

Portada

AGRADECIMIENTOS:

ÍNDICE DEL CONTENIDO

ABSTRACT.....	10
1. INTRODUCCIÓN	12
2. DOCUMENTO DE OBJETIVOS DEL PROYECTO (DOP)	14
2.1. OBJETIVOS DEL PROYECTO	14
2.2. ALCANCE	15
2.2.1. PROCESOS TÁCTICOS	16
2.2.2. PROCESOS OPERATIVOS	16
2.2.3. PROCESOS FORMATIVOS	17
2.3. MÉTODO DE TRABAJO	17
2.4. PLANIFICACIÓN ESTIMADA	18
2.4.1. ESTIMACIÓN DE COSTES	19
2.5. PLAN DE CONTINGENCIA	22
2.5.1. PROBLEMAS EN LOS PROCESOS TÁCTICOS	22
2.5.1.1. PÉRDIDA DE INFORMACIÓN	22
2.5.1.2. RETRASOS EN LAS ESTIMACIONES DE TIEMPO Y EN EL TRABAJO	22
2.5.2. PROBLEMAS EN LOS PROCESOS OPERATIVOS	22
2.5.2.1. ELECCIÓN ERRÓNEA E INEXPERIENCIA DE LA TECNOLOGÍA	22
2.5.2.2. PROBLEMAS CON EL SOFTWARE SELECCIONADO	23
2.6. RECURSOS	23
2.7. ANÁLISIS DE FACTIBILIDAD	23
3. CAPTURA DE REQUISITOS.....	24
3.1. MODELO DE CASOS DE USO.....	24

3.1.1. CASOS DE USO DEL USUARIO	24
3.1.1.1. CASO DE USO TRADUCIR.....	25
3.1.1.2. CASO DE USO INTERCAMBIAR IDIOMAS.....	25
3.1.1.5. CASO DE USO CAMBIAR IDIOMA.....	26
3.2. MODELO DE DOMINIO	26
3.3. DESCRIPCIÓN DE LA INTERFAZ GRÁFICA DEL USUARIO.....	27
3.3.1. PANTALLA DE BIENVENIDA	27
3.3.2. MENÚ PRINCIPAL.....	28
3.3.3. PANTALLA DE LA FUNCIÓN “TRADUCIR”.....	28
3.3.4. CAMBIAR EL IDIOMA.....	30
4. ANÁLISIS.....	32
4.1. CASO DE USO TRADUCIR.....	32
4.2. CASO DE USO INTERCAMBIAR IDIOMAS.....	33
4.3. CASO DE USO CAMBIAR IDIOMA.....	34
5. ARQUITECTURA DEL SISTEMA.....	36
5.1. CAPA DE PRESENTACIÓN Y DE DOMINIO	36
5.2. CAPA DE GESTIÓN DE DATOS	37
6. ELECCIÓN TECNOLÓGICA	40
6.1. J2ME (JAVA MICROEDITION)	40
6.1.1. MÁQUINA VIRTUAL JAVA (JVM)	42
6.1.2. CONFIGURACIONES	42
6.1.3. PERFILES	42
6.2. J2ME WIRELESS TOOLKIT.....	44
7. DISEÑO.....	46
7.1. DIAGRAMAS DE INTERACCIÓN.....	46
7.1.1. OPERACIÓN ELEGIR IDIOMA	46
7.1.2. OPERACIÓN TRADUCIR.....	47

7.1.3. OPERACIÓN INTERCAMBIAR.....	49
7.2. DIAGRAMA DE CLASES	50
8. IMPLEMENTACIÓN.....	52
8.1. ¿QUÉ ES UN MIDLET?	52
8.2. ESTRUCTURA GENERAL DEL MIDLET	53
8.3. IMPLEMENTACIÓN DE LAS INTERFACES GRÁFICAS MEDIANTE APIS (INTERFAZ DE PROGRAMACIÓN DE APLICACIONES)	55
8.4. MANEJO DE LOS BOTONES Y EVENTOS	57
8.5. ACCESO AL DICCIONARIO.....	58
8.6. RESTRICCIONES DEL CLDC	60
8.6.1. NO EXISTE SOPORTE PARA OPERACIONES EN PUNTO FLOTANTE.	60
8.6.2. LIMITACIONES EN EL MANEJO DE ERRORES	60
8.6.3. LIBRERÍAS LIMITADAS	61
8.6.4. SEGURIDAD	61
8.7. DESCRIPCIÓN DE LAS CLASES	62
9. PRUEBAS.....	64
9.1. PRUEBAS DE “CAJA NEGRA”	64
9.1.1. PULSAR UN BOTÓN DE LA INTERFAZ	64
9.1.2. ESCRIBIR UNA PALABRA EN LA FUNCIÓN TRADUCIR.....	64
9.1.2.1. CAMPO DE TEXTO VACÍO	65
9.1.2.2. PALABRA CORRECTA	65
9.1.2.3. PALABRA INEXISTENTE O MAL ESCRITA	65
9.1.2.4. LÍMITE DEL CAMPO DE TEXTO	65
9.1.3. LÍMITE DE PALABRAS	65
9.2. PRUEBAS DE “CAJA BLANCA”	66
9.2.1. FORMATO DE LAS INTERFACES GRÁFICAS.....	66
9.2.2. CONTROL DEL ACCESO AL FICHERO	66

9.3. OTRAS PRUEBAS	66
10. IMPLANTACIÓN	68
10.1. REQUERIMIENTOS	68
10.2. EMPAQUETAMIENTO	69
10.3. DESCARGA AL DISPOSITIVO.....	69
11. GESTIÓN DEL PROYECTO	70
12. CONCLUSIONES	73
12.1. FUTURAS MEJORAS DEL SISTEMA.....	74
13. APÉNDICES	76
13.1. MANUAL DEL USUARIO	76
13.1.1. TECLAS	76
13.1.2. ENTRAR A LA APLICACIÓN.....	77
13.1.3. NAVEGACIÓN POR EL MENÚ	78
13.1.4. AJUSTES DE IDIOMA	79
13.1.5. TRADUCIR UNA PALABRA	81
13.1.5.1. INTERCAMBIAR IDIOMAS.....	82
13.1.5.2. TRADUCCIÓN CORRECTA.....	83
13.1.5.3. TRADUCCIÓN INCORRECTA.....	84
13.2. GUÍA DE INSTALACIÓN	85
ECLIPSEME	86
JAVA IDE (INTEGRATED DEVELOPMENT ENVIROMENT) Y JDK	85
SUN JAVA WIRELESS TOOLKIT	89
13.3. EMULADORES	91
14. BIBLIOGRAFÍA	93

INDICE DE ILUSTRACIONES

Figura 1: Diagrama EDT	16
Figura 2: Lista de tareas planificadas	20
Figura 3: Diagrama Gantt de la Planificación Estimada	22
Figura 4: Casos de Uso del actor Usuario	26
Figura 5: Modelo de Dominio del Traductor	27
Figura 6: Pantalla de Bienvenida.....	28
Figura 7: Menú Principal en castellano y euskera.....	29
Figura 8: Traducir en castellano y euskera.....	29
Figura 9: Intercambio de idioma de la figura 8	30
Figura 10: Pantalla para seleccionar el idioma.....	31
Figura 11: Diagrama de secuencia de “Traducir”	33
Figura 12: Diagrama de secuencia de “Intercambiar idiomas”	34
Figura 13: Diagrama de secuencia de “Cambiar Idioma”	35
Figura 14: Arquitectura del Sistema.....	37
Figura 15: Relación entre las APIs de la plataforma Java.....	42
Figura 16: Arquitectura J2ME y estructura del Traductor	44
Figura 17: Diagrama de interacción de “Elegir Idioma”	47
Figura 18: Diagrama de interacción de “Traducir”	48
Figura 19: Diagrama de interacción de “Cambiar”	50
Figura 20: Diagrama de Clases del “Traductor”	51
Figura 21: Estados de un <i>MIDlet</i>	54
Figura 22: Sistema sin el uso del <i>Thread</i>	60
Figura 23: Sistema con el uso del <i>Thread</i>	60
Figura 24: Tabla de Clases	64
Figura 25: Gráfica de relación de horas por proceso real.....	71
Figura 26: Gráficas de la distribución de los procesos tácticos estimados y reales	71
Figura 27: Gráficas de la distribución de los procesos operativos estimados y reales...	72
Figura 28: Gráficas de la distribución de los procesos formativos estimados y reales ..	72
Figura 29: Comparativa de porcentajes de tiempo	73

Figura A1: Teclas de control del emulador	79
Figura A2: Pantalla de Inicio.....	80
Figura A3: Pantalla del Menú inicial (castellano y euskera).....	81
Figura A4: Selección del idioma	82
Figura A5: Guardar cambios	83
Figura A6: Traducir una palabra (castellano y euskera).....	84
Figura A7: Funciones “Traducir” disponibles.....	85
Figura A8: Intercambio de idioma.....	85
Figura A9: Traducción correcta.....	86
Figura A10: Traducción incorrecta.....	87
Figura A11: Instalación del JDK (paso 1).....	88
Figura A12: Instalación del JDK (paso 2).....	89
Figura A13: Instalación de nuevos componentes en el Eclipse.....	90
Figura A14: Instalación del EclipseME (paso 1).....	90
Figura A15: Instalación del EclipseME (paso 2).....	91
Figura A16: Instalación del Wireless Toolkit	92
Figura A17: Importación del WTK	93
Figura A18: Configuración del WTK.....	93

0. ABSTRACT

El sistema que se presenta en este proyecto permite la traducción de palabras euskera – castellano y castellano - euskera a través de un dispositivo móvil. Se trata de un proyecto realizado para la titulación de Ingeniería Técnica en Informática de Sistemas (ITIS) dirigido por el Sr. Germán Rigau Claramunt.

Todas las tareas que se presentan aquí han sido realizadas por la autora de este documento, no se ha precisado de un trabajo conjunto. El tutor ha sido el encargado de facilitar un diccionario real de 69108 traducciones en las dos direcciones. La aplicación se ha desarrollado en el entorno de desarrollo Eclipse, aunque es portable a otros entornos que soporten la plataforma Java, como NetBeans.

1. INTRODUCCIÓN

El objetivo principal de este proyecto es desarrollar una aplicación que ofrezca al usuario de un dispositivo móvil la posibilidad de traducir una palabra de castellano a euskera y viceversa.

La idea del traductor surgió a raíz de una conversación con un compañero que se encontraba en un albergue estudiando euskera. Comentó que en ocasiones necesitaba la traducción de una palabra y no tenía un diccionario a mano. Por lo que debatiendo sobre el tema llegamos a la conclusión de que existían traductores para todo tipo de dispositivos y en diferentes idiomas, pero no para el móvil en euskera.

La evolución del teléfono móvil ha permitido disminuir su tamaño y peso. Además, junto con la incorporación de software más atractivo y el desarrollo de pantallas más nítidas y con mayor resolución gráfica, hacen de este dispositivo un elemento muy apreciado hoy en día. El avance de esta tecnología ha hecho que se les pueda incorporar todo tipo de aplicaciones, como por ejemplo, juegos, reproductores de música MP3 y video, navegadores, fotografía y video digital, videollamada y hasta Televisión Digital.

Una muestra clara es el último lanzamiento del iPhone 3G, un reproductor de música con pantalla panorámica que además es un teléfono y un dispositivo de acceso a Internet que incluye correo electrónico y mapas con GPS.

Las compañías de telefonía móvil y empresas como Google ya están pensando en nuevas aplicaciones para añadir a este aparato. Algunas de esas ideas son: localizador e identificador de personas, una guía de recetas multimedia, guías de turismo,...¹

¹(Más información en http://code.google.com/android/images/adc1r1_deck.pdf)

Vista la importancia actual de estos dispositivos y la variedad de funciones que se ofrecen se presenta la necesidad de conocer las tecnologías y plataformas utilizadas. Por lo que, para inicializarse en este campo, se plantea programar una aplicación para este tipo de elementos. Así surge el proyecto que se presenta en este documento, la implementación de un traductor con soporte para dispositivos móviles.

A lo largo de la memoria se presentará información necesaria para el seguimiento de este trabajo. Incluye el documento de objetivos del proyecto establecido al principio de la etapa, así como las fases de la captura de requisitos, análisis y diseño. Se describen también las herramientas y tecnologías más convenientes que se han utilizado para la implementación y las pruebas.

Finalmente se recoge la experiencia acumulada y las conclusiones personales que ha aportado la creación de este trabajo. También se incluyen las posibles mejoras que se pueden aplicar al sistema para hacerlo mucho más eficiente y funcional.

Esta memoria pretende ser una guía eficaz para comprender el desarrollo del proyecto y la arquitectura del programa, con el fin de que el lector no pierda ningún detalle sobre este sector de las nuevas comunicaciones.

2. DOCUMENTO DE OBJETIVOS DEL PROYECTO (DOP)

Este capítulo recoge los pasos y estrategias necesarias para diseñar la planificación del proyecto. Con el fin de detallarlos se mostrarán: los objetivos del proyecto, datos relacionados con las estimaciones de tiempo, los riesgos y sus soluciones, recursos necesarios, etc.

2.1. Objetivos del proyecto

Este proyecto es la creación de un software para un dispositivo móvil capaz de traducir una palabra introducida por el usuario, dando como resultado la traducción de dicha palabra, o en su defecto, las más parecidas a ella.

Dispondrá de una función que permita cambiar el idioma fuente de la palabra, alternando entre euskera y castellano para poder utilizarlo en ambos sentidos. También deberá ofrecer la posibilidad de cambiar el idioma de la interfaz, ya sea el euskera o el castellano. De esta manera el usuario podrá navegar por las distintas pantallas del programa con más facilidad.

Puesto que los teléfonos móviles disponen de una capacidad de memoria limitada, convendrá que la aplicación no ocupe mucho espacio y la estructura de datos deberá ser lo más sencilla posible. Por lo que será importante encontrar el soporte y el lenguaje de programación más adecuado para llevar a cabo este objetivo.

2.2. Alcance

Una buena planificación ayuda a organizar y estructurar cualquier proyecto y a prevenir imprevistos que pueden surgir a lo largo del mismo. Siendo conscientes de ello, debemos dividir el trabajo en fases que favorezcan esa situación y ser capaces de llevar un control sobre ellas, disponiendo las tareas de modo que convengan con el trabajo y el tiempo planificado. Como resultado dispondremos de un diagrama de la Estructura de Descomposición del trabajo (EDT) como el siguiente:

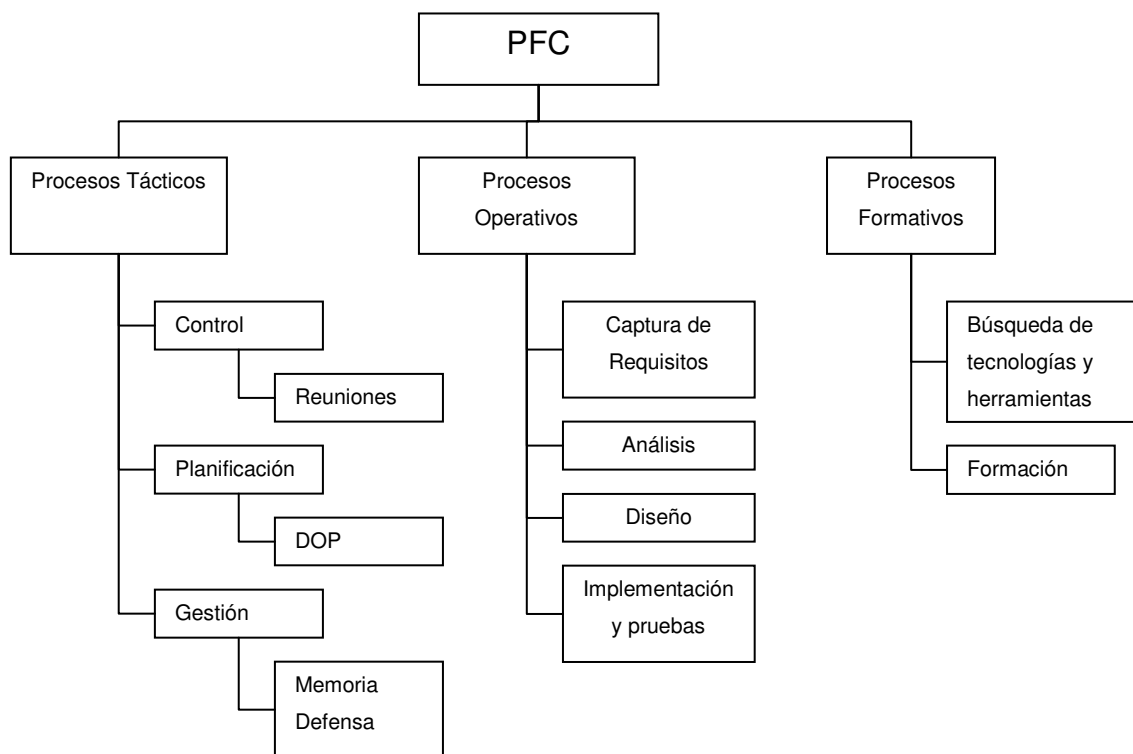


Figura 1: Diagrama EDT

En la figura 1 se pueden distinguir las diferentes partes que componen el proyecto, siguiendo un orden en la ejecución de algunos de ellos como son la captura de requisitos, el análisis y el diseño antes que la implementación y las pruebas.

A continuación describiremos cada una de las fases que lo componen más a fondo:

2.2.1. Procesos Tácticos

En esta fase se incluyen la gestión y la planificación del proyecto. Esto es necesario para llevar un control del desarrollo del mismo. En él se especifican los pasos que tomaremos y el esfuerzo que vamos a emplear en ellos para poder entregar el trabajo en un plazo fijado. Además, se fijan unas reuniones con el tutor para verificar que los objetivos se cumplen y establecer nuevos plazos.

En la gestión se recogen tanto el documento que formaliza la aceptación del proyecto (la memoria), como la defensa. En la primera de ellas se reflejarán todas las tareas llevadas a cabo; ya que un proyecto no solo se compone de una aplicación ejecutable, sino que también debe disponer de una documentación que recoja todas las decisiones tomadas y un manual que guíe al usuario a utilizarlo. Es importante elaborar una memoria clara y precisa en todos los detalles, ya que el tribunal dispondrá una copia para su evaluación. Incluso en un futuro podrá servir a una persona interesada en mejorar y/o ampliar las prestaciones del sistema. En la defensa se realizará una presentación ante el tribunal para explicar la creación y el desarrollo del trabajo.

Por otra parte, en la planificación, a parte de distinguir las etapas que compondrá el trabajo, se deberá estimar la duración que utilizaremos en cada tarea. En el apartado 2.4 se especifica más sobre este tema en el que se verá reflejada la planificación en un diagrama.

2.2.2. Procesos Operativos

Antes de dedicarnos plenamente a la implementación del programa se deben identificar los casos de uso que va a disponer y los servicios que queremos que ofrezca. Un primer análisis para la búsqueda de los casos de uso se realiza en la etapa de la captura de requisitos. Finalizada esta fase comenzará otro análisis para describir de forma más detallada las funcionalidades del sistema.

Una vez realizado el estudio exhaustivo de la aplicación, se procederá a su diseño. Una tarea que mostrará cómo esta compuesto nuestro sistema, en el que se podrá apreciar en cada funcionalidad las clases que tomarán parte y la conexión entre ellas.

Al finalizar la tarea de analizar y diseñar nuestra aplicación, se procederá a implementar la misma. En esta etapa utilizaremos la tecnología escogida en las primeras fases para desarrollarla.

En la etapa de pruebas se realizarán simulaciones periódicamente en la fase final de la implementación para localizar errores que puedan surgir. De esta manera podremos solucionarlos de un modo constante sin necesidad de remediarlos todos al final.

2.2.3. Procesos Formativos

En esta fase se distinguen dos apartados. La primera, la búsqueda y elección de las tecnologías a utilizar, se debatirá una vez decidido “qué” queremos hacer (finalizados el DOP, la captura de requisitos y el análisis).

La segunda de ellas es la formación durante el proyecto, que tratará sobre el aprendizaje en el uso de las herramientas y lenguajes necesarios. Existe el caso de no haber utilizado nunca alguno de ellos, por lo que lo mejor será iniciarse en el tema antes de poder utilizarlo en nuestro trabajo.

