UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA:

DESARROLLO DE UN MOTOR DE VIDEOJUEGOS CONVERSACIONALES SOBRE WAP PARA DISPOSITIVOS MÓVILES.

INGENIERÍA TELECOMUNICACIÓN

Málaga, 2007 Javier González Monroy

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Titulación: Ingeniería de Telecomunicación

Reunido el tribunal examinador en el día de la fecha, constituido por:

D. _____

D. _____

D. _____

para juzgar el Proyecto Fin de Carrera titulado:

DESARROLLO DE UN MOTOR DE VIDEOJUEGOS CONVERSACIONALES SOBRE WAP PARA DISPOSITIVOS MÓVILES.

del alumno D. Javier González Monroy

dirigido por D. Francisco Javier González Cañete

ACORDÓ POR: _____ OTORGAR LA CALIFICACIÓN DE

_____

y, para que conste, se extiende firmada por los componentes del Tribunal, la presente diligencia.

Málaga, a ____de_____de Año

El Presidente: El Secretario: El Vocal:

Fdo. _____ Fdo. _____ Fdo. _____

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DESARROLLO DE UN MOTOR DE VIDEOJUEGOS CONVERSACIONALES SOBRE WAP PARA DISPOSITIVOS MÓVILES.

REALIZADO POR:

Javier González Monroy

DIRIGIDO POR:

Francisco Javier González Cañete

DEPARTAMENTO DE: Tecnología Electrónica.

TITULACIÓN: Ingeniería de Telecomunicación

Palabras claves: WAP, WML, WMLScript, XHTML-MP, ECMAScript, XML, servlets, JSP, arquitectura MVC, videojuego conversacional.

RESUMEN:

Atendiendo al importante avance de la telefonía móvil en cuanto al sector demográfico sobre la que se aplica, así como en el continuo aumento de prestaciones y características de sus terminales, lo que posibilita su uso más allá de establecer comunicaciones entre diversos terminales, el objetivo del presente proyecto es ofrecer un motor sobre el que poder desarrollar y ejecutar videojuegos conversacionales para este tipo de dispositivos. Por una parte se pretende dotar de un sistema sencillo y cómodo de usar que posibilite la creación de diferentes videojuegos de similares características, al mismo tiempo que se establece un motor sobre el que ejecutar dichos videojuegos, adaptando los contenidos a las características de cada terminal.

Málaga, Septiembre de 2007

A mi familia y a los que formarán pronto parte de ella.

Agradecimientos

En primer lugar agradecer a aquel que me dio fuerzas en los momentos más difíciles de la carrera, al que me ayudó a ver las cosas desde otro punto de vista cuando me ofuscaba en ver las cosas siempre del mismo color, al que me ha guiado hasta donde ahora mismo estoy, a mi Dios.

Tampoco puedo olvidarme de mi familia, la cual siempre me apoyó en mis decisiones y a la que debo todo lo que soy. En especial a mis padres, a mi abuela, a Alex y a mi hermano Sergio.

A mis compañeros de clase, con los que he compartido en estos años alegrías, esfuerzos y penas, y con los que mantengo una gran amistad.

Finalmente agradecer el esfuerzo y gran dedicación de mi tutor Francisco Javier González Cañete, por esas largas horas de Messenger hablando del proyecto cuando yo estaba lejos, por sus consejos, por su ayuda en todos los aspectos.

Gracias a todos.

ÍNDICE DE CONTENIDOS

Índice de Contenidos ..... **i**
Relación de Acrónimos ..... **v**
Índice de Figuras ..... **vii**
Índice de Tablas ..... **ix**

CAPÍTULO 1: INTRODUCCIÓN **1**
CAPÍTULO 2: TECNOLOGÍAS Y HERRAMIENTAS EMPLEADAS **9**
2.1 INTRODUCCIÓN 9
2.2 INTELIGENCIA CLIENTE/SERVIDOR 9
2.3 JAVA 10
2.3.1 INTRODUCCIÓN 10
2.3.2 VENTAJAS FRENTE A OTRAS TECNOLOGÍAS 11
2.4 SERVLETS 13
2.4.1 CARACTERÍSTICAS 14
2.4.2 ESTRUCTURA DE UN *SERVLET* 15
2.4.3 CICLO DE VIDA 18
2.4.4 JERARQUÍA DE ARCHIVOS 19
2.5 JSP 21
2.6 ARQUITECTURA MVC 23
2.7 XML 25
2.7.1 XSD 25
2.8 WAP 26
2.8.1 INTRODUCCIÓN 26
2.8.2 ARQUITECTURA WAP 1.X 27
2.8.3 ARQUITECTURA WAP 2.0 29
2.8.4 WAP 1.X Vs WAP 2.0 31
2.9 WML 34
2.9.1 INTRODUCCIÓN 34
2.9.2 WML A GRANDES RASGOS 34
2.9.2.1 Características principales 34
2.9.2.2 Sintaxis WML 35
2.9.2.3 Barajas y Cartas 36
2.9.2.4 Texto y Formato 37
2.9.2.5 Eventos y Tareas 38
2.9.2.6 Enlaces 40
2.9.2.7 Tipos MIME 41
2.10 WMLSCRIPT 42
2.10.1 INTRODUCCIÓN 42
2.10.2 BENEFICIOS Y CARACTERÍSTICAS 42
2.10.3 LLAMADAS A FUNCIONES 44
2.10.4 SINTAXIS 45
2.10.5 DECLARACIÓN DE FUNCIONES 46
2.10.6 BIBLIOTECAS ESTÁNDAR 48
2.11 XHTML 50
2.11.1 INTRODUCCIÓN 50
2.11.2 VISTA GENERAL 51
2.11.3 VENTAJAS 52
2.11.4 REGLAS SINTÁCTICAS 52

2.11.5 XHTML-MP 53
2.11.5.1 Cambios respecto a WML 54
2.11.5.2 Declaración XML y DOCTYPE 55
2.11.5.3 Tipos MIME 55
2.12 ECMASCRIPT –MP 56
2.12.1 INTRODUCCIÓN 56
2.12.2 CARACTERÍSTICAS 56
2.13 HOJAS DE ESTILO WCSS 57
2.13.1 INTRODUCCIÓN 57
2.13.2 HOJAS DE ESTILO WCSS 58
2.14 SERVIDOR DE APLICACIONES TOMCAT 59
2.15 EMULADORES 59
2.16 XMLPAD 61
CAPÍTULO 3: DISEÑO DE LA APLICACIÓN 63
3.1 INTRODUCCIÓN 63
3.2 VISIÓN GENERAL 63
3.3 SISTEMA DE ARCHIVOS 64
3.3.1 INTRODUCCIÓN 64
3.3.2 ARQUITECTURA DEL SISTEMA DE ARCHIVOS 64
3.3.3 ARCHIVOS DE CONFIGURACIÓN DEL VIDEOJUEGO 66
3.3.3.1 Consecuencias de una acción 75
3.4 FUNCIONAMIENTO DE LA APLICACIÓN 80
3.4.1 CONFIGURABILIDAD DE LA VISTA 82
3.4.2 PROCESOS DE LA APLICACIÓN 84
3.4.2.1 Presentación 84
3.4.2.2 Registro de usuario 85
3.4.2.3 Identificación de usuario 87
3.4.2.4 Ejecución del videojuego 88
3.5 CONSIDERACIONES ADICIONALES SOBRE EL DISEÑO 92
3.5.1 CREACIÓN DE SESIÓN 92
3.5.2 PRESENTACIÓN DE LOS CONTENIDOS 93
3.5.3 SOPORTE PARA IMÁGENES 95
CAPÍTULO 4: MANUAL DE USUARIO 99
4.1 MANUAL DE USUARIO ADMINISTRADOR 99
4.1.1 ESPECIFICACIÓN DEL VIDEOJUEGO 99
4.1.2 ELECCIÓN DE LA VISTA 99
4.1.3 ESTRUCTURA Y CONTENIDOS 101
4.2 MANUAL DE USUARIO JUGADOR 104
4.2.1 MANUAL DE USUARIO PARA VISTA XHTML 104
4.2.2 MANUAL DE USUARIO PARA VISTA WML 106
CAPÍTULO 5: PLAN DE PRUEBAS 109
5.1 MEDICIONES DE TRÁFICO GENERADO 109
5.1.1 CONSIDERACIONES PREVIAS 109
5.1.2 MEDICIONES REALIZADAS 110
5.1.2.1 Mediciones para XHTML 110
5.1.2.2 Medicioes para WML 112
5.2 PLAN DE PRUEBAS DETALLADO 115
5.2.1 PRUEBAS CON EMULADOR Y SERVIDOR LOCAL 115
5.2.2 PRUEBAS CON EMULADOR Y SERVIDOR EXTERNO 117
5.2.3 PRUEBAS CON DISPOSITIVOS MÓVILES REALES 117
CAPÍTULO 6: CONCLUSIONES Y LINEAS FUTURAS 119

BIBLIOGRAFÍA 123
APÉNDICE A: FICHEROS XML PARA LA AVENTURA ORIGINAL 125
APÉNDICE B: VALIDACIÓN DE LOS DOCUMENTOS XML 139

RELACIÓN DE ACRÓNIMOS

aDeSe	Asociación española de Distribuidores y Editores de Software de Entretenimiento.
API	Aplication Program Interface.
ASP.NET	*Active Server Pages* con tecnología *.NET*
CDMA	Code Division Multiple Access
CGI	Common Gateway Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSS2	Cascading Style Sheets versión 2
DAMPS	Digital-Advanced Mobile Phone Service
DTD	Document Type Definition
ECMAScript-MP	ECMA Script Mobile Profile
ESMP	EcmaScript Mobile Profile
GPRS	General Packet Radio Service
GSM	Global System for Mobile comunications
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IIS	Internet Information Services
IP	Internet Protocol
JAR	Java ARchives
JDK	Java Development Kit
JSP	Java Server Pages
MIME	Multipurpose Internet Mail Extensions
MVC	Modelo Vista Controlador
NPC	Non-Playable Character
OMA	Open Mobile Alliance
PERL	Practical Extraction and Reporting Language
PHP	Php *Hypertext Preprocessor*
PHS	Personal Handyphon System
SGML	Standard Generalized Markup Language
SSL	Secure Sockets Layer

TCP	Transmission Control Protocol
UMTS	Universal Mobile Telecommunications System
URL	Universal Resource Locator
WAE	Wireless Application Environment
WAP	Wireless Aplication Protocol
WBMP	Wireless BitMaP
WCSS	WAP CSS o Wireless CSS
WDP	Wireless Datagram Protocol
WML	Wireless Markup Language
WMLScript	Wireless Markup Language Script
WSP	Wireless Session Protocol
WTLS	Wireless Transport Layer Security
WTP	Wíreless Transaction Protocol
WWW	*World Wide Web*
W3C	World Wide Web Consortium
XHTML	eXtensible Hypertext Markup Language
XHTML-MP	eXtensible Hypertext Markup Language -Mobile Profile
XML	eXtensible Markup Language

ÍNDICE DE FIGURAS

Figura 1.1 Imágenes de diferentes Aventuras Conversacionales 3
Figura 1.2 Imágenes de algunos juegos de Aventuras AD.....5
Figura 2.1 Diferencias entre Java y los lenguajes tradicionales.....11
Figura 2.2 Estructura de un *servlet*.....16
Figura 2.3 Ciclo de vida de un *servlet*.....19
Figura 2.4 Jerarquía de archivos para *servlets* 20
Figura 2.5 Flujo petición/respuesta cuando se llama a una página JSP 22
Figura 2.6 Relación entre las capas del modelo MVC 23
Figura 2.7 Arquitectura Web 27
Figura 2.8 Arquitectura WAP 1.X 28
Figura 2.9 Pila de protocolos de WAP 1.X 28
Figura 2.10 Arquitectura WAP 2.0 30
Figura 2.11 Pila de protocolos de WAP 2.0 31
Figura 2.12 Separación de las torres de protocolos WAP en un dispositivo móvil 33
Figura 2.13 WMLScript comparado con ESMP 57
Figura 3.1 Arquitectura del sistema de archivos 65
Figura 3.2 Ejemplo del archivo *user.xml*.....67
Figura 3.3 Fragmento del archivo *map.xml*.....68
Figura 3.4 Estructura de un elemento de localización en *map.xml* 69
Figura 3.5 Fragmento del archivo *object.xml*.....71
Figura 3.6 Fragmento del archivo *actions.xml* 73
Figura 3.7 Diagrama de funcionamiento de la aplicación.....81
Figura 3.8 Organización de los archivos según la arquitectura MVC.....82
Figura 3.9 Organización de los contenidos en el servidor 83
Figura 3.10 Diferentes llamadas al motor 85
Figura 3.11 Diagrama de flujo de *engine.class* 92
Figura 3.12 Menús y listas desplegables en XHTML 94
Figura 3.13 Menús y listas desplegables en WML.....94
Figura 3.14 Imágenes de la Aventura Original para la vista XHTML.....96
Figura 3.15 Imágenes de la Aventura Original para la vista WML 96
Figura 3.16 Fragmento del archivo *map.xml* para el caso de no usar imágenes 97
Figura 4.1 Estructura de directorios para nuevos videojuegos.....100

Figura 4.2 Estructura de directorios perteneciente a cada videojuego..... 101
Figura 4.3 Acceso e identificación de nuevo usuario para la vista XHTML..... 105
Figura 4.4 Información y formulario enviados al usuario según localización..... 105
Figura 4.5 Acceso e identificación de nuevo usuario para la vista WML 106
Figura 4.6 Información y formulario para la vista WML 107

ÍNDICEDE TABLAS

Tabla 2.1 Comparación entre WAP 1.X y WAP 2.0 33
Tabla 2.2 Caracteres restringidos de WML.....37
Tabla 2.3 Etiquetas para formato de texto en WML 38
Tabla 2.4 Eventos en WML 39
Tabla 2.5 Tipos MIME para WAP 1.X 41
Tabla 2.6 Funciones de la librería estándar WMLBrowser.....48
Tabla 2.7 Funciones de la librería estándar Lang. 49
Tabla 2.8 Funciones de la librería estándar Dialogs. 50
Tabla 5.1 Medidas de tráfico para la vista XHTML 111
Tabla 5.2 Medidas de tráfico para la vista WML.....113
Tabla 5.3 Medidas de tráfico generado al moverse dentro de una baraja 114

CAPÍTULO 1: INTRODUCCIÓN

Datos recientes sobre el mercado español, constatan que el sector de los videojuegos es un valor en alza en la actualidad, con tasas de crecimiento elevadas y estables en los últimos años y con previsiones de seguir así. Esto indica que el sector de los videojuegos no es solo un mundo en los que los más jóvenes invierten su tiempo de ocio, sino que es toda una industria que día a día genera más beneficios y puéstos de trabajo en nuestro país.

