UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Extensión de funcionalidades de una aplicación para la programación en Java para personas con diversidad funcional

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Font Vicedo, Andrés

Tutor: Martínez Hinarejos, Carlos David

2019 - 2020

Resumen

En este proyecto se van a extender las funcionalidades del *plugin* COPS, una aplicación para facilitar la programación en el entorno Eclipse a personas con diversidad funcional. Esto se va a conseguir mejorando por una parte el alcance del reconocedor de voz integrado en COPS, iATROS, y por otra añadiendo la funcionalidad de navegación y uso de menús de Eclipse mediante órdenes de voz.

Se va a comenzar con un estudio sobre las distintas opciones relacionadas con la programación y el reconocimiento de voz que hay disponibles y cuáles son sus características diferenciales.

Una vez hecho esto, para llevar este proyecto a cabo se va a realizar un proceso de mantenimiento perfectivo. Primero se hará un análisis del sistema COPS y de sus componentes, incluido iATROS. En este análisis se realizará un estudio exhaustivo del código a través de técnicas de ingeniería inversa, para así obtener los máximos conocimientos sobre su funcionamiento interno.

Una vez estudiada la aplicación, se continuará con un proceso de ingeniería de requisitos para poder resolver el problema de la mejor manera posible y tener unos requisitos contra los que realizar las pruebas.

Una vez especificados todos los requisitos, se procederá a diseñar e implementar la solución propuesta. Tras esto, se pasará a un proceso de prueba de software para verificar que se han cumplido todos los requisitos especificados.

Palabras clave: Mantenimiento, accesibilidad, Reconocimiento de voz, Eclipse.

Abstract

This project will extend the functionalities of the COPS plug-in, an application to facilitate programming in the Eclipse environment for people with functional diversity. This will be achieved by improving the range of the voice recogniser integrated into COPS, iATROS, on the one hand, and by adding the navigation and menu-driven functionality of Eclipse via voice commands on the other.

Firstly, a study will be carried out on the different options related to programming and voice recognition available and what their differential characteristics are. Once this has been done, a process of perfective maintenance will be done in order to carry out this project. First, an analysis of the COPS system and all its components, including iATROS, will be done. In this analysis, an exhaustive study of the code is made through reverse engineering techniques, in order to obtain the maximum knowledge of its internal functioning.

Once the application has been studied, a requirement engineering process is carried out to solve the problem in the most optimal way and to have requirements against which to do the tests.

Once specified, the proposed solution will be designed and implemented. After this, a software testing process will be done to verify that all requirements have been fulfilled.

Keywords: Maintenance, accessibility, speech recognition, Eclipse.

Tabla de contenidos

Contenido

1. Introducción 7
   1.1 Motivación 9
   1.2 Objetivos 9
   1.3 Impacto esperado 10
   1.4 Estructura 10
   1.5 Convenciones 11
2. Estado del arte 13
   2.1 Conclusiones sobre el estado del arte 18
3. Análisis del problema 19
   3.1 Plan de trabajo 19
   3.2 Tecnología utilizada 22
   3.3 Estudio y análisis del sistema COPS 23
   3.4 Identificación y análisis de soluciones posibles 32
   3.5 Solución propuesta 34
4. Diseño de solución 43
   4.1 Acciones de actualización 43
   4.2 Diseño detallado 43
5. Desarrollo de la solución propuesta 47
6. Pruebas 51
7. Conclusiones 57
   7.1 Relación con los estudios cursados 58
8. Trabajos futuros 59
9. Bibliografía 61

1. Introducción

Eclipse es una plataforma de desarrollo de software de código abierto diseñada para ser extendida de forma indefinida a través de *plugins*. Fue pensada desde el principio para convertirse en una plataforma de integración de herramientas de desarrollo. Además, no tiene en mente un lenguaje específico, sino que es un entorno de desarrollo integrado (IDE en inglés) genérico, por lo que da soporte a multitud de lenguajes. Sin embargo, el lenguaje más popular entre los desarrolladores en Eclipse es Java, apoyándose en el famoso *plugin* JDT[1], que ya viene incluido en la distribución estándar de la herramienta. Esto es una pequeña muestra de lo grande e implicada que está su comunidad de usuarios, constantemente extendiendo las funcionalidades con nuevos *plugin*, como el ya mencionado JDT, EGit[2] (que integra Git en Eclipse) o ADT[3] (para el desarrollo de aplicaciones Android).

En este marco está a COPS (Computer Programming using Speech) [5]. COPS es también un *plugin* para Eclipse y, como tal, extiende la funcionalidad de la plataforma de desarrollo, en este caso hacia la accesibilidad. Lo que COPS permite es que personas con diversidad funcional puedan trabajar de manera más cómoda a la hora de escribir código. Esto lo consigue sobre todo con sus dos mayores funcionalidades: el reconocimiento de voz y la síntesis de voz. Con la primera, permite escribir código con la voz, mientras que con la síntesis nos es posible revisar el código a través de la reproducción del fragmento seleccionado.

El funcionamiento del *plugin* es sencillo. Una vez abierto Eclipse, abriendo la vista de COPS, se mostrará el código del fichero que esté abierto clonado en una nueva pestaña, que de forma predeterminada tendrá un fondo negro con la fuente en amarillo, para obtener un mayor contraste. Este aspecto se puede ver en la Figura 1.

[1] https://www.eclipse.org/jdt/overview.php
[2] https://projects.eclipse.org/projects/technology.egit
[3] https://stuff.mit.edu/afs/sipb/project/android/docs/tools/sdk/eclipse-adt.html

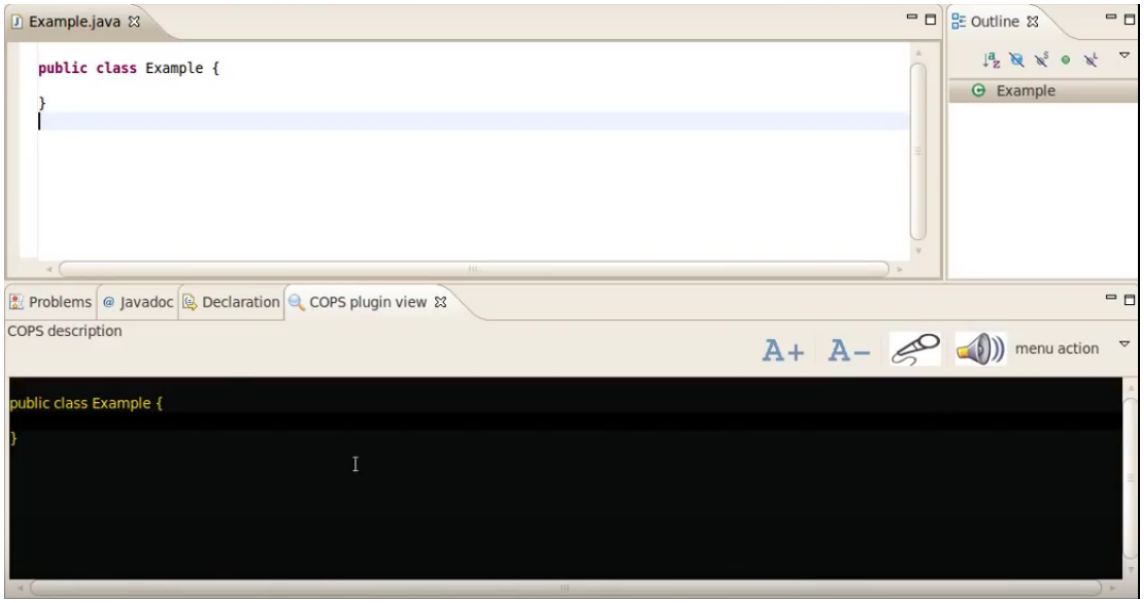

Figura 1. Vista principal de COPS.

Esta configuración de fuente y fondo puede modificarse desde las preferencias de la vista, guardarse y cargarse en la siguiente sesión, como puede verse en la Figura 2.

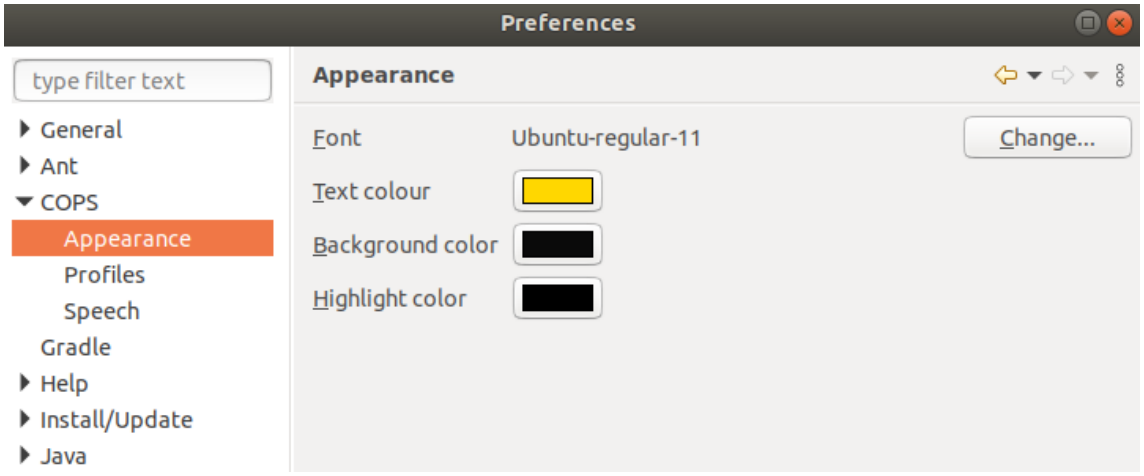

Figura 2. Vista de las preferencias de COPS.

Para las funciones principales están los iconos de aumentar el tamaño de la fuente, disminuirlo, un micrófono y un altavoz. Al hacer el primer clic en el micrófono comenzará la grabación; con el segundo clic se parará y con el audio guardado se escribirá el código dictado. Por otro lado, seleccionando una parte del texto y haciendo clic en el altavoz, se puede escuchar el código seleccionado. El proceso de síntesis se

apoya en FreeTTS[4] mientras que el dictado se apoya en iATROS [15], del que se hablará más adelante.

Además de estas funciones, COPS nos ofrece otras funcionalidades como cambiar el color y tamaño de la fuente, además del fondo para ofrecer un texto en alto contraste. Todo esto desde una interfaz clara y sencilla.

1.1 Motivación

El motivo de la elección de este proyecto ha sido la gran presencia del software en nuestras vidas y las barreras que las personas con diversidad funcional tienen para acceder a él. Casi cualquier elemento cotidiano tiene su chip integrado con el que procesa la información de uso y nos muestra en nuestro teléfono móvil esas métricas que no sabíamos que necesitamos. Esto lleva a que cada vez más gente esté interesada en cómo funcionan estos sistemas que la rodean, lo que deriva a que cada vez haya más interés por la programación y el desarrollo de software en general, y esto no excluye a las personas con diversidad funcional. Estas personas deben tener alternativas adaptadas para poder acceder al mundo del desarrollo de software en las mejores condiciones posibles y así disfrutar de este gran mundo que es la programación.

1.2 Objetivos

El primer objetivo es conocer a fondo el funcionamiento e implementación de COPS, junto con el uso del sistema iATROS en el que se apoya. Una vez alcanzado, el objetivo principal es ampliar la funcionalidad del *plugin* gracias a ese conocimiento adquirido, añadiendo la posibilidad de navegar por los menús de la plataforma Eclipse también por órdenes de voz, de manera similar al funcionamiento del dictado de voz para código. Además de esto, se va a ampliar el vocabulario de iATROS para reconocer las nuevas órdenes de voz para la navegación de los menús, además de añadir nuevos términos de programación y así dar un mayor soporte a la parte de dictado de código.

[4] https://freetts.sourceforge.io/

1.3 Impacto esperado

El impacto esperado de esta ampliación de funcionalidad es el de dar mayor soporte a las personas con diversidad funcional, en este caso dando aún más protagonismo al dictado por voz. Con la posibilidad de navegar por menús con órdenes de voz se evita un mayor uso del ratón para navegar por los mismos, reduciéndolo a dos clics y evitando así la incomodidad que puede suponer el buscar un menú concreto. Respecto a la ampliación del vocabulario de iATROS, se busca conseguir por una parte la posibilidad de navegar por los menús y, por otra, evitar en la medida de lo posible correcciones en el código que pueden surgir tras hacer el dictado por voz debido a la falta de términos propios de la programación.

1.4 Estructura

Se comenzará explicando en el capítulo 2 el estado del arte, es decir, qué técnicas han sido las más utilizadas en el reconocimiento de voz hasta ahora y qué alternativas a COPS se pueden encontrar en el mercado. En el capítulo 3 se va a exponer el problema y se va a proponer un proceso de mantenimiento perfectivo para abordarlo. Tras elegir las herramientas para el desarrollo, en el punto 3.3, se hace un análisis profundo, a través de técnicas de ingeniería inversa del *plugin* COPS en mayor parte, y del componente iATROS. Visto esto, en el siguiente punto, el 3.4, se exponen las distintas soluciones que se pueden aplicar al problema, con sus problemas y sus beneficios. Finalmente, en el punto 3.5 se eligen las soluciones a aplicar y se justifican dentro del marco del proyecto. En el siguiente capítulo, el 4, se diseña la solución que se ha decidido implementar, desarrollando sus requisitos y teniendo en cuenta el sistema heredado. Una vez diseñada la solución, se procede a su desarrollo, en el capítulo 5. Tras esto, en el capítulo 6 se desarrollan las pruebas que se han llevado a cabo para comprobar que se cumplen los requisitos especificados en el diseño. Finalmente, el capítulo 7 son las conclusiones de todo el proyecto y la relación con el grado en Informática. Para acabar, se indican posibles trabajos futuros y en el punto 9 está la bibliografía.

1.5 Convenciones

Para los nombres de carpetas, archivos, y términos de programación como código, nombres de clases y nombres de métodos, se empleará la letra Courier.

Para términos en inglés, se utilizará la letra Arial cursiva.