2.3. Método de trabajo

El proyecto se dividirá en tres fases a lo largo del curso. La primera de ellas se realizará en el primer cuatrimestre. Debido a las asignaturas pendientes en este periodo, gran parte del tiempo se invertirá en el estudio, pero sin dejar a un lado el proyecto. Por lo que en esta primera fase se llevará a cabo un borrador del Documento de Objetivos del Proyecto y se investigará sobre las tecnologías utilizadas por los teléfonos móviles.

En la segunda fase, durante el segundo cuatrimestre, se analizarán las funciones que deseamos que disponga nuestro programa. El desarrollo del sistema se realizará en cascada, es decir, la captura de requisitos precederá al análisis, le seguirá el diseño del sistema y luego la implementación. Antes de nada, se piensa que es recomendable formarse en los nuevos programas y lenguajes escogidos previamente, ya que de esta manera se sabrá mejor hasta dónde pueden llegar nuestras expectativas en el software.

La implementación se desarrollará en la tercera fase, durante el verano. Ésta será la parte más complicada y la que más tiempo consumirá del trabajo. Al ser la primera vez que se va a programar en este tipo de dispositivos, se implementarán inicialmente algunos ejemplos sencillos para ir experimentando.

Se mantendrá el contacto con el tutor vía e-mail, y en caso de ser necesaria una reunión se establecerá una fecha mediante un mensaje de correo electrónico.

Una vez finalizado el proyecto y la memoria, se les proporcionará al tutor y al tribunal una copia del documento y un CD con la aplicación. Para una clara exposición ante el tribunal evaluador se deberá realizar una serie de diapositivas que reflejen todo el trabajo que conlleve este proceso, las cuales nos ayudarán a la hora de defenderlo de manera precisa e intuitiva

2.4. Planificación Estimada

La distribución de las tareas que se realizarán durante el proyecto son las siguientes, junto con el tiempo estimado que tomaremos en ellas. Posteriormente se muestran estas tareas en el diagrama de Gantt.

- | | |
|----------------------------|---------|
| - Desarrollo del DOP | 7 días |
| - Búsqueda de herramientas | 5 días |
| - Captura de Requisitos | 7 días |
| - Análisis | 10 días |

-
- Diseño 10 días
 - Implementación 50 días
 - Pruebas 32 días
 - Memoria 22 días
 - Formación 119 días?

No es posible definir un plazo fijo para la tarea de la formación, ya que se realizará durante todo el proyecto. A pesar de iniciarse en las herramientas tecnológicas antes de implementar, se irán aprendiendo nuevos conceptos a medida que aparezcan dudas y complicaciones

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	DOP	7 días	lun 29/10/07	mar 06/11/07	
2	Búsqueda de Herramientas	5 días	lun 12/11/07	vie 16/11/07	
3	(Preparar exámenes)	45 días	lun 03/12/07	vie 01/02/08	
4	Captura de Requisitos	7 días	lun 03/03/08	mar 11/03/08	
5	Análisis	10 días	mié 12/03/08	mar 25/03/08	4
6	Diseño	10 días	mié 26/03/08	mar 08/04/08	5
7	Implementación	50 días	mié 09/04/08	mar 17/06/08	6
8	Pruebas	32 días	mié 18/06/08	jue 31/07/08	7
9	Memoria	22 días	vie 01/08/08	lun 01/09/08	
10	Formación	119 días?	vie 01/02/08	mié 16/07/08	

Figura 2: Lista de tareas planificadas

Las tareas que requerirán más tiempo, en principio, son las de formación e implementación. Puesto que es la primera vez que voy a utilizar un lenguaje de programación utilizado en los móviles, necesitaré más tiempo del habitual en documentarme y en conocer el entorno en el que voy a trabajar

2.4.1. Estimación de costes

Se estima que la duración total del proyecto rondará sobre las 400 horas. Es el resultado de desglosar las tareas de la siguiente manera:

- DOP	2 horas/día
- Búsqueda de herramientas	1 hora/día
- Captura de requisitos	2 horas/día
- Análisis	2 horas/día
- Diseño	2 horas/día
- Implementación	2'5 horas/día
- Pruebas	2 horas/día
- Memoria	3 horas/día
- Formación	3 horas/día ²

Junto con los días planificados de las tareas (Figura 2) hacen un total de 418 horas.

² Utilizando como muestra una media de 30 días

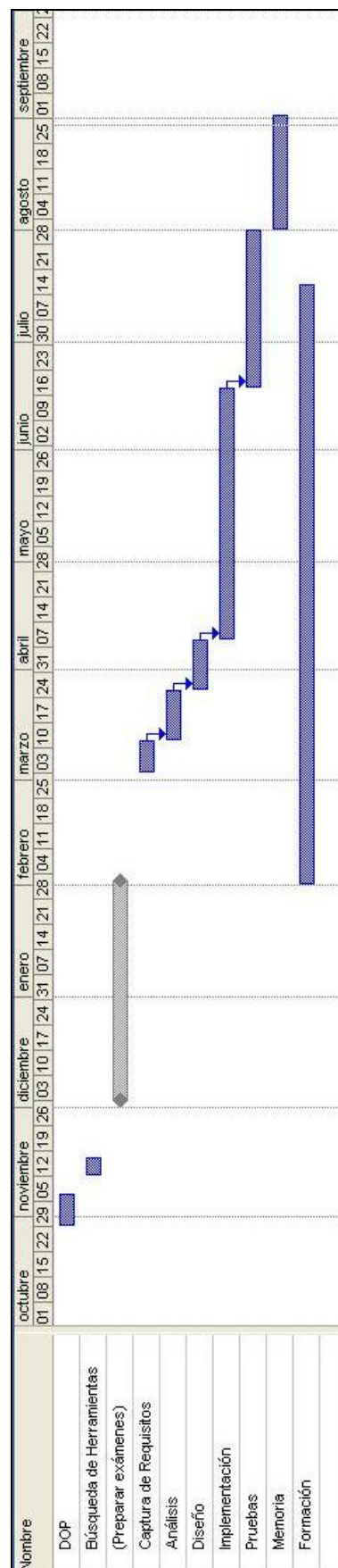


Figura 3: Diagrama Gantt de la Planificación Estimada

2.5. Plan de Contingencia

Es necesario idear un buen plan que ayude a paliar los problemas más frecuentes que surgen al realizar un proyecto. Las soluciones propuestas deberán intentar no afectar al trabajo ya realizado en la medida de lo posible.

2.5.1. Problemas en los Procesos Tácticos

2.5.1.1. Pérdida de información

Es probable que algunos datos se pierdan o se almacenen incorrectamente debido a un corte de electricidad, algún fallo del sistema, etc. Por lo que es recomendable crear copias de seguridad habitualmente en diferentes dispositivos que guarden el trabajo realizado hasta el momento.

2.5.1.2. Retrasos en las estimaciones de tiempo y en el trabajo

En el caso de que no se puedan cumplir los plazos fijados se deberá re-planificar el trabajo que resta por hacer en el intervalo de tiempo que disponemos hasta la fecha de entrega, poniendo más empeño y constancia para que no vuelva a ocurrir el mismo problema.

2.5.2. Problemas en los Procesos Operativos

2.5.2.1. Elección errónea e inexperiencia de la tecnología

La mala elección de las herramientas o la inexperiencia al utilizarlos es probablemente el riesgo más importante. Cuanto más tarde nos demos cuenta de este problema más difícil será solucionarlo. Hay que considerar que el campo en el que se desarrolla el proyecto no es dominado por el alumno. Los sistemas de los dispositivos móviles son aún un tema en el que las facultades de informática no se han adentrado en profundidad.

Por tanto, se propone el estudio de las distintas herramientas existentes y hacer una comparativa entre ellas antes de seleccionar alguna. De esta forma se conocerán a priori sus capacidades y las limitaciones que presenta. Además se deberán estudiar manuales, tutoriales y guías de usuario para experimentar con ejemplos sencillos.

2.5.2.2. Problemas con el Software seleccionado

Si el entorno de desarrollo en el que estamos trabajando presenta problemas o nos parece incómodo, el avance del proyecto se ralentizará. Debemos disponer de otras posibilidades que soporten el trabajo que tenemos realizado para migrar la labor de un entorno a otro y poder seguir adelante.

2.6. Recursos

Obviamente es necesaria la utilización de un PC (en este caso con un sistema operativo Windows). Para poder probar la aplicación en un dispositivo real se deberá utilizar un teléfono móvil que soporte la aplicación. En caso de no disponer ninguno, habrá que buscar emuladores que simulen la ejecución del sistema. Para la formación y la implementación serán de gran ayuda los tutoriales y las guías. Generalmente, las herramientas tecnológicas disponen de sus propios manuales de usuario.

2.7. Análisis de factibilidad

Una vez estudiadas las tareas que se deberán realizar debemos decidir si es posible la creación del traductor. Vista la gran aceptación de las aplicaciones en los teléfonos móviles mucha gente se ha lanzado a programar sus propios servicios. En Internet existen bastantes herramientas que facilitan la elaboración y la instalación del sistema en nuestros ordenadores. Además, existen un gran número de foros dedicados exclusivamente a la programación móvil que ayudan con las dudas de los interesados en el tema.

Como conclusión y debido a la cantidad de información que se puede encontrar, la construcción de una aplicación traductora para un dispositivo móvil parece factible.

3. CAPTURA DE REQUISITOS

En este primer análisis del sistema se recogen el Modelo de Casos de Uso (MCU) y el Modelo de Dominio (MD) para la gestión del Traductor. Además se añaden los casos de uso con una descripción de alto nivel. Un Caso de Uso recoge una secuencia de eventos que realiza un actor que está utilizando el programa. Describe un proceso de principio a fin relativamente amplio.

3.1. Modelo de Casos de Uso

Se quiere desarrollar una aplicación que traduzca la palabra que el usuario ha introducido mediante el teclado del móvil. El traductor deberá permitir intercambiar los idiomas fuente (euskera y castellano) del término. La aplicación también controla el idioma de la misma, otorgando al usuario la oportunidad de navegar por el sistema en el idioma preferido.

Cuando se ejecuta la aplicación, el usuario observará un menú principal que muestra las operaciones que se pueden realizar.

Se ha identificado un único tipo de actor para la gestión del Traductor: el Usuario del teléfono móvil. Será el encargado de navegar por su dispositivo para entrar a la aplicación. Mediante la pulsación de las teclas interactuará con el sistema.

3.1.1. Casos de Uso del Usuario

La figura 4 presenta los casos de uso del usuario. Tendrá la posibilidad de realizar una serie de tareas con el sistema, las cuales son: traducir una palabra (**Traducir**), alternar entre los idiomas de traducción (**Intercambiar Idioma**) y cambiar el idioma de la aplicación (**Cambiar Idioma**).

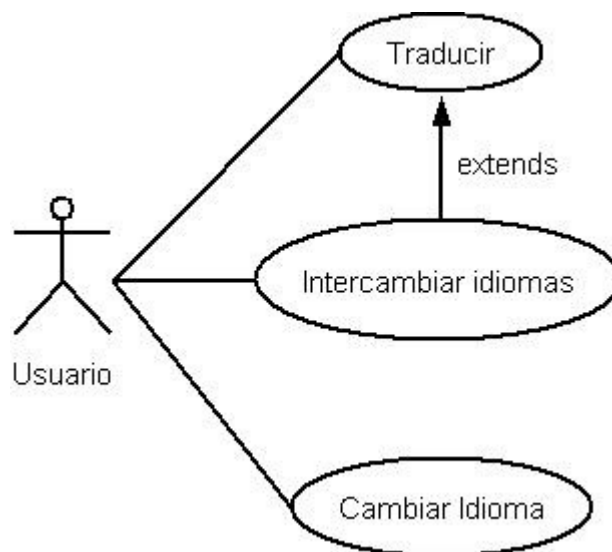


Figura 4: Casos de Uso del actor Usuario

3.1.1.1. Caso de Uso Traducir

Este caso de uso permite al Usuario traducir una palabra del idioma fuente al idioma destino y viceversa.

Se introducirá la palabra un en campo habilitado para tal efecto y se procederá a la traducción. El sistema buscará la palabra en la estructura de datos y, en caso de encontrarla, le mostrará el resultado de la transcripción por pantalla. Si por el contrario la palabra deseada no se encuentra almacenada, la aplicación generará un listado de los términos más parecidos a la palabra origen.

3.1.1.2. Caso de Uso Intercambiar Idiomas

Este caso de uso se produce cuando el idioma de la palabra a traducir es el contrario del mostrado en la pantalla. El sistema deberá intercambiar el sentido de la traducción, es decir, si se muestra euskera – castellano tendría que cambiar a castellano – euskera. Para ello realizará un cambio de estado que advierta a la aplicación de tal evento.

Este caso de uso estará disponible una vez hayamos avanzado al caso de uso de Traducir.

3.1.1.5. Caso de Uso Cambiar Idioma

Este caso de uso le permite al usuario del dispositivo utilizar el sistema con el idioma que prefiera. Podrá elegir entre dos opciones para que la aplicación se muestre con el idioma escogido.

3.2. Modelo de Dominio

En la figura 5 se presenta el modelo de dominio del sistema Traductor. El diagrama incluye básicamente, la estructura del almacenamiento de los datos. Se trata de una representación de cómo se distribuirán las palabras para realizar las consultas.

Un **diccionario** se compone de al menos una palabra (y su traducción), ya sea euskera – castellano o castellano – euskera.

Una **palabra** se caracteriza por el nombre, su traducción y una letra que denomina la categoría, pudiendo ser n (nombre), v (verbo), a (adjetivo) y r (adverbio). A cada palabra del idioma fuente, le corresponde una o más traducciones de la palabra destino.

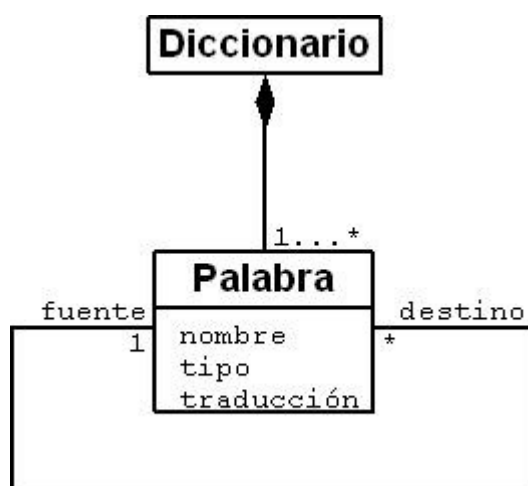


Figura 5: Modelo de Dominio del Traductor

3.3. Descripción de la Interfaz Gráfica del Usuario

El Traductor constará de un menú principal que da entrada a cada uno de los casos de uso del sistema a los que se puedan acceder directamente. Inicialmente se visualizará una pantalla de presentación de la aplicación que dará acceso a dicho menú. La interfaz mostrará las posibilidades admisibles según el estado donde se encuentre

A continuación se presentan los bocetos de las interfaces que dispondrá nuestro sistema.

3.3.1. Pantalla de Bienvenida



Figura 6: Pantalla de Bienvenida

La figura 6 presenta la pantalla de Bienvenida dónde se invitará al usuario a entrar a la aplicación. Avanzará a la siguiente pantalla seleccionando sobre “Entrar / Sartu”. En caso de desear salir de la aplicación, se pulsará sobre “Salir / Irten”.

3.3.2. Menú Principal

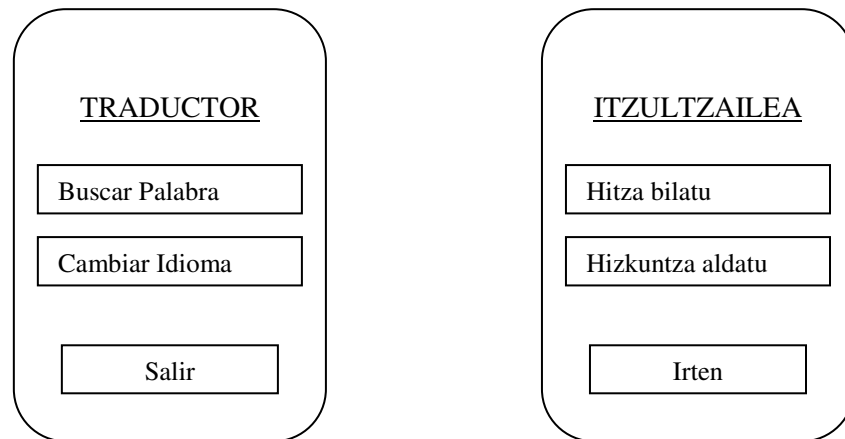


Figura 7: Menú Principal en castellano y euskera

La interfaz muestra los servicios que nos presta el sistema. La primera opción nos llevaría a la pantalla de la función principal “Traducir”. La segunda, en cambio, nos traslada a la pantalla que permite elegir el idioma para visualizar la aplicación.

Por último, la tercera opción ofrece la posibilidad de salir de la aplicación.

3.3.3. Pantalla de la función “Traducir”

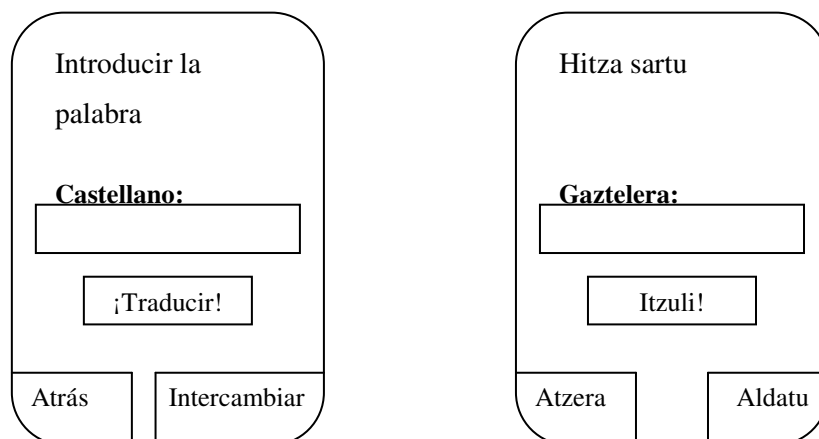


Figura 8: Traducir en castellano y euskera

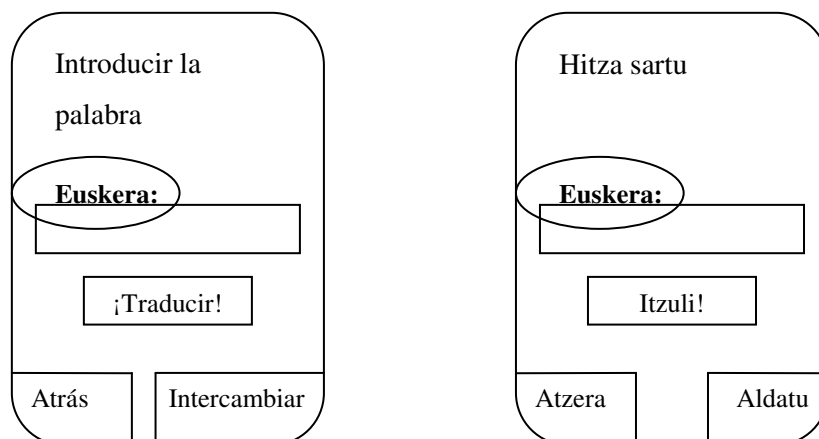


Figura 9: Intercambio de idioma de la figura 8

Las imágenes 8 y 9 muestran las pantallas correspondientes al caso de uso “Traducir”. Estarán compuestas de un campo habilitado donde se escribirá la palabra que el usuario necesite traducir. El botón central “¡Traducir!” o “Itzuli!” será el encargado de que el sistema se ocupe de buscar dicho término en es diccionario para después devolver el resultado obtenido.

La función que realiza el intercambio de idiomas es alternar entre las pantallas de las imágenes 8 y 9. La diferencia se centra en el texto que acompaña al campo donde se escribe la palabra (delimitado en los bocetos de la figura 9 con una elipse). De esta manera el sistema sabrá sobre qué idioma deberá buscar la traducción.

Las opciones posicionadas en las esquinas inferiores corresponden a volver a la pantalla anterior y a intercambiar el idioma del campo de texto respectivamente.