La aDeSe (Asociación española de Distribuidores y Editores de Software de Entretenimiento), que agrupa a más del 85% del mercado nacional, confirma este crecimiento imparable en los últimos años, con cifras que indican que aproximadamente el 40% del mercado del ocio audiovisual está relacionado con los videojuegos [1], ya sean para video consolas, ordenadores personales, o como recientemente están apareciendo en el mercado, dispositivos portátiles dedicados en su totalidad a la ejecución de videojuegos (N-Game de Nokia, PSP de Sony, Nintendo DS, etc.).

No obstante, cada día son más las alternativas que dispone el usuario para poder jugar en cualquier lugar, dado que el sector de las videoconsolas está adaptando terminales para ser fácilmente portables como los ejemplos anteriormente citados. El claro inconveniente de esta alternativa es el precio, dado que una videoconsola portátil cuesta en torno a los 170€ en la actualidad, precio que muchos usuarios no están dispuéstos a pagar para poder divertirse en su tiempo de ocio.

En este contexto surgen los videojuegos para dispositivos móviles (teléfonos, agendas electrónicas, etc.), los cuales basándose en la tremenda implantación que éstos dispositivos tienen en la actualidad, y en las características que implementan en el soporte de tecnologías Java y navegación por internet, hacen que el desarrollo de éstos videojuegos solventen los problemas de las videoconsolas portátiles, aunque también es cierto con características gráficas muy inferiores.

Según un estudio de eBrain (empresa española centrada en el desarrollo de videojuegos para teléfonos móviles), más de la mitad de los usuarios encuestados afirmaban haber jugado o jugar con cierta asiduidad a este tipo de videojuegos [2]. Para un 80,9% de los consumidores, la mayor ventaja de jugar a través del móvil es que no requiere una consola o PC y permite una total libertad geográfica, es decir, el teléfono móvil hace posible el jugar en cualquier parte sin conllevar un gasto añadido en la adquisición del terminal, dado que más del 80% de la población española dispone de uno o varios dispositivos móviles en la actualidad.

Hay una gran variedad de videojuegos para dispositivos móviles, desde los clásicos arcade, pasando por juegos de aventura, hasta los basados en la estrategia. En su mayor parte son videojuegos basados en J2ME (versión Java para dispositivos móviles), y todos ellos tienen la característica común de estar limitados en la capacidad gráfica y en la potencia de cálculo necesaria para poder ejecutarlos, dado que son características restrictivas en éstos dispositivos. Otra alternativa para desarrollar videojuegos destinados a ser usados en estos dispositivos, son las aventuras conversacionales.

Las aventuras conversacionales fueron los primeros juegos llevados al ordenador, en una época en la que las capacidades gráficas de los ordenadores eran nulas o muy inferiores a las grandes prestaciones que ofrece hoy cualquier ordenador de sobremesa. Este aspecto hace a este tipo de videojuego un candidato muy interesante en el momento actual, dado que los dispositivos móviles como teléfonos y agendas electrónicas sufren de carencias gráficas similares, sobre todo por el reducido tamaño de sus pantallas.

Una aventura conversacional es un juego en el que al jugador se le presenta inicialmente una historia, el contexto en el que se desarrollará toda la acción del juego, principalmente usando textos escritos en la pantalla. Así, estos juegos suelen estar basados en una serie de localizaciones, cada una con su correspondiente descripción textual y habitualmente para enriquecerla, acompañada de una imagen que ilustra el contenido. Todas esas localizaciones se unen de diferentes formas, ya sean por ser localidades colindantes o usando escaleras y otros artilugios en caso de varios niveles, formando un mapa virtual del mundo ficticio creado en el juego.

En cada situación mostrada, el jugador debe teclear o seleccionar de una lista la acción que desea realizar. El juego interpreta esa acción y genera una serie de consecuencias, proponiendo una nueva situación al jugador, que debe volver a solucionarla al igual que la anterior. Todas esas situaciones son coherentes y están relacionadas siguiendo el argumento del juego, que por supuesto no siempre es lineal, dando cierta libertad al jugador (incluso llegando a realizar acciones que le causarán la muerte), que le permite personalizar el juego a su forma de pensar y actuar.

Algunos ejemplos de estas a venturas conversacionales pueden verse en la Figura 1.1, donde se muestra las diferentes formas de mostrar el contenido al usuario. En la primera imagen (Cobra’s arc) el usuario dispone de una serie de acciones tipo que deben ser elegidas, se muestra una imagen y un texto que describen la localización actual, adicionalmente se añaden a la izquierda los objetos que posee el usuario en su inventario.

En los otros dos ejemplos (Megacorp y Don Quijote), el formato de presentación es más simple, basándose en una imagen de diferente tamaño para cada una de ellas, y un texto explicativo de la situación. En ambos casos, las acciones que se deseen realizar deben ser escritas textualmente, y será el interprete del videojuego quién ejecute la acción, si esta tiene sentido y está permitida.

Cobra's arc *Megacorp* *Don Quijote*

Figura 1.1 Imágenes de diferentes Aventuras Conversacionales

En la mayoría de las aventuras, la acción transcurre como si el jugador estuviera viviendo el papel del protagonista en primera persona, donde la importancia y el interés recaen principalmente en la originalidad de la historia mostrada y en el argumento planteado en ella. Esto es así puesto que el género suele requerir del uso de la inteligencia y de la astucia (y en muchos casos de la paciencia) del jugador a la hora de elegir correctamente las acciones a llevar a cabo para poder conseguir el objetivo inicialmente planteado en la historia.

Uno de los puntos fuertes de toda aventura conversacional, junto con la originalidad de la historia, son los objetos. El jugador puede cogerlos, dejarlos, usarlos por si solos o junto con otros objetos, y examinarlos para obtener mayor información sobre ellos y así poder resolver las situaciones que se le plantean. Estos objetos incluyen desde puertas que dan acceso a otras localizaciones, a NPC (Non-Playable Character), que no son más que personajes de la historia manejados por el propio juego con los que el usuario debe interaccionar para poder avanzar.

Las aventuras conversacionales más antiguas no contenían gráficos, eran sólo texto, lo que hacía desarrollar al máximo la imaginación del usuario, comparándose con libros interactivos. Unos años después (comienzos de los 80), ya contaban con gráficos que mostraban el paisaje, o los personajes de cada situación.

El primer juego conversacional se creó en 1973 y se llamó *"Adventure"*. Fue creado por *William Crowther* y *Don Woods* usando el lenguaje de programación fortran. El fenómeno comenzó a extenderse a finales de los 70 y principios de los 80 en Inglaterra y Estados Unidos, de donde surgieron gran cantidad de títulos.

En España, el despegue de este género se produjo en 1988 con la publicación de "La Aventura Original", el primer lanzamiento de Aventuras AD, el cual era una adaptación libre de "Adventure". No obstante había ya un gran número de aventuras similares que no terminaron de tener éxito en nuestro país, principalmente por la barrera del idioma.

Aventuras AD fue en cierto modo un referente español en este sector, y entre 1988 y 1992 Aventuras AD publicó seis juegos a la venta, todos ellos con un gran éxito de ventas. Estos juegos son:

- La Aventura Original (1988). Adaptación libre de "Adventure" de Crowther y Woods, todo un clásico de las aventuras conversacionales a nivel mundial y referente en la creación de otros títulos posteriores. En España fue la primera aventura que fue traducida al castellano. Ver Figura 1.2.
- Jabato (1989). Basado en un famoso personaje de cómic, se ambientaba en pleno auge del Imperio Romano. Su principal novedad era el manejo de varios personajes simultáneos, que recorrían gran parte de Europa y África.
- Cozumel (1990). El primer título de la trilogía "Ci-u-Than" ambientado en el Caribe en la primera mitad de siglo. El jugador encarnaba la figura del explorador Doc Monro, el cual naufragaba en la costa de la isla de Cozumel, donde comenzaba sus grandes aventuras.
- La Aventura Espacial (1990). Supuso un paréntesis en la realización de la trilogía de "Ci-u-Than". Estaba ambientado en el futuro, y tenía cierto toque experimental.
- Los Templos Sagrados (1991). Segunda parte de la trilogía "Ci-u-Than". Un juego de transición lleno de puzzles, también ambientado en la selva caribeña.
- Chichén Itzá (1992). Tercera parte de la trilogía, para muchos el juego más completo, con gran número de NPC y búsquedas por el fantástico mapa que ofrece.

Aventura Original *Jabato* *Chichén Itzá*

Figura 1.2 Imágenes de algunos juegos de Aventuras AD

Hoy día aún pueden encontrarse la mayoría de los títulos de las principales aventuras conversacionales disponibles para diferentes emuladores, que simulan el comportamiento de los antiguos ordenadores para las que fueron creadas (ordenadores de 8 bits como Commodore 64, Amstrad CPC, ZX Spectrum, y primeros ordenadores de 16 bits como el caso de Amiga, Atari). No obstante este tipo de videojuegos declinó a favor de las más conocidas aventuras gráficas, que como su nombre indica evolucionaron de las anteriores en cuanto a la forma de desarrollar una historia, usar diferentes objetos y sobre todo en el uso de la inteligencia como principal arma del jugador, pero con el aliciente de una compleja estructura gráfica que acompaña al usuario a nuevas sensaciones y proporciona un entorno más interactivo. Naturalmente esto fue posible gracias al aumento de las prestaciones que sufrieron los ordenadores de la época.

Pero este género no ha caído en el olvido ni mucho menos, actualmente existen clubes dedicados a las aventuras conversacionales como el caso de CAAD (Club de Aventuras AD), el cual dispone de foros, y espacios Web para el tratamiento de este genero (www.caad.es), donde se realizan diversas actividades entre las que destacan los concursos de mini-aventuras, creadas por los colaboradores o aficionados al genero. Igualmente se pueden encontrar noticias, relatos de eventos y toda la actualidad de este mundo en el fanzine de la SPAC (Sociedad para la Preservación de las Aventuras Conversacionales), que actualmente se encuentra alojada dentro del CAAD, y el cual es congénere de otros de habla no hispana como el fanzine inglés SPAG (*Society for the Promotion of Adventure Games*).

Teniendo en cuenta las limitaciones de los actuales dispositivos móviles, los datos tan favorables que se presentan para el mercado de los videojuegos en este sector, y las estupendas características que ofrecen las aventuras conversacionales a este respecto, el presente proyecto pretende desarrollar una plataforma sobre la que poder confeccionar y ejecutar de forma sencilla y totalmente configurable, videojuegos conversacionales para

dispositivos móviles, basándose para ello en la tecnología WAP (*Wireless Aplication Protocol*), de la que constan en su mayoría todos los dispositivos.

Se desarrollará una aplicación capaz de ejecutar diferentes videojuegos conversacionales simultáneamente y para diversos usuarios, de forma que se podrá tener varios usuarios por cada uno de los juegos implementados.

Cada juego estará basado en una serie de archivos XML (*eXtensible Markup Language*) que configurarán desde los textos e imágenes a mostrar, hasta los objetos a utilizar o las acciones a realizar por el usuario. Estos archivos serán usados por la aplicación junto con una serie de archivos encargados de la presentación de los contenidos, para realizar las acciones que requieran los usuarios del juego, en cada una de las situaciones mostradas.

