea nulo error retornará lista de *empty verbs* vacía.

```
{
    "emptyverbs":[{
            "lexeme":"consult",
            "desinence":"a"
        },
        {
            "lexeme":"consult",
            "desinence":"ar"
        },
        {
            "lexeme":"busc",
            "desinence":"ar"
        },
        {
            "lexeme":"busc",
            "desinence":"ame"
        }]
}
```

Resultado esperado `LinkedList<EmptyVerb>`:

- EmptyVerb1:
  Lexeme: consult.
  Desinences: a, ar.
- EmptyVerb2:
  Lexeme: busc.
  Desinences: ar, ame.

Figura 8.2. Json de empty verbs para la prueba PU02

PU03.- En esta prueba se comprobará el correcto funcionamiento del método `ReadSignals` al cuál se le pasa un *StreamReader* para leer el siguiente diccionario de *signals* de la figura 8.3.

Tras ser este método ejecutado se espera obtener una `LinkedList<Signal>` que recoja las señales, con sus identificadores y descripciones. Si el *StreamReader* es nulo, se retornará una lista vacía de señales.

```
{
    "signals":[{
            "id":"CICA1",
            "description":"Temperatura del edificio"
        },
        {
            "id":"CICA2",
            "description":"Punto de rocío a 10 metros"
        },
        {
            "id":"CICA3",
            "description":"Sigma de la dirección del viento"
        },
        {
            "id":"CICA4",
            "description":"Variación del viento en la torre de reserva"
        }]
}
```

Figura 8.3. Json de señales para la prueba PU03

Resultado esperado `LinkedList<Signal>:`

- Signal1:
  Id: CICA1.
  Description: Temperatura del edificio.
- Signal2:
  Id: CICA2.
  Description: Punto de rocío a 10 metros.
- Signal3:
  Id: CICA3.
  Description: Sigma de la dirección del viento.
- Signal4:
  Id: CICA4.
  Description: Variación del viento en la torre de reserva

PU04.- En esta prueba se comprobará el correcto funcionamiento del método `ReadDocTypes` al cuál se le pasa un *StreamReader* para leer el siguiente diccionario de tipos de documentos de la figura 8.4.

Tras ser este método ejecutado se espera obtener una LinkedList<DocType> que recoja los tipos de documentos, con sus campos *type* y *common.* Si el *StreamReader* es nulo, se retornará una lista vacía de tipos.

```
{
    "doctypes":[{
            "type":"Grafica",
            "common":"grafica"
        },
        {
            "type":"Sinoptico",
            "common":"sinoptico"
        },
        {
            "type":"Diagrama",
            "common":"diagrama"
        },
        {
            "type":"Mapa",
            "common":"mapa"
        }]
}
```

Figura 8.4. Json de tipos de documentos para la prueba PU04

Resultado esperado `LinkedList<DocType>:`

- DocType1:
  Type: Grafica.
  Common: grafica.
- DocType2:
  Type: Sinoptico.
  Common: sinoptico.
- DocType3:
  Type: Diagrama.
  Common: diagrama.
- DocType4:
  Type: Mapa.
  Common: mapa

- **Pruebas unitarias del componente NLPProcessor**

A continuación, se van a especificar las pruebas unitarias relacionadas con el componente NLPProcessor, el cual implementa a la interfaz IProcessor, que como se ha dicho anteriormente define los métodos siguientes:

Nombre método	Retorno	Parámetros
LoadResources	void	assembly : Assembly
Tokenizer	List<string>	sentence : string
RemoveVerbs	List<string>	tokens : List<string>
RemoveStopWords	List<string>	tokens : List<string>
ChooseAction	List<string>	tokens : List<string>

Tabla 8.2. Métodos de la interfaz IProcessor

En este apartado, se definirán las pruebas a realizar para comprobar el correcto funcionamiento de los métodos de la tabla anterior (ver tabla 8.2), a excepción del método `LoadResources` el cuál solo hace llamadas a los métodos del componente *JsonReaderNLP*, y el comportamiento de esos métodos ya ha sido verificado.

PU05.- En esta prueba se va a comprobar el funcionamiento del método `Tokenizer.` Este método recibirá una sentencia y el escenario de éxito esperado es una lista que contenga las palabras de la anterior sentencia separadas, en minúsculas y sin puntos de acentuación.

Sentencia introducida:

"Consultar el histórico de la señal CICA1 de 3 semanas"

Escenario de éxito:

List<string> = {"consultar", "el", "histórico", "de", "la", "señal", "cica1"}

En el caso de que la cadena de texto introducida esté vacía o sea nula, se retornará una lista vacía.