3.3.4. Cambiar el idioma

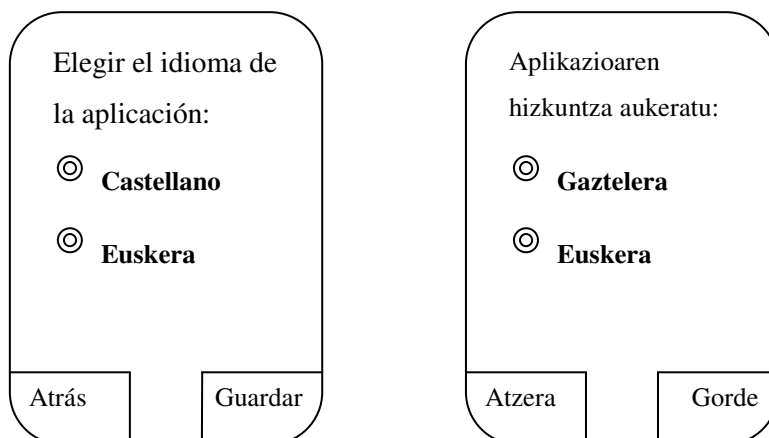


Figura 10: Pantalla para seleccionar el idioma

En esta pantalla se podrá seleccionar el idioma de la interfaz gráfica del Traductor. Al ser euskera y castellano los idiomas en los que se puede traducir una palabra, serán estas dos las opciones disponibles (aunque podrían ser otras). Una vez escogido el idioma habrá que pulsar sobre “Guardar” para que se realicen las modificaciones y mostrar el cambio realizado correctamente.

Pulsando “Atrás” volveremos a la pantalla del menú principal (figura 7).

4. ANÁLISIS

En este apartado de la memoria se realiza un análisis más detallado de los casos de uso que hemos obtenido en el Capítulo 3. Cada caso de uso se explicará con un diagrama de secuencia y su contrato.

Un diagrama de secuencia es una representación que muestra los eventos generados por los actores externos, su orden y los eventos del sistema. El objetivo es poder identificar mejor los eventos que se van a formar y el comportamiento del sistema sobre esos eventos.

Por otro lado, el contrato describe textualmente el comportamiento de la aplicación cuando se invoca una operación: si se producen cambios en las variables, cuáles son las salidas que proporciona, etc.

4.1. Caso de Uso Traducir

Diagrama de secuencia:

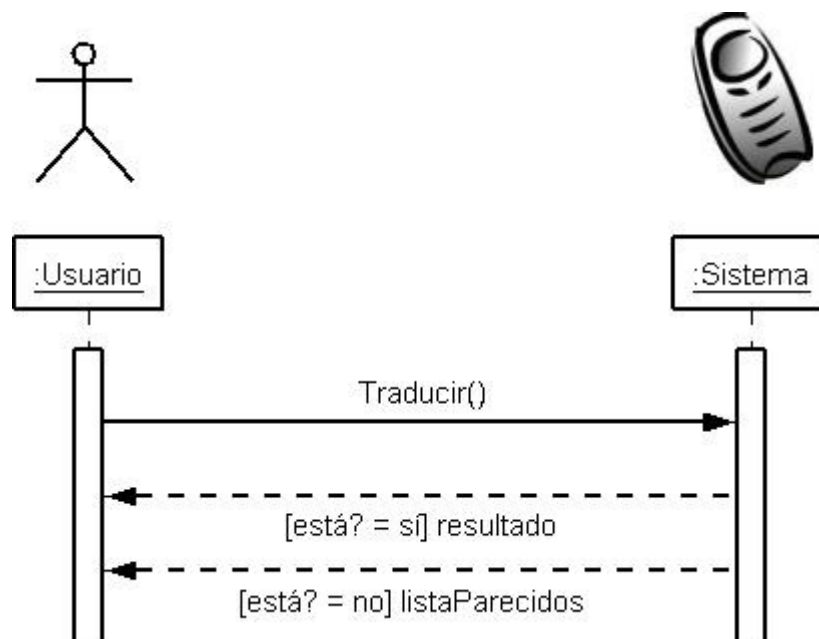


Figura 11: Diagrama de secuencia de “Traducir”

Contrato: Traducir ()

- Nombre: Traducir
- Responsabilidades: Extraer la palabra introducida y buscar la palabra en el diccionario. Devolver el resultado obtenido o una lista de los más parecidos.
- Precondición: Palabra no nula.
- Poscondición:
- Salida: Si la palabra introducida se encuentra en el diccionario el sistema mostrará la palabra origen y su resultado.

En caso de que la palabra esté mal escrita y/o no aparezca entre almacenadas se mostrará una pantalla con un listado de los términos más parecidos a la palabra origen.

4.2. Caso de Uso Intercambiar Idiomas

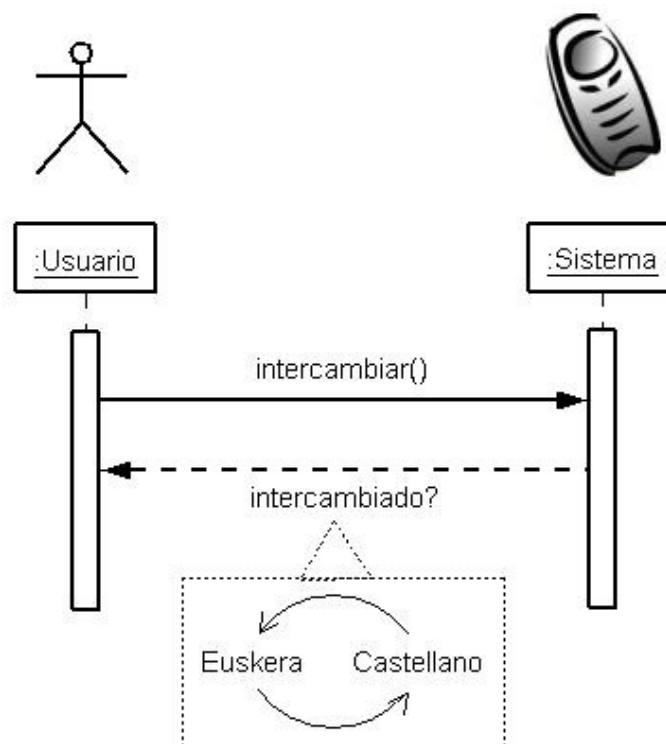
Diagrama de secuencia:

Figura 12: Diagrama de secuencia de “Intercambiar idiomas”

Contrato: intercambiar ()

- Nombre: intercambiar ()
- Responsabilidades: Cambiar el campo de texto de la pantalla actual para que coincida con el idioma de la palabra a traducir.
- Precondición: Estar situado en la pantalla de la función “Traducir”.
- Poscondición: La variable booleana, que controla si está intercambiada la pantalla o no, cambia.
- Salida

4.3. Caso de Uso Cambiar Idioma

Diagrama de secuencia:

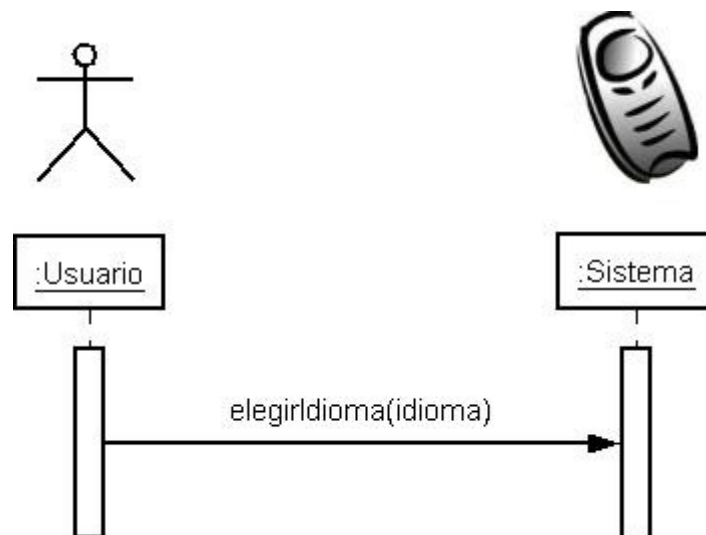


Figura 13: Diagrama de secuencia de “Cambiar Idioma”

Contrato: elegirIdioma (idioma)

- Nombre: elegirIdioma (idioma)
- Responsabilidades: Ofrecer la posibilidad de elegir entre los dos idiomas que muestra la pantalla.
- Precondición: Estar situado en la pantalla “Cambiar Idioma”.
- Poscondición: Marcar la posición de la opción elegida y habilitar poder guardar los cambios.
- Salida:

5. ARQUITECTURA DEL SISTEMA

Una vez decidido “qué” se va a hacer hay que pensar “cómo” lo haremos. En este capítulo se determina la organización estructural del Sistema Informático que queremos crear. Dependiendo de la información que tenemos sobre la programación para los móviles se seleccionarán los elementos estructurales más adecuados.

Nuestro sistema constará con dos capas que separarán las responsabilidades de la aplicación. La primera de ellas será la capa de presentación y de dominio, responsable de la interacción con el usuario y de las funcionalidades del sistema (control de la navegación entre las pantallas). Y posteriormente, la segunda será la capa de gestión de datos que se ocupará del almacenamiento de los datos.

La figura 14 muestra la distribución de las capas del sistema jerárquicamente.

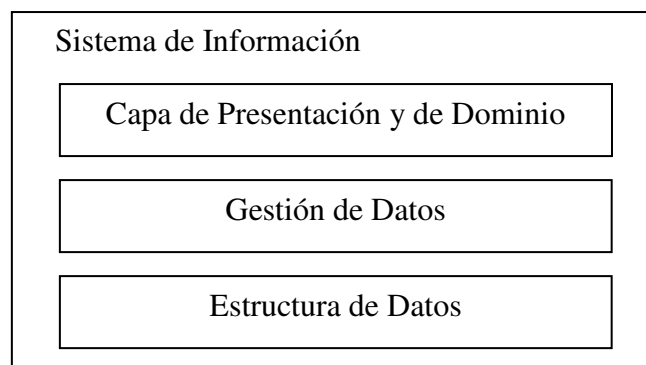


Figura 14: Arquitectura del Sistema

5.1. *Capa de Presentación y de Dominio*

Es la capa que conoce cómo obtener los datos del diccionario y como presentárselos al usuario. La sección 3.3 describe unos bocetos que representan las pantallas que mostrará el sistema. El objetivo principal es realizar esas pantallas de la manera más intuitiva posible y que sean de fácil manejo.

Al tratarse de una aplicación para un móvil la implementación deberá realizarse con un lenguaje apto para este tipo de dispositivos. Actualmente, la plataforma J2ME (Java MicroEdition) es la opción más segura para estos casos. Se trata de un subconjunto mínimo de los paquetes incluidos en Java que en consecuencia, permite crear clases de tamaño reducido.

La elección de Java es debido a su gran robustez e independencia de la plataforma donde se ejecutase el código. Con los años, Java ha progresado enormemente en varios ámbitos como servicios HTTP, servidores de aplicaciones, acceso a bases de datos (JDBC)... Se ha ido adaptando a las necesidades tanto de los usuarios como de las empresas ofreciendo soluciones y servicios tanto a unos como a otros. Debido a la explosión tecnológica de estos últimos años Java ha desarrollado soluciones personalizadas para cada ámbito tecnológico. La edición Java 2 Micro Edition fue presentada en 1999 por Sun Microsystems³ con el propósito de habilitar aplicaciones Java para pequeños dispositivos.

Aparentemente, esta disposición en dos capas puede parecer poco eficiente ya que al combinar las capas de presentación y de dominio las funcionalidades tienden a ser menos portables, cambiables y re-usables (ya que dependen de las interfaces gráficas del usuario y el diseño de las pantallas). Pero J2ME incluye un paquete adicional que proporciona un grupo de objetos básicos para el diseño de las interfaces que favoreces la portabilidad entre dispositivos.

En el siguiente capítulo se describirá esta plataforma y sus componentes más detalladamente para poder entender mejor la estructura de un programa en J2ME

5.2. Capa de Gestión de Datos

Esta capa es la responsable de interaccionar con el diccionario almacenado.

³ Sun Microsystems, Inc. (<http://es.sun.com/>)

Se diseñará un gestor de archivos implementado en Java que solamente se preocupará de cargar parte del diccionario a un *buffer*. Después deberá tratarlo debidamente para poder buscar una palabra o un listado de las más parecidas. Únicamente estará capacitado para devolver el resultado de la traducción, no para presentarlo visualmente al usuario.

6. ELECCIÓN TECNOLÓGICA

Como hemos comentado en la sección de la Capa de Presentación y Dominio, para nuestra aplicación era necesaria la utilización de un lenguaje de programación apto para dispositivos móviles, con la posibilidad de ser fácil de entender y que no creara un programa de gran capacidad. Una vez consultados foros de programación para móviles y compañeros con experiencia en la materia, se llegó a la conclusión final de que la plataforma J2ME, o Java 2 MicroEdition, era la más indicada para esta tarea. Además, disponía de la ventaja de poder ser desarrollado en entornos de trabajo que soportasen Java como Eclipse o NetBeans, añadiendo el kit de herramientas Wireless Toolkit (a partir de ahora WTK) de Sun Microsystems y el paquete con el entorno de desarrollo de Java j2sdk.

Las simulaciones sobre la interfaz de un teléfono móvil se pueden ejecutar directamente desde el espacio de trabajo donde implementaremos gracias al WTK. Por lo que no es necesaria la instalación de otras aplicaciones que ocupen espacio en nuestro ordenador.

Las características tecnológicas de las herramientas seleccionadas están descritas en las siguientes secciones:

6.1. J2ME (Java MicroEdition)

Java Micro Edition es la especificación de un subconjunto de la plataforma Java orientada al desarrollo de aplicaciones para dispositivos pequeños con capacidades limitadas tanto en pantalla gráfica como de procesamiento y memoria (teléfonos móviles, PDA's, handhels, buscas...). Actualmente las compañías telefónicas y los fabricantes de móviles están adaptando sus productos implantando protocolos y dispositivos necesarios para soportarla.

Sun Microsystems, empresa que diseñó el lenguaje Java, dividió la plataforma en distintos grupos dependiendo de las necesidades del usuario, creando así 3 ediciones: J2SE (Standard Edition), J2EE (Enterprise Edition, orientada a un entorno más empresarial) y J2ME (Micro Edition). En todas ellas se incluye la máquina virtual de Java y comparten un conjunto amplio de interfaces de programación de aplicaciones (APIs) básicas, principalmente los paquetes `java.lang` y `java.io`.

Se puede decir que J2ME representa una versión simplificada de J2SE, aunque con algunas diferencias y restricciones. La edición Micro agrupa algunas de las clases que tiene en común con la edición estándar y añade un paquete más denominado `javax.microedition`, ya que como se ha descrito antes contiene varias limitaciones de proceso y capacidad gráfica, y por lo tanto habrá clases destinadas solo para J2ME.

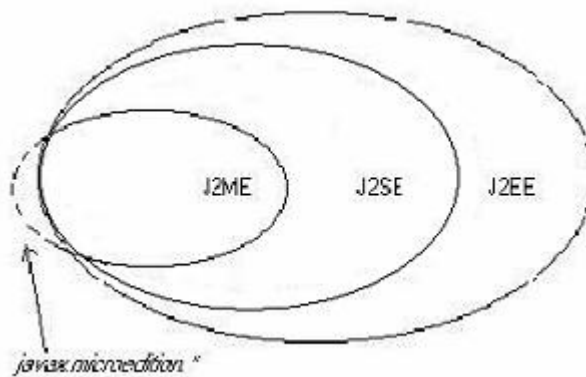


Figura 15: Relación entre las APIs de la plataforma Java

Una aplicación implementada en J2ME se compone de una selección de una máquina virtual Java, una configuración oportuna (CLDC o CDC), unos perfiles conocidos como MIDP (Mobile Information Device Profile) y, opcionalmente, de otros paquetes. Analicemos más a fondo estos componentes:

6.1.1. Máquina Virtual Java (JVM)

Las implementaciones tradicionales de JVM son en general muy pesadas en cuanto a ocupación de memoria, por lo que J2ME define dos máquinas virtuales adecuadas al ámbito de los dispositivos en los cuales queramos implementar. Su utilización dependerá de la configuración que seleccionemos para nuestra aplicación.

La máquina virtual de la configuración CDC se denomina CVM (Compact Virtual Machine); la de CLDC en cambio, KVM (proveniente de Kilobyte, referencia a la baja ocupación de memoria).

6.1.2. Configuraciones

Una configuración en Java Micro Edition es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos específicos. Como ya se ha visto, hay dos configuraciones diferentes, CLDC (Connected Limited Device Configuration) y CDC (Connected Device Configuration).

Para este proyecto se ha utilizado CLDC, puesto que está orientada a dispositivos con limitaciones en cuanto a capacidad gráfica, cómputo y memoria. Algunas de estas restricciones vienen dadas por el uso de la KVM. La CDC en cambio, está orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, decodificadores de televisión digital, televisores con Internet, algunos electrodomésticos y sistemas de navegación en automóviles.

Cuando la configuración CLDC se combina con un perfil de tipo MIDP, provee una plataforma Java sólida para desarrollar aplicaciones que se ejecuten en dispositivos con restricciones, como es en este caso.

6.1.3. Perfiles

El conjunto de APIs que controlan el ciclo de vida del programa, la interfaz de usuario, etc. componen el perfil. Éste identifica el grupo de dispositivos por la funcionalidad que proporcionan (móviles, electrodomésticos...) y el tipo de aplicaciones que se ejecutarán en ellos.

Por lo tanto, según el dispositivo que se quiera implementar variará el perfil a utilizar: Foundation Profile (orientado a dispositivos que carecen de interfaz gráfica), PDA profile y MIDP profile, entre otros.

Este último es el utilizado en este proyecto. El perfil MIDP ofrece las funcionalidades básicas requeridas por los servicios de los teléfonos móviles como son interfaces de usuario, conectividad a redes, almacenamiento local de datos y control sobre las aplicaciones. Las aplicaciones realizadas con MIDP son conocidas como *MIDlets*.

En resumen, un *MIDlet* es una aplicación Java realizada con el perfil MIDP sobre una configuración CLDC, dando como resultado programas como el descrito en esta memoria.

La siguiente imagen muestra de manera gráfica como se solapan las distintas partes de la plataforma J2ME. El recuadro marcado describe la estructura que sigue la aplicación “Traductor”.

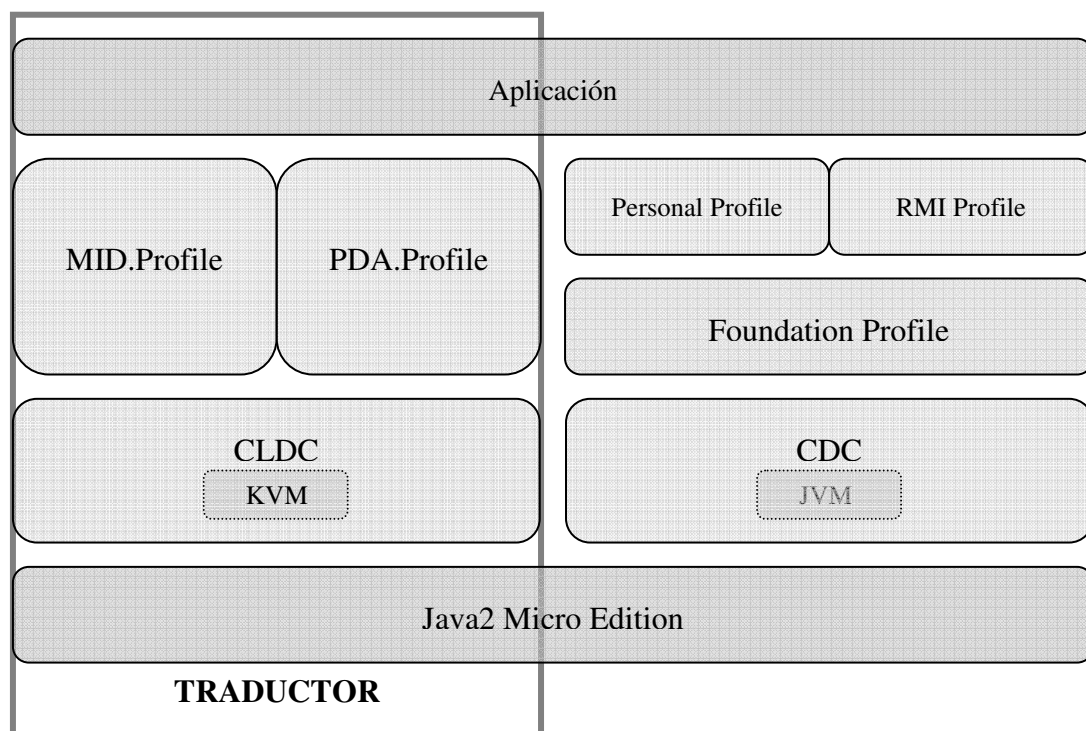


Figura 16: Arquitectura J2ME y estructura del Traductor

6.2. J2ME Wireless Toolkit (WTK)

Es una “caja de herramientas” para desarrollar aplicaciones Wireless que estén basadas en MIDP y CLDC, y diseñada para ejecutar en teléfonos móviles y otros pequeños dispositivos. El kit incluye entornos de emulación, posibilidad de optimizar y mejorar el rendimiento, documentación y ejemplos que ayudarán al desarrollador a crear programas de manera eficiente y eficaz.