La aplicación que se pretende crear será totalmente independiente de los juegos incluidos en ella, siendo de esta forma una plataforma que soporte la generación de nuevos juegos de forma sencilla y optimizada. Adicionalmente y para dar mayor cobertura al mercado de los dispositivos móviles, la aplicación será capaz de soportar diferentes protocolos de transporte, en concreto será capaz de soportar las dos versiones del protocolo WAP que actualmente están en uso (WAP 1.X y WAP 2.0), aunque la aplicación estará diseñada para poder soportar otros protocolos que en un futuro se estimen oportunos.

El presente documento está estructurado en diversos capítulos para poder explicar con detalle todas las partes del proyecto:

1. Introducción: es el presente capítulo, en el que se hace una somera descripción y justificación del proyecto.
2. Tecnologías y herramientas empleadas: en el cual se explican y justifican cada una de las herramientas software y hardware usadas, comparándolas a otras similares para ver sus ventajas e inconvenientes.
3. Diseño de la aplicación: donde se detallan las diferentes partes que la componen, y se realiza un estudio pormenorizado de su funcionamiento.
4. Instalación y Manual de usuario: en el que se explica paso a paso cómo instalar cada uno de los elementos que conforman el sistema para su correcto funcionamiento, así como una descripción detallada sobre el uso y creación de nuevos juegos para la plataforma.
5. Plan de pruebas: donde se detallan cada una de las pruebas realizadas al sistema para comprobar su correcto funcionamiento.

6. Conclusiones y líneas futuras: en el que se hace un balance del proyecto, así como el estudio de posibles mejoras o ampliaciones.

CAPÍTULO 2: TECNOLOGÍAS Y HERRAMIENTAS EMPLEADAS

2.1 INTRODUCCIÓN

En la realización de este proyecto se han puesto en conjunción multitud de herramientas tanto software (en su mayoría) como hardware, que a lo largo de este capítulo serán explicadas y comentadas, así como la justificación de su uso, en contraposición a otras con propósitos equivalentes. Se dará una somera explicación de cada una de las tecnologías disponibles y de la medida en que han participado en el presente proyecto. Cabe destacar que todas las herramientas software empleadas son de libre uso y distribución siguiendo unas premisas fundamentales a lo largo de todo el proyecto que son el bajo coste y la estandarización.

2.2 INTELIGENCIA CLIENTE/SERVIDOR

En el contexto de este proyecto, en el que se establecerá una comunicación mediante el protocolo WAP entre un servidor Web y uno o varios terminales móviles, es necesario realizar un estudio previo de las diferentes alternativas que se nos ofrecen a la hora de brindar de inteligencia a cada una de esas partes (clientes y servidores).

En el lado del cliente (dispositivos móviles), disponemos de:

- **Scripts→** Un lenguaje de script es un lenguaje interpretado por el propio navegador, que generalmente se introduce directamente sobre el código de etiquetas usado en la aplicación, ya sea HTML (*HyperText Markup Language*), XHTML (*eXtensible Hypertext Markup Language*), WML (*Wireless Markup Language*) etc. usando para ello unas etiquetas especificas (generalmente <script>). Para el caso de este proyecto, que se fundamenta en una comunicación basada en WAP, se disponen de los lenguajes WMLScript (*Wireless Markup Language Script*), y ECMAScript-MP (*ECMA Script Mobile Profile*).

- **Applets**→ Son mucho más potentes que los scripts, puesto que son clases de Java completas (archivos.class) que se cargan y se ejecutan en el cliente. Por ello ofrecen mucha mayor funcionalidad.

Las opciones para añadir inteligencia en el lado del servidor son:

- **CGI** (*Common Gateway Interface*)→ Es una importante tecnología de la Web que permite a un cliente (explorador Web) solicitar datos de un programa ejecutado en un servidor Web. CGI especifica un estándar para transferir datos entre el cliente y el programa. Es un mecanismo de comunicación entre el servidor Web y una aplicación externa.
- **Servlets/JSP** (*Java Server Pages*)→ Son la respuesta o equivalencia en Java de los CGI. Se comportan en un servidor como los applets en un cliente, es decir, un programa que se ejecuta en el marco de un servicio de red, y que responde a las peticiones de los clientes.

Teniendo en cuentas las características de los terminales móviles respecto a capacidad de procesamiento y velocidad, el peso de la aplicación debe recaer sobre el lado del servidor. Se ha optado por una tecnología basada en Java, usando *servlets* y JSP, dada su robustez y portabilidad entre otras ventajas que veremos con más detalle en posteriores apartados. No obstante, para no sobrecargar el medio radio con una comunicación continua y excesiva entre servidor y cliente, se ha optado por una tecnología de Script para los clientes móviles, de forma que éstos puedan realizar pequeños cálculos y operaciones, mejorando notablemente la comunicación en el medio radio.

2.3 JAVA

2.3.1 INTRODUCCIÓN

Prácticamente desde que empezaron a surgir lenguajes de programación de alto nivel su ciclo de vida siempre ha sido el mismo: nacen, se extienden, y pasado un cierto tiempo surge otro que lo supera y el primero cae en el olvido. Desde hace unos años parece que este paradigma ha cambiado, o al menos se ha frenado bastante con la aparición de Java. Y es que este lenguaje ofrece unas ventajas que no se pueden ignorar.

Cabe destacar entre ellas fundamentalmente dos: es seguro y es multiplataforma.

- Por una parte es seguro porque es un lenguaje que, por naturaleza, no permite acceder a la máquina físicamente más que a través de librerías que sí dependen del sistema operativo y que no son estándares de Java.
- Por otra parte es multiplataforma, ya que se ejecuta, a través de los bytecodes, sobre la máquina virtual de Java, que actúa como interfaz (esta sí es dependiente del sistema operativo) y que se puede instalar de forma gratuita en cualquier sistema operativo, consiguiendo con ello cumplir el famoso lema en inglés *"write once, run anywhere"*, tal y como se muestra en la Figura 2.1.

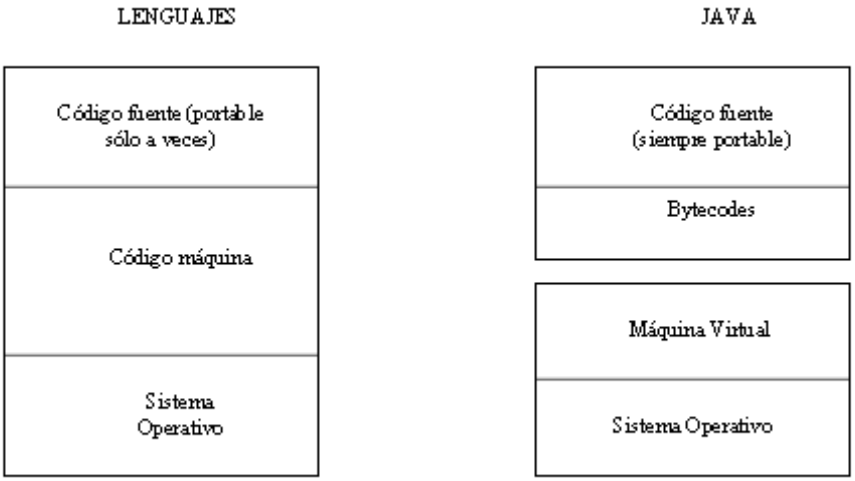

Figura 2.1 Diferencias entre Java y los lenguajes tradicionales

Son estas y otras ventajas las que lo hacen merecedor de estar presente como lo está en cada vez más aplicaciones de la WWW (*World Wide Web*) y en caso de este proyecto, de ser el eje fundamental sobre el cual se sustenta toda la aplicación.

En lo que respecta a este proyecto las claves para elegir Java han sido, el hecho de ser multiplataforma, muy robusto, ampliamente usado y la gratuidad del mismo. En el apartado 2.3.2 se realiza una comparación con otras tecnologías similares.

2.3.2 VENTAJAS FRENTE A OTRAS TECNOLOGÍAS

La primera tecnología que se puede ocurrir con un propósito parecido a las JSP de Java, es CGI, que constituye el sistema más antiguo que existe para la programación de las páginas dinámicas de servidor. Actualmente se encuentra un poco desfasado por diversas razones entre las que destaca la dificultad con la que se desarrollan los programas y la pesada carga que supone para el servidor que los ejecuta, ya que habitualmente es

interpretado en el servidor en PERL (*Practical Extraction and Reporting Language*), o ya compilado en C.

La tecnología PHP (Php *Hypertext Preprocessor*) habría sido otra buena opción a considerar. Es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación. Al igual que JSP se escribe dentro del código HTML, por lo que comparte en este sentido también sus ventajas. Así, la única razón de usar JSP frente a PHP es por homogeneidad del proyecto, que emplea Java como base, ya que básicamente son iguales.

El último contendiente de JSP son las ASP.NET (*Active Server Pages* con tecnología *.NET*), que tienen dos diferencias fundamentales respecto a las JSP:

- Las páginas ASP.NET están escritas en cualquier lenguaje soportado por el marco de trabajo .NET de Microsoft como *Visual Basic.NET*, C# o *JScript.NET*, mientras que las JSP están en lenguaje Java. Por lo tanto, las páginas JSP son independientes de la plataforma y las páginas ASP.NET no lo son.
- Las páginas JSP usan tecnología JavaBeans como arquitectura de componentes y las páginas ASP.NET usan componentes .NET.

Como diferencias más importantes a la hora de elegir JSP frente a ASP.NET tenemos las siguientes:

a. Velocidad y Escalabilidad: Aunque las páginas ASP.NET son cacheadas, siempre son interpretadas, en cambio las páginas JSP son compiladas en *servlet* Java y cargadas en memoria la primera vez que se las llama, y son ejecutadas para todas las llamadas siguientes. Esto le da a las páginas JSP la ventaja de la velocidad y escalabilidad sobre las páginas ASP.NET.
b. Etiquetas Extensibles: Las páginas JSP tienen una característica avanzada conocida como etiquetas extensibles. Este mecanismo permite a los desarrolladores crear etiquetas personalizadas. En otras palabras, las etiquetas extensibles permiten extender la sintaxis de las etiquetas de las páginas JSP. Esta cualidad no tiene su equivalente en ASP.NET.
c. Libertad de Elección: Las páginas ASP.NET sólo funcionan con Microsoft IIS (*Internet Information Services*). El uso de páginas ASP.NET requiere un compromiso con los productos de Microsoft, mientras que las páginas JSP no

imponen ningún servidor Web ni sistema operativo. Esta es quizá la razón más importante para interesarse por las JSP en detrimento de las ASP.NET.

d. La tecnología JSP es de libre uso (gratuita) mientras que ASP.NET es una tecnología propietaria.

Dentro de la tecnología Java podemos ver otra posibilidad al uso de las JSP, es usar *servlet*, que sirven para el mismo propósito, y de hecho se usan en este proyecto para un objetivo concreto. Pero al igual que CGI, los *servlets* carecen de la división entre la presentación y el código en sí, cosa que en las JSP se soluciona con el uso de JavaBeans. Las JavaBeans son clases de Java de las que hacen uso las JSP para establecer, modificar, y usar sus propiedades, y en función de ellas mostrar lo que interese por pantalla.

2.4 SERVLETS

Los *servlets* de Java son un componte clave del desarrollo en la parte del servidor con Java. Un *servlet* es una clase Java que puede ser cargada dinámicamente como una pequeña extensión de un servidor para aumentar su funcionalidad. Se podría definir un *servlet* como un programa escrito en Java que se ejecuta en el marco de un servicio de red (un servidor HTTP, por ejemplo), y que recibe y responde a las peticiones de uno o más clientes [García de Jalón'99].

Los *servlets* son comúnmente utilizados para crear contenido dinámico para una página Web, formando lo que se llaman aplicaciones Web. Mientras una página Web estática simplemente muestra un contenido fijo por el que podemos navegar, una aplicación Web puede ser más interactiva. Construir páginas Web al vuelo es útil (y comúnmente usado) por varias de razones:

- La página Web está basada en datos enviados por el usuario. Por ejemplo, las páginas de resultados de los motores de búsqueda se generan de esta forma, y los programas que procesan pedidos desde sitios de comercio electrónico también.
- Los datos cambian frecuentemente. Por ejemplo, un informe sobre el tiempo o páginas de cabeceras de noticias podrían construir la página dinámicamente, quizás devolviendo una página previamente construida y luego actualizándola.
- Las páginas Web que usan información desde bases de datos corporativas u otras fuentes. Por ejemplo, usaríamos esto para hacer una página Web en una tienda online que liste los precios actuales y el número de artículos en stock.

Los *servlets* nacieron como la alternativa Java a los programas CGI. Mientras que con los CGI se usan varios procesos para manejar distintos programas o peticiones, los *servlets* son manejados por hilos de ejecución dentro de un único proceso correspondiente al servidor Web. Por tanto, para cada petición que se realice a un CGI, se deberá cargar el programa en memoria; en cambio, con un *servlet* ya se tendrá cargado en la memoria del servidor.

Otra ventaja de los *servlets* es su portabilidad. Se puede utilizar el mismo *servlet* en cualquier plataforma que soporte Java y en cualquiera de los servidores Web que soporten esta tecnología.