2. Estado del arte

El panorama en el mundo del reconocimiento de voz ha cambiado mucho desde su nacimiento, comenzando con sistemas rudimentarios de pocas palabras, pasando por sistemas más clásicos con un vocabulario mayor y llegando al aprendizaje profundo que se utiliza hoy en día.

Los inicios del reconocimiento de voz se pueden remontar al año 1952, cuando Davis, Bidulph y Balashek fabricaban el primer reconocedor, en este caso de dígitos. Sin embargo, estos primeros reconocedores solo funcionarán con un usuario específico para el que se ha preparado la máquina [1].

Paralelamente a este trabajo, en los Laboratorios RCA en 1956 se desarrollaba una máquina de escribir fonética. Su funcionamiento era sencillo: a partir de una señal de audio se obtenía un dibujo característico para cada sonido. En ese momento se preguntaron qué sería mejor reconocer: fonemas, sílabas o palabras. Por una parte. los fonemas daban resultados alejados a la realidad; por el otro, las palabras, si bien al ser reconocidas acertaban la gran mayoría de veces, ocupaban demasiada memoria como para tener un vocabulario amplio, por lo que finalmente pasaron a reconocer sílabas [2].

Una vez entrados en la era informática, los avances cada vez son mayores, pero no es hasta 1971 cuando se toma realmente en serio el ámbito del reconocimiento de voz, precisamente cuando Estados Unidos lanza el mayor proyecto de reconocimiento del habla hasta entonces, el ARPA-SUR (*Advanced Research Projects Agency - Speech Understanding Research*) [3]. A pesar de que los resultados no fueran los esperados, este movimiento dio un gran impulso a la investigación a la vez que las aproximaciones al problema son cada vez más acertadas, como es la de los modelos ocultos de Markov. Los modelos ocultos de Markov serían una aproximación efectiva al problema hasta la llegada de las redes neuronales que maximizarían su rendimiento, pero que, en aquel momento, debido a las limitaciones tanto de hardware como de algoritmos, no eran viables [2]. Los modelos ocultos de Markov son una técnica de modelización de datos secuenciales en los que el objetivo es determinar los parámetros desconocidos (ocultos) de dicha secuencia a partir de los parámetros

observables. Este modelo, junto al algoritmo de Viterbi (que busca la secuencia más probable de estados ocultos) es la base del reconocimiento de voz clásico [4].

En el caso que nos ocupa, COPS utiliza iATROS para reconocer la voz, que también trabaja con una variante de los modelos ocultos de Markov, los de densidad continua. En su vocabulario están presentes todas las palabras reservadas de Java, además de los números, las letras individualmente y símbolos como el paréntesis, los signos de puntuación, corchetes, etc. [5].

Tras este gran avance, el siguiente paso era usar redes neuronales para el entrenamiento, pero la ciencia no estaba preparada aún, por lo que el entrenamiento de los sistemas de reconocimiento de voz se realizaba con modelos de mezclas gaussianas. Estos modelos realizan una *clusterización*, es decir, clasificar, dentro de un grupo o población, distintos subgrupos o subpoblaciones. A pesar de que este modelo es bastante efectivo, no es demasiado eficiente dadas las características de la voz y sus ondas. No es el caso de las redes neuronales, que tienen la posibilidad de aprender mucho más, y de una forma mucho más eficiente. En la actualidad el hardware permite su uso, y esto lo ha convertido en la técnica más utilizada en el reconocimiento de voz [6]. Grandes empresas como Google y Amazon utilizan estas técnicas en sus dispositivos y siguen mejorando el resultado gracias a los datos de sus usuarios.

Una vez visto un poco de contexto sobre las etapas del reconocimiento de voz, se van a ver a las alternativas o productos disponibles para la programación por voz en Eclipse. En este aspecto, no se puede encontrar gran variedad de software en forma de *plugin*, excepto algún caso que se comentará a continuación, siendo normalmente la opción más buscada usar software externo a la plataforma de desarrollo, tanto para escribir como para navegar en los menús.

En cuanto a los *plugin*, el primero es SpeechClipse, que actualmente ya no está disponible. Este *plugin* cuenta únicamente con un sistema de reconocimiento de voz para la navegación por menús, a diferencia de COPS, que cuenta además con sistemas de alto contraste, síntesis de voz y cambio del tamaño de la fuente.

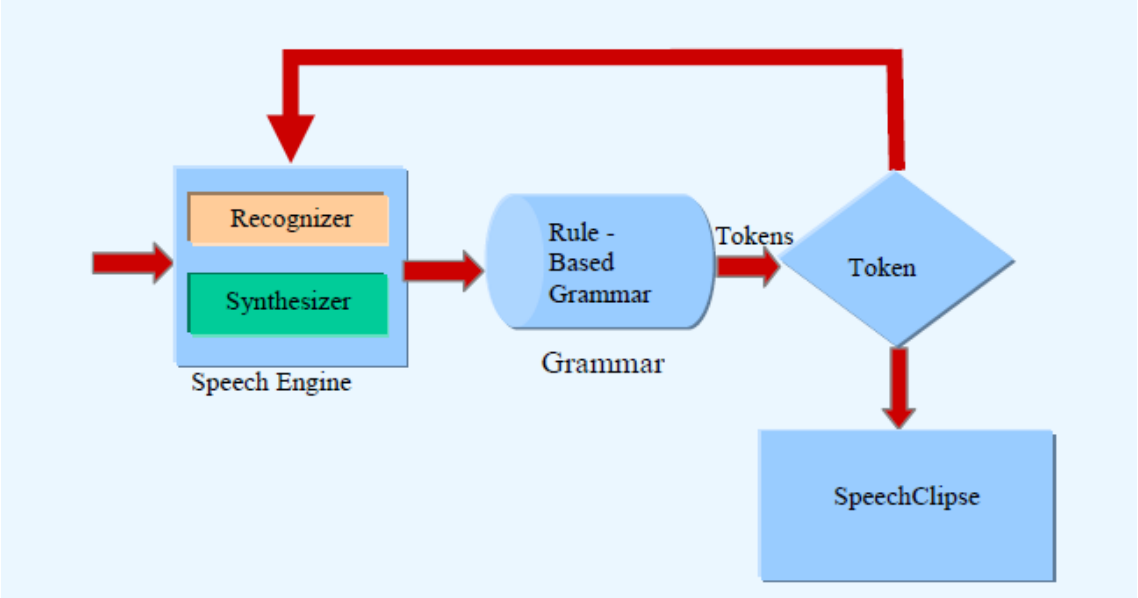

Figura 3. Diagrama de flujo de control de SpeechClipse [7].

SpeechClipse usa una implementación de la API Java Speech sobre la herramienta de reconocimiento de voz de Windows y define su gramática con Java Speech Grammar Format. Su funcionamiento se basa en el esquema representado en la Figura 3. La gramática define qué debe o qué va a escuchar el reconocedor de voz. Esta gramática incluye una parte de dictado libre y otra basada en reglas; en el caso de SpeechClipse, solo utiliza la basada en reglas para que su funcionamiento sea más preciso. La opción de dictado libre suele resultar más natural para el usuario a la hora del reconocimiento, pero por el momento no se ha implementado. Su funcionamiento es sencillo: a partir de la entrada del micrófono, comprueba si lo dictado es un evento simple (por ejemplo, abrir el menú 'archivo'); en caso de que sea así, pasa el evento a 'Robot Class', que lo transforma en una entrada de teclado, abriendo la opción seleccionada. En caso de que se trate de más de una acción, las encadena y las manda a la 'Robot Class' para que se ejecuten de la misma forma, como puede verse en la Figura 4 [7].

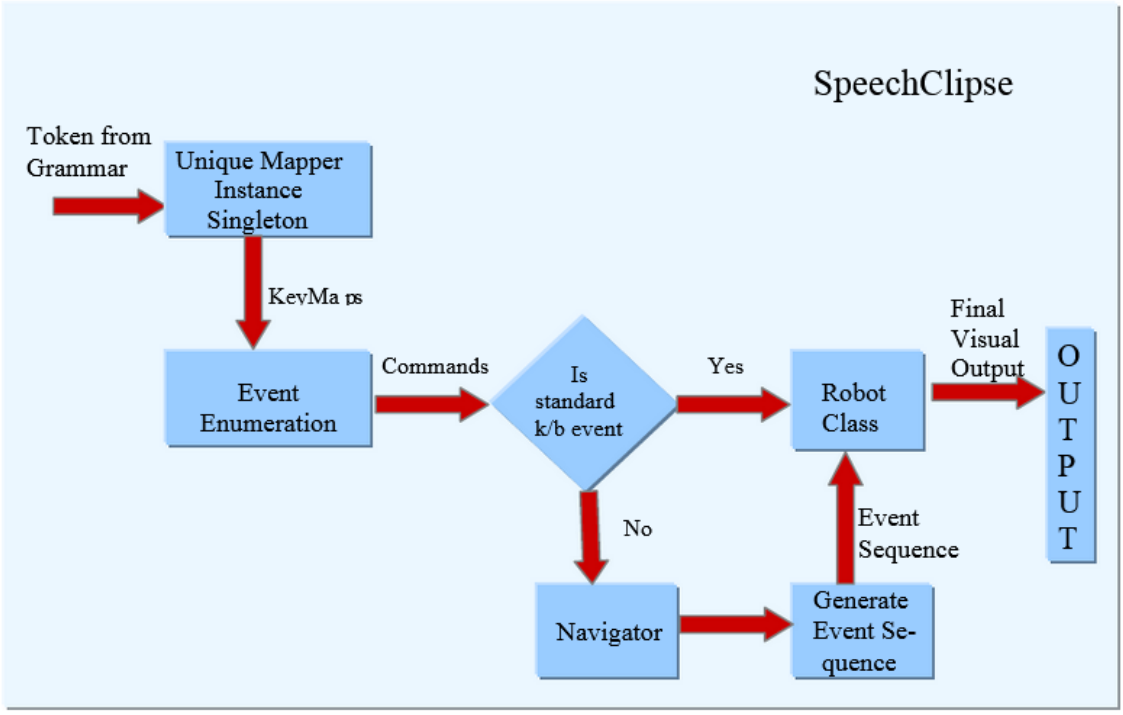

Figura 4. Flujo de control interno de SpeechClipse [7].

Otro *plugin* relacionado con el reconocimiento de voz es Polaris[5] (programing with voice in Eclipse). Este *plugin* permite realizar algunas acciones básicas como cortar texto, copiarlo y abrir diálogos para nuevos archivos a través de la voz. Como se puede ver en la Figura 5, la interfaz de Polaris presenta alguna similitud con la de COPS, ya que, pese a no tener las mismas funciones, conserva esa presentación en forma de gran botón integrado en la propia interfaz, aunque en este caso en la barra superior. No existe demasiada información acerca de su arquitectura y funcionamiento interno.

[5] https://sourceforge.net/projects/polarisspeachrecognition/

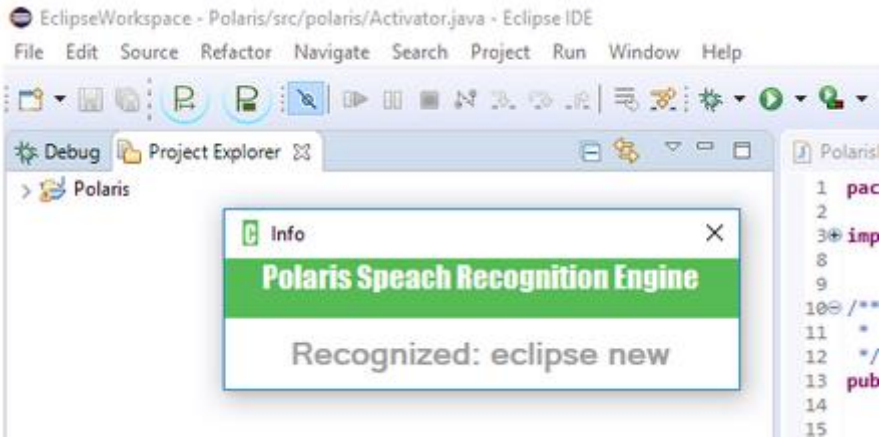

Figura 5. Interfaz de Polaris.

La siguiente herramienta no se instala como un *plugin*, pero también está integrada en Eclipse; se trata de Ok Eclipse[6]. En este caso también es una herramienta que se basa en órdenes de voz, en vez de transcripción de código como tal. Tiene órdenes como 'crea una clase con cuatro variables `Integer`; transformación de código Java a Python con una orden 'convert' o generar la estructura de pruebas unitarias, entre otras funciones similares. Para realizar el reconocimiento de voz utiliza la API de Google Speech.

Fuera del entorno Eclipse, aparecen más alternativas para el dictado de voz, ya sea para escribir texto o código.

Una de estas herramientas es Dragon[7], de la empresa Nuance. Esta herramienta principalmente es utilizada para el dictado de textos, y aunque inicialmente no está pensada para ello, hay personas que lo utilizan para programar, ya no en Eclipse, si no en general en cualquier IDE y en cualquier lenguaje. No obstante, debido a su concepción y lo intrincado de los lenguajes de programación, resulta difícil en algunos momentos escribir código con la herramienta.

Otra herramienta, esta discontinuada, es VoiceCode[8]. Esta herramienta sí que está directamente desarrollada para la programación por voz y es independiente del entorno de programación que se esté utilizando. Esta herramienta trabaja juntamente con Dragon para el reconocimiento de voz y además añade soporte para accesorios de manos libres para el manejo del ratón como es SmartNav.

[6] https://github.com/dharmangbhavsar/ok-eclipse
[7] https://www.nuance.com/es-es/dragon.html
[8] https://voicecode.io/

2.1 Conclusiones sobre el estado del arte

Como se ha podido ver en este apartado, no existen demasiadas alternativas para el desarrollo en Eclipse, pero las que están incluyen habitualmente control de menús por órdenes de voz por delante de reconocimiento de voz a la hora de programar. Fuera de Eclipse, el dictado de voz es la opción más avanzada, en algún caso (como en VoiceCode) adaptado a la programación, en otros para el texto plano. En cualquier caso, se puede observar que existe una falta de software integrado en Eclipse que cumpla tanto con la navegación de menús como con un mejor reconocimiento de voz, por lo que esos son puntos a reforzar en COPS.