Por otra parte, ofrece la función de poder cargar otras interfaces gráficas que emulan la imagen de un móvil. De esa manera el usuario puede trabajar sobre el dispositivo que vaya a utilizar en la realidad para comprobar las restricciones que conlleva. Algunas empresas fabricantes de móviles como Nokia y Motorola⁴, disponen de emuladores con muchos de sus modelos para una mejor simulación.

Esta herramienta es la responsable de crear los ficheros oportunos una vez compilado el programa. Genera dos ficheros que son los que se deben añadir a la memoria del dispositivo para su uso, uno es un archivo comprimido .JAR que es el que contiene el programa en sí y un archivo .JAD (Java Archive Descriptor) que contiene diversa información sobre la aplicación.

⁴ Véase Capítulo 13, Apartado 3 (Emuladores)

7. DISEÑO

7.1. Diagramas de Interacción

En este capítulo se encuentran los diseños de las operaciones resultantes de la fase del análisis. Cada operación de los casos de uso tiene asociado un diagrama de secuencia y una breve explicación de los eventos que realiza. Los diagramas de secuencia (o interacción) explican gráficamente cómo los objetos (instancias y clases) interactúan entre sí a través de mensajes.

7.1.1. Operación Elegir Idioma

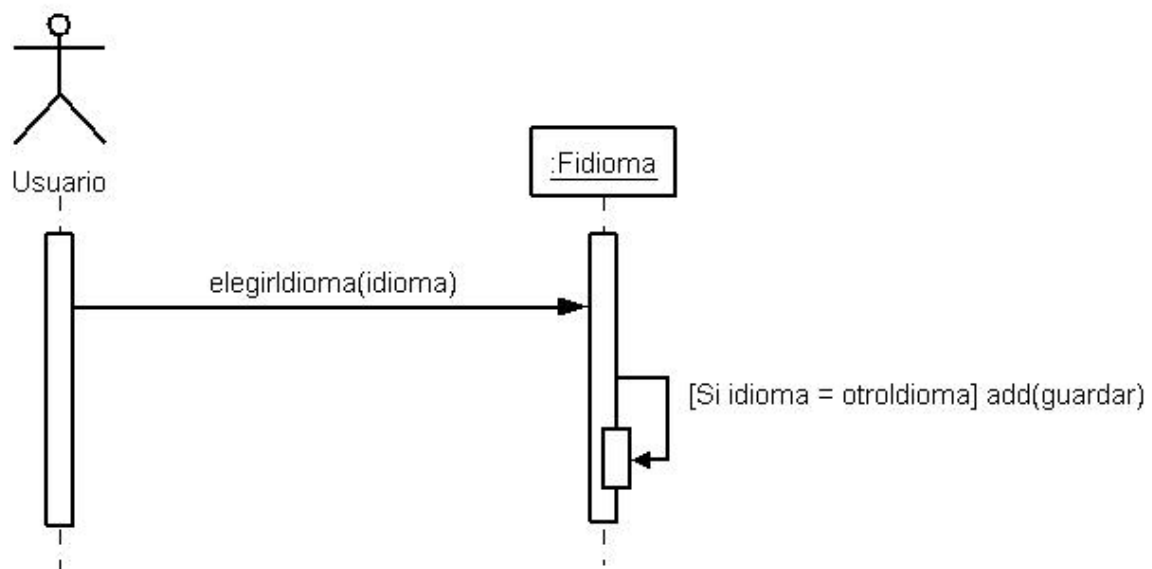


Figura 17: Diagrama de interacción de "Elegir Idioma"

Esta operación se podrá realizar situados en la pantalla para cambiar el idioma. Se mostrarán dos posibilidades entre las que elegir el idioma del sistema. Si la lengua elegida es la contraria a la que está marcada se añadirá una opción que permitirá realizar el cambio de las pantallas al idioma escogido.

7.1.2. Operación Traducir

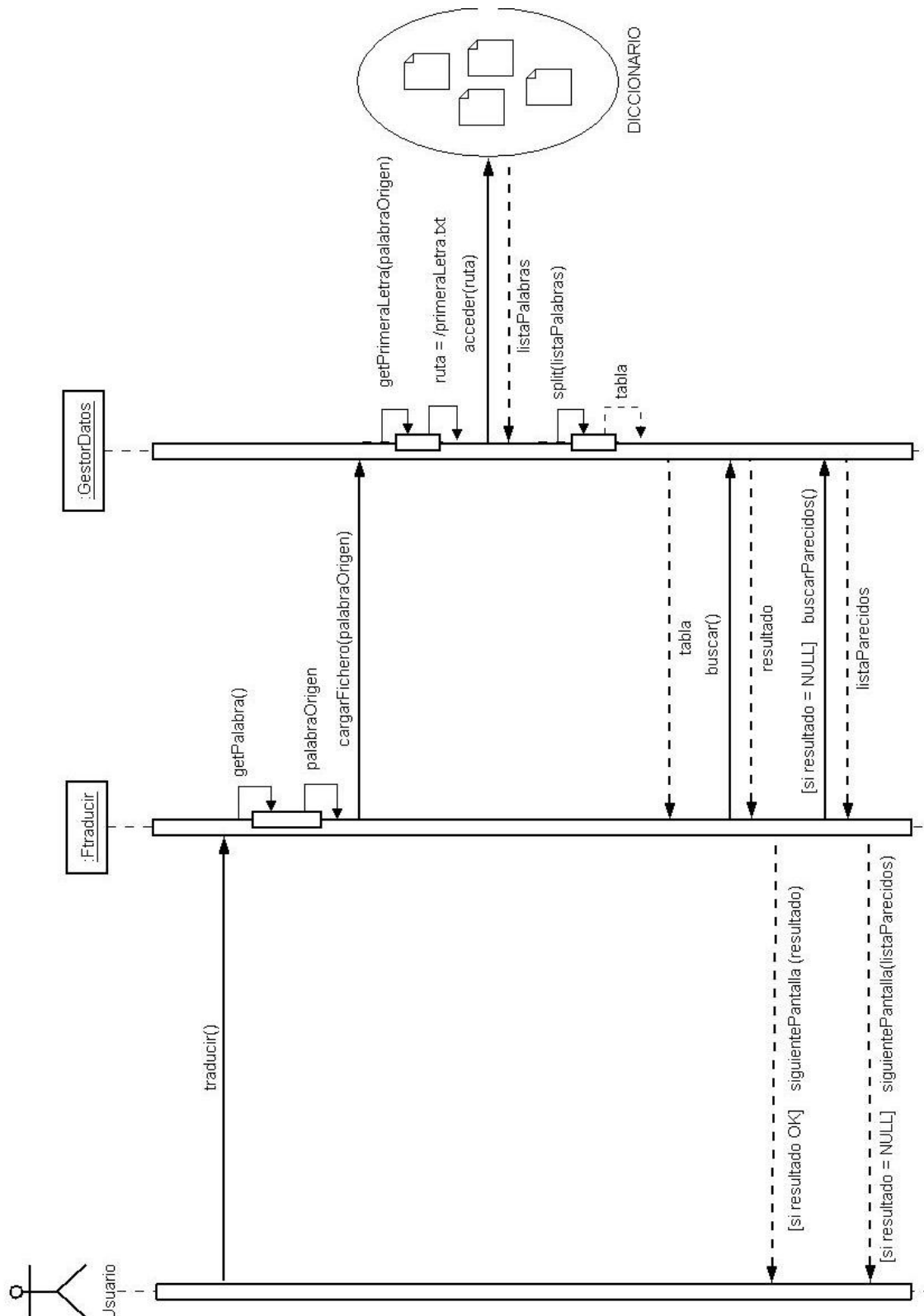


Figura 18: Diagrama de interacción de “Traducir”

Al ser ésta la tarea más fundamental del sistema es también la más desarrollada y complicada en su diseño. Lo que muestra el diagrama de la figura 18 son las acciones que se llevarán a cabo cuando el usuario utilice la operación “Traducir”. Como requisito para obtener algún resultado es indispensable haber escrito previamente una palabra en el campo de texto habilitado. En caso de no haber introducido nada la operación estará inhabilitada y no se mostrará ningún tipo de resolución.

Cuando se presiona “Traducir” el sistema extrae la palabra a traducir y llama a una operación del gestor de datos (`cargarFichero`). El gestor es el encargado de realizar el método correspondiente para acceder al diccionario almacenado y extraer las palabras que contengan la primera letra en común. Una vez obtenidos esos datos, el sistema los trata (operación `split()`) de manera que se pueda guardar cada elemento (palabra origen, categoría, traducción) en una tabla. Después, la función `buscar()` itera sobre la tabla obtenida.

Dependiendo del resultado de la búsqueda, la clase “Ftraducir” se comportará de dos maneras: Si se ha obtenido un resultado óptimo, la aplicación mostrará en una nueva interfaz la palabra introducida por el usuario y su traducción. Por el contrario, si no se ha obtenido ninguna traducción, se creará una lista con las palabras más parecidas y se mostrarán en una nueva pantalla.

7.1.3. Operación Intercambiar

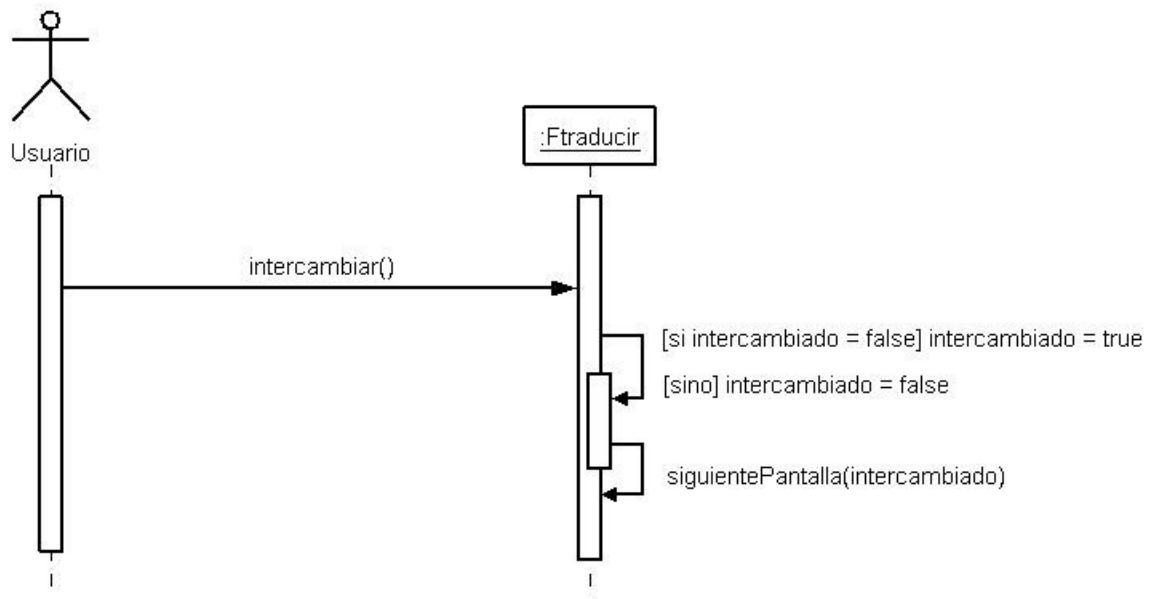


Figura 19: Diagrama de interacción de “Cambiar”

Lo que realizará esta función será alternar entre pantallas. Hace una comprobación del estado de la variable intercambiado y en base a ello la modifica y avanza a la siguiente interfaz.

Por ejemplo, si estamos situados en la pantalla “Traducir 1” (con el texto “Castellano” en el apartado habilitado para la escritura) lo que ocurrirá al llamar a la función será sustituir esa pantalla por la de “Traducir 2” (en el mismo apartado aparecerá el texto “Euskera”).

7.2. Diagrama de Clases

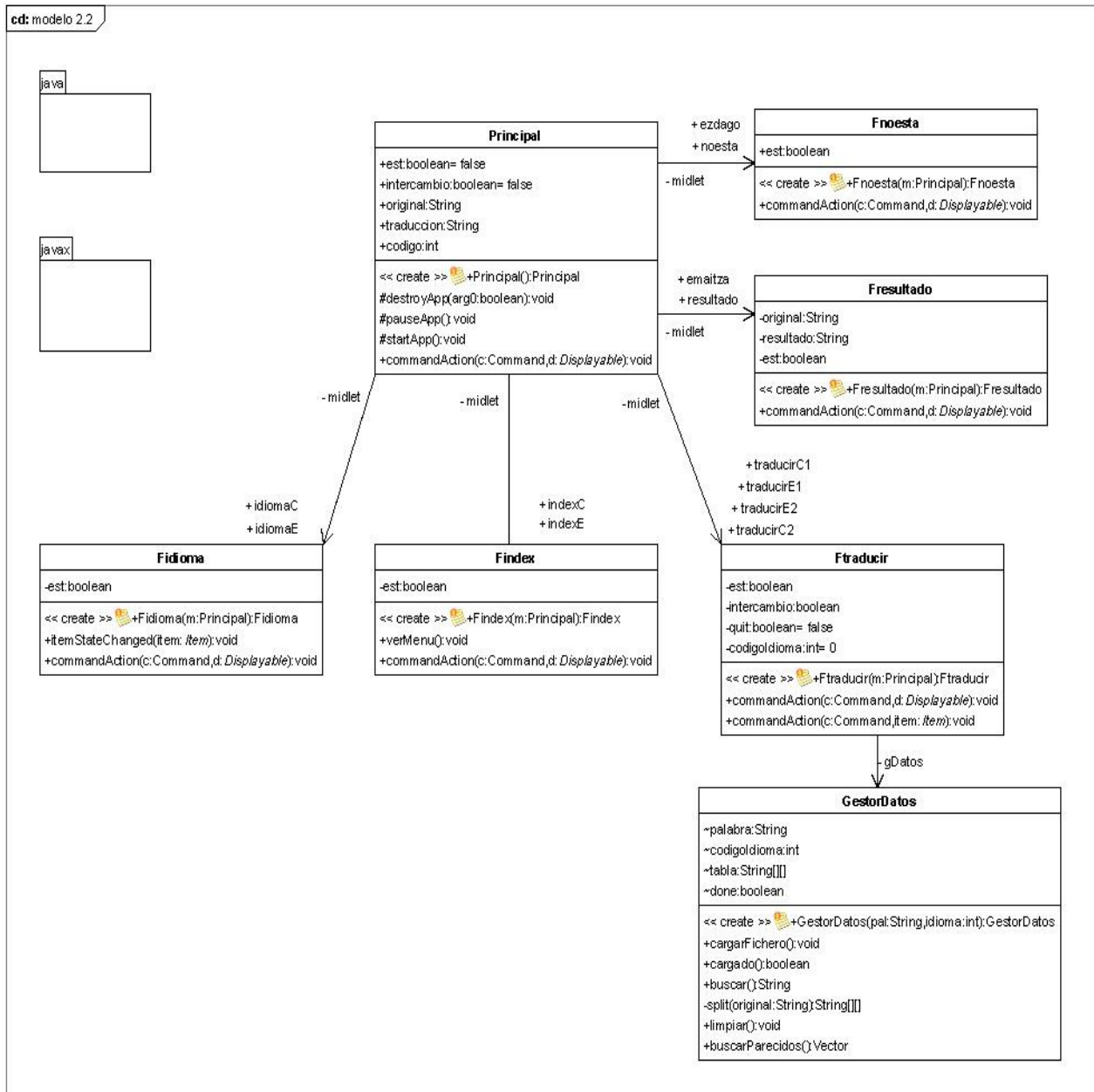


Figura 20: Diagrama de Clases del “Traductor”

La ilustración 20 muestra el diagrama de clases generado por ingeniería inversa. Se ha hecho uso del programa “Poseidón for UML Standard Edition” (copia de evaluación)⁵.

⁵ Gentleware (<http://www.gentleware.com/eval.html>)

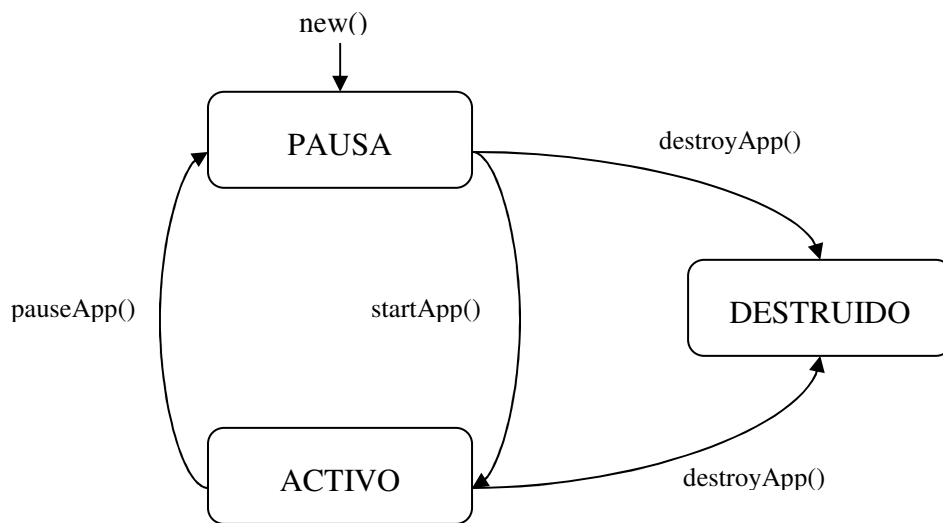
8. IMPLEMENTACIÓN

A continuación se detallan las características de la aplicación software: su estructura, funcionalidades y demás aspectos relacionados directamente con el producto desarrollado aquí.

8.1. ¿Qué es un *MIDlet*?

Como ya hemos comentado en el capítulo 6, los *MIDlets* son aplicaciones creadas usando la especificación MIDP diseñados para ser ejecutados en dispositivos de poca capacidad gráfica, de cómputo y de memoria. En estos dispositivos reside el AMS (Application Management System) o Gestor de Aplicaciones, software encargado de ejecutar, pausar o destruir los *MIDlets* y gestionar los recursos que estos ocupan.

Nuestro sistema es una aplicación J2ME (un *MIDlet*) que se ejecuta en la memoria del teléfono móvil. Una vez cargado en la memoria puede transitar entre tres estados diferentes: Activo, en pausa y destruido. Cuando un *MIDlet* comienza su ejecución, está en el estado “Activo” pero, ¿qué ocurre si durante su ejecución recibimos una llamada o un mensaje? El gestor de aplicaciones debe ser capaz de cambiar el estado de la aplicación en función de los eventos externos al ámbito de ejecución de la aplicación que se vayan produciendo. En este caso, el gestor de aplicaciones interrumpiría la ejecución del *MIDlet* sin que se viese afectada la ejecución de éste y lo pasaría al estado de “Pausa” para atender la llamada o leer el mensaje. Una vez que terminemos de trabajar con el *MIDlet* y salgamos de él, éste pasaría al estado de “Destruído” donde sería eliminado de la memoria volátil del dispositivo. Una vez finalizada la ejecución del *MIDlet* podemos volver a invocarlo las veces que queramos ya que éste permanece en la zona de memoria persistente hasta el momento que deseemos desinstalarlo.

Figura 21: Estados de un *MIDlet*

El gestor de aplicaciones cambia el estado de los *MIDlets* haciendo una llamada a cualquiera de los métodos mostrados en la figura 21, aunque un *MIDlet* también puede cambiar de estado por sí mismo.

J2ME habilita el paquete exclusivo `javax.microedition.midlet` que define las aplicaciones MIDP y su comportamiento respecto al entorno de ejecución.