2.4.1 CARACTERÍSTICAS

Las principales características de los *servlets* son:

1. **Portabilidad**. Debido a que los *servlets* se escriben en Java, son altamente portables a través de los sistemas operativos y de los servidores Web.
2. **Potencia**. Los *servlets* pueden aprovechar completamente toda la potencia del núcleo de la API (*Aplication Program Interface*) Java: networking y acceso a URL (*Universal Resource Locator*), multihilo, manipulación de imágenes, compresión de datos, conectividad con bases de datos, internacionalización, llamadas a procedimientos remotos, serialización de objetos, etc.
3. **Eficiencia**. La invocación de *servlets* es muy eficiente, mucho más que un CGI tradicional. Con CGI tradicional, se arranca un nuevo proceso para cada solicitud HTTP (*HyperText Transfer Protocol*). Si el programa CGI hace una operación relativamente rápida, la sobrecarga del proceso de arrancada puede dominar el tiempo de ejecución. Con los *servlets*, la máquina Virtual Java permanece arrancada, y cada petición es manejada por un hilo o hebra Java de peso ligero. De forma similar, en CGI tradicional, si hay N peticiones simultáneas para el mismo programa CGI, el código de este problema se cargará N veces en memoria. Sin embargo, con los *servlets*, hay N hebras pero sólo una copia de la clase Servlet. Los *servlet* también tienen más alternativas que los programas normales CGI para optimizaciones como las caches de cálculos previos, mantener abiertas las conexiones de bases de datos, etc.

4. **Seguridad**. Debido a que están escritos en Java, los *servlets* heredan los mecanismos de seguridad que proporciona dicho lenguaje. Además, la API de los *servlets* en sí ha sido implementada para ser segura. El mecanismo automático de recolección de basura y la ausencia de gestión de punteros por parte del usuario (aunque Java los maneja interna y transparentemente), hacen que los *servlets* sean seguros en cuanto a problemas de manejo de memoria como punteros con referencia inválidas. Los *servlets* pueden manejar los errores fácilmente, debido al mecanismo de manejo de excepciones de Java. Si un *servlet* realiza una división por cero o alguna otra operación ilegal, se puede capturar la excepción y realizar las operaciones oportunas para subsanar el fallo.
5. **Elegancia**. El código de un *servlet* es elegante, ya que es orientado a objetos, modular y bastante simple. Una de las razones de su simplicidad es la propia API de los *servlets*, que incluye métodos y clases para manejar la mayoría de las tareas necesarias para el desarrollo de un *servlet*. Por ejemplo el manejo de cookies o de sesiones están incluidos dentro de clases de la API.
6. **Integración**. Los *servlets* están estrechamente integrados con el servidor. Esta integración permite al *servlet* cooperar con el servidor. Por ejemplo, un *servlet* puede usar el servidor para traducir rutas de ficheros, comprobar autorizaciones y, en algunos casos, añadir usuarios a la base de datos del servidor.
7. **Extensibilidad y flexibilidad**. La API de los *servlets* ha sido diseñada para ser fácilmente extensible. Incluye clases optimizadas para el desarrollo de *servlets* HTTP, pero en un futuro, podrían añadirse clases optimizadas para otros tipos de *servlets*, bien por parte de Sun o por parte de terceros. Los *servlets* también son bastante flexibles. Pueden ser utilizados para generar una página Web completa.

2.4.2 ESTRUCTURA DE UN SERVLET.

Un *servlet* es una clase Java, y como tal sigue la relación entre las clases e interfaces de Java, muy determinada por el concepto de herencia. Esta relación se entiende mucho mejor mediante una representación gráfica tal como la que puede verse en la Figura 2.2. En dicha figura se representan las clases con letra normal y las interfaces con cursiva.

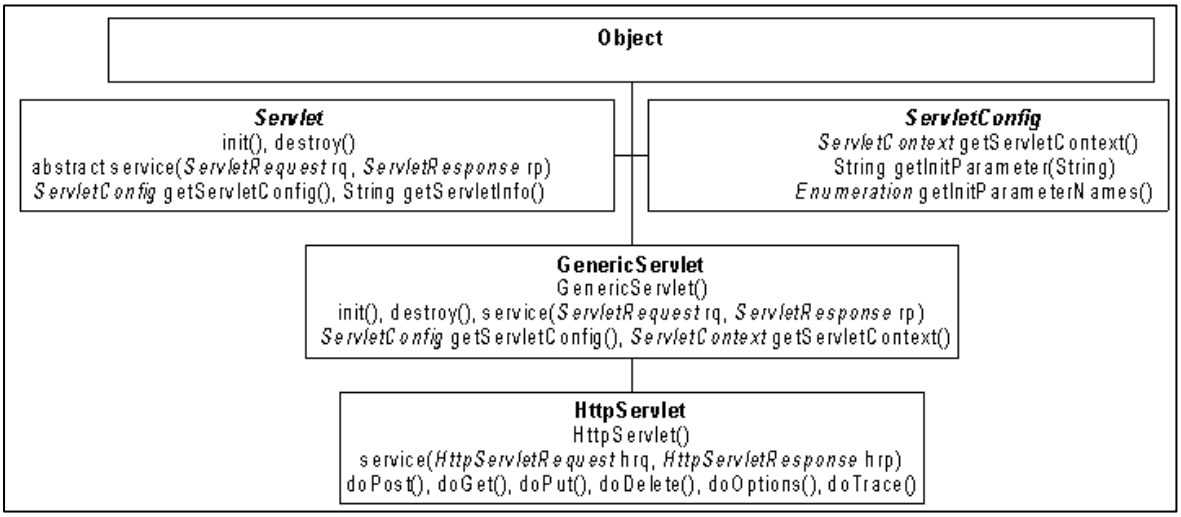

Figura 2.2 Estructura de un servlet

La clase GenericServlet es una clase abstracta puesto que su método service() es abstracto. Esta clase implementa dos interfaces, de las cuales la más importante es la interfaz Servlet.

La interfaz Servlet declara los métodos más importantes de cara a la vida de un *servlet*: init() que se ejecuta sólo al arrancar el *servlet*; destroy() que se ejecuta cuando va a ser destruido y service() que se ejecutará cada vez que el *servlet* deba atender una solicitud de servicio.

Cualquier clase que derive de GenericServlet deberá definir el método service(). Es muy interesante observar los dos argumentos que recibe este método, correspondientes a las interfaces ServletRequest y ServletResponse.

La primera de ellas referencia a un objeto que describe por completo la solicitud de servicio que se le envía al *servlet*. Si la solicitud de servicio viene de un formulario HTML, por medio de ese objeto se puede acceder a los nombres de los campos y a los valores introducidos por el usuario; puede también obtenerse cierta información sobre el cliente (ordenador y browser).

El segundo argumento es un objeto con una referencia de la interface ServletResponse, que constituye el camino mediante el cual el método service() se conecta de nuevo con el cliente y le comunica el resultado de su solicitud. Además, dicho método deberá realizar cuantas operaciones sean necesarias para desempeñar su cometido: escribir y/o leer datos de un fichero, comunicarse con una base de datos, etc.

El método service() es realmente el corazón del *servlet*.

En la práctica, salvo para desarrollos muy especializados, todos los *servlets* deberán construirse a partir de la clase HttpServlet, sub-clase de GenericServlet. La clase HttpServlet ya no es abstracta y dispone de una implementación o definición del método service(). Dicha implementación detecta el tipo de servicio o método HTTP que le ha sido solicitado desde el browser y llama al método adecuado de esa misma clase (doPost(), doGet(), etc.).

Cuando el programador crea una sub-clase de HttpServlet, por lo general no tiene que redefinir el método service(), sino uno de los métodos más especializados (normalmente doPost()), que tienen los mismos argumentos que service():dos objetos referenciados por las interfaces ServletRequest y ServletResponse.

En la Figura 2.2 aparecen también algunas otras interfaces, cuyo papel se resume a continuación:

- La interfaz ServletContext permite a los *servlets* acceder a información sobre el entorno en que se están ejecutando.
- La interfaz ServletConfig define métodos que permiten pasar al *servlet* información sobre sus parámetros de inicialización.
- La interfaz ServletRequest permite al método service() de GenericServlet obtener información sobre una petición de servicio recibida de un cliente. Algunos de los datos proporcionados por GenericServlet son los nombres y valores de los parámetros enviados por el formulario HTML y una *Input Stream*.
- La interfaz ServletResponse permite al método service() de GenericServlet enviar su respuesta al cliente que ha solicitado el servicio. Esta interfaz dispone de métodos para obtener un *Output Stream* o un *Writer* con los que enviar al cliente datos binarios o caracteres, respectivamente.
- La interfaz HttpServletRequest deriva de ServletRequest. Esta interfaz permite a los métodos service(), doPost(), doGet(), etc. de la clase HttpServlet recibir una petición de servicio HTTP. Esta interfaz permite obtener información de la cabecera de la petición de servicio HTTP.
- La interfaz HttpServletResponse extiende ServletResponse. A través de esta interfaz los métodos de HttpServlet envían información a los clientes que les han pedido algún servicio.

2.4.3 CICLO DE VIDA

El ciclo de vida de los *servlets* permite a los servidores conseguir las prestaciones de los programas CGI y la seguridad de la programación a bajo nivel de servidores. Un motor de *servlets* puede ejecutar sus *servlets* en una máquina virtual Java. Debido a esto, distintos *servlets* pueden compartir datos eficientemente. Además, al poder persistir en memoria entre cada petición se produce un ahorro de memoria considerable.

Este ciclo de vida se puede resumir en tres pasos (como se ve en la figura 2.3), aunque cada servidor podrá implementarlos de la manera que elija. Los pasos son:

1. Crear e inicializar el *servlet*.
2. Manejar las peticiones que realicen los clientes.
3. Destruir el *servlet* y borrarlo de la memoria.

En la creación, el servidor llama al método init() de la clase GenericServlet. Cuando un *servlet* es cargado por primera vez, el método init() es llamado por el servidor HTTP. Este método no será llamado nunca más mientras el *servlet* se esté ejecutando. Esto permite al *servlet* efectuar cualquier operación de inicialización potencialmente costosa en términos de **CPU** (*Central Processing Unit*), ya que ésta sólo se ejecutará la primera vez. Esto es una ventaja importante frente a los programas CGI, que son cargados en memoria cada vez que hay una petición por parte del cliente

Si el servidor permite pre-cargar los *servlets*, el método init() será llamado al iniciarse el servidor. Si el servidor no tiene esa posibilidad, será llamado la primera vez que haya una petición por parte de un cliente. El método init() tiene un único argumento, que es una referencia a un objeto de la interfaz ServletConfig, que proporciona los argumentos de inicialización del *servlet*. Este objeto dispone del método getServletContext() que devuelve una referencia de la interfaz ServletContext, que a su vez contiene información acerca del entorno en el que se está ejecutando el *servlet*.

El servidor garantiza que el método init() termina su ejecución antes de que sea llamado cualquier otro método del *servlet*.

Después de la creación, el servidor está listo para atender peticiones. Lo hará llamando al método service(). Este método es el corazón de los *servlets*. Acepta como parámetros objetos ServletRequest, que contienen la petición del cliente, y ServletResponse, que dispone de métodos para devolver información al cliente.

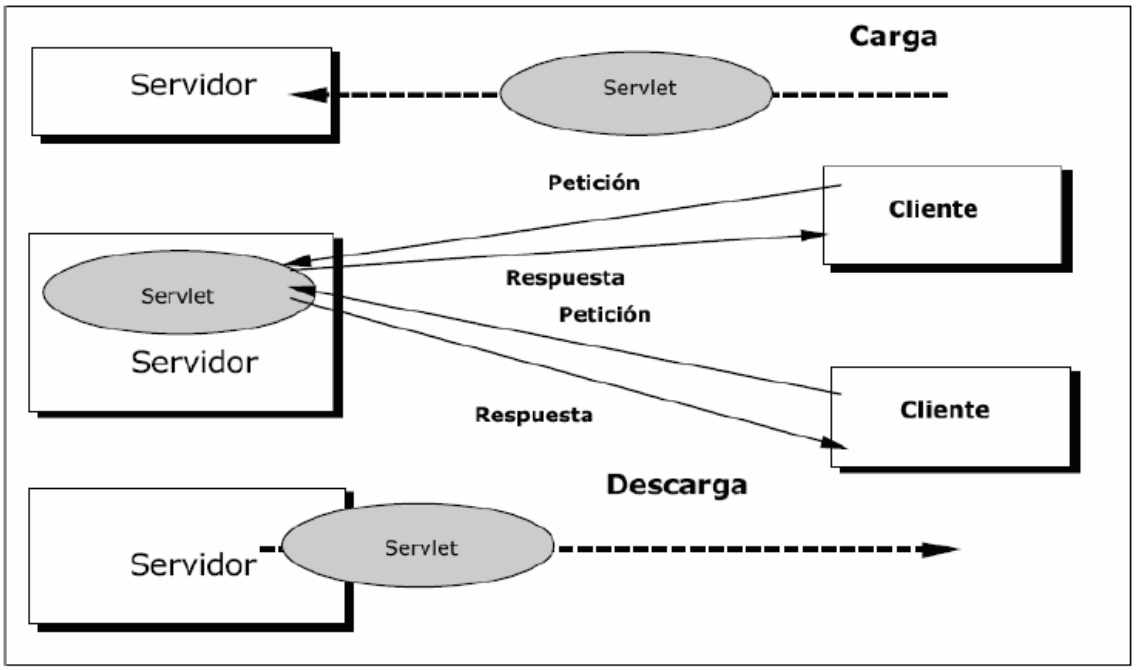

Figura 2.3 Ciclo de vida de un servlet

Finalmente, cuando el servidor va a destruir el *servlet*, hace una llamada al método destroy(). Esto puede ser debido, por ejemplo, a que haya que cargar una nueva versión del *servlet* o cuando se apaga el servidor. Todos los recursos del sistema bloqueados por init() son liberados al invocar este método, y se garantiza que solo se llamará una vez durante la vida del *servlet*.