PU06.- En esta prueba se va a verificar que el método `RemoveVerbs` funciona correctamente. El método recibirá una lista de palabras y, de estas palabras se deberán eliminar los verbos que coincidan con algún *empty verb* de la lista `LinkedList<EmptyVerb>.`

Como en las pruebas unitarias se requiere probar cada componente de forma independiente y sin hacer uso de otros, la lista de *empty verbs* se rellenará en forma de prueba sin utilizar los diccionarios en formato JSON para no depender del componente *JsonReaderNLP*.

Por lo tanto, el método recibirá una lista de palabras que filtrará eliminando todos los verbos que coincidan con los de la lista de *empty verbs*.

Lista de *empty verbs*:

LinkedList<EmptyVerb> = {("consult", {"ar", "a"}), ("busc", {"ar", "ame"})}

Lista de palabras:

List<string> = {"consultar", "el", "histórico", "de", "la", "señal", "cica1"}

Escenario de éxito:

List<string> = {"el", "histórico", "de", "la", "señal", "cica1"}

PU07.- En esta prueba se va a verificar que el método `RemoveStopWords` funciona correctamente. El método recibirá una lista de palabras y, de estas palabras, se deberán eliminar las palabras que coincidan con alguna *stop word* del conjunto HashSet`<StopWord>.`

Al igual que en la prueba PU06, la lista de *stop words* se rellenará en forma de prueba sin utilizar los diccionarios en formato JSON para no depender del componente *JsonReaderNLP*.

Conjunto de *stop words*:

HashSet<StopWord> = {"el", "los", "la", "las", "de"}

Lista de palabras:

List<string> = {"el", "historico", "de", "cica1"}

Escenario de éxito:

List<string> = {"historico", "cica1"}

PU08.- El método cuyo funcionamiento se va a probar es `ChooseAction`. El método recibirá una lista de palabras de las cuales determinará la acción que se debe ejecutar y los parámetros de la acción. Para el análisis de la palabra de la lista, será necesario el uso de una lista que contenga los tipos de documentos del sistema. Por lo tanto, como en las pruebas anteriores, la lista de tipos se rellenará en forma de simule la lista con el contenido del diccionario y así no depender del componente *JsonReaderNLP*.

Lista de tipos de documentos:

```
LinkedList<DocType> = {("Diagrama", "diagrama"), ("Sinoptico", "sinoptico")}
```

- **Caso selección de Search <type> <name> con *type* y *name* no nulos.**

  Lista de palabras:

  ```
  List<string> = {"buscar", "diagrama", "pruebas_cica1"}
  ```

  Escenario de éxito:

  ```
  List<string> = {"search", "diagrama", "pruebas_cica1"}
  ```

- **Caso selección de Search <type> <name> con *type* nulo y *name* no nulo.**

  Lista de palabras:

  ```
  List<string> = {"search", "pruebas", "temperatura"}
  ```

  Escenario de éxito:

  ```
  List<string> = {"search", null, "pruebas temperatura"}
  ```

- **Caso selección de Search <type> <name> con *type* no nulo y *name* nulo.**

  Lista de palabras:

  ```
  List<string> = {"search", "sinoptico"}
  ```

  Escenario de éxito:

  ```
  List<string> = {"search", "sinoptico", null}
  ```

- **Caso selección de ShowRealTime <signal>**

  Lista de palabras:

  ```
  List<string> = {"tiempo", "real", "cica1"}
  ```

  Escenario de éxito:

  ```
  List<string> = {"real time", "cica1"}
  ```

- **Caso selección de ShowHistoric <signal> <number> <measure>**

  Lista de palabras:

  ```
  List<string> = {"historico", "cica1", "3", "semanas"}
  ```

  Escenario de éxito:

  ```
  List<string> = {"historic", "cica1", "3", "semanas"}
  ```

- **Caso selección de ShowGraphic <signal> <number> <measure>**

Lista de palabras:

```
List<string> = {“grafica”, “cica1”, “3”, “semanas”}
```

Escenario de éxito:

```
List<string> = {“grafica”, “cica1”, “3”, “semanas”}
```

Para el caso de las acciones `ShowRealTime`, `ShowHistoric` y `ShowGraphic`, si alguno de los parámetros es nulo se retornará una lista vacía.