8.2. Estructura general del *MIDlet*

Una aplicación en J2ME debe extender la clase `MIDlet` para que el AMS pueda gestionar sus estados y tener acceso a sus propiedades. La estructura que comparten todos los *MIDlets*, incluso el de este programa llamado “Principal”, es el siguiente:

```

import javax.microedition.midlet
public class miMIDlet extends MIDlet{
    public miMIDlet(){...}
    protected void destroyApp(){...}
    protected void pauseApp(){...}
    protected void startApp(){...}
  
```

Estos métodos son los que obligatoriamente tienen que poseer todos los *MIDlets*, ya que la clase que hemos creado tiene que heredar los tres métodos abstractos que posee la clase *MIDlet* y que han de ser implementados por cualquier *MIDlet*. El constructor se usa para crear los objetos que poseerá la aplicación y crear la referencia a la pantalla del dispositivo. En el método `startApp()` se activa la pantalla inicial de nuestro programa haciendo uso del método `setCurrent()`. El método `pauseApp()` está vacío ya que no es necesario su uso, aunque podríamos emitir si quisiéramos un mensaje al usuario informándole de que la aplicación está en estado de pausa. En el método `destroyApp()` se liberan recursos de memoria y se notifica al AMS que el *MIDlet* ha sido finalizado.

Todo *MIDlet* debe poseer por lo menos un objeto *Display*, clase que representa el manejador de pantalla y los dispositivos de entrada. Puede obtener información sobre las características de la pantalla, además de ser capaz de mostrar los objetos que componen nuestras interfaces. Para conseguir esta instancia se emplea la siguiente llamada realizada desde el constructor:

```
Display pantalla = Display.getDisplay(this)
```

De esta forma nos aseguramos de que el objeto *Display* esté a nuestra disposición durante toda la ejecución.

Hay que tener en cuenta que cada vez que salimos del método `pauseApp`, entramos en el método `startApp`, por lo que la construcción de las pantallas y demás elementos que forman parte de nuestro *MIDlet* las creamos en el método constructor.

En nuestra aplicación, la clase “Principal” es la encargada de heredar la clase *MIDlet* y sus métodos. En ella se inicializan todas las pantallas y elementos que conforman el sistema, de manera que podamos navegar entre ellas. También se implementa en el método `startApp()` la pantalla inicial de bienvenida para que sea cargada al ejecutar el *MIDlet* por primera vez.

8.3. Implementación de las interfaces gráficas mediante APIs (Interfaz de Programación de Aplicaciones)

Teniendo en cuenta la diversidad de aplicaciones que se pueden realizar para los dispositivos que soportan MIDP y los elementos que nos proporcionan la configuración CLDC y el perfil MIDP las interfaces se pueden dividir en dos grupos: APIs de alto nivel y APIs de bajo nivel.

La interfaz de alto nivel usa componentes tales como botones, cajas de texto, formularios, etc. Estos elementos son implementados por cada dispositivo y la finalidad de usar las APIs de alto nivel es su portabilidad. Al usar estos elementos, perdemos el control del aspecto de la aplicación ya que la estética de estos componentes depende del dispositivo donde se ejecute.

Sin embargo, también existen las interfaces de usuario de bajo nivel que ofrecen la opción de definir nuestros propios objetos y añadirlos a las librerías ya existentes para poder utilizarlos después. Al crear una aplicación usando las APIs de bajo nivel tendremos un control completo sobre los recursos del dispositivo, lo que implica tener que dibujar cada pantalla (definir la figura, el texto, los matices de colores y la posición de los elementos) y controlar los eventos de bajo nivel en cada una de ellas, como por ejemplo el rastreo de pulsaciones de teclas. Implementar en este nivel es más conveniente para un programador experto que conozca todas las posibilidades existentes.

Aunque el programa en alto nivel sea menos llamativo, dispone de una ventaja muy importante: la portabilidad. Usando los APIs básicos se gana un alto grado de portabilidad de la misma aplicación entre distintos dispositivos, mientras que los de bajo nivel dependerán del soporte gráfico del terminal.

Las interfaces gráficas de este proyecto han sido implementadas con APIs de alto nivel. El paquete `javax.microedition.lcdui` incluye las clases necesarias para crear interfaces de usuario. Cada pantalla se crea por separado, donde además de crear los elementos que forman parte de ella, captura los eventos que produzcan los botones que insertemos y escribiremos el código de la acción asociada a ese evento. Con este procedimiento, podemos crear aplicaciones con el número de pantallas que deseemos, ya que al encontrarse el código de cada pantalla en un archivo `.java` diferente, su depuración y mantenimiento es mucho más simple. Hay que pensar también en las posibles futuras actualizaciones o modificaciones de nuestro *MIDlet*. Al estar modularizada nuestra aplicación, el trabajo de modificación del código asociado a las pantallas es mucho más sencillo y se realizaría de una manera más rápida.

Las pantallas están formadas por listas (como en el caso del menú principal) y por formularios. La clase `Form` es un componente que actúa como contenedor de un número indeterminado de objetos como son `StringItem` (cadenas de texto) y `TextField` (campos de texto) utilizados en este caso. Por otro lado, la clase `List` permite construir pantallas que poseen una lista de opciones. Dispone de 3 tipos diferentes de lista, pero han sido dos los utilizados en este proyecto:

- **IMPLICIT**: Utilizado en la clase “Index”, la elección de un elemento provoca un evento.
- **EXCLUSIVE**: Un sólo elemento puede ser seleccionado una vez. Es muy útil en la clase “Fidioma”, que da la posibilidad de elegir entre euskera y castellano.
- **MULTIPLE**: Cualquier número de elementos pueden ser seleccionados a la vez.

El *MIDlet* “Traductor” básicamente se ocupará de darnos a elegir qué función queremos realizar a través de un objeto `List` y comunicarse con cada uno de los formularios de cada operación.

8.4. Manejo de los botones y eventos

Un objeto de la clase `Command` mantiene información sobre un evento. Generalmente se implementan en los *MIDlets* cuando se quiere detectar y ejecutar una acción simple. Al declarar un objeto de este tipo se especifica el texto que aparecerá en la pantalla, el tipo de objeto que se quiere crear (`OK`, `CANCEL`, `BACK`, `ITEM`, `STOP...`) para asignarle una apariencia u otra y la prioridad con la que el AMS establece un orden.

Para que un botón de tipo `Command` realice la acción que se desea de acuerdo a su tipo, se tiene que implementar el método que hereda la interfaz `CommandListener`. Es más, cualquier *MIDlet* que incluya `Commands` tiene que implementar esa interfaz con sus correspondientes métodos. En este caso, sólo se incluye el método `commandAction(Command c, Displayable d)`, donde se indica la acción a realizar cuando se produzca un evento en el botón `c` que se encuentra en el objeto `d`.

El manejo de eventos de un formulario se hace de manera muy similar al de los `Commands`. En este caso hay que implementar la interfaz `ItemStateListener` que contiene el método abstracto `itemStateChanged(Item item)`. Cuando se realiza algún tipo de acción en un ítem de un formulario, se ejecuta el código asociado a ese ítem que esté definido en el método `itemStateChanged(Item item)` de igual forma que con el método `commandAction(...)`. Este tipo de evento ha sido utilizado a la hora de traducir una palabra. En la pantalla “Ftraducir” se muestra un elemento con apariencia diferente que al presionarlo ejecuta la acción referida a la traducción.

Los botones generales “Entrar”, “Atrás”, “Guardar” e “Intercambiar Idiomas” están implementados como objetos de tipo `Command`, en los que cada uno tiene una acción diferente a ejecutar.

8.5. Acceso al Diccionario

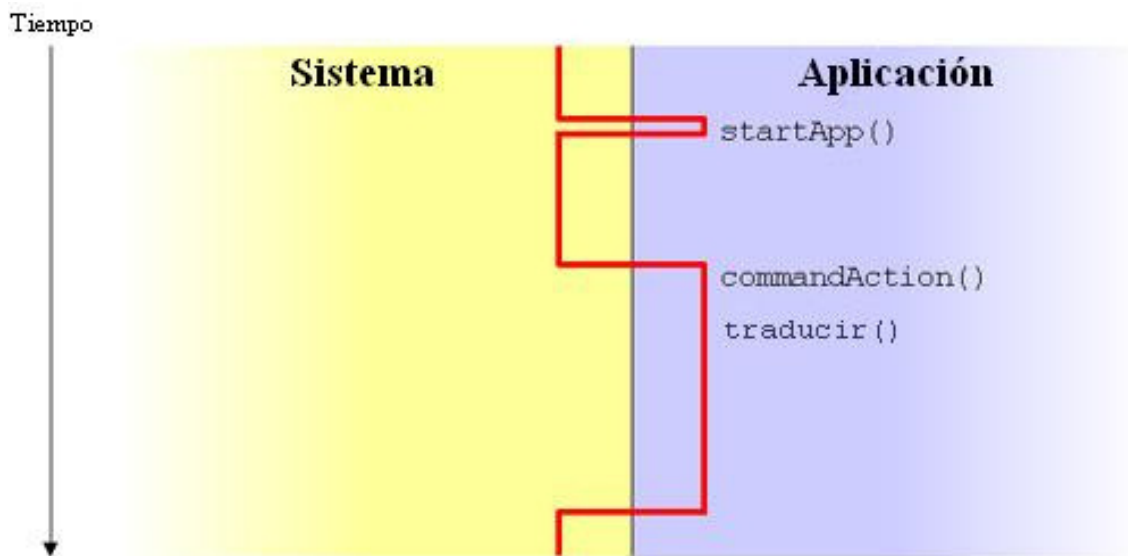
Se deliberó utilizar en la capa de datos un Sistema Gestor de Bases de Datos (SGBD) específico para los móviles, ya que estos dispositivos no soportan el SGBD utilizado en las aplicaciones Web. A medida que se analizaban las funcionalidades de la aplicación se descubrió que las palabras no iban a ser modificadas o borradas por el usuario, solo serían consultadas. Por lo tanto, el uso de una base de datos para únicamente realizar consultas era innecesario.

Finalmente se tomó la decisión de distribuir las palabras del diccionario en varios ficheros de texto y almacenarlos en la propia aplicación. De esta manera, se simplificaba el acceso a los datos desde el gestor.

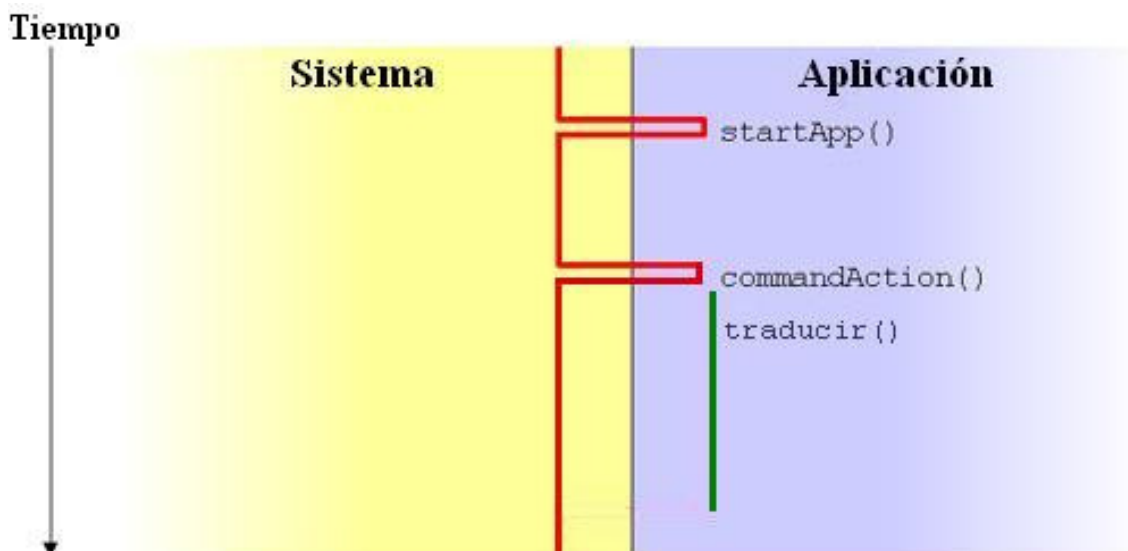
La conexión al diccionario se realiza de distinta manera dependiendo del tipo del almacenamiento de los ficheros. El paquete `javax.microedition.io` contiene numerosas clases que permiten crear y manejar diferentes conexiones de red y el paquete `java.io` se encarga de proporcionar las clases necesarias para leer y escribir en estas conexiones. Este tipo de conexión es útil cuando los archivos están almacenados en un servidor.

Dado que en este sistema los ficheros de texto están incluidos en el *jar* de la propia aplicación, la lectura de estos es diferente. Se accede a ellos a través de del método `getResourceAsStream(String)` que devuelve un `InputStream` asociado al fichero que se pasa por parámetro al método anterior. Esta acción está implementada en la clase “GestorDatos”, en la que también se implementan las operaciones de tratamiento de datos y búsquedas.

Por motivos de seguridad, la carga del fichero se ejecuta en un proceso hilo o *thread*. En la lectura de los datos puede ocurrir una excepción de tipo `IOException` que bloquee nuestro Terminal. Para evitarlo, se ejecuta en un segundo plano que, en caso de bloquearse, permita al usuario poder utilizar el resto de funciones o salir de la aplicación.

Figura 22: Sistema sin el uso del *Thread*

La imagen 22 muestra el exceso de trabajo que puede llegar a soportar la aplicación en caso de que la operación tenga que ejecutar código pesado. Incluso si la operación se queda bloqueada, el método no acabaría y no retornaría el control al sistema, provocando así un interbloqueo. El uso de un proceso hilo evita este tipo de situaciones (imagen 23).

Figura 23: Sistema con el uso del *Thread*

8.6. Restricciones del CLDC

La configuración CLDC se ocupa de las siguientes áreas:

- Lenguaje Java y características de la máquina virtual.
- Librerías del núcleo de Java (`java.lang.*` y `java.util.*`).
- Entrada / Salida.
- Comunicaciones
- Seguridad.

Lo que se intenta con la configuración CLDC es mantener lo más pequeño posible el número de áreas abarcadas. Es mejor restringir el alcance de CLDC para no exceder las limitaciones de memoria.

Las diferencias entre CLDC y la plataforma J2SE se deben, principalmente, a dos aspectos: Diferencias entre el lenguaje Java y el subconjunto de éste que conforman las APIs de CLDC y diferencias entre la JVM y la JVM que usa CLDC. Profundicemos un poco en algunas de estas diferencias y a explicarlas con más detenimiento:

8.6.1. No existe soporte para operaciones en punto flotante.

Esta capacidad ha sido eliminada porque la mayoría de los dispositivos CLDC no poseen el hardware que les permita realizar estas operaciones, y el coste de realizar operaciones en punto flotante a través de software es muy alto.

8.6.2. Limitaciones en el manejo de errores

Una máquina virtual Java en CLDC soporta el manejo de excepciones. Sin embargo, el conjunto de las clases de error incluidas en las librerías CLDC ha sido limitado. Sólo cuenta con un subconjunto de las excepciones disponibles para J2SE, por lo que el manejo de éstas queda limitado. La clase `java.lang.Error` y todas sus subclases han sido completamente eliminadas de las librerías CLDC.

Estas limitaciones se pueden explicar debido a que muchos de los sistemas que hacen uso de CLDC poseen su propio manejo interno de errores.

8.6.3. Librerías limitadas

Un cierto número de características han sido eliminadas de la JVM/CLDC porque las librerías incluidas en CLDC son bastante más limitadas que las librerías incluidas en J2SE. La mayoría de las librerías incluidas en CLDC son un subconjunto de las incluidas en las ediciones J2SE y J2EE de Java. Esto es así para asegurar la compatibilidad y portabilidad de aplicaciones. En total, CLDC usa unas 37 clases provenientes de los paquetes `java.lang`, `java.util` y `java.io`.

8.6.4. Seguridad

Debido a las características de los dispositivos englobados bajo CLDC, en los que se hace necesaria la descarga de aplicaciones y la ejecución de éstas en dispositivos que almacenan información muy personal, se hace imprescindible hacer un gran hincapié en la seguridad. Hay que asegurar la integridad de los datos transmitidos y de las aplicaciones.

Los dispositivos incluidos en CLDC se encuentran ante un modelo de seguridad similar al de los *applets* (programas Java que se ejecutan en un navegador Web). En ellas la ejecución se realiza en una zona de seguridad denominada *sandbox*. Este modelo establece que sólo se pueden ejecutar algunas acciones que se consideran seguras. Existen, entonces, algunas funcionalidades críticas que están fuera del alcance de las aplicaciones. De esta forma, los programas deben cumplir unas condiciones previas antes de ser ejecutadas en los dispositivos móviles:

- Los ficheros de clases Java deben ser verificados como aplicaciones Java válidas.
- Sólo se permite el uso de APIs autorizadas por CLDC.
- Sólo se puede acceder a características nativas que entren dentro del CLDC.
- Una aplicación ejecutada bajo KVM no debe ser capaz de dañar el dispositivo dónde se encuentra. De esto se encarga el verificador de clases que se asegura que no haya referencias a posiciones no válidas de memoria.

Estas restricciones han hecho que la primera estrategia de implementación que tenía pensada se tuviese que modificar. Es el caso de la utilización de `BufferedReader` para el tratamiento del fichero, ya que como no forma parte del subpaquete de `java.io` no es posible su utilización. Además, no es posible generar objetos que ocupen mucho espacio porque saltaría una excepción de tipo `java.lang.OutOfMemoryException`. Para que la carga del fichero a la tabla sea lo más pequeña posible se han dividido las palabras por idioma y la primera letra de ellas. Los nombres de esos ficheros seguirán el formato “cX” o “eX”, donde la c y la e responden a Castellano y Euskera respectivamente y la X cada letra del abecedario. El sistema cargará solo una parte del diccionario dependiendo de la palabra introducida por el usuario y el idioma.

En resumen, dependiendo del dispositivo utilizado las opciones para poder implementar cambian, puesto que a más capacidad de memoria y más resolución gráfica mejores prestaciones se pueden añadir. No existirían tantas restricciones y se podrían utilizar más clases y mejores interfaces gráficas.

8.7. Descripción de las Clases

Finalmente, las clases implementadas para la aplicación “Traductor” se describen de manera resumida y esquemática en la tabla que se muestra a continuación.

CLASE	DESCRIPCIÓN
Principal	Clase que hereda el <code>MIDlet</code> , encargada de inicializar y activar el sistema. Posee también la instancia del <code>Display</code> para el cambio de las pantallas.
Findex	Clase que implementa la pantalla del menú principal. Muestra una lista de tipo <code>IMPLICIT</code> de las funciones posibles a realizar.
Ftraducir	Implementa la interfaz gráfica de la pantalla de la función “Traducir”. En ella se realiza el llamamiento al gestor de datos mediante la utilización de un <code>thread</code> .
GestorDatos	Responsable de acceder al fichero deseado y cargar los datos en el buffer. Implementa los métodos <code>split()</code> (para el tratamiento de las palabras), <code>buscar()</code> y <code>buscarParecidos()</code> .

Fresultado	Interfaz gráfica de la pantalla con el resultado de la traducción si la hubiera. Depende de la palabra introducida en Ftraducir.
Fnoesta	Se encarga de mostrar una lista de las palabras más parecidas a la insertada. La implementación también depende del resultado de Ftraducir.
Fidioma	Implementa la función que realiza el cambio del idioma. Muestra una lista de tipo EXCLUSIVE.

Figura 24: Tabla de Clases

9. PRUEBAS

Una vez finalizada la fase de diseño y habiendo implementado la aplicación de manera básica, se lleva a cabo la fase de pruebas. Esta fase consiste en poner a prueba nuestro código para comprobar si es correcto o responde a lo que se pretende en un principio; para en caso contrario, poder realizar los cambios pertinentes.

Un buen caso de prueba es aquel que tiene una probabilidad muy alta de descubrir un nuevo error. Se deben diseñar y ejecutar grupos de pruebas que detecten de forma sistemática distintos tipos de errores. En este caso las pruebas se centran en las funcionalidades siguiendo una estrategia de pruebas de “caja negra”, es decir, el juego de pruebas se diseña considerando las responsabilidades del componente. Para solucionar los errores surgidos en ese grupo de pruebas se analiza el código de los componentes que los han causado de manera unitaria.

9.1. Pruebas de “caja negra”

Este grupo de pruebas permite detectar errores en la asignación de responsabilidades y la interfaz del componente. Se realizarán ensayos en las pulsaciones de los botones y en las funcionalidades que ofrece el sistema.

9.1.1. Pulsar un botón de la interfaz

Al presionar sobre un evento que ejecute una acción debe realizarla. Es el caso de los botones “Entrar”, “Salir”, “Atrás”, “Traducir”, “Intercambiar Idiomas” y “Guardar”. Se comprueba que pulsar uno de ellos produzca la acción correspondiente y correctamente.