2.4.4 JERARQUÍA DE ARCHIVOS

Una vez creado nuestros *servlets* (archivos.Java), debemos compilarlos (se obtienen archivos.class) y colocarlos en el contenedor de *servlets* utilizado. Esto no es tan simple como en un servidor Web normal, en el que es suficiente con colocar los archivos en el directorio de salida del servidor, para los *servlets* se necesita seguir una estructura de directorios especificada, tal y como muestra la Figura 2.4.

Todos los ficheros cuelgan del directorio Webapps (perteneciente al contenedor de *servlets*), pero no se pueden colocar directamente ahí. De ese directorio descienden otros subdirectorios, que es donde efectivamente se colocan las aplicaciones.

Cada directorio bajo Webapps, corresponde a un contexto; hay un contexto raíz ROOT y tantos otros contextos como se quiera definir (generalmente uno por cada proyecto realizado). Dentro de cada contexto, la estructura de directorios es también siempre la misma.

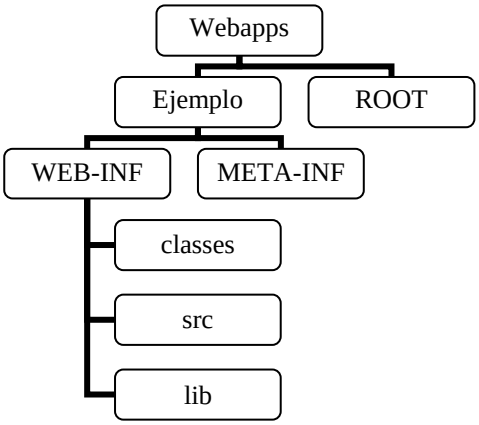

Figura 2.4 Jerarquía de archivos para *servlets*

/*.html *.jsp *.css → Este directorio base (con el nombre del proyecto) contiene los elementos que comúnmente son utilizados en un sitio, documentos en HTML, JSP, CSS (*Cascading Style Sheets*) y otros elementos visibles al usuario Web. Para aplicaciones grandes pueden crearse subdirectorios en los que organizar éstos contenidos.

/WEB-INF/Web.xml → Contiene elementos de seguridad de la aplicación así como detalles sobre los *servlets* que serán utilizados dentro de la misma.

/WEB-INF/classes/ → Contiene las clases Java adicionales (archivos.class) a las del JDK (*Java Development Kit*). En este directorio es donde se introducen los archivos.class de nuestros *servlets*. Estas clases estarán disponibles para todas las otras clases de la aplicación.

/WEB-INF/lib/ → Contiene los archivos JAR (*Java ARchives*) que serán utilizados por la aplicación. Caso de necesitar algún JAR externo, como es el caso de usar alguna librería externa de Java, esta se introducirá aquí.

/**WEB-INF/src/** → Directorio usado para guardar de forma opcional las fuentes de los ficheros. Aquí se puede guardar los archivos.Java de los *servlets* empleados.

Se ha mencionado la existencia de un archivo llamado “Web.xml”. El formato que debe seguir este archivo de configuración de los *servlets* usados es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<Web-app version="2.4" xmlns=http://Java.sun.com/xml/ns/j2ee
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://Java.sun.com/xml/ns/j2ee http://Java.sun.com/xml/ns/j2ee/Web-
app_2_4.xsd">
   <servlet>
      <servlet-name>NOMBRE</servlet-name>
      <servlet-class>NOMBRE</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>NOMBRE</servlet-name>
      <url-pattern>/RUTA/NOMBRE</url-pattern>
   </servlet-mapping>