8.2. Pruebas de integración

En este apartado se probará que el sistema funciona correctamente durante la integración de unos componentes con otros. Para este tipo de pruebas se va a seguir la estrategia *bottom-up*:

PI01.- Se realizará la integración del componente *JsonReaderNLP* y *NLPProcessor*. Para ello, en vez de rellenar a forma de prueba las listas de *stop words*, *empty verbs,* señales y tipos de documentos, se utilizarán los métodos implementados por *JsonReaderNLP* para leer los datos de los diccionarios en formato JSON y cargarlos en las listas.

PI02.- En la segunda prueba se integrará el componente *GestionNLPVM* con los componentes *NLPActions* y *NLPProcessor*. De esta manera se comprobará que al activar el método `ButtonProcClicked` de *GestionNLPVM* y pasarle una sentencia, ésta se procesa haciendo uso de los métodos del componente *NLPProcessor* y se ejecuta la función seleccionada del componente *NLPActions*.

Sentencia que se pasará al método `ButtonProcClicked`

“Consultar el tiempo real de la señal CICA1”

Resultado esperado

“ShowRealTime (CICA1)”

El sistema con todos sus componentes será probado en las pruebas de aceptación ya que el componente *SpeechToText* requiere de la interacción con la interfaz para su uso.

8.3. Pruebas de aceptación

En este apartado se muestran las pruebas que se han llevado a cabo enfrente de los responsables del proyecto. Con ellas los responsables han verificado y comprobado si la prueba de concepto cumplía los requisitos funcionales especificados.

Como se ha dicho con anterioridad, el sistema no reconoce los identificadores de señales, por lo tanto, para poder realizar las pruebas de aceptación se ha de dictar la descripción de la señal

y no su identificador para que sea reconocida. Por ello, en aquellos casos que se necesite indicar la señal de la que se requiere información se utilizará la siguiente estructura:

Id: CICA1, Descripción: Temperatura del edificio

PA01.- Comprobar procesado para la estructura Buscar<type> <name>

1. El usuario pulsa el botón 'REC'.
2. El sistema lanza el pop-up de la tarea de grabación.
3. El usuario dice "Buscar gráfica de temperatura".
4. El sistema recoge el audio dictado por el usuario, lo transcribe a texto.
5. El sistema muestra en el campo de texto el resultado de la transcripción.
6. El usuario pulsa el botón 'PROC'.
7. El sistema procesa la consulta.
8. El sistema muestra "Search (grafica , temperatura)".

PA02.- Comprobar procesado para la estructura Tiempo real <signal>

1. El usuario pulsa el botón 'REC'.
2. El sistema lanza el pop-up de la tarea de grabación.
3. El usuario dice "Mostrar el tiempo real de la señal temperatura del edificio".
4. El sistema recoge el audio dictado por el usuario, lo transcribe a texto.
5. El sistema muestra en el campo de texto el resultado de la transcripción.
6. El usuario pulsa el botón 'PROC'.
7. El sistema procesa la consulta.
8. El sistema muestra "ShowRealTime (CICA1)".

PA03.- Comprobar procesado para la estructura Histórico <signal> <number> <measure>

1. El usuario pulsa el botón 'REC'.
2. El sistema lanza el pop-up de la tarea de grabación.
3. El usuario dice "Quiero consultar el histórico señal temperatura del edificio de hace tres días".
4. El sistema recoge el audio dictado por el usuario, lo transcribe a texto.
5. El sistema muestra en el campo de texto el resultado de la transcripción.
6. El usuario pulsa el botón 'PROC'.
7. El sistema procesa la consulta.
8. El sistema muestra "ShowHistoric(CICA1, 3, dia)".

PA04.- Comprobar procesado para la estructura Gráfica <signal> <number> <measure>

1. El usuario pulsa el botón 'REC'.
2. El sistema lanza el pop-up de la tarea de grabación.
3. El usuario dice "Muéstrame la gráfica de la señal de la temperatura del edificio de hace cuatro semanas".
4. El sistema recoge el audio dictado por el usuario, lo transcribe a texto.
5. El sistema muestra en el campo de texto el resultado de la transcripción.
6. El usuario pulsa el botón 'PROC'.
7. El sistema procesa la consulta.
8. El sistema muestra "ShowGraphic(CICA1, 4, semana)".

PA05.- Comprobar procesado de una consulta sin una estructura correctamente definida

1. El usuario pulsa el botón 'REC'.
2. El sistema lanza el pop-up de la tarea de grabación.
3. El usuario dice "Muéstrame el histórico de la señal temperatura del edificio".
4. El sistema recoge el audio dictado por el usuario, lo transcribe a texto.
5. El sistema muestra en el campo de texto el resultado de la transcripción.
6. El usuario pulsa el botón 'PROC'.
7. El sistema procesa la consulta.
8. El sistema muestra "El sistema no dispone de ninguna acción que coincida con la estructura de su consulta".