3. Análisis del problema

Vistas las opciones que actualmente existen, parece claro que los ámbitos a mejorar en COPS son un mayor vocabulario y el aumento, en la medida de lo posible, de sus funcionalidades.

Respecto al vocabulario, iATROS ya cuenta con todas las palabras reservadas de Java, además de letras, números y signos de puntuación necesarios para programar. Es por ello por lo que el foco debe ponerse en ampliar los siguientes términos más utilizados, los identificadores (palabras o textos que se utilizan para nombrar diferentes elementos del lenguaje). Si bien es imposible cubrir todos los identificadores, ya que cualquier combinación de letras puede serlo, sí que existen muchos que son comúnmente utilizados, a los que habrá que dar soporte.

Por otra parte, se quiere aumentar la funcionalidad del *plugin.* Para ello, se ha optado por hacer más accesible la navegabilidad por los menús y opciones que ofrece Eclipse a través de órdenes de voz, como el dictado de voz que se utiliza para escribir código.

3.1 Plan de trabajo

Dada la naturaleza del problema, queda claro que se está ante un caso de mantenimiento de software. En el mantenimiento de software, aunque su definición pueda variar según la fuente, existen cuatro tipos de mantenimiento: correctivo, adaptativo, preventivo y perfectivo [8].

- El mantenimiento correctivo está indicado para el caso en el que un software necesita la corrección de algún defecto.
- El mantenimiento adaptativo se indica en los casos en los que se necesita realizar un cambio de entorno.
- El mantenimiento preventivo mejora las propiedades sin cambiar sus especificaciones funcionales, por ejemplo, mejorando su mantenibilidad.
- Por último, está el mantenimiento perfectivo, que es el que corresponde al problema planteado. El mantenimiento perfectivo es el que realiza cambios en

la especificación, normalmente debidos a cambios de requisitos del producto software, lo que normalmente lleva a la mejora o adición de nuevas funcionalidades. En el caso de COPS, esas mejoras son la navegación por menús y la ampliación del vocabulario. Aunque el mantenimiento correctivo es el primero que se suele asociar al mantenimiento del software, el mantenimiento perfectivo es el más habitual [8].

En cuanto a las actividades principales en el proceso de mantenimiento del software, se identifican tres categorías: la compresión del software y los cambios a realizar, la modificación del software y la realización de pruebas.

- La comprensión del software supone comprender la funcionalidad y estructura interna del software ya existente y cómo aplicar los nuevos cambios.
- La modificación del software en sí consiste en la implementación de los cambios estudiando las repercusiones que puede tener en el software ya existente, para evitar nuevos defectos.
- Por último, están las pruebas, donde además de cumplir los nuevos requisitos especificados para la mejora, se han de seguir cumpliendo los del software original.

Durante el mantenimiento pueden surgir diferentes obstáculos. Entre ellos, el más importante es los problemas que pueda traer el código heredado. No obstante, en este caso se ha realizado un buen trabajo en el desarrollo de COPS, ya que la mayoría de los métodos y clases están comentados y estructurados, facilitando así la comprensión del código.

Otro aspecto para tener en cuenta son las leyes del mantenimiento, una serie de leyes de las cuales se ha querido destacar dos relacionadas con el problema.

- La primera es la continuidad del cambio, es decir, en el momento en el que un programa es escrito, ya está desfasado, surgen nuevas necesidades que cubrir y hay que seguir manteniendo el producto. Como se ha podido ver en el mercado, a pesar de existir pocas alternativas, vocabularios más amplios y posibilidad de navegación por menús es algo que existe y que el usuario va a demandar.
- La segunda ley, la de crecimiento continuo, está muy relacionada con la de continuidad del cambio. El software normalmente se desarrolla con limitaciones, ya sean de tiempo, presupuesto o cualquier otro recurso, lo que

hace que según pasa el tiempo los requisitos descartados se vuelvan necesarios. En el caso de COPS, se mencionan ya futuros trabajos que probablemente no fueron desarrollados por algún tipo de limitación en el momento del desarrollo [5].

Una vez visto los aspectos más destacados que tener en cuenta, se debe realizar un plan de mantenimiento [10]. En este caso, se va a utilizar el plan propuesto por el estándar ISO/IEC 14764 [9] con alguna modificación, dado que este plan está concebido para organizaciones y proyectos de mayor envergadura.

El primer punto en este plan de mantenimiento es la introducción. En este punto se va a comenzar describiendo el sistema que se va a mantener y el estado desde el que se va a partir el software en cuestión. En este caso, partimos de un software antiguo pero que ha recibido un mantenimiento. Todas las clases están comentadas y la mayoría de sus métodos están documentados. Por la parte del reconocedor iATROS no hay mucha documentación más allá de comentarios en los propios ficheros. Una vez visto esto, se debe justificar el mantenimiento, en este caso es debido a la falta de opciones de accesibilidad. Existen aplicaciones que permiten la navegación por menús y cuentan con un vocabulario más amplio, por lo que el objetivo es añadir esas funciones a COPS.

En el segundo punto se debe definir el tipo de mantenimiento a realizar. En este caso, únicamente se trata de mantenimiento perfectivo, ya que existe un trabajo previo de mantenimiento preventivo en el proyecto. La definición de mantenimiento perfectivo según el estándar ISO varía ligeramente del anteriormente comentado, pero aun así encaja con el trabajo a realizar. La definición es la siguiente: “Modificación de un producto software, después de su entrega, para mejorar su rendimiento o su mantenibilidad.” [9].

El siguiente y último punto es el más importante, la organización del mantenimiento. En este punto se va a comentar qué herramientas se utilizarán para el desarrollo del proyecto. Luego de esto, se va a realizar un análisis del software del que se parte, donde se obtendremos una compresión completa del mismo. Para esto, principalmente realizaremos una lectura profunda del código que más tarde habrá que ampliar. Una vez hecho esto se usarán técnicas de ingeniería inversa para realizar una abstracción del sistema a través de diagramas de flujo y otros modelados para facilitar su comprensión. El siguiente paso será analizar el problema, especificando unos

requisitos y proponiendo diferentes soluciones, buscando la de mejor encaje en el software. Tras todo esto solo quedará la propia modificación del software y su documentación, tras la que se procederá a las pruebas, validando así tanto los requisitos especificados anteriormente como el correcto funcionamiento del resto de las partes.

3.2 Tecnología utilizada

Para el desarrollo de este software y la redacción de la memoria, se han utilizado las siguientes tecnologías:

- Ubuntu: Ubuntu 18.04 ha sido la distribución de Linux elegida como sistema operativo para el desarrollo del software debido a la naturaleza de iATROS (este solo puede ser ejecutado en entornos Linux). Entre las distribuciones de Linux disponibles, se ha elegido Ubuntu por su similitud con los sistemas Windows y su interfaz amigable, una ayuda para usuarios inexpertos en sistemas Linux. Respecto a la versión, se eligió la opción con soporte a largo plazo de entre las disponibles, en ese momento, Ubuntu 18.04.
- Eclipse: Eclipse ha sido el IDE donde se ha desarrollado la mejora de COPS, ya que éste mismo ha sido desarrollado en el propio Eclipse en la parte relativa al *plugin*. Además de esto, Eclipse es un IDE muy popular en el mundo del desarrollo y más en los proyectos en lenguaje Java.
- Java: Para la programación del software se ha recurrido al lenguaje de programación Java, dado que el desarrollo inicial de COPS está realizado en este mismo lenguaje.
- Kate: Tanto para la edición de los archivos de iATROS como para la toma de notas u otras necesidades, Kate ha sido la opción escogida.
- ObjectAid UML: ObjectAid UML ha sido la herramienta elegida para realizar la parte de análisis del problema, ya que, con ella, en pocos clics se puede obtener un diagrama UML de las clases que se haya escogido, facilitando así el proceso de análisis del código al tener un apoyo visual del conjunto.
- Balsamiq mockups: La siguiente herramienta es Balsamiq mockups. Se ha elegido esta herramienta para la realización de los distintos *mockups,*

necesarios a la hora de definir requisitos. Con Balsamiq mockups es posible hacer *mockups*, incluso con enlaces entre ellos, facilitando mucho el proceso de validación de requisitos y permitiendo ver de manera clara cómo puede quedar una implementación antes de llevarla a cabo.

- LucidChart: Lucidchart ha sido la herramienta que se ha elegido para la realización de los distintos diagramas de flujo con los cuales explicar visualmente el funcionamiento tanto de iATROS independientemente como de COPS junto a iATROS. Esta herramienta además proporciona un enfoque del planteamiento distinto, con el cual es normal identificar partes que antes habían pasado desapercibidas.
- Eutranscribe: Eutranscribe ha sido la herramienta escogida para realizar la transcripción de texto a fonética. Se trata de un script de Perl en el que, a partir de una entrada de texto, se genera una salida con la transcripción correspondiente. Se ha elegido esta herramienta ya que fue diseñada específicamente para iATROS y es la única que genera los resultados necesarios para que funcione el reconocedor de voz.

3.3 Estudio y análisis del sistema COPS

COPS se puede dividir en dos subsistemas claros. Por un lado está iATROS y por el otro el *plugin* en sí.

El subsistema iATROS es el que realiza todo el proceso de reconocimiento de voz. Para utilizarlo, lo primero que será necesario es ejecutar `iatros-run`. El ejecutable `iatros-run` es el que arranca el sistema de reconocimiento de iATROS. En caso de que se quiere hacer algún cambio en la configuración, habrá que modificar el archivo `cops.conf`. En este archivo se deciden varias opciones para la ejecución del reconocedor, siendo algunas de ellas el vocabulario que se va a utilizar (iATROS está disponible en español o inglés), la ruta donde debe estar el archivo de entrada para el reconocimiento o los modelos de lenguaje, de los que se hablará a continuación. Una vez decidida la configuración y ejecutado `iatros-run`, el sistema de reconocimiento estará listo para recibir una entrada y devolver el texto correspondiente. La entrada que este proceso necesita es un cepstrum. Un cepstrum es la representación de una señal al aplicarle la transformada de Fourier inversa. Este será el formato necesario

para realizar la conversión a texto, que se conseguirá a través de procesos internos de iATROS.

Para llegar a entregar un cepstrum, primero es necesario comenzar con un audio en formato WAV (Waveform Audio Format), un formato de audio sin compresión. Con esto, se deberá ejecutar el archivo `wav2raw`, que transforma archivos de WAV a RAW, un formato de audio también sin compresión. Una vez se tenga el archivo de audio RAW, ejecutando el archivo `raw2CC` se estará llamando a `iatros-speech-cepstral`, que generará el archivo `cops.CC`, es decir, el cepstrum necesario para realizar el reconocimiento. En el momento en el que aparezca este archivo, si iATROS está corriendo, realizará el reconocimiento y dará como salida el archivo final `cops.out`, donde estará el texto reconocido a partir del audio inicial y que se deberá manejar para realizar las acciones oportunas. Este archivo queda en la carpeta temporal del sistema `/tmp`. Todo este flujo se describe gráficamente en la Figura 6.

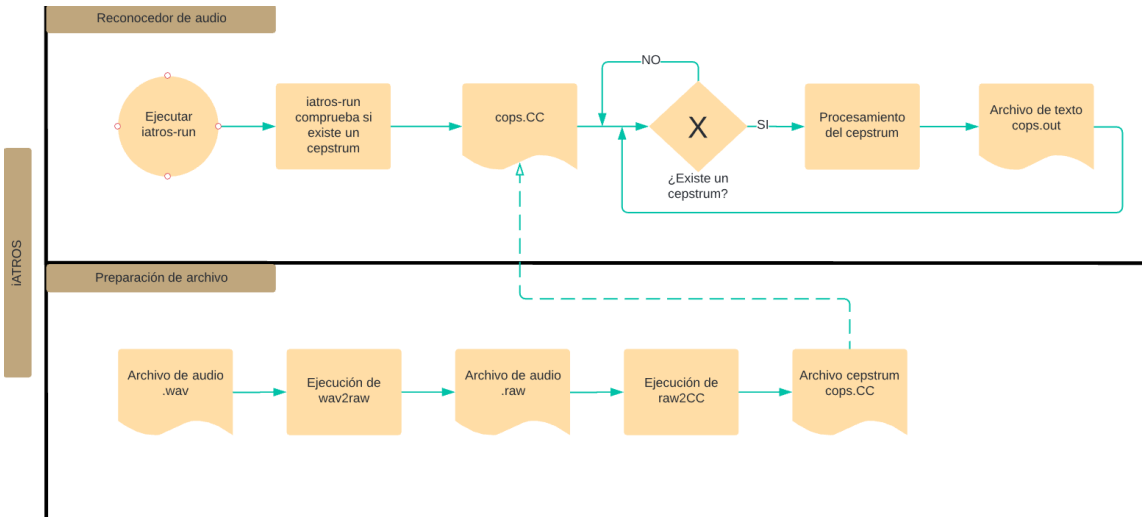

Figura 6. Flujo de iATROS.

Volviendo a la configuración de iATROS, se ha comentado tanto la posibilidad de elección de idioma de reconocimiento como la elección de modelos de lenguaje. Comenzando por los idiomas, puede verse que su vocabulario está definido en los archivos `cops.lx` y `cops_english.lx`, dentro de la carpeta `models`. En la Figura 7 puede verse como la estructura del archivo consta de tres columnas: el símbolo o palabra reconocido, el peso, menor a 1 en caso haya más de una pronunciación para el mismo termino, y la fonética, preparada para el reconocedor de iATROS.

```
) 0.5    z i e @ a p a r e n t e s i s

? 1.0   i n t e @ o g a n t e

@ 1.0   a @ o b a

abstract 1.0    a b s t r a k t

boolean 1.0    b u l e a n

break 1.0    b r e k

byte 1.0    b a i t

case 1.0    k e i s

catch 1.0    k a t c

char 1.0    c a r

class 1.0    k l a s

const 1.0    k o n s t

continue 0.5    k o n t i n i u
```

Figura 7. Vocabulario de iATROS.