9.1.2. Escribir una palabra en la función Traducir

En este apartado se detallarán los diferentes casos de prueba que pueden ocurrir al introducir una palabra en el campo de texto de la pantalla.

9.1.2.1. Campo de texto vacío

Si el usuario no introduce ninguna palabra y presiona el botón que realiza la función de Traducir, el sistema no hará nada. A ojos del usuario parecerá que la opción está invalidada. La operación de traducir solo se ejecutará si hay algún carácter escrito.

9.1.2.2. Palabra correcta

Cuando el usuario introduzca una palabra que se encuentre en algún fichero de texto, la aplicación debe mostrar una nueva pantalla con el resultado de la traducción. Además, tendrá la posibilidad de volver a la pantalla anterior para poder introducir otra palabra.

9.1.2.3. Palabra inexistente o mal escrita

Si por el contrario, se ha escrito mal un término y por consiguiente no aparece en el diccionario, la aplicación le muestra al usuario una lista de palabras parecidas a introducida. Esta función permite al usuario darse cuenta del error y volver atrás para reescribir la entrada.

9.1.2.4. Límite del campo de texto

El campo de texto está limitado a 15 caracteres como máximo. Si no se estableciese un límite esta capacidad podría ser mayor que la que el dispositivo puede mostrar a la vez (por ejemplo si una tecla se queda bloqueada o presionada).

9.1.3. Límite de palabras

A la hora de traducir se carga parte del diccionario en un objeto para poder tratarlo, esta carga le supone mucha memoria al dispositivo por lo que puede llegar a limitar el número de palabra a traducir. Al final del método se elimina el contenido de la tabla para liberar espacio, pero de todas maneras se ha comprobado si puede ocurrir una excepción relativa a la memoria.

Se ha introducido una palabra en el campo de texto y se ha procedido a traducirla un número de 20 veces. El dispositivo ha respondido correctamente sin bloquearse, aunque un monitor de memoria activado muestra que por cada acción se requieren unos pocos de bytes.

9.2. Pruebas de “caja blanca”

Este tipo de pruebas se diseñan analizando y ejecutando el código del componente. Permite detectar errores de flujo de control y caminos de tratamiento de errores.

9.2.1. Formato de las interfaces gráficas

Si al ejecutar el código las pantallas no se muestran como se deseaba (mal posicionamiento, tipo de letra...) se revisa el código de la pantalla en cuestión para modificarlo.

9.2.2. Control del acceso al fichero

Para estudiar un correcto control del acceso al fichero necesario se muestra por la salida estándar una serie de sentencias que imprimen el camino que sigue: La ruta del fichero al que ha accedido, si la carga al *buffer* ha sido correcta...

Cuando ocurren errores en la conexión se analiza el resultado de la salida estándar para averiguar en qué parte del camino ha ocurrido el problema y poder solucionarlo en el menor tiempo y esfuerzo posible.

9.3. Otras pruebas

Existen otros tipos de casos de prueba para ensayar las interacciones entre componentes del sistema, como son las pruebas de integración, de validación y de sistema.

Dado que por falta de tiempo no ha sido posible la implantación de la aplicación en un teléfono móvil real, no se han ejecutado este tipo de pruebas para ver el funcionamiento en un entorno real. Era interesante realizarlas ya que el emulador no presenta restricciones de memoria y no interactúa con otras funciones del dispositivo.

10. IMPLANTACIÓN

10.1. Requerimientos

La especificación CLDC define un estándar para pequeños dispositivos conectados con recursos limitados con las siguientes características:

- 160 Kb a 512 Kb de memoria total disponible para la plataforma Java.
- Procesador de 16 bits o 32 bits.
- Bajo consumo, normalmente el dispositivo usa una batería.
- Conectividad a algún tipo de red, normalmente inalámbrica, con conexión intermitente y ancho de banda limitado (unos 9600 bps).

CLDC está diseñado para ejecutarse en una gran variedad de dispositivos, desde aparatos de comunicación inalámbricos como teléfonos móviles y buscapersonas, hasta organizadores personales, terminales de venta, etc. Las capacidades del hardware de estos dispositivos varían considerablemente y por esta razón, los únicos requerimientos que impone la configuración CLDC son los de memoria. La configuración CLDC asume que la máquina virtual, librerías de configuración, librerías de perfil y la aplicación debe ocupar una memoria entre 160 Kb y 512 Kb. Más concretamente:

- 128 Kb de memoria no volátil para la JVM y las librerías CLDC.
- Al menos 32 Kb de memoria volátil para el entorno de ejecución Java y objetos en memoria.

Este rango de memoria varía considerablemente dependiendo del dispositivo al que hagamos referencia. Como requerimientos software, generalmente, la configuración CLDC asume que el dispositivo contiene un mínimo Sistema Operativo (S.O.) encargado del manejo del hardware de éste. Este S.O. debe proporcionar al menos una entidad de planificación para ejecutar la JVM.

10.2. Empaquetamiento

Antes de nada, el entorno de desarrollo utilizado debe encargarse de compilar y preverificar las clases java para convertirlas en ficheros de tipo *.class*.

En esta fase se va a empaquetar el *MIDlet* y dejarlo totalmente preparado para su descarga sobre el dispositivo. Para ello hay que tener que construir dos archivos: Un archivo JAR con los ficheros que forman el *MIDlet*, y un archivo descriptor de la aplicación que es opcional. El archivo JAR está formado por los siguientes elementos:

- Un archivo manifiesto que describe el contenido del archivo JAR.
- Las clases Java que forman el *MIDlet*.
- Los archivos de recursos usados por el *MIDlet*.

10.3. Descarga al dispositivo

En la teoría, para cargar la aplicación a un teléfono móvil real, basta con copiar el archivo *.JAR* a la memoria del dispositivo. La descarga se puede realizar por medio de USB, infrarrojos y *bluetooth* si lo soporta.

En la práctica no se ha podido comprobar si es factible este método, debido a la falta de tiempo en poder buscar más información y los plazos fijados de la entrega del proyecto. Pero se tiene constancia de ello como una mejora del sistema.

11. GESTIÓN DEL PROYECTO

En este capítulo se recoge el esfuerzo realizado y su distribución durante el proyecto. Con los datos obtenidos se realiza una comparativa con las tareas planificadas y los tiempos estimados.

TIEMPO TOTAL INVERTIDO	560 HORAS
-------------------------------	------------------

Frente a las 400 planeadas, se deduce que ha sido necesario empeñar mas esfuerzo en las actividades. Además, algunos plazos fijados no se han cumplido, por lo que algunos procesos han precisado más horas en menores intervalos de tiempo. Los siguientes gráficos muestran el tiempo necesario (en horas) para realizar los procesos.

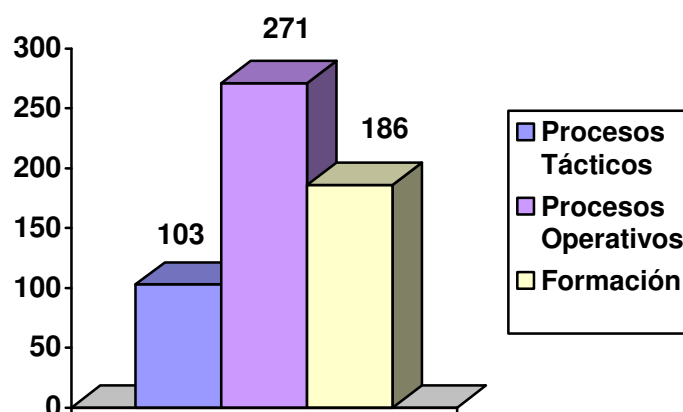
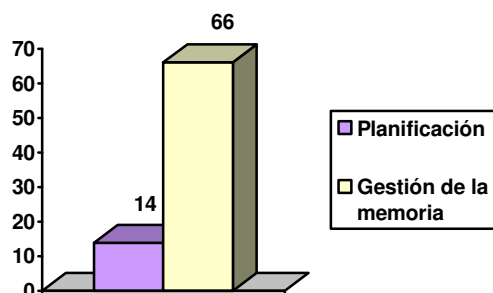


Figura 25: Gráfica de relación de horas por proceso real

Procesos Tácticos Estimados



Procesos Tácticos Reales

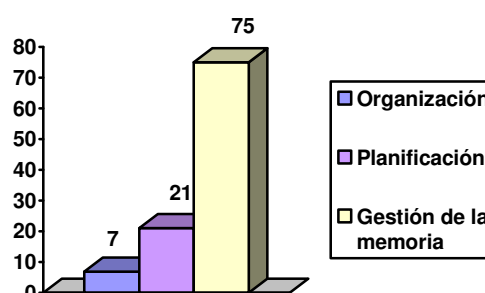
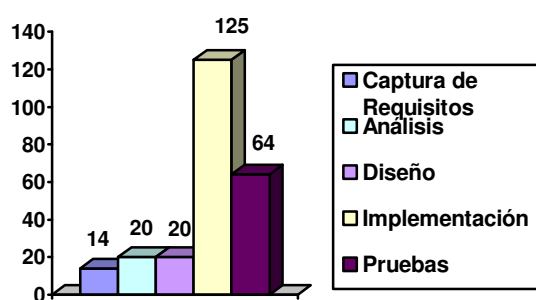


Figura 26: Gráficas de la distribución de los procesos tácticos estimados y reales

En la figura 26 se muestra el esfuerzo estimado y el real. Se aprecia que la tarea de organización no aparece en los procesos planificados del DOP⁶. Esto es debido a la reorganización sufrida durante el proyecto. Puesto que algunos plazos estimados no se han cumplido, ha sido necesario dedicar un tiempo a volver a organizar el trabajo restante.

Procesos Operativos Estimados



Procesos Operativos Reales

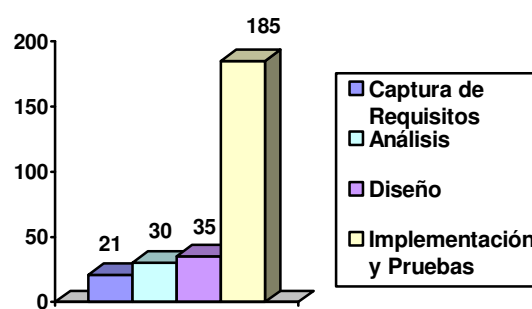
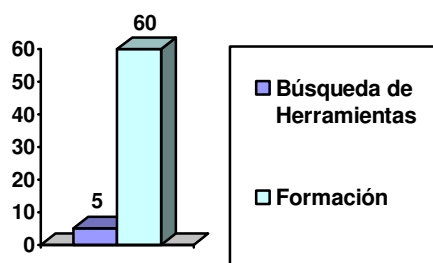


Figura 27: Gráficas de la distribución de los procesos operativos estimados y reales

En este caso la diferencia se centra en la distribución de las tareas de implementación y pruebas. En la planificación se estimó que primero se implementaría un programa básico y después se realizarían las pruebas. Finalmente, se recurrió a efectuar las dos tareas conjuntamente. Se implementaban los métodos y se probaban individualmente hasta que funcionasen correctamente.

Procesos Formativos Estimados



Procesos Formativos Reales

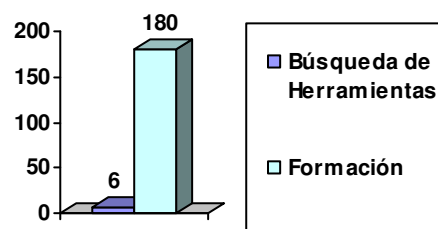


Figura 28: Gráficas de la distribución de los procesos formativos estimados y reales

⁶ Véase Capítulo 2, Apartado 4 (Planificación Estimada)

En la imagen 28 es notable el gran aumento de esfuerzo que precisó la tarea de formación. En la gráfica de la izquierda, el cálculo se ha realizado suponiendo una media de 30 días en la formación. El esfuerzo de la formación real (45 días) no difiere mucho del estimado. Pero se necesitaron más horas para realizar los ejemplos y para solucionar los problemas que surgieron durante la implementación y las pruebas.

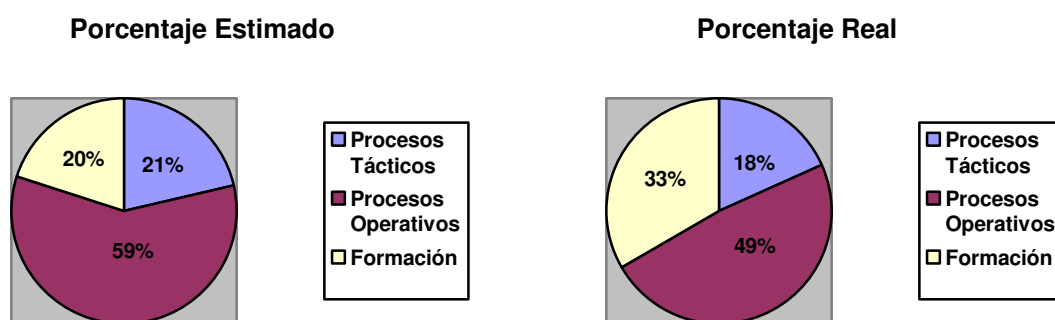


Figura 29: Comparativa de porcentajes de tiempo

En resumen (figura 29), se destaca un menor porcentaje en los procesos operativos. La razón de tal efecto se basa en que las pruebas se realizaron durante la implementación. Por lo que al ejecutar la aplicación para comprobar si el código era correcto se procedía también a verificar los casos de prueba. Se aprecia también un mayor porcentaje en el ámbito de la formación. Han sido necesarias muchas horas de documentación y consultas para comprender la estructura de una aplicación en J2ME, y se han realizado pequeños ejemplos para entender las prestaciones que ofrecía.

12. CONCLUSIONES

La finalidad principal de este proyecto era ofrecer a un usuario de un teléfono móvil la posibilidad de traducir una palabra al euskera o castellano cuando la situación lo necesitase. Estoy satisfecha ya que creo que se ha cumplido ese propósito. He sido capaz de crear por mi misma un sistema que permite sustituir un diccionario a cambio de un poco de espacio en nuestro móvil y facilitar la búsqueda de una traducción.

Me hubiera gustado disponer de más tiempo para poder implantar la aplicación en un teléfono móvil real. De esa manera podría comprobar las limitaciones que pueda ocasionar mi código y mejorarlo. No descarto la posibilidad de seguir trabajando en él en un futuro, ya que me ha interesado mucho la programación para los móviles.

La plataforma J2ME me ha parecido algo complicada cuando se trata de dispositivos de poca capacidad. Es la primera vez que programo con *MIDlets*, y aún habiendo hecho mil y una aplicaciones básicas para controlar los componentes de las interfaces o los comandos, en su desarrollo siempre aparecía una situación en la que me quedaba estancada hasta encontrar alguna solución. En algunos ámbitos como son la lectura de un fichero externo y el tratamiento sobre el mismo se pueden agilizar si se consigue más experiencia en el entorno. Por otro lado, las interfaces gráficas de este trabajo son básicas al ser de alto nivel, se podrían mejorar con la creación de pantallas más atractivas usando interfaces gráficas de bajo nivel.

Sin embargo, a pesar de las dificultades, la plataforma J2ME me ha mostrado un gran abanico de posibilidades en la implementación de aplicaciones. El campo de las nuevas tecnologías móviles esta creciendo notablemente hoy en día. No hay más que fijarse en la publicidad sobre juegos Java y las nuevas capacidades y mejoras en los futuros teléfonos móviles y PDAs. La formación en este lenguaje es una apuesta segura para iniciarse en las nuevas plataformas móviles.

Muestra de ello es la reciente aparición de *Android*⁷, una plataforma de programación software para dispositivos móviles que incluye un sistema operativo, middleware y aplicaciones clave. Te ofrece toda una variedad de APIs y herramientas para el desarrollo de aplicaciones utilizando el lenguaje de programación Java. Apadrinado por Google, mucha gente se ha adentrado en el mundo de la programación para los móviles, exponiendo una gran variedad de aplicaciones que se pueden lograr con el uso del GPS, la cámara, el video, etc.

12.1. Futuras mejoras del Sistema

Como ya se ha comentado varias veces a lo largo de esta memoria, es posible mejorar la apariencia de las pantallas con el uso de APIs de bajo nivel a costa de perder portabilidad. Por otro lado, el acceso a los datos en el diccionario sería más ágil si se utilizarían tablas *Hash* (acceso directo a la posición de la clave), pero se perdería la funcionalidad de poder devolver una lista de las más parecidas en caso de no encontrar una traducción para la palabra. Se han consultado varias páginas para saber si existe alguna tabla *hash* que pueda buscar valores cercanos al *hash* de la clave, pero no se han encontrado resultados.

Por último, se podría añadir una nueva funcionalidad la cual acceda a un servidor Web para buscar actualizaciones en los ficheros de texto que contengan nuevas palabras. Una nueva función en el menú conectaría con la página Web de un servidor en la que el administrador irá añadiendo nuevas palabras que puedan surgir. Para ello sería necesario implementar una clase que cree una conexión de red y diseñar una página que guarde los ficheros de las palabras, las cuales estarán guardadas en una base de datos para poder trabajar sobre ellas. La idea no ha podido ser estudiada para añadirla a la aplicación al surgir los últimos días de entrega, por lo que quedará abierta para una próxima vez o para algún otro programador que le interese ampliar la aplicación.

⁷ Página oficial del proyecto Android: <http://code.google.com/android/>

Finalmente, y no necesariamente como una mejora, comentar la viabilidad de colgar la aplicación en Internet como libre distribución, para que cualquier otro usuario pueda descargarlo o implementarlo a su gusto.

13. APÉNDICES

En este capítulo se incluyen una guía de usuario para la utilización del sistema y las instalaciones de los entornos y programas necesarios para su ejecución en un PC. Además, se incluye un pequeño apartado sobre los emuladores para los teléfonos móviles y PDAs

13.1. MANUAL DEL USUARIO

Esta guía servirá para que el lector conozca las prestaciones que ofrece la aplicación y como poder navegar entre ellas de manera sencilla. A continuación se describe cada pantalla que mostrará el programa, junto con una imagen de la misma para mayor claridad de donde nos situamos, explicando la función de cada botón y la próxima pantalla que se mostraría.

13.1.1. Teclas

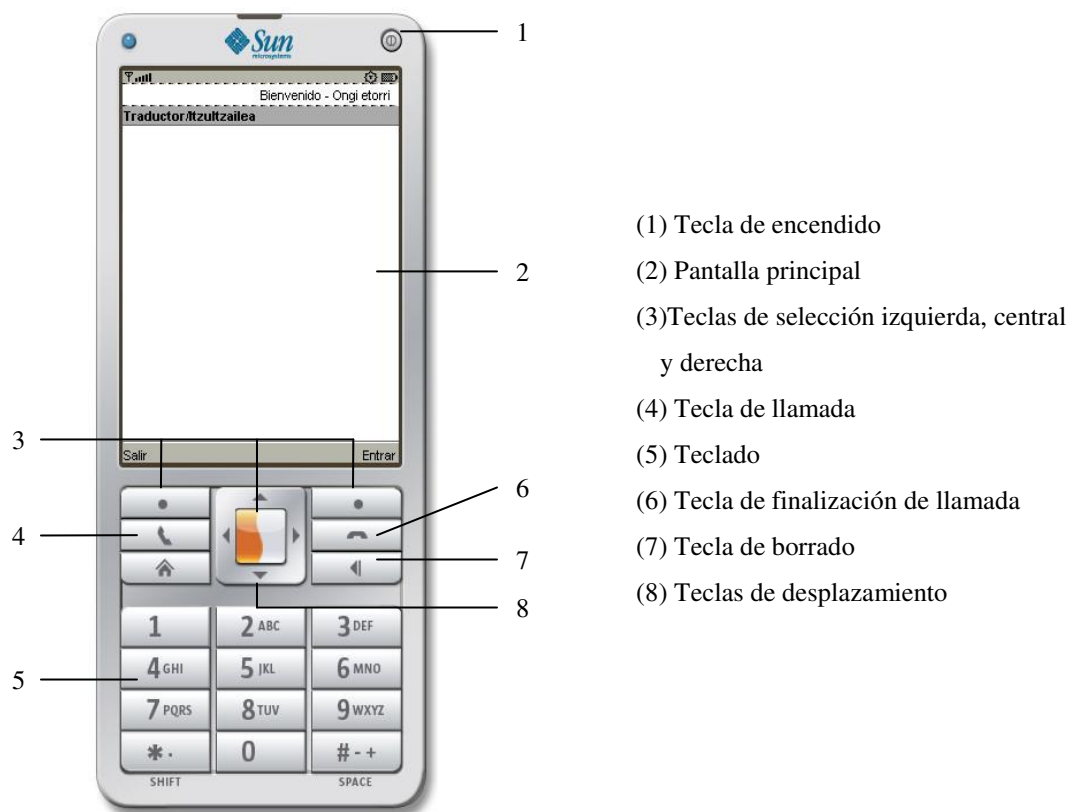




Figura A1: Teclas de control del emulador


13.1.2. Entrar a la aplicación



Figura A2: Pantalla de Inicio

Cuando se ejecuta el programa se muestra una pantalla inicial que da la bienvenida al usuario (figura A2).