En las primeras iteraciones las pruebas fueron realizadas en presencia de los responsables depurando los fallos del procesador y teniendo en cuenta los nuevos formatos de consulta que se podían utilizar.

Sin embargo, en la última reunión con los responsables, éstos se han encargado de efectuar las pruebas que se han mostrado anteriormente. Las pruebas han sido exitosas siempre que se ha seguido la estructura del comando, en el caso contrario el sistema ha indicado al usuario que la consulta estaba mal formulada. El tiempo de aprendizaje de los comandos a los que obedece el sistema es pequeño y una vez asimilada la estructura de éstos es difícil que el usuario se equivoque y, por lo tanto, que el sistema falle.

Capítulo 9

Conclusión y trabajos futuros

En el presente capítulo se va a valorar de forma general el resultado del proyecto y comentarán las líneas de trabajo que, en mi opinión, se deberían abordar para continuar su desarrollo en el futuro.

9.1. Conclusiones

Siendo uno de los objetivos del proyecto encontrar dos *frameworks* dedicados a la traducción de voz a texto y al procesamiento del lenguaje natural con los siguientes requisitos: gratuito, sin necesidad de conectarse a internet e integrable en Xamarin.Forms, puede decirse que se ha satisfecho parcialmente. Son los siguientes aspectos los que no han permitido cumplir completamente el objetivo:

- Se ha encontrado la librería CMUSphinx que puede realizar la transcripción *Speech-to-Text* de la consulta del usuario y satisface todos los requisitos del proyecto. Sin embargo, no se ha conseguido implementar el módulo dedicado a esta tarea con la librería encontrada.

  En su defecto, se ha implementado el módulo con la librería Speech de Google integrada en Android ya que era la librería que más se acercaba a cumplir los requisitos funcionales. Sin embargo, no puede identificar en el dictado por voz del usuario los identificadores de las señales.

- Por otra parte, se ha encontrado un *framework* con el que poder realizar el procesamiento de la consulta denominado OpenNLP, sin embargo, para utilizarlo en nuestro sistema había que utilizar un *port* y que permitiese el procesado de lengua española. Al no encontrar solución, se decidió implementarlo por completo. Este procesador desarrollado ha funcionado correctamente para las consultas y pruebas que se han realizado hasta el momento.

Por último, quiero decir que este proyecto me ha servido tanto para tener un primer contacto con el mundo laboral, como para poner en práctica los conocimientos que he adquirido durante la carrera sobre diseño y desarrollo de software. Por primera vez me he encontrado con unos exigentes requisitos para el desarrollo de un proyecto, lo que ha hecho que me dé cuenta de que la carrera no solo me ha servido para aprender conceptos teóricos orientados al desarrollo de software, sino que también he adquirido capacidades de adaptabilidad y de resolución de problemas con los que no me había encontrado anteriormente.

9.2. Trabajos futuros

Como se puede deducir del apartado anterior, el sistema no cumple el 100% de los requisitos especificados, por ello las líneas de trabajo que se proponen para continuar con el desarrollo del sistema se listan a continuación:

- **Implementar *Speech Recognizer* de acuerdo con los requisitos**
  Aun habiendo dedicado ya mucho tiempo a la implementación del *Speech Recognizer* con la librería CMUSphinx y no conseguir buenos resultados, se propone como línea de futuro indagar más sobre el funcionamiento de la librería para poder solucionar los errores en la actual implementación o buscar otra forma de utilizar la librería en el sistema, ya que es la única de todas las librerías encontradas que satisface todos los requisitos.
  Además, ya sea con la librería CMUSphinx u otra, habrá que capacitar al reconocedor para que pueda reconocer los identificadores de señales en el audio de la consulta dictada por el usuario.
- **Sustituir procesador actual por una herramienta de procesado externa**
  Aunque con las pruebas realizadas se pueda decir que el procesador desarrollado funciona correctamente, su implementación es arcaica respecto a las herramientas a las que se puede acceder de forma gratuita o no. Por lo tanto, se propone continuar con la búsqueda de un *port* que permita utilizar la librería OpenNLP para la sustitución del actual procesador.