Por otra parte, tenemos los modelos. iATROS cuenta con cuatro modelos de lenguaje: `ident.gr`, `numero.gr`, `caracter.gr` y `cadena.gr`. En estos archivos se están definiendo una serie de autómatas finitos para determinar qué va a poder reconocer iATROS. El primer fichero determina los posibles identificadores, el segundo para números, el tercero los caracteres individuales y el último para cadenas de caracteres. Dado que los ficheros de iATROS son demasiado extensos, se ha creado un ejemplo simplificado para explicar su funcionamiento en la Figura 8. El autómata comienza en el estado 0, desde ahí puede pasar de nuevo al estado 0 o avanzar al 1, como indica la segunda columna. Para los cambios de estado, sea volver al mismo o pasar al siguiente, tendrá que tomar el valor de la tercera columna, con una probabilidad del 0.25 para todos los casos. En el caso del ejemplo, se ha añadido el parámetro #ANT#, que indica que el valor anterior tomado se concatenará con el siguiente, que puede ser ‘x’ o ‘y’. Por tanto, desde el estado 0 se podrá volver al estado 0 añadiendo ‘x’ o ‘y’ a la cadena y se podrá ir al estado 1, añadiendo ‘A’ o ‘B’. Una vez en el estado 1, ya sea añadiendo ‘Z’ o ‘W’ se llegará al estado 2, que es el estado final y no tiene más transiciones. Una cadena resultante de este ejemplo podría ser: `xyxyyxAZ`.

```
State 0 c = 1 i = 1

0   0   "#ANT#x"   p = 0.25
0   0   "#ANT#y"   p = 0.25
0   1   "#ANT#A"   p = 0.25
0   1   "#ANT#B"   p = 0.25

State 1 c = 1 f = 1

1   2   "#ANT#Z"   p = 0.5
1   2   "#ANT#W"   p = 0.5

State 2 c = 1 f = 1
```

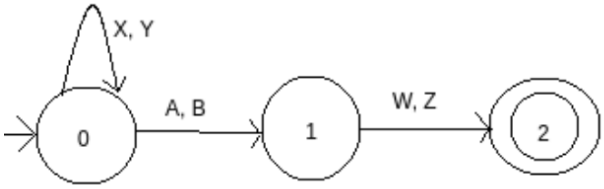

Figura 8. Autómata finito.

Una vez analizado iATROS, el siguiente paso es estudiar el *plugin* COPS. Internamente, COPS cuenta con los siguientes elementos: una carpeta `icons` para las imágenes que se utilizan en el *plugin*, una carpeta con las librerías necesarias para el funcionamiento del *plugin*, un archivo `plugin.xml` para definir algunos de los aspectos visuales y, por último y más importante, las fuentes, donde hay cinco paquetes: `codeshine`, `codeshine.preferences`, `codeshine.speech`, `codeshine.utils` y `codeshine.views`. La estructura completa puede verse en la Figura 9.

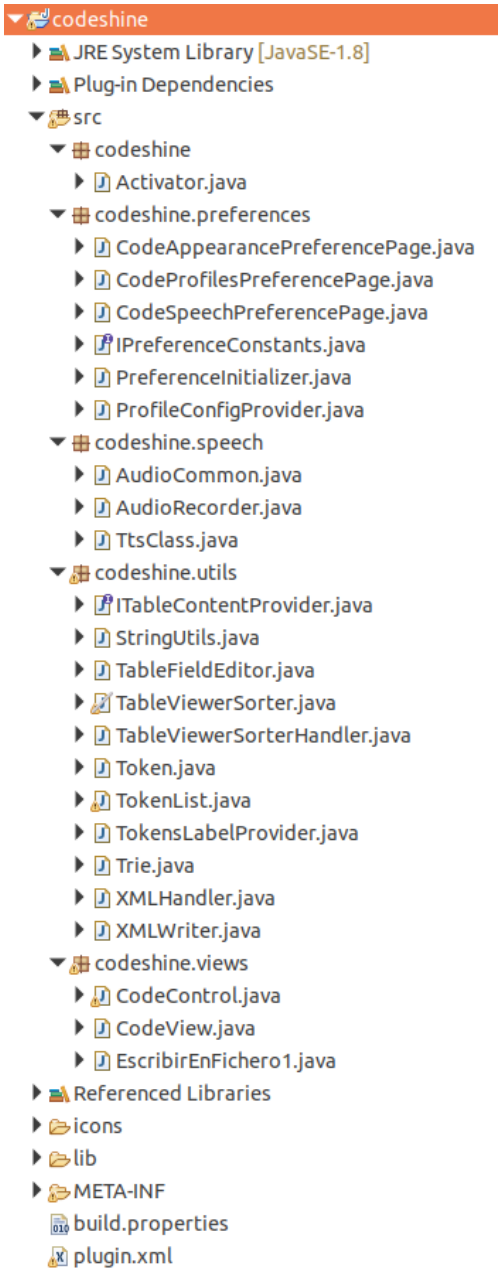

Figura 9. Carpetas y paquetes de COPS.

Comenzando por las fuentes, en el paquete `codeshine` está la clase `Activator`. En los *plugin* de Eclipse, la clase `Activator` es el punto de inicio; se carga en el arranque y es la clase que informa de que se trata de un *plugin* de interfaz de Eclipse, como sucede en el caso de COPS.

El siguiente paquete es el de `preferences`. En este paquete se encuentran las clases relacionadas con la gestión de preferencias y perfiles de COPS. La primera clase que analizar es `CodeAppearancePreferencePage`. En esta clase se define la página de preferencias de visualización del código. Podemos ver esta página una vez

ejecutado COPS, abriendo las preferencias, donde se puede elegir color de letra, de fondo o fuente, entre otros. Lo mismo sucede con `CodeSpeechPreferencePage`, en este caso para las preferencias de la síntesis de voz, donde es posible cambiar, por ejemplo, su tono. Para complementar el menú de preferencias se encuentra `CodeProfilesPreferencePage`. Esta clase es la que define la ventana en la que se puede guardar y cargar distintos perfiles, perfiles que permiten guardar la configuración de la sesión y cargarla en la siguiente. En caso de querer modificar un perfil, existe la clase `ProfileConfigProvider`, que permite precisamente esto. Una vez creados o editados, estos perfiles se guardan como archivos XML en la carpeta que decida el usuario. Además de estas clases, se encuentra la clase `PreferenceInitializer`, que es la encargada de cargar las opciones predeterminadas en el momento de iniciar el *plugin* y `IPreferenceConstans`, una interfaz que cuenta con todas las constantes que se pueden modificar en las preferencias de COPS.

El siguiente paquete es `speech`, donde se encuentran las clases relacionadas tanto con la síntesis de voz como con el reconocimiento. La primera clase es `TtsClass`. Esta clase es la que gestiona la reproducción de audio en COPS, apoyándose principalmente en FreeTTS[9]. FreeTTS es un paquete de recursos *open source* para la síntesis de voz completamente construido en Java muy usado para la síntesis de voz en este lenguaje de programación. Luego de esto, encontramos la clase `AudioCommon`, que cuenta con varios métodos utilizados para la gestión de audio. Esta clase proviene de Java Sound Resources[10], un paquete con multitud de recursos para el uso de la API de sonido de Java. Por último, tenemos la clase `AudioRecorder`. Esta clase es una de las más importantes para la resolución del problema planteado, ya que es donde se realizan varios procesos importantes en el reconocimiento de voz. La clase es una modificación de la original `AudioRecorder` que también se puede encontrar en Java Sound Resources. Inicialmente esta clase está preparada para la grabación de audio, pero en esta modificación se le han añadido varios atributos y métodos para que se realice también la preparación de ese audio que va a ser reconocido por iATROS. En el constructor se puede observar que, además de inicializar los parámetros necesarios para la grabación de audio, se están inicializando variables y procesos que luego serán importantes en el reconocimiento.

[9] https://freetts.sourceforge.io/
[10] http://jsresources.sourceforge.net/index.html

```
strPath= "/home/andres/eclipse-workspace-tfg/codeshine/iatros_cops/";
    strFilename = strPath+"cops.wav";

    // iAtros
    iatrosCeps=strPath+"raw2CC";
    iatrosOff=strPath+"iatros-run";
    outFile=new File("/tmp/cops.out");
    iatros = null;

    // Conversión de wav a raw

    convert=strPath+"wav2raw";
    // Se ejecuta el reconocedor iAtros
    try {
      iatros = Runtime.getRuntime().exec("/bin/bash "+iatrosOff);
    } catch (IOException e) {
        Runtime.getRuntime().exit(-1);
    }
```

Figura 10. Código de `AudioRecorder`

Como se puede ver en el fragmento de código mostrado en la Figura 10, se indica dónde y cómo se guardará el archivo de audio, se preparan las variables para la ejecución que se realizará después de los archivos `wav2raw` y `raw2CC` y finalmente se ejecuta `iatros-run`, que queda esperando a que exista un archivo `cops.CC` con el cepstrum necesario para realizar la conversión. Pasando a los métodos, la clase cuenta con un método `startRecording` para comenzar a grabar y otro `stopRecording` para detenerla. Cada vez que se comienza a grabar, se restauran los atributos necesarios y se crea una nueva instancia de *Recorder*, que permite realizar la grabación. Una vez vistos estos métodos, se puede pasar al método `performRecognition`. Este método es el que va a ir ejecutando, paso a paso, las acciones que se han descrito en la parte de iATROS para realizar el reconocimiento de voz. Primero ejecuta `wav2raw` para pasar el audio al formato deseado; tras esto, ejecuta `raw2CC` para obtener el cepstrum que necesita iATROS para el reconocimiento; por último, entra en un bucle, esperando a que el archivo `cops.out`

exista en la carpeta `/tmp` del sistema. Como se ha ejecutado `iatros-run` al invocar el constructor de la clase, `iatros-run` recibirá automáticamente el cepstrum generado y comenzará el reconocimiento, que en cuanto finalice dejará como salida el archivo `cops.out`. Una vez aparece el archivo, se lee su contenido y se escribe donde esté el cursor de texto en el código en el momento de finalizar la grabación. Todo este proceso se ilustra en la Figura 11.

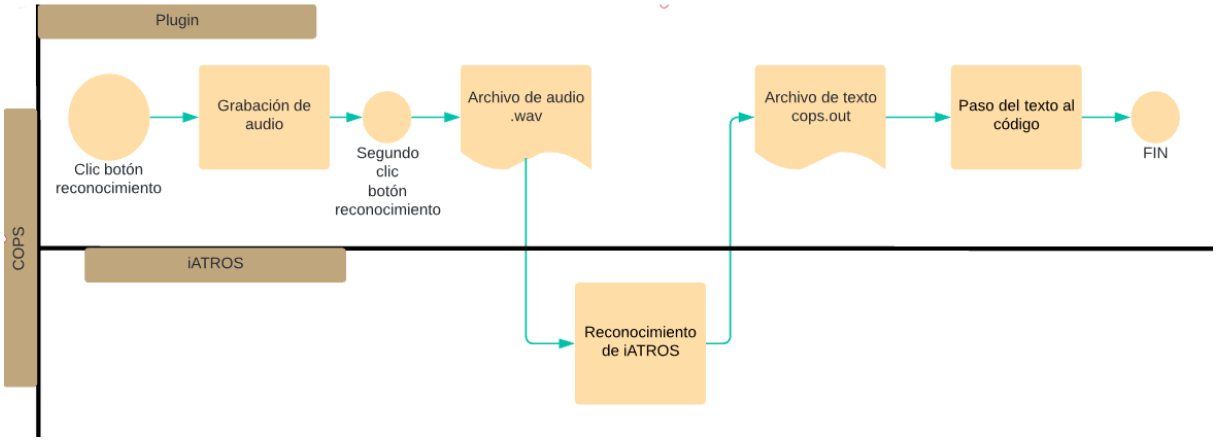

Figura 11. Flujo de COPS.

El siguiente paquete que estudiar es el paquete `utils`. En este paquete se encuentran las clases `Token`, `TokenList` y `TokenLabelProvider`. Estas clases ayudan a la gestión y visualización de, por ejemplo, la selección de perfiles de preferencias. Las demás clases, como `TableFieldEditor` y `TableViewerSorter`, tienen el mismo propósito: facilitar la gestión y visualización de datos para el *plugin*.

Por último, está el paquete `views`, donde están las clases necesarias para que la interfaz de COPS se muestre y ejecute correctamente. La primera clase a analizar es `CodeControl`. En el constructor de `CodeControl` se crea un *listener* (que se encargan de controlar los eventos, esperando a que el evento se produzca para realizar la acción que se defina). El *listener* es para el caso en el que se pulse una tecla, ya que COPS tiene atajos de teclado para algunas de sus funciones, como aumentar y disminuir el tamaño de la fuente, activar la síntesis de voz del código seleccionado y abrir las preferencias. Respecto a los métodos, primero encontramos los distintos métodos `set` para cada parámetro de la visualización, como el fondo y el color de la letra. Tras estos, el siguiente método a comentar es `speaker`. Este método

es llamado desde la clase `CodeView`, como se detallará más adelante, y es el encargado de preparar la síntesis de voz a través de los siguientes pasos:

- Se comprueba si se ha seleccionado algún texto en el editor de COPS, que será el texto para reproducir como audio.
- En caso afirmativo, se crea un archivo de audio `sinte.wav` vacío y una instancia de `EscribirEnFichero1`, para escribir la síntesis en el audio.
- Se pasa el texto a reproducir al método `toFile` de `EscribirEnFichero1` para escribirlo como un audio en `sinte.wav`.
- Finalmente se crea un `AudioClip` para reproducir el audio.

La siguiente clase a comentar del paquete es `EscribirEnFichero1`. Esta es la clase que hace las llamadas necesarias para realizar la síntesis de voz. Su constructor debe invocarse con la voz que se ha decidido utilizar para la reproducción del texto. Cuando se invoca, en el constructor se crea una instancia de la clase `Voice` (`Voice` es una clase del paquete de FreeTTS en la que se realiza el proceso de pasar de texto a voz). A esta instancia de Voice se le asigna una voz a través de una instancia de `VoiceManager`, a la que se le pasa el `String` de la voz que se desea, indicada anteriormente en el constructor, y se consulta si la voz está disponible para la reproducción. `VoiceManager` es también una clase del paquete de FreeTTS, en este caso para la gestión de voces.