- Seleccione la función “Entrar/Sartu” de la tecla de selección () derecha si desea entrar al menú principal de la aplicación.
- En caso de querer salir del sistema, seleccione la función “Salir/Irten” de la tecla de selección () izquierda.

En cualquier momento se puede salir de la aplicación pulsando sobre el botón de finalización de llamada 

13.1.3. Navegación por el Menú



Figura A3: Pantalla del Menú inicial (castellano y euskera)




En la pantalla de la imagen A3 se visualizan dos opciones, con nombres bastantes intuitivos, que son:

Traducir Palabra

Accede al programa principal que nos permitirá introducir una palabra para su posterior traducción.

Cambiar Idioma

Es la opción que accede a una lista de idiomas a los que se puede trasladar la aplicación, en este caso está disponible en euskera y en castellano.

1. Para acceder al menú, seleccione “Entrar” en la pantalla de inicio
2. Desplácese por el menú mediante las teclas de navegación arriba y abajo.
( y )
3. Seleccione una opción con el botón de selección central 

Para salir de la aplicación, seleccione la función de la tecla de selección izquierda “Salir”.

13.1.4. Ajustes de Idioma



Figura A4: Selección del idioma

1. Seleccione “Cambiar Idioma” en la pantalla del menú principal (Véase figura A3) pulsando en la tecla de selección central.


2. Desplácese por los idiomas disponibles mediante las teclas de navegación arriba y abajo
3. Seleccione una opción sin marcar con la tecla central ()
4. Pulse sobre la función “Guardar” de la tecla de selección derecha (Figura A5)



Figura A5: Guardar cambios

5. Para volver al menú principal, pulse sobre la tecla de selección izquierda de la función “Atrás”.

13.1.5. Traducir una palabra

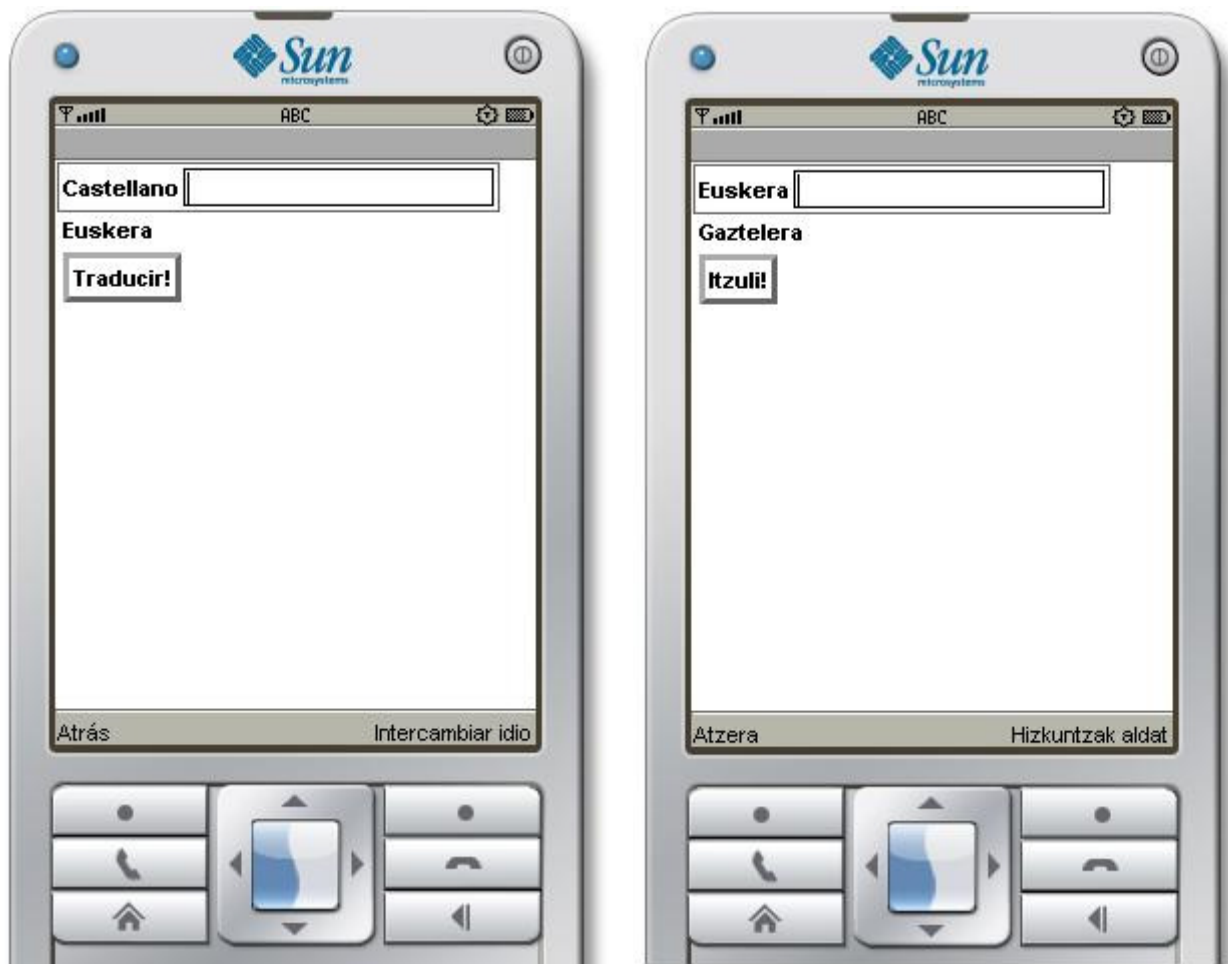





Figura A6: Traducir una palabra (castellano y euskera)

1. Seleccione la opción “Traducir palabra” del menú principal.
2. Escriba la palabra que desee traducir en el campo de texto⁸, para corregir algún carácter pulsar .
3. Desplácese al botón “Traducir” de la pantalla principal ( o ).
4. Pulse sobre la tecla de selección central o en la tecla de selección derecha “Menú” > “Traducir” (Figura A7)

⁸ En el simulador del WTK es posible escribir y borrar por medio del teclado.

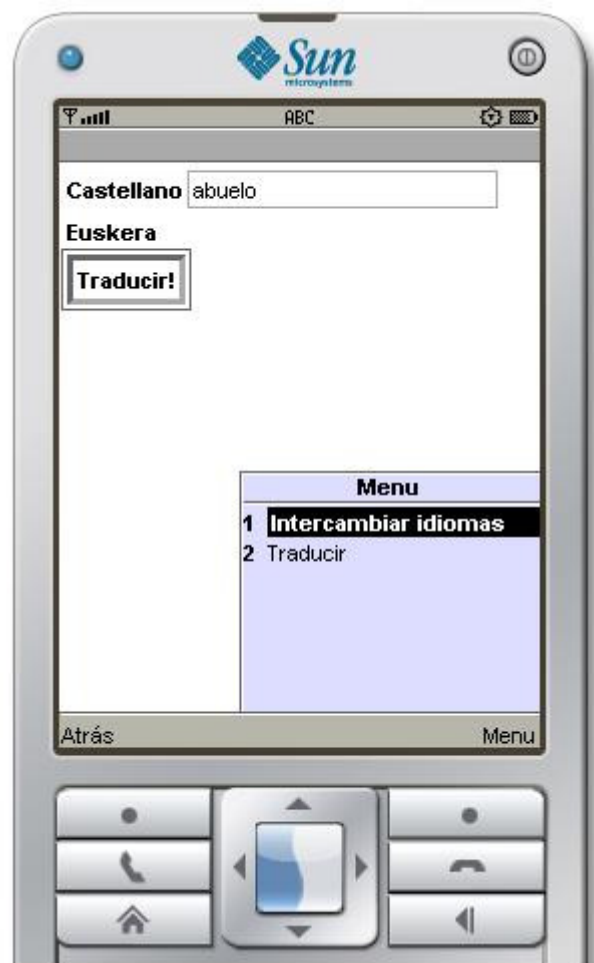


Figura A7: Funciones “Traducir” disponibles

13.1.5.1. Intercambiar Idiomas

La función “Intercambiar idiomas” está disponible en la pantalla de la opción “Traducir Palabra”. Se puede seleccionar de dos maneras:

- Seleccionando la función “Intercambiar Idiomas” de la tecla de selección derecha.
- Desde el menú desplegable de la función “Traducir”.

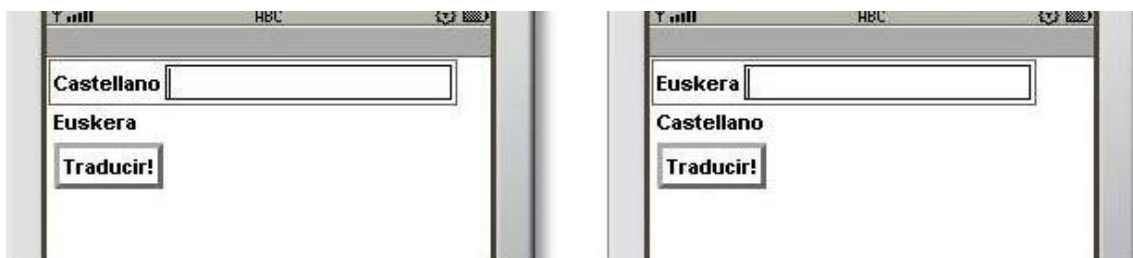


Figura A8: Intercambio de idioma

Para volver al menú principal de la aplicación, seleccionar “Atrás” (tecla de selección izquierda)


13.1.5.2. Traducción correcta

Si la palabra introducida es correcta, se mostrará una pantalla con el resultado de la traducción.



Figura A9: Traducción correcta

La función “Menú” de la tecla de selección derecha muestra la leyenda de las categorías.

Para volver a la pantalla de la opción “Traducir Palabra” pulsar la tecla de selección izquierda “Atrás” ()

13.1.5.3. Traducción Incorrecta

Si la palabra está mal escrita o no se encuentra en el diccionario, el sistema crea una lista con las palabras más parecidas a la introducida y las muestra (Figura A10).

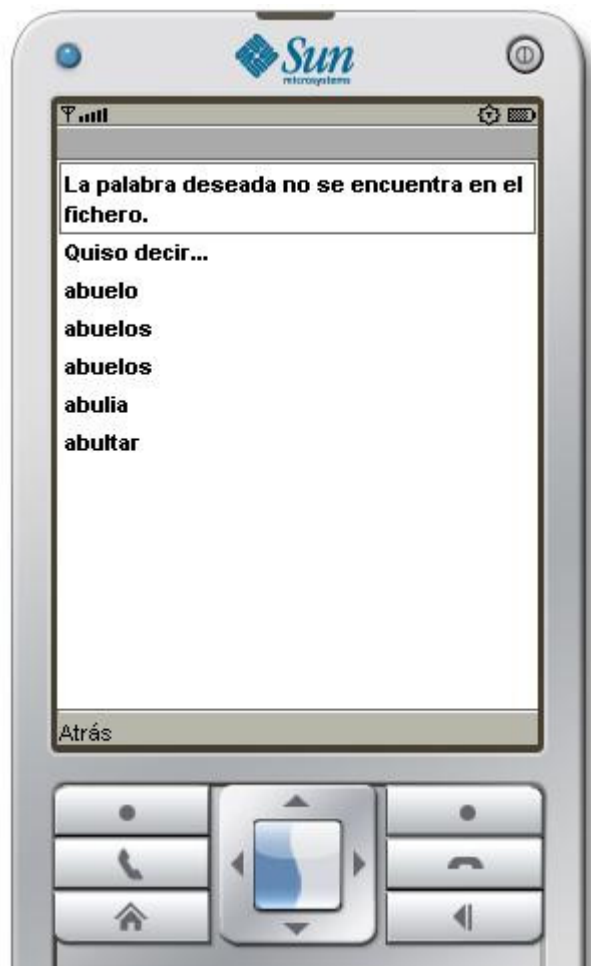


Figura A10: Traducción incorrecta

Para volver a la pantalla anterior seleccionar la función “Atrás” de la tecla de selección izquierda.

13.2. GUÍA DE INSTALACIÓN

Como hemos comentado en el apartado de las herramientas tecnológicas, ha sido necesaria la instalación de algunos programas para poder trabajar en el proyecto. A continuación se describirán los pasos que se deberán tomar para una correcta instalación de los programas para soportar J2ME y sus componentes. Es necesario haber descargado previamente el siguiente software:

- Java 2 Standard Edition (JDK 6) (<http://java.sun.com/javase/downloads/index.jsp>)
- Eclipse (<http://www.eclipse.org/downloads/>)
- Sun Java Wireless Toolkit (<http://java.sun.com/products/sjwtoolkit/download.html>)
- EclipseME (<http://eclipseme.org/>)

JAVA IDE (Integrated Development Environment) y JDK

Como suponemos, es necesario disponer de un entorno de trabajo para Java. En este trabajo se ha utilizado el IDE Eclipse, aunque no se descartan otros entornos de desarrollo como el NetBeans. Antes de nada se ha instalado el Kit de Desarrollo de Java *j2sdk*:

- Hacemos doble clic en el icono del instalador. Una vez aceptadas las condiciones se mostrará una pantalla de este tipo.

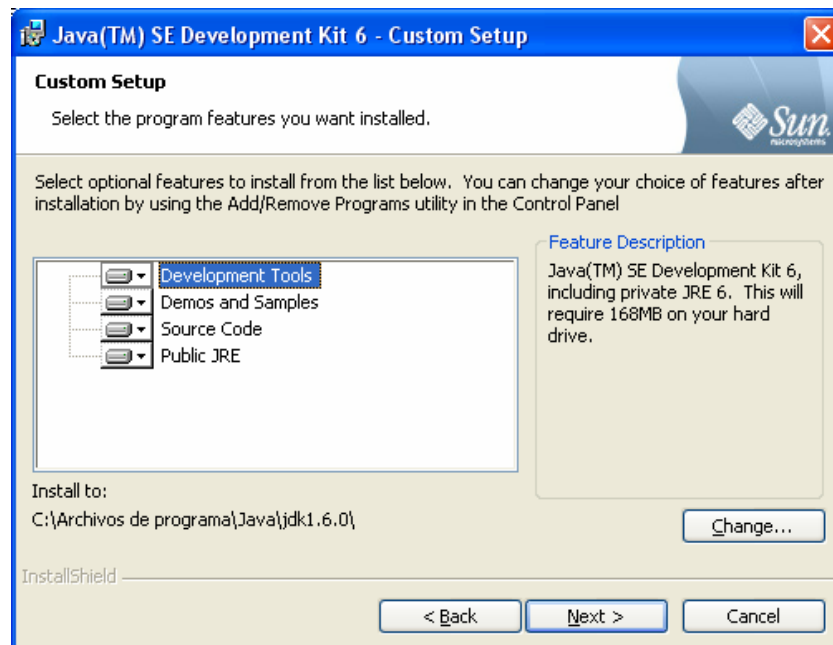


Figura A11: Instalación del JDK (paso 1)

- Seleccionamos la primera opción que hace referencia a las herramientas de desarrollo y hacemos clic en “Siguiente”.
- La pantalla mostrará los componentes que deseamos instalar.

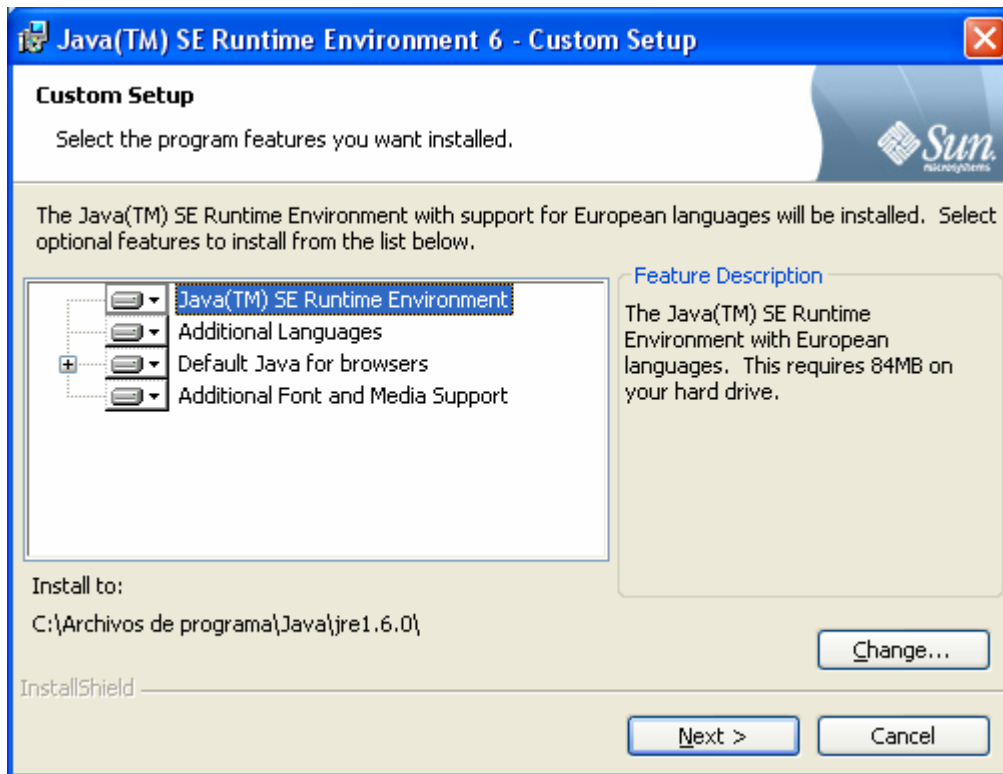


Figura A12: Instalación del JDK (paso 2)

- Si pulsamos “Siguiente” se procederá a la instalación del entorno de ejecución (JRE) y las opciones adicionales que hemos elegido previamente.
- Instalamos Eclipse: para ello descomprimos el archivo bajado y dejamos la carpeta Eclipse en la ruta que prefiramos.

EclipseME

EclipseME es un *plugin* de Eclipse para ayudar a desarrollar *MIDlets* J2ME. EclipseME realiza el trabajo de conectar el Wireless Toolkit al entorno de desarrollo del Eclipse.

- Iniciamos Eclipse (~ruta\eclipse.exe)
- Accedemos en la barra del menú a *Help* → *Software Updates* → *Find and Install*
- Seleccionamos *Search for new features to install*

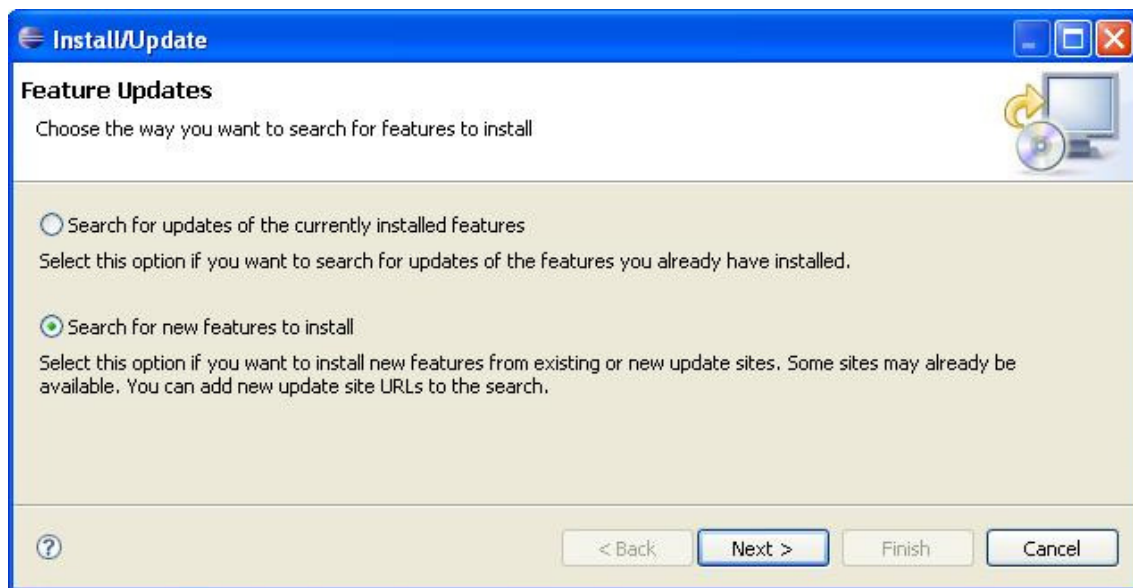


Figura A13: Instalación de nuevos componentes en el Eclipse

- Hacemos clic en *New Archived Site*. Buscamos el archivo *zip* del EclipseME (eclipseme.feature_1.6.5_site.zip) y pulsamos *Finish*.
- Marcamos el elemento a instalar.