</Web-app>
```

Tal y como se aprecia, el archivo comienza con las cabeceras que lo identifican como un archivo XML. A continuación se define el elemento raíz del documento, llamado Web-app junto a una referencia para la comprobación de la sintaxis del documento. Dentro de este elemento se deben crear los elementos hijos mostrados para cada *servlet* de la aplicación, ya que serán los encargados de contener la información útil para la ejecución de éstos. Por cada *servlet*, se dispone dentro del elemento raíz de un elemento <servlet> que contendrá el nombre del archivo y la clase a la que pertenece, y un elemento <servlet-mapping> el cual contiene la información para realizar la llamada a ese archivo, esto es el nombre del archivo y la dirección con la que se quiere llamar. Esto es muy útil, puesto que se pueden crear usando este <url-pattern> directorios virtuales donde alojar los *servlets*, aumentando de esa forma la seguridad.

2.5 JSP

JSP es una tecnología orientada a crear páginas Web con tecnología Java. Sólo con decir esto ya se hacen evidentes dos características básicas. Por una parte es multiplataforma, y por tanto se pueden crear aplicaciones que se ejecuten en diferentes servidores Web que funcionen en diferentes sistemas operativos. Por otra parte, siguen siendo páginas Web, por lo que, aunque están mezcladas con etiquetas especiales para programar scripts escritos en Java, siguen estando compuestas por código HTML/XML.

De esta forma, se pueden componer en cualquier editor HTML/XML o, simplemente, en un editor de texto como el bloc de notas en Windows.

Se generan archivos con extensión .jsp que incluyen, dentro de la estructura de etiquetas del lenguaje deseado para la presentación de contenidos (HTML, XHTML, WML, ect.), las sentencias Java a ejecutar en el servidor. Antes de que sean funcionales los archivos, el motor JSP lleva a cabo una fase de traducción de esa página en un *servlet*, implementado en un *archivo.class* (Bytecodes de Java). Esta fase de traducción se lleva a cabo habitualmente cuando se recibe la primera solicitud de la página .jsp, aunque existe la opción de precompilar en código para evitar ese tiempo de espera la primera vez que un cliente solicita la página.

Como se ha dicho, internamente las JPS son *servlets*, el servidor se encarga de compilar la página JSP y a partir de ese momento son un *servlet* como cualquier otro. Para entender el proceso por el que se transforman en *servlet* se puede ver un esquema como el de la Figura 2.5.

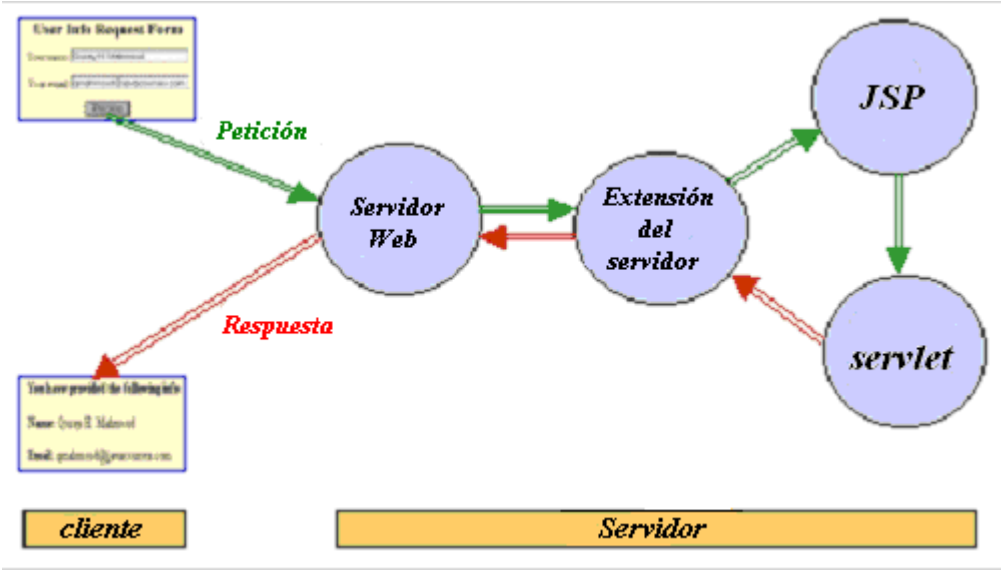

Figura 2.5 Flujo petición/respuesta cuando se llama a una página JSP

Una vez el archivo .jsp haya sido compilado en un *servlet*, la próxima vez que un usuario requiera ese archivo .jsp el motor *servlet* se encarga de dar la respuesta con el *servlet* ya compilado, a menos que la página haya cambiado, en cuyo caso se recompila y se vuelve a poner a disposición del servidor.

2.6 ARQUITECTURA MVC

MVC (Modelo Vista Controlador) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el controlador es el Sistema de Gestión de Base de Datos y el modelo es el modelo de datos.

Modelo: Es la representación específica de la información con la cual el sistema opera, ya sea usando bases de datos, archivos u otros medios.

Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario, adaptándose a las necesidades de éste.

Controlador: Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Puede considerarse como la parte inteligente de una aplicación que implemente la arquitectura MVC, realizando las acciones requeridas por las peticiones del usuario.

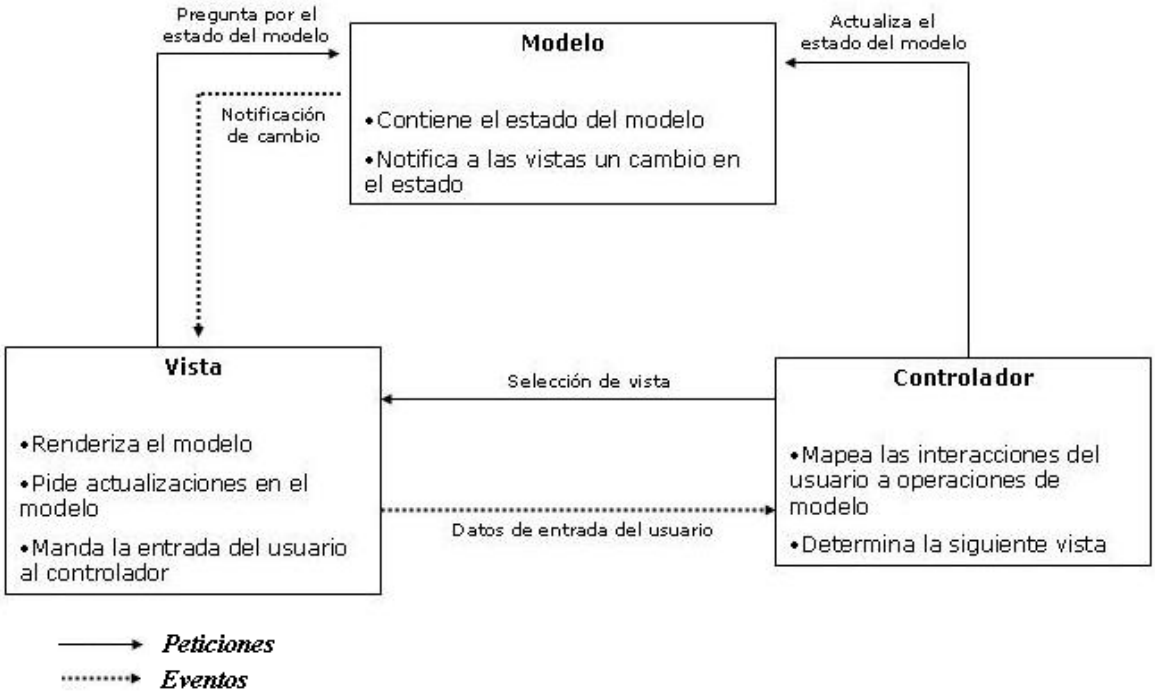

Figura 2.6 Relación entre las capas del modelo MVC

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que sigue el control generalmente es el mostrado en la figura 2.6, que se explica a continuación:

El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace,…). El controlador recibe (por parte de los objetos de la interfaz-

vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (*handler*) o *callback*. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

La aplicación de la arquitectura MVC a este proyecto se fundamenta en la separación que esta arquitectura propone en cuanto a datos, control, y vista al usuario. En concreto la vista se realiza usando la tecnología JSP de la que se ha hablado en el apartado 2.5, la cual combina la generación de páginas HTML, XHTML, WML, etc. junto con el código Java empleado en el servidor, consiguiendo de esa forma la generación de una vista dinámica y adaptada al terminal móvil al que está orientado.

Por otra parte el control de la aplicación se realiza usando también una tecnología Java, en concreto se ha optado por el uso de *servlets* (apartado 2.4), los cuales no tienen en ningún momento salida hacia el usuario, y simplemente centran su trabajo en responder a las peticiones de los usuarios, modificando para ello los archivos que presentan el modelo y haciendo llamadas para actualizar la vista.

Finalmente el modelo estará compuesto por una serie de archivos XML, dispuestos en una estructura de directorios de la que se hablará en posteriores apartados.

2.7 XML

El lenguaje extensible de marcas, abreviado XML, describe una clase de objetos de datos llamados documentos XML y parcialmente describe el comportamiento de programas de computador que pueden procesarlos. XML es un perfil de aplicación o forma restringida de SGML (*Standard Generalized Markup Language*), al igual que HTML. La diferencia fundamental del XML con el lenguaje HTML es que, aún usando etiquetas como HTML, las etiquetas XML describen el contenido en vez de presentarlo.

Puesto que se trata de describir los datos y éstos pueden ser de muy diversa índole, el lenguaje XML permite crear etiquetas propias así como los atributos de las mismas. La única restricción a la que se debe atender es un conjunto de normas de sintaxis que están pensadas con el fin de garantizar la consistencia de los datos representados.

En teoría HTML es un subconjunto de XML especializado en presentación de documentos para la Web, mientras que XML es un subconjunto de SGML especializado en la gestión de información para la Web. En la práctica XML contiene a HTML aunque no en su totalidad. La definición de HTML contenido totalmente dentro de XML y por lo tanto que cumple a rajatabla la especificación SGML es XHTML, del que se tratará con más detalle en el siguiente apartado.

Dada la flexibilidad a la vez que la organización que ofrecen éstos documentos en cuanto a la información que pueden contener, han sido elegidos para formar el sistema de archivos que forman el modelo (siguiendo la arquitectura MVC) para este proyecto.

2.7.1 XSD

Dada la extensibilidad de los documentos XML que permiten definir nuevas etiquetas y atributos, en ciertas ocasiones es necesario restringir mediante normas o reglas la estructura del documento, a fin de evitar errores y conseguir un documento correctamente estructurado. Ésto puede conseguirse usando XSD (*XML Schema Definition*), que es un lenguaje de esquema utilizado para describir la estructura y las restricciones de los contenidos de los documentos XML de una forma muy precisa, más allá de las normas sintácticas impuestas por el propio lenguaje XML.

.

Los documentos esquema (con extensión .xsd) se concibieron como una alternativa a las DTD (*Document Type Definition*), más compleja, intentando superar sus puntos débiles y buscar nuevas capacidades a la hora de definir estructuras para documentos XML. La

principal aportación de XMLSchema es el gran número de los tipos de datos que incorpora, así como de elementos, ya que se trata de un documento escrito a su vez en el lenguaje XML. Los componentes con los que formar un esquema son:

- Elemento (*element*)
- Atributo (*attribute*)
- Tipo simple (*simple type*)
- Tipo complejo (*complex type*)
- Notación (*notation*)
- Grupo modelo nombrado (*named model group*)
- Grupo de atributos (*attribute group*)
- Restricción identidad (*identity constraint*)

Todos estos elementos, junto con la gran variedad de estructuras ofrecida por el lenguaje XML, ofrecen una poderosa herramienta a la hora de definir las restricciones necesarias a los documentos XML creados.

2.8 WAP

2.8.1 INTRODUCCIÓN

WAP o protocolo de aplicaciones inalámbricas es un estándar abierto internacional (salvo para Japón que usa mayoritariamente Imode) para aplicaciones que utilizan las comunicaciones inalámbricas, p.ej. acceso a servicios de Internet desde un teléfono móvil.

Se trata de la especificación de un entorno de aplicación y de un conjunto de protocolos de comunicaciones para normalizar el modo en que los dispositivos inalámbricos, se pueden utilizar para acceder a correo electrónico, grupo de noticias y otros.

El organismo que se encarga de desarrollar el estándar WAP fue originalmente el WAP Forum, fundado por cuatro empresas del sector de las comunicaciones móviles, Sony-Ericsson, Nokia, Motorola y Openwave (originalmente Unwired Planet). Desde 2002 el WAP Forum es parte de la OMA (*Open Mobile Alliance*), consorcio que se ocupa de la definición de diversas normas relacionadas con las comunicaciones móviles, entre ellas las normas WAP.

WAP define un entorno de aplicación y una pila de protocolos para aplicaciones y servicios accesibles a través de terminales móviles. Consiste en un conjunto de

especificaciones que permiten que los desarrolladores diseñen aplicaciones de interconexión para terminales móviles.

La tecnología WAP permite que los usuarios de estos dispositivos puedan acceder a servicios disponibles en Internet. Sin embargo, existen algunas consideraciones a tener en cuenta al diseñar éstos servicios para usuarios móviles, fundamentalmente debidas a las características de los terminales: pantalla significativamente más pequeña que la de un ordenador personal, teclados más limitados que los de un ordenador, limitaciones en la memoria disponible, tanto memoria RAM como memoria para almacenamiento persistente, y limitaciones en la capacidad del procesador, en comparación con la memoria y procesador de un ordenador personal típico. Las redes de telefonía móvil ofrecen también unas prestaciones por lo general menores que los accesos a Internet, si bien con las redes de tercera generación como UMTS (*Universal Mobile Telecommunications System*) las prestaciones mejoran de manera importante.

2.8.2 ARQUITECTURA WAP 1.X

WAP está basado en una arquitectura cliente-servidor en la que se amplía la arquitectura típica de la Web para que se tenga acceso desde una red inalámbrica. En la arquitectura Web, el cliente es un ordenador con un navegador que realiza peticiones HTTP al servidor, que es el encargado de devolver las páginas solicitadas (Figura 2.7).

La arquitectura WAP 1.X extiende la estructura Web introduciendo un elemento intermedio, el Proxy WAP, cuyas funciones más importantes incluyen la de traducción (Gateway) entre los protocolos usados en Internet y el medio radio, además de poder ser utilizado como caché de páginas visitadas para que el rendimiento del sistema sea óptimo (Figura 2.8).

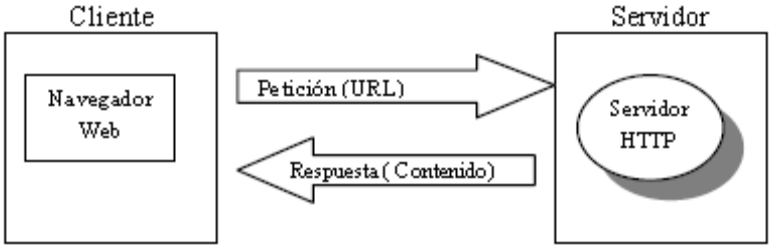

Figura 2.7: Arquitectura Web

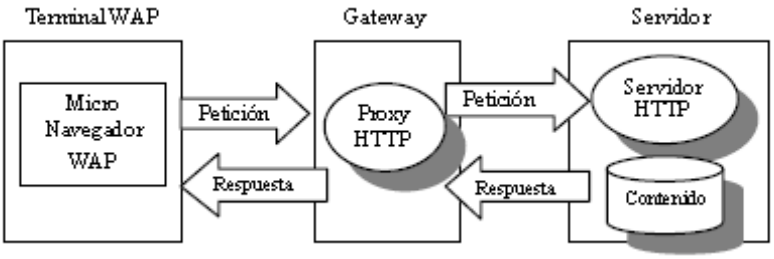

Figura 2.8: Arquitectura WAP 1.X

En este esquema, el cliente es un terminal móvil con capacidad de navegación WAP que hace las peticiones de las páginas a un Proxy WAP, que es el encargado de traducirlas a peticiones HTTP, las cuales son enviadas al servidor Web que contiene la página solicitada. El servidor devuelve la página al Proxy y éste la recodifica para ser enviada al cliente de una manera eficiente a través del medio radio.

La razón de que se hayan desarrollado diferentes protocolos para el medio radio, y por tanto, que sea necesaria la utilización de un Proxy WAP, es debido a que es un medio en el que la cantidad de datos a enviar debe ser lo menor posible, ya que el ancho de banda es reducido, y la probabilidad de pérdidas es elevada. La torre de protocolos para la versión WAP 1.x, puede verse en la Figura. 2.9.

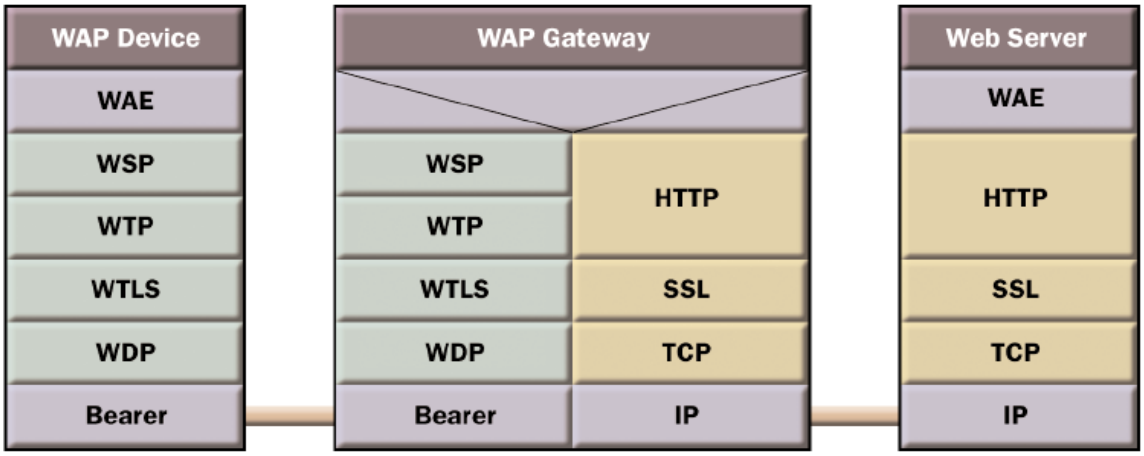

Figura 2.9 Pila de protocolos de WAP 1.x

En la parte Web se distinguen los protocolos clásicos del mismo, como son HTTP, SSL (*Secure Sockets Layer*), TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*), aunque se añade una capa más, el WAE (*Wireless Application Environment* – Entorno de aplicación inalámbrica), ya que las páginas WAP que vayan a ubicarse en un servidor

tienen que tener un formato específico basado en un lenguaje especialmente desarrollado para WAP, el WML que es un lenguaje de marcas similar al HTML, y su lenguaje de script asociado, el WMLScript, de los cuales se tratará más adelante.

En la parte radio de la arquitectura, y en la zona más alta de la torre de protocolos, nos encontramos también el WAE, ya que el terminal WAP debe incorporar un micronavegador que sea capaz de interpretar los comandos recibidos en WML y WMLScript. Seguidamente nos encontramos el protocolo WSP (*Wireless Session Protocol* – Protocolo Inalámbrico de Sesión) que posee las funcionalidades del HTTP, pero mediante una codificación más compacta. WSP es el encargado del mantenimiento de las sesiones. WTP (*Wíreless Transaction Protocol* – Protocolo Inalámbrico de Transacciones) proporciona varias clases de servicio de transacciones, así como seguridad usuario-usuario.

Existe una capa orientada a la seguridad, WTLS (*Wireless Transport Layer Security* – Capa de Seguridad de Transporte Inalámbrico) que proporciona integridad y privacidad de los datos, así como servicios de autentificación. No obstante, existe un importante riesgo de la seguridad e integridad de los datos en WAP 1.X, debido a que WTLS solo actúa en los enlaces radio, mientras que en la pasarela los datos deben estar en abierto (sin cifrar), lo que implica una posible vulnerabilidad. En un nivel inferior nos encontramos WDP (*Wireless Datagram Protocol* – Protocolo Inalámbrico de Datagramas), que sirve de capa de transporte y como intermediario entre los protocolos portadores y los de las capas superiores. En la parte más baja de la torre de protocolos nos encontramos los posibles portadores, entre los que se encuentran, además de GSM (G*lobal System for Mobile comunications*) y GPRS (*General Packet Radio Service*), otros como DAMPS (*Digital-Advanced Mobile Phone Service*), CDMA (*Code Division Multiple Access*), PHS (*Personal Handyphon Syste*m), etc., hecho que demuestra que WAP es independiente de la tecnología usada como portadora del servicio.