En cuanto a los métodos, el primero es `toFile`. Este método llama al método `speak` de la instancia de `Voice` creada en el constructor, donde internamente se realiza la síntesis y reproducción de voz, y graba el resultado en el archivo que se le indique como parámetro. Además de este método, tiene otros como `listAllVoices`, con el que, a través de `VoiceManager`, da la lista de las posibles voces, o `getExtension`, para obtener la extensión de un archivo y por tanto su formato.

La última clase del paquete es `CodeView`, que implementa los métodos para que la interfaz funcione. El primer método a comentar sería `selectionChanged`; este método notifica al *listener* de la clase cuándo cambia la selección en el editor de Eclipse, de manera que cuando se cambia de archivo a editar, la pantalla clonada de COPS también cambia. Los siguientes métodos son `init` y `dispose`. El primero inicializa la interfaz de COPS y, además, crea un objeto de tipo `AudioRecorder`. (`AudioRecorder` es una clase del paquete `speech` de COPS que, como se ha

comentado anteriormente, ejecuta la grabación y los procesos necesarios para que el archivo de audio pueda ser reconocido y pasado a texto). `dispose`, por su parte, termina el `AudioRecorder` y los *listener* que tiene la interfaz. El siguiente método es `makeActions`, y es el encargado de crear los botones de la interfaz de COPS. Estos botones en realidad son objetos de tipo `Action` que se crean para cada acción, y a los que se les asigna una imagen, un texto y la acción que deben realizar. Por ejemplo, para las acciones de aumentar y disminuir texto, simplemente se llama al método correspondiente de la clase `CodeControl`. Sin embargo, en la acción del reconocimiento de voz, que es más compleja, hay más código. Para empezar, al inicio de la ejecución se comprueba, mediante una variable de control, el estado de la acción, ya que la acción de reconocer la voz pasa por dos fases: el comienzo de la grabación con el primer clic y el final de ésta y su procesamiento con el segundo clic. En la primera fase se llama al método `startRecording` de la clase `AudioRecorder` y cambia la imagen del icono para representar visualmente que se ha comenzado la grabación, además de actualizar la variable de control, con la que se controla la fase actual. En la segunda fase, se vuelve a la imagen original, actualiza la variable de control y se llama a los siguientes métodos de `AudioRecorder`:

- `stopRecording`: Detiene la grabación.
- `performRecognition`: Inicia el proceso de transformación del audio, preparándolo para el reconocimiento.
- `restoreRecording`: Restaura el estado de `AudioRecorder`, dejándolo preparado para la siguiente grabación.

Finalmente, con el método `updateText` de `codeControl` se escribe el texto en el editor de Eclipse.

3.4 Identificación y análisis de soluciones posibles

Dado los problemas planteados y el sistema ya estudiado, surgen distintas soluciones.

Por un lado, está la parte de iATROS. Dado que el sistema ya cuenta con todas las palabras reservadas de Java y algunos identificadores [5], se ampliará el número de estos últimos. Para ampliar el vocabulario de identificadores habrá que elegir una

muestra de los más comunes y, con la herramienta Eutranscribe, realizar la transcripción. Para la navegación por menús será igualmente necesario añadir los nuevos términos al vocabulario de iATROS. Una vez hecho esto, se creará el modelo de lenguaje para la navegación por menús y adaptará el sistema para hacer uso de él. Finalmente, en caso de que sea necesario, se deberá modificar el modelo de los identificadores por falta de soporte a algún termino.

Respecto al *plugin* en sí, se quiere poder navegar por los menús de Eclipse a través de ordenes de voz para evitar, en la medida de lo posible, el uso del ratón o el teclado. Para este problema surgen varias soluciones.

La primera, más cercana al diseño que ya tiene el *plugin,* es añadir un segundo botón, como el del micrófono que ya existe para dictado de voz, que al primer clic comience la grabación y al segundo la finalice, y tras ello la procese y genere una salida. Una vez se tenga esa salida, se manejará para abrir los menús que se hayan indicado. El movimiento por los menús se realizará con una clase que traduce las órdenes dadas en atajos de teclas para navegar. El principal inconveniente de esta solución es que, en el momento de dar la orden de voz, el usuario debe saber cuál es el nombre del menú y los submenús por los que desea navegar. En caso de que no se reconozca la orden completa, el avance por los menús podría quedarse a medio camino, con lo que el usuario debería de terminar la acción con el ratón o comenzar desde el principio el reconocimiento de la orden, suponiendo un inconveniente para el usuario.

Una solución cercana a la primera, pero más precisa, es contar con el mecanismo de navegación por menús y, además, añadir la posibilidad de ejecutar las opciones más comunes evitando tener que navegar por el menú completo. Estas acciones son, por ejemplo, ‘Search, ‘New project’ o ‘Run as’, entre otras.

Una tercera opción para la navegación por menús es la navegación en tiempo real. Con esta opción, al realizar el primer clic comienza una grabación de unos segundos donde el usuario dice el primer menú que quiere abrir. La grabación finaliza y se procesa como en los casos anteriores, solo que esta vez el usuario no hace un segundo clic para finalizarla. Mientras esta grabación se procesa, otra nueva se ha abierto, y el usuario, ya con el menú en pantalla y las opciones que tiene para elegir, decide en qué submenú entrar y se repite el proceso hasta llegar a la acción deseada. Con esta opción se solventa el problema de que el usuario tenga que saber todos los

submenús que necesite utilizar, pero surge un problema mayor. El usuario no sabe cuándo el *plugin* comienza y finaliza las grabaciones, y aunque de alguna manera se mostrase en pantalla, no deja de ser anti-intuitivo. Esta solución intenta imitar las opciones más populares actualmente, donde se reconoce la voz del usuario en tiempo real, pero debido al funcionamiento de iATROS, no existe una forma óptima de realizar algo similar con COPS. El resultado más probable es audios cortados a mitad de palabra del usuario, provocando que iATROS no pueda realizar el reconocimiento y frustración del usuario porque no sabe cuándo tiene que dar la orden.

3.5 Solución propuesta

Vistas las distintas soluciones posibles, se ha decidido por una parte aumentar el vocabulario de iATROS añadiendo los menús de Eclipse y los identificadores más comunes. Respecto a la navegación por menús, se ha optado por un sistema de dos clics, similar al del reconocimiento de voz, pero con atajos a las opciones más populares, como ‘New’ o ‘Search’, para así evitar una complejidad excesiva a la hora de navegar por Eclipse.

Tras ser elegidas las soluciones a implementar, se van a especificar los requisitos para más tarde realizar una implementación y unas pruebas adecuadas. Para empezar, un requisito de software define qué tiene que hacer el sistema y bajo qué circunstancias debe operar. Además de este tipo de requisito, existen otros niveles como los requisitos de negocio, referidos a motivos económicos, y los de usuario, enfocados a las acciones que necesitan realizar [11]. Se comenzará con la especificación de requisitos del sistema de navegación por menús del *plugin*.

La nueva funcionalidad de navegación y uso de menús de Eclipse trae consigo varios requisitos en todos los niveles. Empezando por el nivel de negocio, pese a que COPS no es un producto en el mercado, se le puede aplicar la misma lógica. En este caso, una nueva funcionalidad y además de gran utilidad como la de uso de menús por voz, es probable que aumente la satisfacción del usuario a la hora de manejar el producto. Así, el requisito puede expresarse como:

- Aumentar la satisfacción de los usuarios gracias a una nueva función de accesibilidad para Eclipse.

No obstante, al tratarse de un objetivo, no puede ser verificado hasta un tiempo después de su uso, ya que hasta entonces no hay respuesta por parte del usuario. Además, aunque parece claro que una nueva funcionalidad puede aumentar la satisfacción del usuario, en el caso de que ésta fuera errática provocaría lo contrario, por lo que no se puede saber a ciencia cierta si va a cumplirse.

El siguiente nivel es el de usuario. Aquí los requisitos se especifican mayormente a alto nivel, sabiendo que no se van a realizar tal y como se redactan, sino que hay más subprocesos que en conjunto satisfacen este requisito. En este caso se especifica un único requisito que lo engloba todo:

- Uso de menús a través de órdenes de voz.

En este caso, el requisito se podrá verificar fácilmente, comprobando si la nueva opción funciona.

El siguiente y ultimo nivel es el de requisitos de software; es el más complejo y se divide en dos, requisitos funcionales y requisitos no funcionales. Los requisitos funcionales definen una función del software a partir de una entrada, el comportamiento con esa entrada y una salida. Por otra parte, los requisitos no funcionalidades definen propiedades o características del sistema. Ambos pueden ser verificados tras la implementación.

Para la identificación de los requisitos software se debe seguir un proceso de ingeniería de requisitos. Este proceso se divide en cinco fases más una inicial, que no se considera dentro del proceso [12]:

0. Estudio de factibilidad
1. Elicitación de requisitos
2. Definición de requisitos
3. Especificación de requisitos
4. Validación de requisitos
5. Gestión de requisitos

Aunque aparezcan listados, exceptuando el primero y el último se trata de un modelo de espiral, es decir, se pasa por los mismos puntos más de una vez, hasta llegar a un consenso con los requisitos, tal y como muestra la Figura 12.

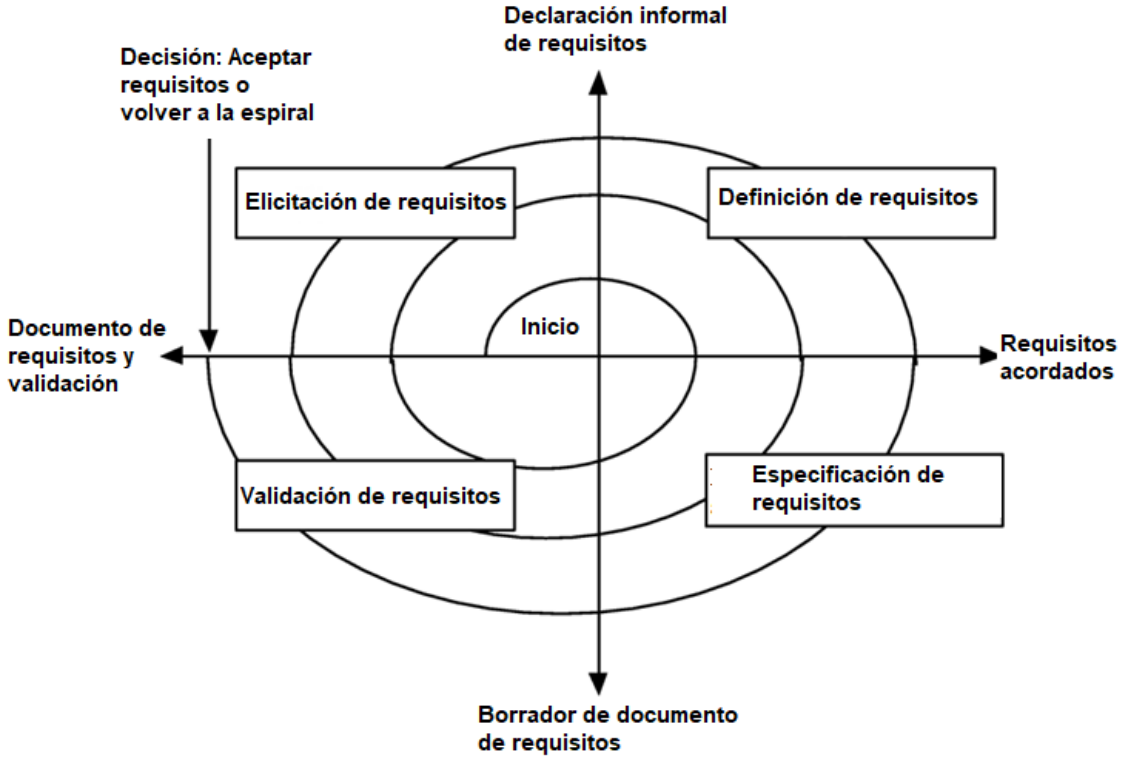

Figura 12. Espiral de requisitos.

En el estudio de factibilidad se estudia si es posible implementar las demandas del usuario. En este caso se da por factible la implementación de las órdenes de voz para uso de menús en Eclipse.

El siguiente punto ya entra en el proceso y es la elicitación de requisitos. En este punto se busca entender los objetivos y motivos para el desarrollo propuesto. Este proceso de elicitación se suele realizar identificando los *stakeholders*, que son elementos que posiblemente estén relacionados con el sistema y pueden ser tanto usuarios como el propio cliente. Tras esto, deberá averiguarse qué necesidades tienen sobre éste [12]. En este caso podrían considerarse como tal tanto los usuarios con diversidad funcional como los usuarios sin ella.

Luego de la elicitación, viene la definición de requisitos; estas definiciones van normalmente orientadas al cliente, es decir, describen la funcionalidad del sistema y sus restricciones de uso, pero siempre desde una óptica del comportamiento externo, sin entrar en tecnologías ni funcionamiento interno. En este punto ya comienza la

división entre requisitos funcionales y no funcionales. Para la nueva funcionalidad se han definido los siguientes requisitos:

1. Requisitos funcionales:
    a. La grabación de audio para el uso de menús comienza en el primer clic en el botón de uso de menús.
    b. La grabación finaliza y se abre el menú correspondiente en el segundo clic al botón de menús.
    c. Solo se puede hacer un reconocimiento de voz simultáneo.
    d. Si el audio no es reconocido por completo, se utilizará la parte reconocida.
    e. Si en el audio nada es reconocido, se notificará por pantalla.

2. Requisitos no funcionales:
    a. El audio se almacenará en formato WAV.
    b. El sistema responderá, tras finalizar la grabación, en menos de 5 segundos.
    c. Tras haberse realizado el reconocimiento de voz, no quedará ningún archivo de audio ni texto por eliminar.

Tras la definición, viene la especificación de requisitos. En esta fase se busca añadir detalle a la definición, expresando qué se necesita de los desarrolladores. En este caso se ha optado por una especificación en lenguaje natural en aquellos requisitos en los que es necesario un mayor nivel de detalle.