En el caso de no encontrar ninguna solución para las líneas de futuro planteadas, se sugiere hablar con el Jefe de Proyecto y plantearle la modificación de los requisitos del proyecto.

Bibliografía

[1]tutorialspoint. *AI- Natural Language Processing.* [19 de julio de 2018]. Disponible en: https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_natural_language_processing.htm

[2]Apiumhub. *Startups y proyectos de procesamiento del lenguaje natural para ver en 2018.* NOVOSELTSEVA, E. 2018. [19 de julio de 2018]. Disponible en: https://apiumhub.com/es/tech-blog-barcelona/startups-proyectos-procesamiento-del-lenguaje-natural/

[3]Centro Virtual Cervantes. *El procesamiento del lenguaje natural, tecnología en transición.* CARBONELL, J. 1992. En: *Actas del congreso de la lengua española: Sevilla.* [20 de julio de 2018]. Disponible en: https://cvc.cervantes.es/obref/congresos/sevilla/tecnologias/ponenc_carbonell.htm

[4]Grup de Fonètica del Departament de Filología Espanyola de la Universitat Autònoma de Barcelona. *El tratamiento computacional de los niveles de análisis lingüístico.* LLISTERRI, J. 2018. [21 de julio de 2018]. Disponible en: http://liceu.uab.cat/~joaquim/language_technology/NLP/PLN_analisis.html

[5]Stanford NLP. *Stanford Tokenizer.* [22 de julio de 2018]. Disponible en: https://nlp.stanford.edu/software/tokenizer.html

[6]Microsoft. *Introduction to the C# Language and the .NET Framework*. 2015. [30 de septiembre de 2018]. Disponible en: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

[7]Microsoft. *What is .NET?*. [30 de septiembre de 2018]. Disponible en: https://www.microsoft.com/net/learn/what-is-dotnet

[8]Microsoft. *IDE de Visual Studio*. [30 de septiembre de 2018]. Disponible en: https://visualstudio.microsoft.com/es/vs/

[9]Wikipedia. *Android*. 2018. [30 de septiembre de 2018]. Disponible en: https://es.wikipedia.org/wiki/Android

[10]TCC. *What is Android OS?.* STRINGFELLOW, A. 2018. [30 de septiembre de 2018]. Disponible en: https://www.tccrocks.com/blog/what-is-android-os/

[11]Microsoft. *Xamarin.Forms*. [30 de septiembre de 2018]. Disponible en: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/

[12]*Introducing JSON*. [30 de septiembre de 2018]. Disponible en: *https*://www.json.org/

[13]Newtonsoft. *Json.NET.* [30 de septiembre de 2018]. Disponible en: https://www.newtonsoft.com/json

[14]Microsoft. *Android Speech.* 2018. [30 de septiembre de 2018]. Disponible en: https://docs.microsoft.com/es-es/xamarin/android/platform/speech

[15]No Magic, Inc. *MagicDraw*. [30 de septiembre de 2018]. Disponible en: https://www.nomagic.com/products/magicdraw

[16]NinjaMock. *Wireframe editor*. [30 de septiembre de 2018]. Disponible en: https://ninjamock.com/features

[17]Atlassian. *Getting Git Right.* [30 de septiembre de 2018]. Disponible en: https://www.atlassian.com/git

[18]*NUnit*. [30 de septiembre de 2018]. Disponible en: https://nunit.org/

[19]Microsoft. *Microsoft Project.* [30 de septiembre de 2018]. Disponible en: *https*://products.office.com/es-es/project/project-and-portfolio-management-software

[20]IBM. *Speech to Text.* [23 de julio de 2018]. Disponible en: https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/curl.html?curl#introduction

[21]Cognitiva. *Speech to Text aplicado a industrias*. VINDAS, R. 2017. [23 de julio de 2018]. Disponible en: https://www.cognitiva.la/blog/ibm-watson-speech-to-text-con-reconocimiento-de-palabras-de-industria/

[22]Microsoft Azure. *Precios de Cognitive Services: Speech Services.* [23 de julio de 2018]. Disponible en: *https*://azure.microsoft.com/es-es/pricing/details/cognitive-services/speech-services/

[23]Google Cloud. *Cloud Speech-to-Text Basics.* 2018. [24 de julio de 2018]. Disponible en: https://cloud.google.com/speech-to-text/docs/basics#phrase-hints

[24]Dialogflow. *Pricing.* [24 de julio de 2018]. Disponible en: https://dialogflow.com/pricing

[25]AWS Amazon. *Amazon Trascribe.* [24 de julio de 2018]. Disponible en: https://aws.amazon.com/es/transcribe/?nc2=h_m1