Como puede advertirse en la Figura 2.9, prácticamente existe una relación uno a uno entre los protocolos de Internet y los del medio radio, de forma que sea relativamente sencillo realizar la traducción entre ambas zonas.

2.8.3 ARQUITECTURA WAP 2.0

En la versión del protocolo WAP 2.0 se extiende la arquitectura de protocolos de forma que se asemeje a la arquitectura Web (Figura 2.10), pero con ciertas mejoras, como es el hecho de que sea el propio servidor el que pueda iniciar el envío de información sin que el

cliente haya realizado ninguna petición (tecnología Push). Una posible utilidad de este servicio es que el servidor avise al cliente cuando se den una serie de condiciones, como puede ser una noticia importante o que le ha llegado un correo electrónico.

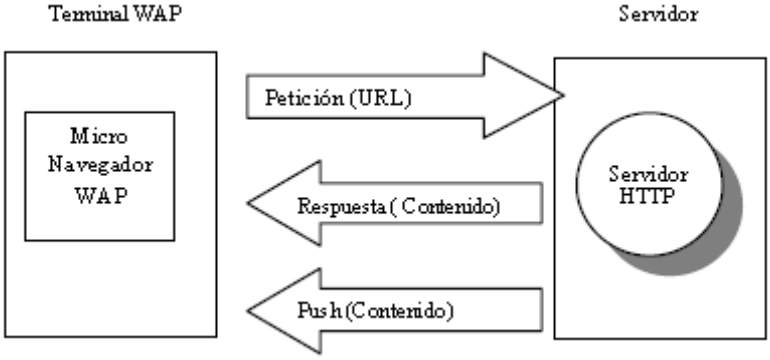

Figura 2.10: Arquitectura WAP 2.0

La arquitectura del WAP 1.x está pensada para optimizar los escasos recursos radio debido al escaso ancho de banda que éste ofrece. Esto sucede con GSM y también con GPRS, aunque la evolución de las tecnologías está consiguiendo que el ancho de banda se esté incrementando notablemente, hasta el punto que con UMTS se puede obtener un ancho de banda de hasta 2 Mbps. UMTS es precisamente el escenario idóneo para aplicar la arquitectura WAP 2.0 y, por lo tanto, conseguir la confluencia hacia IP.

La nueva versión de WAP, WAP 2.0, está presente en los teléfonos móviles de nueva generación (a partir de 2004). Esta versión es una reingeniería de WAP que utiliza XHTML-MP (*eXtensible Hypertext Markup Language -Mobile Profile*) como lenguaje de presentación de contenidos, en lugar de WML y mejora el soporte de los gráficos (incluye color).

En cuanto a los protocolos usados (Figura 2.11), en la capa de transporte se usa TCP y en la de aplicación, HTTP. Así pues, WAP 2.0 ha adoptado los protocolos de Internet, facilitando de esa forma la comunicación entre ambos. WAP 2.0 además especifica opciones tanto en TCP como en HTTP para mejorar las prestaciones de dichos protocolos sobre redes de comunicaciones móviles.

Los mecanismos de seguridad usados ya son compatibles con los de Internet por lo que los problemas de seguridad de WAP 1 se resuelven. La pasarela WAP no es estrictamente necesaria en WAP 2.0, pero su presencia puede tener funciones útiles, como caché Web y para dar soporte a las opciones de TCP y HTTP antes mencionadas.

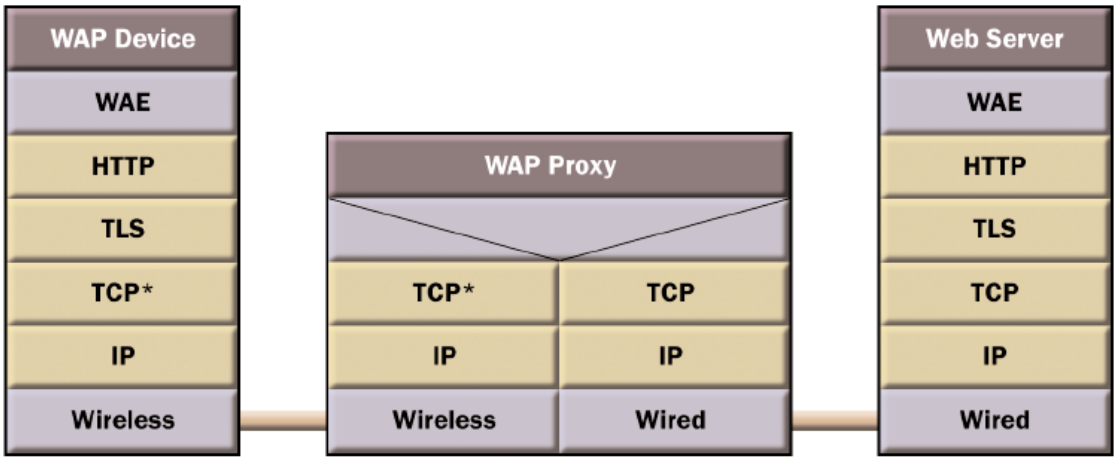

Figura 2.11 Pila de protocolos de WAP 2.0

En cuanto a lenguajes de definición de contenidos y scripts, hay también una gran diferencia entre la primera versión de WAP y esta segunda versión. En la versión 1 de WAP, los contenidos son mostrados al usuario siguiendo el lenguaje de marcas WML en conjunción con un simple lenguaje de script denominado WMLScript. Por su parte, WAP 2 establece un lenguaje de presentación basado en XML, denominado XHTML-MP el cual es un subconjunto de XHTML adaptado a las características más restrictivas de los terminales móviles. En conjunción a este lenguaje de presentación, WAP 2 soporta WCSS (WAP CSS o Wireless CSS) que es una versión reducida del estándar CSS2 (*Cascading Style Sheets versión* 2) comúnmente aplicado junto con HTML en aplicaciones Web. WCSS es usado para definir de forma separada la presentación del contenido de una página XHTML-MP, de esta forma, separando el contenido de la presentación final, puede optimizarse de forma sencilla la misma página XHTML-MP a diferentes dispositivos móviles según sus características.

Finalmente WAP 2 soporta también un lenguaje de script basado en ECMAscript, en este caso denominado ESMP (*EcmaScript Mobile Profile*), el cual es mucho más potente que WMLScript como se verá en posteriores apartados.

2.8.4 WAP 1.X Vs WAP 2.0

En el presente epígrafe se pretende realizar una comparación entre ambas versiones de dicho protocolo, con el fin de conocer las limitaciones y ventajas, que cada uno de ellos ofrece a la hora de realizar aplicaciones móviles sobre WAP.

- WAP 1.X se define principalmente para una comunicación basada en texto, tanto para navegación Web, como para el envío de información hacia Internet. No obstante la capacidad para soportar información gráfica es muy reducida en esta versión (aunque es cierto que los terminales usados en el momento de su definición no soportaban tampoco dichas características gráficas, por lo que no fue un requisito indispensable el realizarlo). WAP 2.0 soporta una mayor capacidad gráfica, tanto en formato de archivos (jpg, gif, gif animado, png, etc.), cantidad de colores a visualizar, así como en capacidad para presentar los contenidos en pantalla, al disponer de WCSS.

- La versión de WAP 2.0 implica un profundo cambio en las características del lenguaje usado para la programación de los contenidos, pero también un profundo cambio en los protocolos usados por WAP para la comunicación entre emisor y receptor.

- En la versión 1 de WAP, definida en 1999, el lenguaje de presentación de contenidos es el WML. En WAP 2.0, dado que se tiende a una arquitectura Web basada en HTML, se utiliza XHTML como lenguaje de presentación de contenidos.

- WAP 1.X define junto con WML un sencillo lenguaje de "scripting", WMLScript, basado en ECMAscript/JavaScript. De la misma forma para WAP 2.0 se dispone de ESMP como lenguaje de “scripting” junto a XHTML, el cual es muy similar a Javascript.

- La pila de protocolos de WAP 1 no es compatible directamente con la de Internet, tal y como puede observarse en la Figura 2.8. Por el contrario, WAP 2.0 presenta una torre de protocolos más acorde a la torre de protocolos de Internet (Figura 2.10).

- La incompatibilidad de la pila de protocolos WAP 1 con la de Internet exige la presencia de un nodo pasarela (WAP Gateway) para hacer de intermediario en la comunicación entre un terminal WAP y un servidor de contenidos WAP residente en Internet. Para WAP 2.0 este nodo pasarela no es obligatorio, pero si recomendable por las funciones extra que puede realizar.

- En WAP 1.X existe un potencial problema de seguridad debido a que WTLS no es muy robusto y además, por no ser compatible con las capas de seguridad usadas en Internet,

en la pasarela WAP los contenidos deben estar en claro. En WAP 2.0 este problema no existe.

- WAP 1 ha sido objeto de fuertes críticas por diversos motivos, que incluyen la pobreza del soporte gráfico donde solo soporta gráficos monocromos WBMP (Wireless BitMaP), las diferencias en las implantaciones de WAP en los terminales de distintos fabricantes, y la escasa seguridad. WAP 2.0 pretende en un esfuerzo que cambia toda su arquitectura solventar todos esos problemas, ofreciendo una mejor más positiva comunicación entre cliente móvil e Internet.

Una breve comparación de ambas versiones WAP se muestra en la tabla 2.1.

	WAP 1.X	**WAP 2.0**
Lenguaje de marcas	WML	XHTML
Protocolo de transferencia	WTP	HTTP
Seguridad	NO	SI (end-to-end)
WAP Push	SOLO TEXTO	SI
Script	WMLScript	ESMP
Presentación	MONOCAPA	WCSS
Soporte PNG, JPEG, GIF	NO	SI

Tabla 2.1 Comparación entre WAP 1.X y WAP 2.0

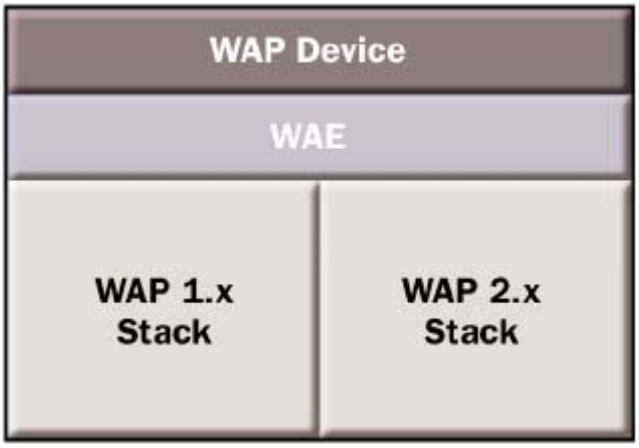

Figura 2.12 Separación de las torres de protocolos WAP en un dispositivo móvil

No obstante, WAP 2.0 es totalmente compatible con WAP 1.x, lo que indica que un terminal que soporte WAP 2.0 puede también ser capaz de visualizar contenidos sobre WAP 1.x. Es por ello que amabas torres de protocolos deben ser capaces de coexistir en un

mismo terminal móvil, trabajando para ello de forma independiente. Esto puede entenderse mejor en la figura 2.12 en la que se representan ambas torres de protocolos y la forma en la que coexisten.

2.9 WML

2.9.1 INTRODUCCIÓN

El *Wireless Markup Language* es un lenguaje cuyo origen está basado en XML. Este lenguaje es usado para construir aplicaciones destinadas a ser visualizadas en los dispositivos móviles como son teléfonos móviles y los asistentes personales digitales (PDA) dotados de tecnología WAP. Concretamente este lenguaje está definido en la primera versión de WAP (WAP 1.x).

WML es un metalenguaje, lo que implica que además de usar etiquetas predefinidas se pueden crear componentes propios y tiene ciertas similitudes con otro lenguaje de etiquetas bastante conocido, el HTML, utilizado para la creación de páginas Web convencionales. Un consorcio formado por Nokia, Phone.com, Motorola y Ericsson, el WAP Forum, definió la sintaxis, variables y elementos utilizados en WML. Algunos fabricantes han desarrollado capacidades adicionales a este estándar.

Al igual que el HTML se sirve de un lenguaje de script como Javascript para dotar de cierto dinamismo a sus documentos, WML dispone del WMLScript que es un lenguaje bastante similar al Javascript, pero con alguna diferencia fundamental que se verá con más detalle en siguientes apartados.

Dado que cada fabricante puede añadir nuevas etiquetas a este lenguaje, la visualización de la página dependerá del dispositivo que se use y de la forma en que este interprete el código, ya que varían entre si. Igualmente no todos los dispositivos que soporten WML soportan forzosamente WMLScript.

2.9.2 WML A GRANDES RASGOS

2.9.2.1 Características principales

Las características principales de WML son:

1. Soporte para imágenes y texto, con posibilidad de texto con formato. No obstante como ya se comentó en el apartado 2.8.2 dedicado a WAP 1.X, aunque

se tiene soporte para imágenes, éstas son de limitada calidad, y formato. De la misma forma pasa con el texto con formato, en el que se pueden aplicar una muy pequeña variedad de formatos de texto.

2. Una pagina WML es como una página HTML con la diferencia de que en WML esas "páginas" contienen una serie de cartas (*card*). Cada carta representa una pantalla en el dispositivo móvil que la visualiza. Al conjunto de estas cartas se les suele llamar baraja (*deck*). Así pues un usuario que haga una petición a una página WML, descargará una baraja compuesta por una o más cartas. En cada momento solo podrá visualizar una de esas cartas.
3. Posibilidad de navegar entre cartas y barajas de la misma forma que se navega entre páginas Web. El navegar entre barajas corresponde a la típica navegación entre páginas usada en HTML, por lo que cada nueva baraja se corresponde a una nueva página WML que debe ser requerida al servidor y descargada en el dispositivo móvil. Por el contrario navegar entre cartas no requiere la descarga de nuevas páginas WML, sino que internamente en la página WML descargada podemos movernos de una carta a otra.
4. Manejo de variables y formularios para el intercambio de información entre el teléfono celular y el servidor. Una peculiaridad muy importante de WML respecto a otros lenguajes de etiquetas como XML, HTML, XHTML, etc. es que en WML podemos usar variables. Esas variables son propias de cada carta o baraja (según su definición) y pueden ser accedidas posteriormente por WMLScript como se verá más adelante, para ayudar a enviar la información de los formularios o de otras variables que se deseen crear.

2.9.2.2 Sintaxis WML

WML es un lenguaje de marcas comprendido dentro del estándar XML, esto conlleva que WML debe cumplir con la sintaxis de XML. A continuación se describen brevemente los rasgos más importantes de esta sintaxis.

- Sensible a mayúsculas/minúsculas. Todos los elementos de WML son sensibles a mayúsculas/minúsculas, esto incluye las etiquetas, los atributos, los identificadores, las variables...
- Conjunto de caracteres. El conjunto de caracteres definido por defecto es el ISO/IEC-10646 que es el mismo que el Unicode 2.0. WAP soporta los

siguientes subconjuntos de Unicode:(UTF-8, ISO-8859-1 o ISO Latin-1, UCS-2). Se definen en la etiqueta:

<?xml version="1.0" encoding="UTF-8"?>

- Etiquetas. Todas las etiquetas en WML se escriben en minúsculas. Hay dos tipos de etiquetas, las que contienen elementos, para lo cual hay una etiqueta de inicio y otra de fin. Los atributos de las etiquetas han de ir siempre en la etiqueta de inicio.

  <etiqueta atributo=”valor”> Otros elementos </etiqueta>

  Y las etiquetas que no contienen elementos en su interior, las cuales tienen el siguiente formato:

  <etiqueta/>

- Comentarios. Los comentarios tienen el siguiente formato:

  <!-- Comentario -->

2.9.2.3 Barajas y cartas

A las páginas WML se les suele llamar barajas porque están compuestas por cartas. Una carta es la unidad de información que un navegador WAP puede mostrar. El navegador nos permite pasar de una carta a otra dentro de la baraja para así poder acceder a todas las cartas.

- Baraja

  Una baraja de cartas se marca con las etiquetas <wml> ...</wml>. Dentro de estas marcas irán todas las cartas de la baraja. Puede contener las etiquetas “head”, “template” y es obligatorio que al menos tenga una etiqueta “card”, esto es, cada baraja puede estar compuesta por una o más cartas, pero como poco una.

- Carta

  Una carta es la unidad de información que se muestra en un navegador WAP, una carta puede contener texto, campos de datos, enlaces, imágenes, formularios, etc. La etiqueta es <card> ... </card> y algunos de los atributos son “title” que nos permite indicar el título de la carta, “id” que nos proporciona una

manera de identificar la carta. El atributo “id” es común para todos las etiquetas WML y nos permite identificar un elemento dentro de un documento WML.

A continuación se muestra un ejemplo de una baraja con dos cartas.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="t1" title="Tarjeta 1">
    - <p>Hola mundo !</p>
</card>
<card id="t2" title="Otra tarjeta">
    - <p>Bienvenido</p>
</card>
</wml>
```