1. Requisitos funcionales:
    a. Cuando el usuario hace clic por primera vez en el botón correspondiente al uso de menús, debe iniciarse la grabación de audio.
    b. Cuando el usuario hace clic por segunda vez en el botón correspondiente al uso de menús, la grabación debe finalizar, guardarse y ser reconocida por iATROS para obtener la salida de texto y abrir el menú indicado a través de atajos de teclado.
    c. Cuando el usuario hace clic por primera vez en el botón correspondiente al uso de menús o al de reconocimiento de voz para escribir código, el botón que no ha sido clicado debe quedar bloqueado

hasta que no se realice el segundo clic al botón inicialmente pulsado y, por tanto, finalice el proceso de reconocimiento.

d. Cuando el usuario hace clic por segunda vez en el botón correspondiente al uso de menús, si el audio no ha sido reconocido en su totalidad se deberá usar la parte reconocida para navegar.
e. Cuando el usuario hace clic por segunda vez en el botón correspondiente al uso de menús, si el audio no ha sido reconocido, se notificará por pantalla mediante una ventana que no se ha realizado el reconocimiento correctamente.

2. Requisitos no funcionales:
    a. El audio se almacenará en formato WAV.
    b. Cuando el usuario hace clic por segunda vez en el botón correspondiente al uso de menús y el audio es correctamente reconocido, el audio debe ser procesado y el menú indicado abierto en menos de 5 segundos. En caso de error el tiempo debe ser también menor a 5 segundos.
    c. Cuando el usuario hace clic por segunda vez en el botón correspondiente al uso de menús y es procesada la salida correspondiente, se deberán eliminar todos los archivos que puedan quedar a raíz del proceso, que son:
        - El archivo de formato WAV.
        - El archivo de formato RAW.
        - El archivo con el cepstrum.
        - El archivo con la salida de iATROS.

El último paso del proceso iterativo es la validación. En este punto se quiere demostrar que los requisitos definen el sistema que el cliente desea, es decir, certificar que es una descripción correcta del desarrollo que se va a realizar. El problema en este punto radica en saber con qué validar estos requisitos. Normalmente este proceso se realiza con los *stakeholders* identificados anteriormente. En las validaciones se comprueba si los requisitos presentados son entendibles, válidos,

consistentes o verificables, entre otras comprobaciones a realizar [13]. En este caso, se ha realizado un *mockup* para la validación.

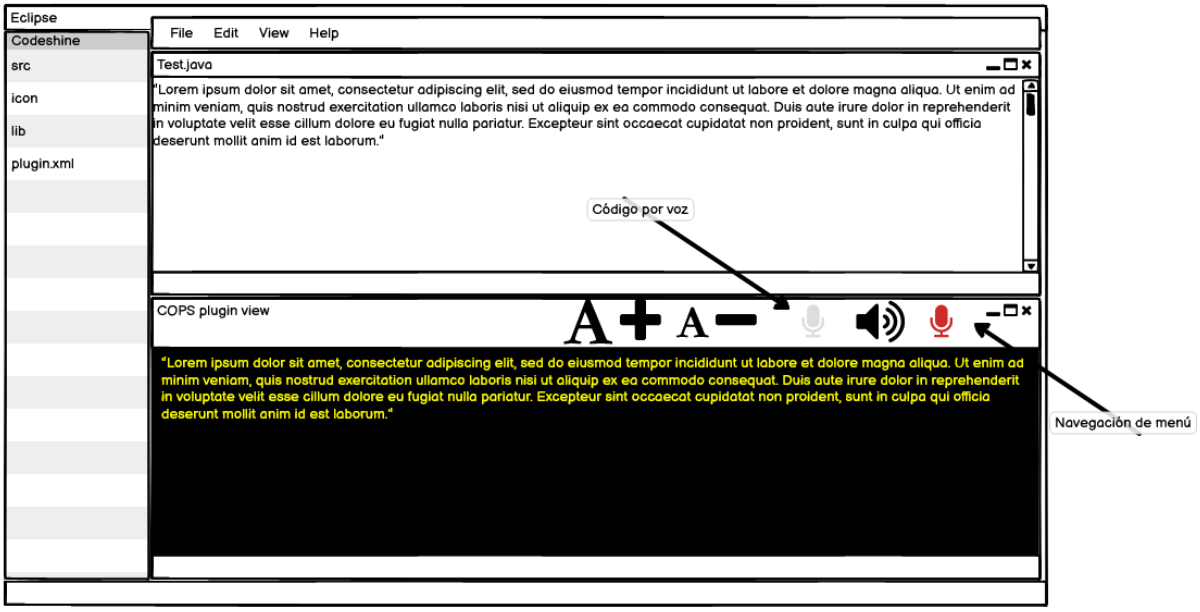

Figura 13. Mockup de los requisitos implementados 1.

En el *mockup* de la Figura 13 puede verse cómo en el momento en el que se clica el botón de grabación, en este caso el de uso de menús, el botón para escribir código se deshabilita. De esta manera, se puede validar el requisito por el cual no se debe poder clicar uno de los dos reconocimientos de audio hasta que se haya realizado el procedimiento completo del que se haya clicado inicialmente.

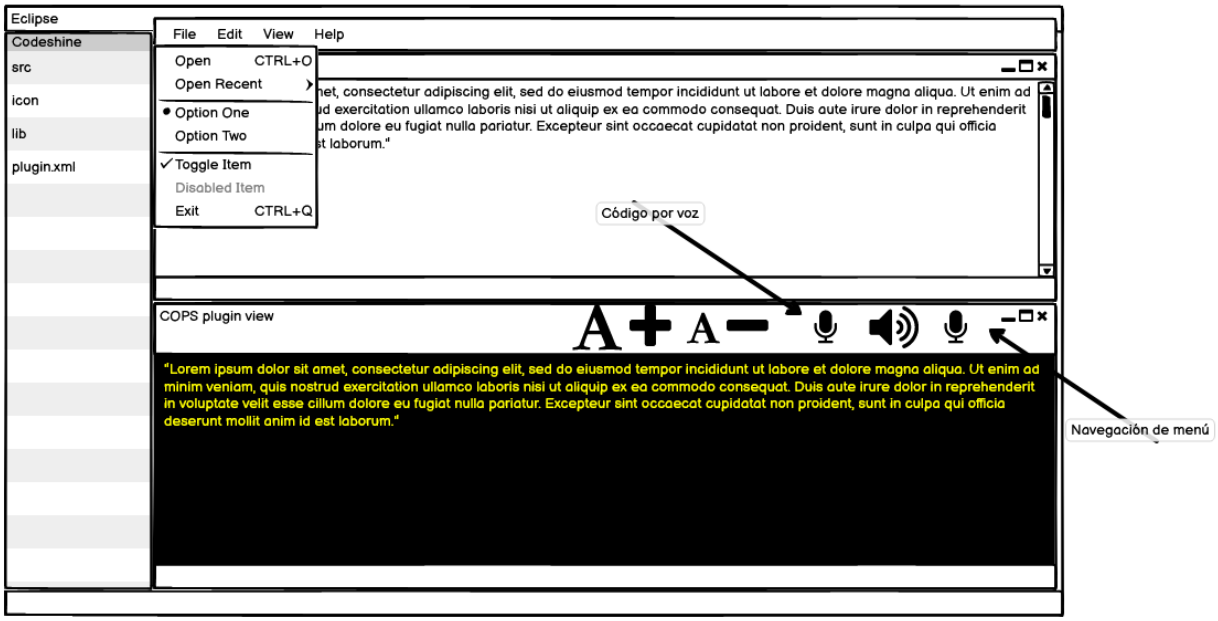

Figura 14. Mockup de los requisitos implementados 2.

Por otra parte, en el *mockup* de la Figura 14 puede observarse cómo, una vez finalizada la grabación y el procesamiento de la orden, ambos botones están habilitados de nuevo y, además, se ha ejecutado la orden, abriendo el menú de 'File' en el caso del ejemplo.

Por último, y ya fuera del proceso iterativo de identificación de requisitos, está la gestión de requisitos. Durante las iteraciones del proceso descrito, incluso después, en la fase de implementación, los requisitos siempre evolucionan, ya fuese por un mejor entendimiento del problema o por cambios en los objetivos. Si estos cambios no están controlados, el coste de adaptarse a ellos puede ser muy alto. Es por eso por lo que ha de existir una trazabilidad entre requisitos; de esta forma, en el caso de que un requisito cambie se puede corregir el error que puede provocar en requisitos relacionados y modificar el documento de requisitos sin tener que empezar de nuevo cuando haya demasiados cambios descontrolados [13]. En este caso, al tratarse de pocos requisitos, no es necesario este tipo de gestión.

Vistos los requisitos de la navegación por menús, se va a pasar a comentar los de iATROS.

Para la parte de iATROS también se pueden definir requisitos tanto funcionales como no funcionales:

1. Requisitos funcionales:
    a. Se reconocen audios con nuevos identificadores.
    b. Se reconocen audios con nombres de menús de Eclipse.

2. Requisitos no funcionales:
    a. Se incluyen los nombres de los menús de Eclipse.

Tras la definición de los requisitos, viene la especificación de éstos. En esta fase se va a añadir detalle a la definición, expresando qué se necesita de los desarrolladores. También se ha optado por una especificación en lenguaje natural.

1. Requisitos funcionales:
    a. Dado un audio en formato WAV en el que se mencionan uno o más menús de Eclipse, aplicando las transformaciones necesarias al archivo, se reconoce el o los menús mencionados en el audio.
    b. Dado un audio en formato WAV en el que se mencionan uno o más identificadores que no estuvieran incluidos anteriormente, aplicando las transformaciones necesarias al archivo, se reconocen el o los identificadores mencionados en el audio.

2. Requisitos no funcionales:
    a. Se incluyen en el vocabulario de iATROS los menús disponibles en la versión de Eclipse 2020-06, excluyendo aquellos que puedan añadirse con la instalación de software externo.

Una vez definida la solución y especificados sus requisitos, se puede pasar al diseño e implementación de ésta.

4. Diseño de solución

Ya con el sistema de iATROS y el de COPS analizados, y especificados todos los requisitos para la solución que se ha propuesto, se van a analizar cómo se va a llevar a cabo las soluciones para la nueva funcionalidad de COPS y de la ampliación de vocabulario de iATROS.

4.1 Acciones de actualización

Para llevar a cabo la solución propuesta existen dos bloques de trabajo:

- Ampliación de vocabulario: En este primer apartado se va a trabajar directamente con iATROS y sus archivos internos, desde su vocabulario a sus modelos de lenguaje.
- Añadir funcionalidad de uso de menús a través de reconocimiento de voz: Esta solución se desarrollará tras la primera, ya que, si se hacen pruebas durante el desarrollo, va a ser necesaria la ampliación del vocabulario para comprobar el funcionamiento. Para el desarrollo de esta funcionalidad se van a hacer cambios mayoritariamente en los paquetes `codeshine.speech` y en `codeshine.views`, para implementar tanto la parte de interfaz como la de funcionamiento interno.

4.2 Diseño detallado

En el caso de la primera solución a implementar, se debe navegar en las carpetas de iATROS, que se instalan en el *workspace* de Eclipse. En la ruta `workspace/codeshine/iatros_cops/model` se encuentra el archivo `cops.lx` , el cual debe modificarse para añadir los nuevos términos que van a ser reconocidos. Como se puede ver en la Figura 15 el documento de texto consta de tres columnas. La primera indica el `String` que se va a reconocer, ya sea un carácter único o una palabra. La segunda columna marca los pesos de cada respuesta, siendo 1 el

máximo. El peso cambia de 1 en el momento en el que se considera que existe más de una pronunciación o forma de expresar un carácter o una palabra. En ese caso, los pesos se dividen entre las distintas opciones, sumando entre ellos 1.

```
}        0.5    z e @ a r y a b e
}        0.5    z i e @ a y a b e
default  1.0    d e f a u l t
do       1.0    d u
double   1.0    d o u b e l
else     1.0    e l s e
extends  1.0    e s t e n d s
final    0.5    f a i n a l
final    0.5    f i n a l
finally  0.5    f a i n a l i
finally  0.5    f i n a y i
float    1.0    f l o a t
for      1.0    f o r
```

Figura 15. Parte de `cops.lx`.

Es aquí donde hay que añadir todo el nuevo vocabulario que se desea incluir en iATROS, tanto de identificadores como de menús de Eclipse. Para ello, con la herramienta Eutranscribe se realizará la transcripción de los términos originales y se añadirá a los vocabularios.

Una vez ampliado el vocabulario, se pasará los modelos de lenguaje. Como se ha comentado anteriormente, iATROS trae cuatro modelos para las cuatro categorías de mensaje que tiene: Identificadores, caracteres, números y cadenas. Es por ello por lo que, para los menús, va a ser necesario un nuevo modelo. En este modelo, cada estado va a corresponder a un menú del que se podrá ir a los submenús y así sucesivamente hasta llegar al último. Respecto al modelo de los identificadores, en caso de que las nuevas incorporaciones no estén cubiertas por el modelo, se deberá modificar el fichero también. Hecho esto el siguiente paso será la configuración de iATROS. Dado que el sistema de navegación de menús es independiente del de reconocimiento de voz para código, se debe crear un sistema paralelo al original para poder realizar ambos reconocimientos en una misma ejecución.

La segunda solución a implementar, añadir la funcionalidad de uso de menús por voz en el *plugin*, va a realizarse mayoritariamente en los paquetes `codeshine.speech`, `codeshine.utils` y `codeshine.views`.

Respecto al paquete `speech`, va a haber que modificar la clase `AudioRecorder`. `AudioRecorder` es la clase que, en su constructor, activa iATROS; además, cuenta con el método `performRecognition`, el cual es llamado en el momento que se hace el segundo clic en el botón de reconocimiento de voz para el dictado de código, y que, en diferentes fases con distintos ejecutables del propio iATROS, transforma el archivo de audio original en el cepstrum que el reconocedor necesita para procesar la entrada. Finalmente, el mismo método lee la salida de iATROS para escribirla en el editor de Eclipse. Primero, en el constructor de esta clase se va a añadir la llamada al nuevo sistema paralelo de reconocimiento para la navegación por menús. Tras esto, se va a crear un nuevo método `performNavigation`, que será llamado cuando se haga el segundo clic en el botón de navegación de menús de Eclipse. Este método, similar a `performRecognition`, realizará las distintas llamadas a los ejecutables de iATROS para la transformación del fichero de audio. Una vez se ha obtenido el texto, `performNavigation` lo enviará a una nueva clase, `Navigation.java`.