[26]*CMUSphinx.* [25 de julio de 2018]. Disponible en: https://cmusphinx.github.io/

[27]ATANASSOVA-BARNES, A. *How to "Talk" to Your Software: Alexa, Google, Watson, and Cortana, a Side-by-Side Comparison of Cloud Speech Recognition APIs.*2018. En: *14th Annual SEI*

Architecture Technology User Network Conference.[25 de julio de 2018]. Disponible en: https://resources.sei.cmu.edu/asset_files/Presentation/2018_017_001_518929.pdf

[28]g2crowd. *Best Voice Recognition Software.* [25 de julio de 2018]. Disponible en: https://www.g2crowd.com/categories/voice-recognition

[29]appus.software. *Owerview of The Best Services for Speech to Text.* BO, I. 2017. [26 de julio de 2018]. Disponible en: https://appus.software/blog/the-most-common-speech-recognition-services

[30]Bussiness Insider. *IBM speech recognition i son the verge of super-human accuracy.* WELLER, C. 2017. [26 de julio de 2018]. Disponible en: https://www.businessinsider.com/ibm-speech-recognition-almost-super-human-2017-3?IR=T

[31]Apple Developer. *iOS 10.0*. [26 de julio de 2018]. Disponible en: https://developer.apple.com/library/archive/releasenotes/General/WhatsNewIniOS/Articles/iOS10.html#//apple_ref/doc/uid/TP40017084-DontLinkElementID_3

[32]Stanford NLP. *CoreNLP.* [27 de julio de 2018]. Disponible en: *https*://stanfordnlp.github.io/CoreNLP/index.html

[33]Apache OpenNLP. *Apache OpenNLP Developer Documentation.* [28 de julio de 2018]. Disponible en: https://opennlp.apache.org/docs/1.9.0/manual/opennlp.html#intro.description

[34]spaCy. *spaCy 101: Everything you need to know.* [29 de julio de 2018]. Disponible en: *https*://spacy.io/usage/spacy-101

[35]LIN, C.S., VALIATH, S. HOPP, T. *Information Extraction -GATE, JAPE, ANNIE-*. 2008. [29 de julio de 2018]. Disponible en: http://kontext.fraunhofer.de/haenelt/kurs/Referate/Hopp_Lin_Valiath_GATE-JAPE-ANNIE-presentation.pdf

[36]CUNNINGHAM, H. [et al.]. *Developing Language Processin Comonents with GATE Version 8 (a User Guide)*. 2018. [30 de julio de 2018]. Disponible en: https://gate.ac.uk/sale/tao/tao.pdf

[37]alias-i. *LingPipe.* [30 de julio de 2018]. Disponible en: http://alias-i.com/lingpipe/index.html

[38]alias-i. *Download Models.* [31 de julio de 2018]. Disponible en: http://alias-i.com/lingpipe/web/models.html

[39]alias-i. *Download LingPipeCore.* [31 de julio de 2018]. Disponible en: *http*://alias-i.com/lingpipe/web/download.html

[40]*sharpnlp.* [28 de julio de 2018]. Disponible en: *https*://archive.codeplex.com/?p=sharpnlp

[41]EcuRed. *Modelo Vista Vista Modelo.* [1 de julio de 2018]. Disponible en: *https*://www.ecured.cu/Modelo_Vista_Vista_Modelo

[42]Microsoft. *The Model-View-ViewModel Pattern.* 2017. [1 de julio de 2018]. Disponible en: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[43]Wintellect. *Model-View-ViewModel (MVVM).* LIKNESS, J. 2014. [1 de julio de 2018]. Disponible en: https://www.wintellect.com/model-view-viewmodel-mvvm-explained/

[44]DifferenceBetween. *Difference between MVVM and MVP.* DOUGLAS, G. 2017. [1 de julio de 2018]. Disponible en: http://www.differencebetween.net/technology/difference-between-mvvm-and-mvp/

[45]GeeksforGeeks. *HashSet in Java.* SINGH, D. y LENKA, C. [24 de agosto de 2018]. Disponible en: *https*://www.geeksforgeeks.org/hashset-in-java/

[46]*Big-O Cheat Sheet*. [24 de agosto de 2018]. Disponible en: http://bigocheatsheet.com/

[47]Microsoft. *Selecting a Collection Class*. 2017. [24 de agosto de 2018]. Disponible en: https://docs.microsoft.com/es-es/dotnet/standard/collections/selecting-a-collection-class