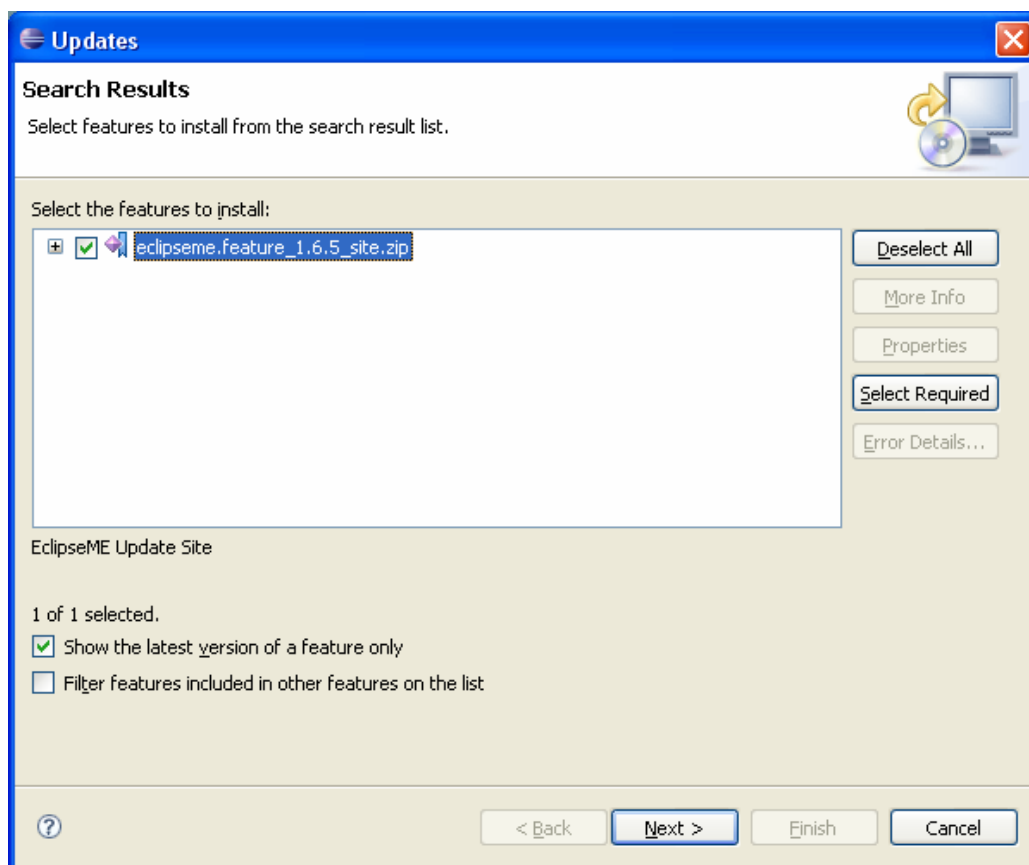


Figura A14: Instalación del EclipseME (paso 1)

- Aceptamos la licencia y comprobamos la ruta por defecto (dentro de la carpeta Eclipse). Terminamos pulsando en *Install All* y nos indica que debemos reiniciar Eclipse.

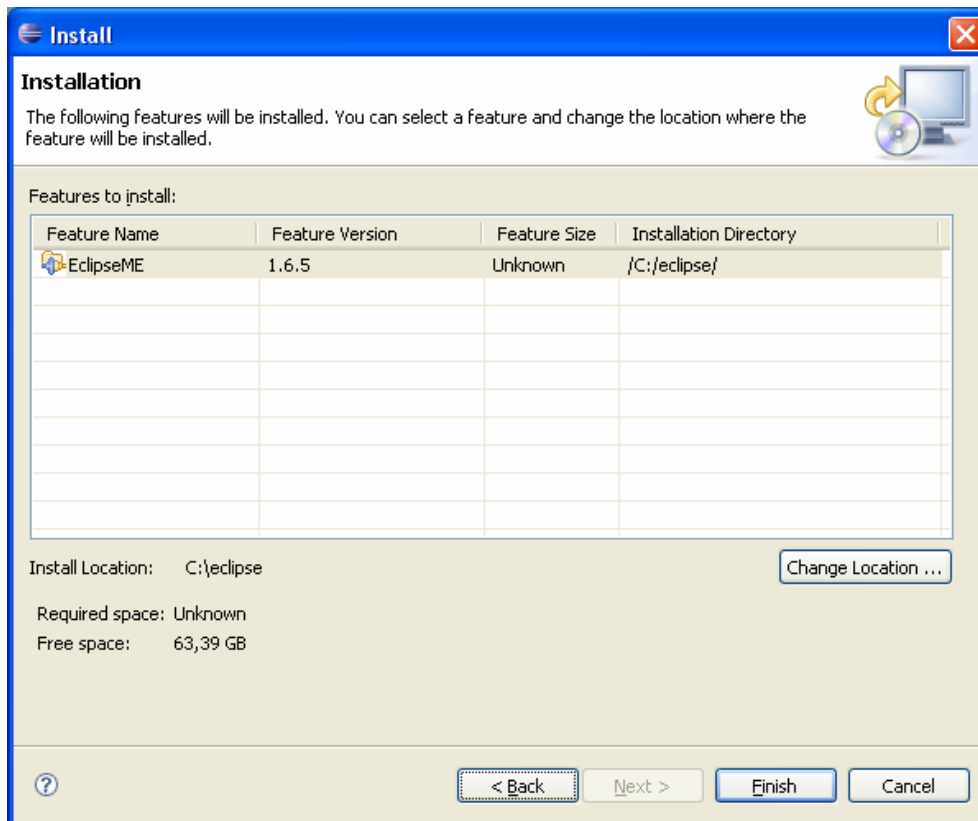


Figura A15: Instalación del EclipseME (paso 2)

Sun Java Wireless Toolkit

El programa lo podemos encontrar en la página de los productos de Sun Microsystems. Para instalarlo no habrá que hacer más que ejecutar la aplicación descargada y una vez especificada la ruta de la carpeta hacer clic en “Siguiente”.

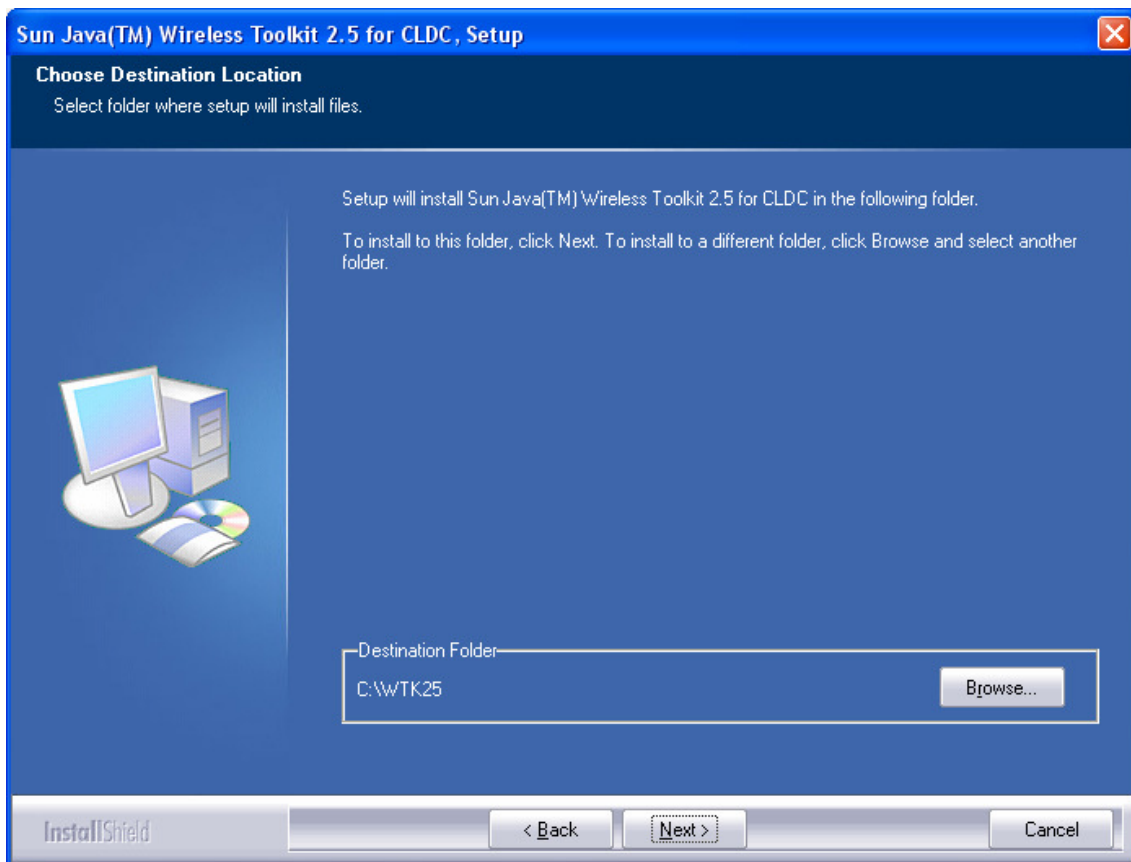


Figura A16: Instalación del Wireless Toolkit

- De nuevo en Eclipse, vamos a *Windows* → *Preferentes* → *J2ME* e indicamos la ruta de nuestro directorio WTK.
- Vamos a *Device Management* y pulsamos *Import*. (Figura A17)
- En *Specify search directory* volvemos a poner el directorio del WTK y pulsamos *Refresh*. Dejamos marcados todos los nuevos elementos que aparecen.

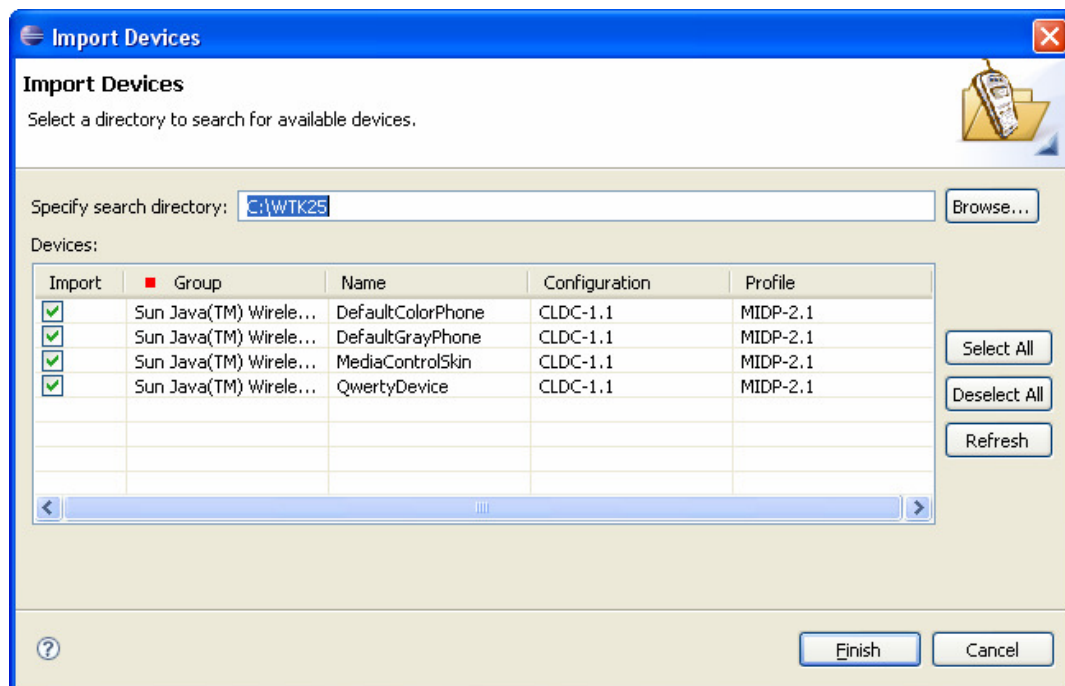


Figura A17: Importación del WTK

- Vamos a *Windows* → *Preferences* → *Java* → *Debug* y dejamos los valores como en la captura. Y en *Java* → *Build Path* seleccionamos *Folders*.

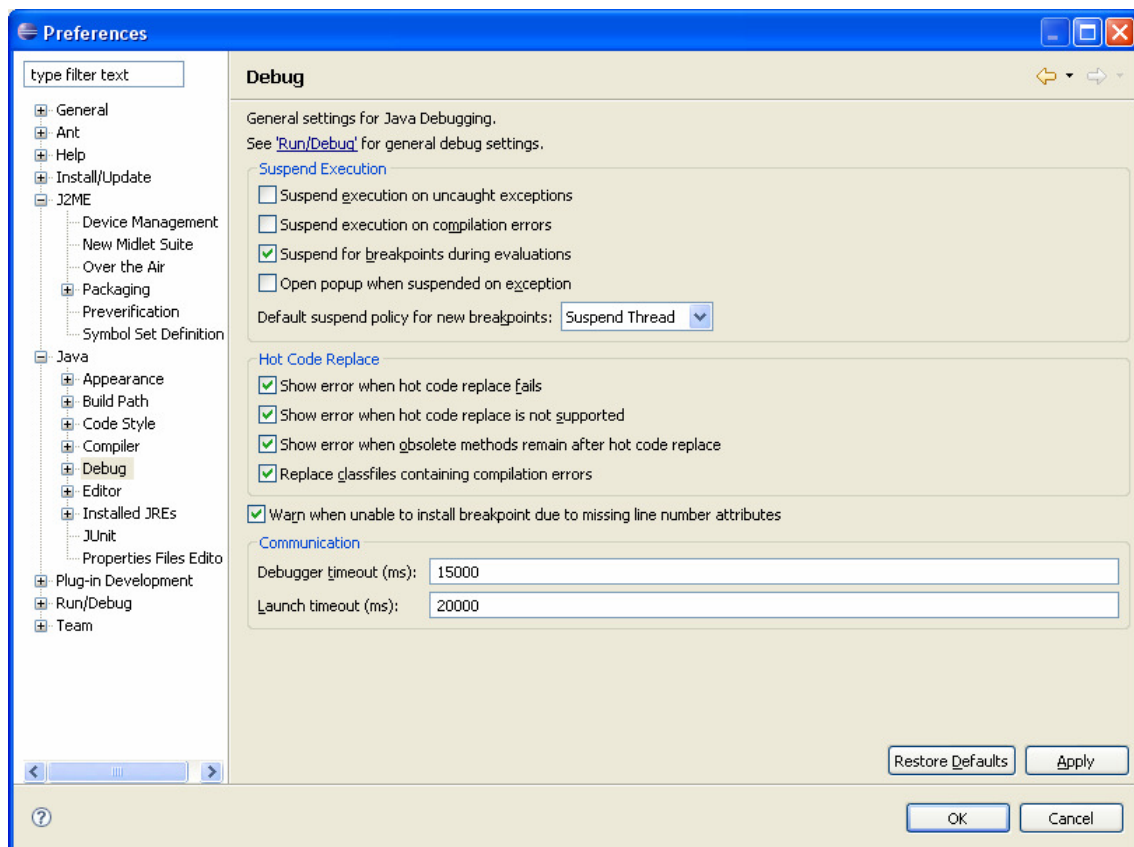


Figura A18: Configuración del WTK

13.3. Emuladores

El emulador ideal sería aquel que imitase el comportamiento de un dispositivo exactamente. Pero la precisión del emulador está correlacionada con la dificultad de desarrollarlo. Los desarrolladores deben que decidir como precisar los emuladores dependiendo del tiempo y las habilidades que tengan.

Una manera de dividir el mundo de los emuladores consiste en diferenciarlos entre los emuladores “de concepto” y los emuladores “de vida real”. Un emulador de concepto no representa al dispositivo específico, pero sirve para demostrar las características generales de un grupo concreto de dispositivos. El emulador del J2ME Wireless Toolkit , por ejemplo, es de tipo concepto que puede ser usado para representar una variedad de dispositivos MIDP. Un emulador de vida real en cambio, es diseñado para imitar la apariencia y comportamiento de un dispositivo actual. Como emulador puede ejecutar parte o todo el código binario como lo ejecutaría el propio dispositivo.

Nokia proporciona ejemplos de los dos tipos. La compañía expone emuladores de concepto para ayudar a los desarrolladores a prepararse para las series de los próximos dispositivos. Luego provee emuladores de vida real cuando las series son en realidad lanzadas al mercado. Las herramientas y descargas están disponibles en http://www.forum.nokia.com/main/resources/tools_and_sdks/.

Desafortunadamente no hay reglas generales para usar estos emuladores. Algunos son empaquetados para ser añadidos al J2ME Wireless Toolkit o a entornos de desarrollo IDE. Otros son paquetes como parte de un kit autónomo de desarrollo o una herramienta de pruebas. A continuación se presentan algunas compañías que ofrecen emuladores para MIDP. Se recomienda leer las instrucciones de instalación y las notas publicadas para entender exactamente como instalarlos y usarlos.

- Motorola (<http://developer.motorola.com/technologies/java/>)

- Sprint
(http://developer.sprint.com/site/global/develop/technologies/java_me/sdk_tools/p_sdk_tools.jsp)
- Sony Ericsson
(<http://www.ericsson.com/mobilityworld/sub/open/technologies/java/index.html>)
- Sun Microsystems
(<http://developers.sun.com/mobility/device/device;jsessionid=CB19AA8403DBED6677747CB825465022>)

El propio Java Wireless Toolkit pone a disposición del desarrollador una guía para la creación de nuestros propios emuladores. En la carpeta *docs* del programa WTK se encuentra el manual *BasicCustomizationGuide.pdf*.

14. BIBLIOGRAFÍA

En este capítulo se recogen las referencias de los manuales y los enlaces de Internet consultados para la realización de este proyecto.

Documentación J2ME:

<http://java.sun.com/javame/index.jsp>

Página oficial de J2ME de la empresa Sun

<http://java.sun.com/javame/reference/apis.jsp>

Documentación sobre APIs

<http://developers.sun.com/mobility/getstart/>

Introducción a J2ME

<http://www.j2medev.com/api/midp/javax/microedition/midlet/package-summary.html>

Documentación sobre el paquete javax.microedition.midlet

<http://www.j2medev.com/api/midp/javax/microedition/lcd/ui/package-summary.html>

Documentación sobre el paquete javax.microedition.lcd.ui

<http://developers.sun.com/mobility/midp/articles/threading/>

Artículo sobre conexiones de red y *Threads*

<http://developers.sun.com/mobility/midp/articles/emulators/>

Artículo sobre los emuladores

<http://developers.sun.com/mobility/midp/articles/wtoolkit/>

Tutorial

<http://developers.sun.com/mobility/midp/articles/getstart/>

Tutorial de inicio para mobile java

<http://today.java.net/pub/a/today/2005/02/09/j2me1.html?page=2#step1>

Tutorial para la creación de MIDlets

http://en.wikipedia.org/wiki/Java_ME

Enciclopedia *on-line* realizada por la comunidad de usuarios

Sun Java Wireless Toolkit:

<File:\\C:\\WTK2.5.2\\docs\\UserGuide.pdf>

Guía de usuario del WTK

<http://java.sun.com/products/sjwtoolkit/>

Página oficial del producto Wireless Toolkit

Eclipse:

<http://www.eclipse.org/>

Página oficial de Eclipse

<http://eclipseme.org/>

Plugin para el Eclipse

Foros:

http://www.forum.nokia.com/main/resources/tools_and_sdks/

<http://www.javahispano.org/>

<http://www.todosymbian.com/>

<http://www.programacion.com/>