La clase `Navigation` va a ubicarse en el paquete `codeshine.utils` y va a constar de una serie de sentencias `switch` con las que se irán abriendo los menús deseados. El primer 'nivel' de sentencias va a contar con los menús básicos de Eclipse, que se pueden ver en la Figura 16, además de los atajos a las opciones más utilizadas como 'Run as' o 'New'. En caso de que una de estas opciones sea la elegida, el `switch` llamará al método correspondiente que, ejecutando un atajo de teclado, abrirá la opción deseada. En caso de que se haya ordenado abrir un submenú y no una opción directa, el `switch` irá recorriendo niveles hasta llegar al menú indicado y ejecutará el atajo de teclado necesario para llegar al punto indicado. En caso de que ninguna palabra del texto coincida con los casos del `switch` significará que no se ha reconocido correctamente, por lo que se mostrará un mensaje indicando que ha fallado el reconocimiento. Para evitar un código ilegible debido a la gran cantidad de menús, las sentencias `switch` no irán anidadas, sino que se creará un método para cada submenú, al que se le llamará con el texto reconocido menos la primera palabra, que será el menú anterior.

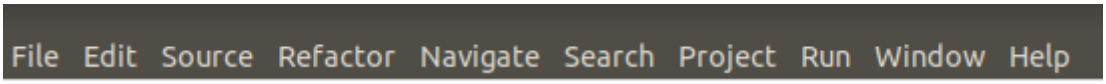

Figura 16. Menús de Eclipse.

Respecto al paquete `codeshine.views`, va a ser necesario cambiar la clase `CodeView`. La clase `CodeView` es la encargada de, por un lado, mostrar los botones en la interfaz de Eclipse y, por el otro, de añadir la funcionalidad a esos botones, por lo que será aquí donde se añadirá la nueva acción para el uso de menús. Lo primero a realizar será crear una variable de control para comprobar si se ha realizado el primer clic o no, de manera similar al funcionamiento de la acción ya existente para el reconocimiento de voz. En caso de que no se haya clicado aún, al hacer clic se llamará al método de grabación de `AudioRecorder` y cambiará el icono para visibilizar que se está grabando. Como se ha comentado en los requisitos, una vez haya comenzado la grabación, el otro botón de reconocimiento debe deshabilitarse. Para esto, a la variable de control que se ha definido se le añadirá un tercer estado, donde una de las acciones quedará inhabilitada. Una vez se realice el segundo clic, la imagen volverá a ser la original, se restaurarán las variables de control y se llamará al método de parar grabación de `AudioRecorder`, seguido de `performNavigation`, el método que se ha definido para la navegación por menú. Finalmente se restaurará el estado del reconocedor.

Con este diseño, se obtendrá el funcionamiento deseado.

5. Desarrollo de la solución propuesta

Una vez planteado el diseño de la solución, se procede a su desarrollo.

Comenzando por iATROS, primero se ha realizado un listado en un documento de texto con los identificadores seleccionados y los menús de Eclipse. Tras esto, con la ejecución de Eutranscribe se ha obtenido la transcripción de todos los términos, y ésta se ha ido añadiendo al fichero `cops.lx`. Tras añadir los términos necesarios al vocabulario, se ha pasado al desarrollo del nuevo modelo de lenguaje. El árbol de menús de Eclipse es muy extenso, con un gran número de menús dentro de más menús, por lo que se ha decidido añadir únicamente los menús principales y el submenú siguiente (como puede verse en la Figura 17), además de las opciones directas. Tanto las opciones directas como los menús principales están en el estado 0. Desde este estado 0, en caso de escogerse una opción directa se pasaría al estado final. Luego, para los menús principales existiría una transición a un nuevo estado con sus submenús, desde los cuales se pasaría ya al estado final.

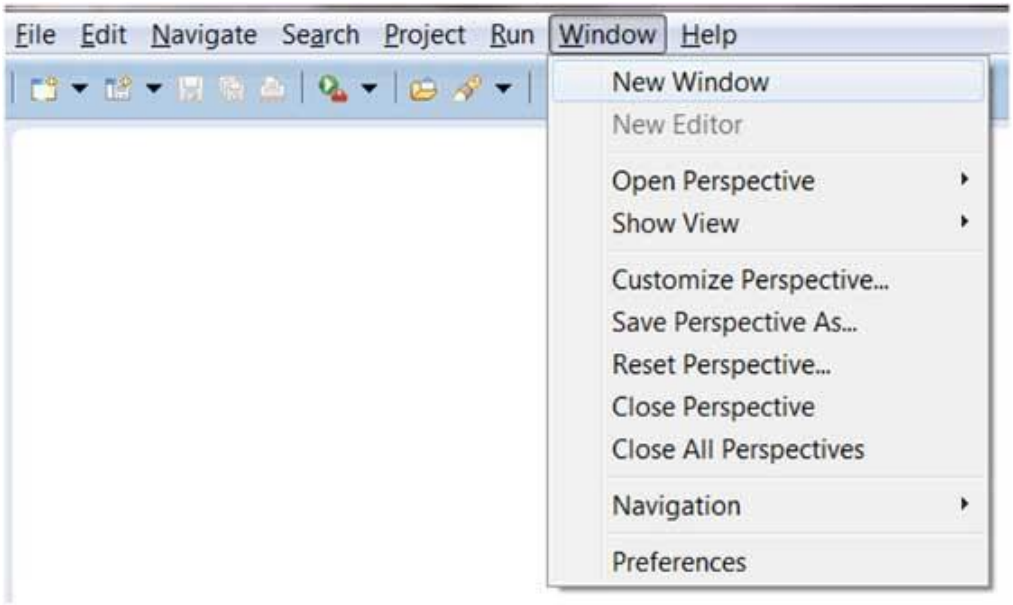

Figura 17. Menús de Eclipse.

Una vez está tanto el vocabulario como el modelo listo, se debe de configurar el sistema de iATROS. Como se ha comentado, va a ser necesario un sistema paralelo, esto es, un nuevo `iatros-run` y todo lo que lo acompaña. Para comenzar, se han creado `iatros-run_menu`, `wav2raw_menu`, `raw2CC_menu`, `file_menu.CC` y

`cops_menu.conf`. Estos archivos van a realizar prácticamente las mismas tareas que los originales, pero modificando ligeramente su código interno para adaptarlo a la nueva entrada. Dado que el nuevo archivo de audio se llamará `cops_menu.wav`, en el fichero `wav2raw_menu` se ha modificado el nombre tanto de la entrada como de la salida para que coincida con el audio inicial. Dicho cambio conlleva también cambiar el fichero `raw2CC_menu` de la misma manera, cambiando la entrada de audio a `cops_menu.raw` y la salida a `cops_menu.CC`. El siguiente fichero modificado es `file.CC`. Este fichero únicamente indica dónde y cuál es el archivo con el cepstrum, por lo que únicamente se ha cambiado el nombre de ese archivo a `cops_menu.CC`. Tras esta, la siguiente modificación ha sido en el archivo de configuración `cops_menu.conf`. En este fichero primero se ha indicado la dirección de `cops_menu.CC`. Tras esto, se han cambiado los modelos para que use el nuevo modelo `menú.gr`. Por último, en `iatros-run_menu` se han realizado las correspondientes correcciones en los nombres de los archivos, siguiendo en la línea de los cambios anteriores.

Una vez finalizadas las modificaciones en iATROS, queda la parte del *plugin* de Eclipse.

Como se ha comentado en el diseño detallado de la solución, primero se va a crear una clase `Navigation`. Esta clase tiene un método `Navigate`, el cual será llamado cuando exista el archivo `cops.out` y se le pasa como parámetro un `String` con el texto reconocido. El método consta de una instrucción `switch` en la cual se lee la primera palabra del texto reconocido. En caso de que sea una opción directa de las disponibles, el `switch` llama al método `play`, en el que se simulan las pulsaciones de teclado necesarias para abrir la opción. En caso de que la primera palabra del `String` sea un menú, se pasa a un método con otro `switch` dentro, pasando como parámetro el `String` inicial menos la palabra leída. En una variable se guarda la pulsación de teclado necesaria para abrir el primer menú indicado y se repite el proceso hasta que no queden más palabras, cuando se llama al método `play` y se abren los menús indicados. En caso de que en el primer `switch` no se encuentre una opción directa o un menú principal, se considera como error y se muestra una ventana indicando que el reconocimiento ha fallado. Como en el modelo de lenguaje, se ha optado por solo incluir menús principales y sus submenús, además de opciones directas.

Tras esto, solo quedan las modificaciones a las dos clases que se han mencionado antes.

La primera clase por modificar es `AudioRecorder`. Lo primero que se ha hecho en esta clase es añadir la ejecución de `iatros-run_menu` en el constructor; de esta manera ya están ambos sistemas corriendo. Tras esto se ha escrito el método `performNavigation`, el cual ejecuta las llamadas a `wav2raw_menu` y `raw2CC_menu`, como hace `performRecognition`. Una vez existe la salida, se lee y se llama al método `Navigate` de la clase `Navigation`.

Por último, se ha modificado la clase `CodeView`. Primero se ha creado una variable de control para las comprobaciones necesarias para las dos acciones de reconocimiento. Tras esto, se ha añadido, en el método `makeActions`, una nueva acción para la navegación por menús. En el cuerpo de la nueva acción hay una sentencia `if`. En caso de que la variable de control esté a 0 (ningún reconocimiento en proceso) inicia la grabación a través de `AudioRecorder` y cambia la imagen tanto suya como la del botón de dictado de código. En caso de que la variable sea 1 (reconocimiento de menú en curso) finaliza la grabación e inicia el reconocimiento, volviendo a las imágenes originales ambos botones. Por último, en caso de que la variable sea 2 (dictado de código en curso) no se hace nada, ya que la acción está bloqueada. Para la acción de reconocimiento para código se ha realizado una modificación similar en la sentencia `if` para tener el mismo comportamiento con la variable de control.

El resultado visual es el de la Figura 18.

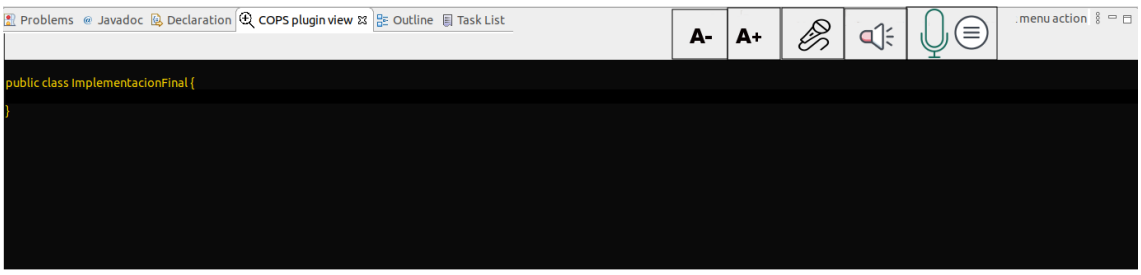

Figura 18. Nueva interfaz de COPS.

6. Pruebas

Las pruebas en el software son un factor muy importante a la hora de determinar la calidad de éste. Para realizar las pruebas del desarrollo, se va a seguir el proceso de prueba común adaptado en parte a las condiciones de este desarrollo.

Las cuatro partes de un proceso de prueba son [14]:

- Pruebas de unidad
- Prueba funcional o de integración
- Prueba del sistema
- Prueba de aceptación

Las primeras pruebas para realizar son las pruebas de unidad. En estas pruebas se comprueba que cada módulo funciona individualmente. El primer módulo de estas pruebas es la clase `Navigation`, encargada de, a partir del texto, abrir el menú o la opción correspondiente. Las pruebas diseñadas para este módulo son de caja blanca, es decir, fijándonos en el código fuente. Dado que existen demasiados caminos, se han preparado cinco pruebas que cumplen todos los posibles escenarios:

- Texto vacío o con término que no son menús ni opciones: En este caso, dado que no se encuentra ningún término que coincida con un menú o una opción directa, se debe mostrar una ventana emergente con un mensaje de que no se ha realizado el reconocimiento correctamente, como se ve en la Figura 19.

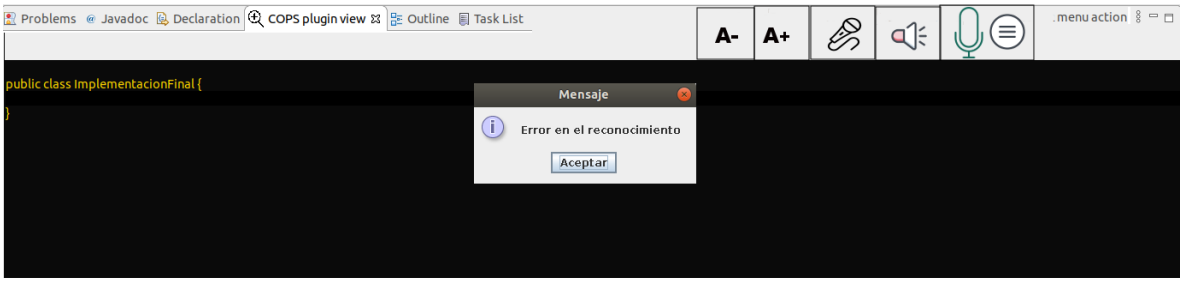

Figura 19. Mensaje de error.

- Texto con reconocimiento a mitad u orden de voz para abrir menú que no finaliza en ninguna opción. En este caso, puede ser que el usuario solo quiera abrir el menú para ver qué opciones tiene para elegir o que el audio no sea reconocido al completo y la orden se quede a medias. En ambos casos debe suceder lo mismo, que se abra todo lo que se haya reconocido. El módulo procede correctamente.

- Texto con opción directa. La entrada de texto que recibirá el módulo será únicamente una opción específica, no un conjunto de menús. En caso de recibir esto, se debe abrir la opción solicitada. El módulo procede correctamente.
- Texto con menú y opción al final que no que está dentro del menú anterior. En este caso, el usuario menciona a qué menús quiere acceder y finalmente dice una opción que no está en ese menú, sea esta una de las opciones directas o no. En este caso se abrirán todos los menús, pero no la opción, ya que no se encuentra dentro del último menú. El módulo procede correctamente.
- Texto con menú y opción al final que sí que está dentro del menú anterior. En este caso, la opción final está dentro del menú anterior mencionado, por lo que debe abrirse la opción mencionada. El módulo procede correctamente.

Con esta batería de pruebas se han podido verificar dos requisitos funcionales:

- Si el audio no es reconocido por completo, se utilizará la parte reconocida.
- Si en el audio nada es reconocido, se notificará por pantalla.

La siguiente prueba de unidad se realizan a la clase `AudioRecorder`. En esta clase se quiere comprobar si el método `performNavigation` funciona según lo esperado. Se realiza la siguiente prueba:

- Existiendo un archivo de audio formato WAV, tras ejecutar `performNavigation`, el archivo de audio ha sido eliminado y ha quedado el archivo `cops_menu.CC`. El módulo procede correctamente.

Las últimas pruebas de unidad son para iATROS. En estas pruebas se quiere comprobar si dado un cepstrum con nuevos y viejos términos de su vocabulario, es capaz de reconocer todo el contenido.

- Dado un archivo cepstrum proveniente de un audio con tanto nuevos términos como los originalmente implementados, se reconoce el contenido de éste. El programa procede correctamente en la mayoría de los casos.
- Dado un archivo cepstrum proveniente de un audio con solo nuevos términos, se reconoce el contenido de éste. El programa procede correctamente en la mayoría de los casos.

En estas pruebas puede verse cómo el resultado no siempre es perfecto, lo que puede deberse a varios factores. La primera explicación a la pequeña tasa de fallos es que se ha realizado una mala adición de los nuevos términos en el vocabulario. Sin embargo, a la hora de reconocer el audio con nuevos y viejos términos, el resultado tampoco es perfecto en todos los casos. Esto no excluye la opción de que haya habido errores en los nuevos procedimientos, pero apunta a que puede tratarse de un problema de hardware o de reconocimiento. Puede ser, por una parte, que el micrófono graba demasiado ruido externo, con lo que no es posible un reconocimiento en condiciones, o que la entonación del usuario es diferente a lo esperado por iATROS y no reconoce correctamente; por otra parte, puede ser simplemente que el reconocimiento de voz no es siempre infalible, por lo que pueden producirse estos fallos.

El siguiente nivel de prueba es el de las pruebas funcionales o de integración. En estas pruebas el software ya está completamente ensamblado y se prueba en conjunto. Para evitar interacciones con el sistema, correspondientes al siguiente nivel, para esta prueba se han modificado las acciones al clicar en cada uno de los botones de reconocimiento para que no se grabe un audio. El audio será grabado anteriormente y preparado en formato WAV para que el sistema pueda tomarlo. Se proponen las siguientes pruebas:

- Clicar en reconocimiento de voz para código y luego en reconocimiento de voz para menú. Al realizar el primer clic, la opción elegida debe cambiar el icono y la otra opción debe inhabilitarse y no realizar ninguna acción. El programa procede correctamente.
- Clicar en reconocimiento de voz para menú y luego en reconocimiento de voz para código. Al realizar el primer clic, la opción elegida debe cambiar el icono y la otra opción debe inhabilitarse y no realizar ninguna acción. El programa procede correctamente.
- Clicar en reconocimiento de voz para menú con un audio con instrucciones de menú, volver a clicar para realizar el reconocimiento. El menú indicado en el audio debe abrirse. El programa procede correctamente en la mayoría de los casos.
- Clicar en reconocimiento de voz para código con un audio con identificadores, originales y nuevos, volver a clicar para realizar el reconocimiento. Los

identificadores indicados en el audio deben escribirse correctamente. El programa procede correctamente en la mayoría de los casos.

Como se ve en las dos últimas pruebas, el reconocimiento de audio es similar a las pruebas anteriores. Por otra parte, no se ha profundizado en diferentes combinaciones de audios como se ha hecho en el apartado de pruebas unitarias de la clase `Navigation`, ya que sería redundante. Respecto a las dos primeras pruebas, nos permite verificar los requisitos funcionales restantes:

- La grabación de audio para el uso de menús comienza en el primer clic en el botón de uso de menús.
- La grabación finaliza y se abre el menú correspondiente en el segundo clic al botón de menús.
- Solo se puede hacer un reconocimiento de voz simultáneo.

Además, permite verificar el siguiente requisito no funcional:

- El audio se almacenará en formato WAV.

Dado que iATROS, para procesar el audio necesita ser en formato WAV, y el proceso no ha fallado en ese punto, se considera verificado.

Las siguientes pruebas son las de sistema, donde el software ya validado se integra en un sistema para probar su funcionamiento. Para este proceso se han propuesto las siguientes pruebas:

- Clicar en reconocimiento de voz para menú, recitar una serie de órdenes de menú, volver a clicar para realizar el reconocimiento. El menú indicado debe abrirse. No debe quedar ningún archivo relacionado con el reconocimiento. El programa responde en menos de 5 segundos. El programa elimina los archivos y realiza el reconocimiento en la mayoría de los casos.
- Clicar en reconocimiento de voz para código, recitar una serie de palabras reservadas de Java e identificadores, volver a clicar para realizar el reconocimiento. El código recitado debe escribirse. No debe quedar ningún archivo relacionado con el reconocimiento. El programa responde en menos de 5 segundos. El programa elimina los archivos y realiza el reconocimiento en la mayoría de los casos.

- Clicar en reconocimiento de voz para menú, no decir nada o nada relacionado con los menús de Eclipse, volver a clicar para realizar el reconocimiento. Debe aparecer el mensaje de error en el reconocimiento. No debe quedar ningún archivo relacionado con el reconocimiento. El programa responde en menos de 5 segundos. El programa procede correctamente.

En las dos primeras pruebas se ha querido comprobar si la grabación de audio se reconoce correctamente y si luego se eliminan todos los archivos relacionados con ésta. Ambos resultados son satisfactorios. En la última prueba se quiere saber si el mensaje de error sigue apareciendo cuando el software se implementa en un sistema. Por último, se comprueba si el tiempo de respuesta es menor a 5 segundos. Con estas pruebas, se han validado los siguientes requisitos no funcionales:

- El sistema responderá, tras finalizar la grabación, en menos de 5 segundos.
- Tras haberse realizado el reconocimiento de voz, no quedará ningún archivo de audio ni texto por eliminar.

Por último, están las pruebas de aceptación. Estas pruebas son realizadas por el usuario final, en este caso, el mismo usuario que ha realizado las pruebas anteriores, por lo que no hay necesidad de repetirlas.

7. Conclusiones

Una vez finalizado el desarrollo tanto del software como de la memoria, se pueden extraer varias conclusiones.

Al inicio de esta memoria se proponen los tres objetivos de este desarrollo: conocer COPS a fondo, implementar la navegación de menús por voz y el aumento del vocabulario de iATROS. Y lo que parece una tarea sencilla puede volverse complicada en un instante.

El mantenimiento del software, en este caso perfectivo, es una tarea complicada, sobre todo en proyectos de gran envergadura para los recursos disponibles (una persona).

La primera aproximación a COPS es difícil en el caso de que no haya una mentalidad de mantenimiento del software, si no de implementación, que es lo que puede ocurrir habitualmente. Tras instalar e iniciar COPS, éste no funciona. Y no es hasta el momento en que se aplican técnicas de mantenimiento de software que el problema va a ser descubierto, y lo mismo sucede con iATROS, que también presentaba problemas de inicio.

Esto deja entrever a qué se puede enfrentar un equipo de mantenimiento en una situación real. COPS es un proyecto importante, pero sin comparación con otras aplicaciones que pueden encontrarse, además de estar comentado en prácticamente todas sus clases, lo que da más fuerza aún al argumento de los procesos de mantenimiento. Si un programa bien mantenido y de un tamaño medio genera algunos quebraderos de cabeza, qué puede ser heredar una aplicación de gran tamaño y sin comentar.

Una vez el problema es abordado de la forma adecuada, la comprensión del conjunto es más sencilla. Aplicando ingeniería inversa, se consigue el conocimiento necesario tanto de COPS como de iATROS para implementar la solución.

Finalmente, se puede decir que tanto el objetivo de comprender el funcionamiento de COPS e iATROS como la navegación y la ampliación de vocabulario ha sido alcanzado plenamente.

Y es por eso y por los resultados obtenidos que se demuestra que el mantenimiento es tan importante, y no debe aplicarse únicamente cuando un software necesita casi reescribirse por completo, generando unos gastos que se podrían haber evitado: se debe aplicar en todas las fases del desarrollo, desde el momento de la creación del software y por todas sus etapas, hasta continuarlo una vez el producto está entregado.

7.1 Relación con los estudios cursados

Este proyecto se relaciona directa y especialmente con la parte de mantenimiento del software, una parte que normalmente no es la más apreciada pero que en estos casos demuestra su uso y sus capacidades. En relación con el mantenimiento se ha hecho uso de técnicas de mantenimiento perfectivo desde las más comunes, como son las actividades propias del mantenimiento del software (comprensión del software y de los cambios a realizar, modificación y pruebas) a la aplicación de leyes del mantenimiento del software e ingeniería inversa (tareas más avanzadas) abriendo aún más la perspectiva de lo que conlleva el mantenimiento del software. Además de esto, se ha apoyado el mantenimiento perfectivo realizado en estándares de calidad como es el ISO/IEC 14764, mejorando la calidad del resultado final. Por último, respecto al mantenimiento, se han realizado procesos de prueba importantes para mejorar el resultado final.

No obstante, además de en el mantenimiento, este proyecto en parte se apoya en el análisis de requisitos, siguiendo el ciclo de vida de un documento de requisitos y pasando por todas sus fases, para así conseguir unos mejores requisitos que verificar con las pruebas.

8. Trabajos futuros

Una vez visto todo el desarrollo realizado, se van a mencionar los posibles trabajos futuros que se podrían aplicar en este proyecto.

El primero y más claro es completar el modelo de lenguaje y la clase `Navigation` de forma que se puedan reconocer todos los menús. Se trata de una tarea relativamente trivial, pero debido al gran volumen de menús que existen, se ha preferido dedicar el esfuerzo a otras partes del proyecto.

Respecto a la navegación por menús, podría ser interesante añadir nuevas órdenes no relacionadas con los menús, similares a las opciones directas, pero ampliando el rango aún más, como pueda ser cambiar de pestaña o minimizar la ventana de edición de código.

Otra línea de trabajo a seguir es la de continuar aumentando el número de identificadores o incluso añadiendo la opción de que el usuario pueda añadirlos por su parte, de manera que no hay que realizar un esfuerzo bruto de escribir todos los identificadores posibles.

9. Bibliografía

1. Esparza Arellano, María Elena, & Avalos Briseño, J. Benito (2003). Reconocimiento de voz. Conciencia Tecnológica, (22). [fecha de Consulta 19 de Julio de 2020]. ISSN: 1405-5597. Disponible en: https://www.redalyc.org/articulo.oa?id=94402206

2. Fernández, V. (1997). Antecedentes y desarrollo de los sistemas actuales de reconocimiento automático del habla 1 (Historical evolution in the current Automatic Speech Recognition System).

3. Rabiner, L. R., & Juang, B. H. (1993). Fundamentals of speech recognition. Englewood Cliffs, N.J: PTR Prentice Hall.

4. Rodríguez-Mateos, Francisco & Bautista, Susana. (2006). Modelos ocultos de Markov para el análisis de patrones espaciales. Ecosistemas: Revista científica y técnica de ecología y medio ambiente, ISSN 1697-2473, Nº. 3, 2006. 15.0

5. Carlos-D. Martínez-Hinarejos, Santiago Sánchez-Alepuz, Natividad Prieto-Sáez. COPS: a computer programming tool to cope with functional diversity. Proceedings of IberSpeech 2012, pp. 371-376, Universidad Autónoma de Madrid, Madrid, 2012

6. G. Hinton et al., "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 82-97, Nov. 2012, doi: 10.1109/MSP.2012.2205597.

7. Shaik, Shairaj & Corvin, Raymond & Sudarsan, Rajesh & Javed, Faizan & Ijaz, Qasim & Roychoudhury, Suman & Gray, Jeff & Bryant, Barrett. (2003). SpeechClipse: an Eclipse speech plug-in. 84-88. 10.1145/965660.965678.

8. DSIC (2020) “Tema 1-Introducción”. Mantenimiento y evolución del Software. Universidad Politécnica de Valencia.

9. ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance," in ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998) , vol., no., pp.1-58, 1 Sept. 2006, doi: 10.1109/IEEESTD.2006.235774.

10. DSIC (2020) “Tema 6- El estándar ISO/IEC 14764”. Mantenimiento y evolución del Software. Universidad Politécnica de Valencia.

11. DSIC (2020) “Tema 1- Ingeniería de requisitos”. Análisis y especificación de requisitos. Universidad Politécnica de Valencia.

12. DSIC (2020) “Tema 3- Proceso de Ingeniería de requisitos”. Análisis y especificación de requisitos. Universidad Politécnica de Valencia.

13. DSIC (2020) “Tema 5- Validación y gestión de requisitos”. Análisis y especificación de requisitos. Universidad Politécnica de Valencia.

14. DSIC (2020) “Tema 7- Pruebas”. Mantenimiento y evolución del Software. Universidad Politécnica de Valencia.

15. Míriam Luján-Mares, Vicent Tamarit, Vicent Alabau, Carlos-D. Martínez-Hinarejos, Moisés Pastor, Alberto Sanchis, and Alejandro Toselli. iatros: A speech and handwritting recognition system. In V Jornadas en Tecnologías del Habla (VJTH'2008), pages 75-78, Bilbao (SPAIN), Nov 2008.