2.9.2.4 Texto y formato

El texto en WML debe ir obligatoriamente entre las etiquetas ... ya que así está definido en el DTD (*Document Type Definition*) de WML. Adicionalmente podemos incluir saltos de línea con la etiqueta <br/>.

"		Comillas
&		Ampersand
'		Apóstrofe
<		Menor que
>		Mayor que
		Espacio fijo

Tabla 2.2 Caracteres restringidos de WML

La etiqueta puede llevar los siguientes atributos:

- align="" Puede contener los valores “left”, “right” y “center”. Indica la alineación del texto al mostrarse en el navegador WAP. Por defecto es “left”.
- mode="" Puede ser “warp” o “nowrap”. “warp” significa que el texto puede ir en varias líneas y “nowrap” quiere decir que el texto no puede ser roto en varias líneas. Por defecto es “warp”.

También hay que reseñar que aunque indiquemos el conjunto de caracteres que vamos a emplear, existen algunos que no pueden escribirse directamente, debido a que son usados en la definición del lenguaje, como son los mostrados en la tabla 2.2.

Aunque los navegadores WAP estén muy limitados en lo referente al apartado visual, se pueden aplicar algunos efectos en el texto (formato de texto) para proporcionar una mejor visualización del contenido al usuario WAP. En la tabla 2.3 se muestran las etiquetas para dar formato al texto en WML 1.X:

Etiqueta apertura	**Etiqueta cierre**	**Significado**
		Enfatizado
		Fuerte
		Cursiva
		Negrita
<u>	</u>	Subrayado
<big>	</big>	Grande
<small>	</small>	Pequeño

Tabla 2.3 Etiquetas para formato de texto en WML

2.9.2.5 Eventos y tareas

WML incluye en su definición un modelo de manejo de eventos. Estos eventos pueden ser ligados a tareas específicas a realizar cuando el evento se produce. Las tareas a realizar cuando un evento se produce son diversas y van desde navegar por el historial de páginas visitadas a ir a una nueva URL especificada previamente en el código.

Los eventos y las tareas nos proporcionan un mecanismo para realizar acciones sobre las cartas, permitiéndonos navegar entre cartas y construir pequeños interfaces para dar más funcionalidad a las páginas.

Para capturar los eventos, WML dispone del elemento <onevent> el cual liga una tarea a un evento especificado por su atributo “type”. No obstante los eventos a poder capturar están predefinidos, y solo estos eventos son tratados por <onevent>. Los eventos soportados en WML son mostrados en la tabla 2.4.

Evento	Descripción
Ontimer	Evento producido tras cumplirse un tiempo establecido (timer). Un timer puede ser configurado usando el elemento <timer>
onenterforward	Evento producirse al entrar en una carta proveniente de otra carta.
onenterbackward	Evento producido al entrar en una carta, tras usar la tarea <prev/>. Es decir al entrar en la carta navegando hacia atrás en el historial
onpick	Evento producido cuando el usuario selecciona un elemento.

Tabla 2.4 Eventos en WML

Como se ha mencionado, los eventos son una poderosa herramienta a usar para ejecutar tareas, pero existen también otras formas de ejecutar las tareas, pero de entre todas ellas, posiblemente la más importante es el elemento <do>, ya que nos permite asignar una tarea sobre una acción.

El elemento <do> puede aparecer tanto a nivel de baraja como de carta:

- A nivel de carta: el elemento <do> puede aparecer dentro de un elemento <card> y puede ser localizado en cualquier lugar dentro del flujo de texto.
- A nivel de baraja: el elemento <do> puede aparecer dentro de un elemento de <template>, indicando que esa acción será aplicada a todas las cartas de la presenta baraja, lo que equivaldría a definir ese mismo elemento <do> en cada una de las cartas.

Los atributos más importantes que contiene la etiqueta <do> son: type="", label="" y name="".

- type="". indica sobre qué botón del navegador se aplica la acción, las más comunes son "accept", "prev" y "help".
- label="". Texto que aparece asociado a la acción.
- name="". Nombre de la acción, es imprescindible si vamos a asignar más de una acción a un mismo tipo.

Un ejemplo del uso del elemento <do> se muestra a continuación.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>

<card id="primera" title="Eventos">
    <do type="accept" label="dos" name="dos">
        <go href="#segunda"/>
    </do>

    <do type="accept" label="tres" name="tres">
        <go href="#tercera"/>
    </do>

    <do type="accept" label="nada" name="tres">
        <noop/>
    </do>

    <p>Primera tarjeta</p>
</card>

<card id="segunda">
<p>Segunda tarjeta</p>
</card>

<card id="tercera">
<p>Tercera tarjeta</p>
</card>
</wml>
```

El contenido de la etiqueta <do> es la tarea que se realizará al seleccionar esa acción. La etiqueta <do> puede contener una de las siguientes tareas, <go>, <prev>, <noop>, <refresh> que indican la acción que se realiza sobre la etiqueta.

1. <go href=""/> Permite ir a la dirección indicada en el atributo href.
2. <prev/> Permite ir a la carta anterior, en la historia del navegador.
3. <noop/> Es una acción que no realiza nada.
4. <refresh>...</refresh> Refresca el contenido de la carta actual, volviéndola a pedir al servidor.

2.9.2.6 Enlaces

En WML se puede al igual que en lenguajes similares, seleccionar parte de un texto o una imagen y que éste sea un enlace a otra página, carta, o que realice una tarea.

Para realizar ésto existen dos etiquetas <anchor>...</anchor> y <a>.... La primera de ellas es más completa, mientras que la etiqueta <a> es más sencilla de usar para casos simples

- La etiqueta <a>... permite seleccionar un texto o imagen y hacer que sea un enlace a otra página, de una manera muy similar a la etiqueta <a>... de HTML. Contiene el atributo “href” en el cual se especifica la URL a donde se dirigirá el enlace. Para moverse a una carta dentro de la propia baraja, tan solo hay que indicar el nombre de la carta deseada precedido del carácter “#”.

  Ir a tarjeta 2

- La etiqueta </anchor>...<anchor> permite además de seguir una URL, indicar que se realice una acción al activar el enlace. Las acciones que se pueden incluir son:

  <go href=""/> Permite ir a la dirección indicada en el atributo “href”.

  <prev/> Permite ir a la tarjeta anterior, en la historia del navegador.

  <refresh>...</refresh> Refresca el contenido de la tarjeta actual, volviéndola a pedir al servidor.

  <setvar name="X" value="Y"/> Crea una variable llamada X con el valor inicial Y.

2.9.2.7 Tipos MIME

Para que el servidor de HTTP sepa que debe servir las páginas *.wml como páginas WAP, hay que informar al servidor de ello, de lo contrario el servidor responderá como que el tipo de fichero es de texto, proporcionando un error en el navegador WAP.

Contenido	Tipo MIME	Extensión
WML	text/vnd.wap.wml	wml
WML compilado	Application/vnd.wap.wmlc	wmlc
WMLScript	text/vnd.wap.wmlscript	wmls
WMLScript compilado	Application/vnd.wap.wmlscript c	wmlsc
Imagen Wireless Bitmap	image/vnd.wap.wbmp	wbmp

Tabla 2.5 Tipos MIME para WAP 1.X

Para indicar al servidor que las páginas de extensión wml son páginas WAP existen los tipos MIME (*Multipurpose Internet Mail Extensions*). Con estos tipos se indica al servidor cómo se debe comunicar con el cliente cuando le solicitan una página wml.

La configuración de tipos MIME para WAP es la mostrada en la tabla 2.5.

2.10 WMLSCRIPT

2.10.1 INTRODUCCIÓN

WMLScript es parte de la capa de aplicación del protocolo WAP, y puede ser usado para añadir procedimientos lógicos en el lado del cliente a las cartas y barajas establecidas en WML. Esto es, poder añadir cierta inteligencia a nuestra aplicación en el ámbito del dispositivo móvil del cliente, no siendo necesario que sea el servidor quién realice esos procedimientos lógicos.

WMLScript está basado en ECMAScript, pero ha sido modificado para adaptarse a las características y requisitos de los dispositivos móviles, tal como es el bajo ancho de banda del que disponen éstos terminales en comparación con el de redes cableadas.

Este lenguaje puede ser usado para dotar de cierta inteligencia a los clientes que navegan por páginas WML, comportándose como un gran complemento en formularios, enlaces, etc. o siendo usado por si solo para realizar las acciones deseadas por el programador, sin basarse en WML.

WMLScript posee una sintaxis y una estructura bien definidas, además todos los datos pertenecientes a este lenguaje son transmitidos por la red inalámbrica en un formato intermedio denominado “bytecode” resultado de la compilación del fichero fuente, consiguiendo un mejor aprovechamiento de los canales radio y un uso optimizado y reducido de memoria del dispositivo móvil.

2.10.2 BENEFICIOS Y CARACTERÍSTICAS

De las dos formas de uso que posee WMLScript, el uso junto con WML es la más importante y usada de ambas. El uso de este lenguaje de script junto con el lenguaje de etiquetas WML, forma una pareja que da una mayor capacidad de personalización y sensación de dinamismo en todas las páginas que el usuario pueda visualizar. De esta forma el programador puede aportar a WML una gran variedad de alternativas que este lenguaje no ofrece por si solo, como pueden ser:

- Comprobar y validar formularios.

- Tener acceso a funcionalidades del dispositivo móvil, como pueden ser realizar llamadas, enviar mensajes, añadir un número a la agenda, etc.
- Generar diálogos internos al dispositivo, como el caso de alertas, diálogos de confirmación, errores, etc.

WMLScript está diseñado para poder aportar estas funcionalidades y otras muchas más usando para ello un entorno programable que se adapta de forma eficiente a entornos de bajo ancho de banda, dado que toda esta funcionalidad podría ser igualmente conseguida en el lado del servidor (sin limitaciones de capacidad de procesamiento, ni de memoria disponible), pero con el inconveniente de un mayor uso del medio inalámbrico, lo que ocasionaría una mayor dependencia del mismo y un mayor coste económico y posiblemente temporal.

Como se ha mencionado anteriormente, WMLScript puede considerarse una versión reducida y adaptada del estándar ECMAScript. No obstante, WMLScript posees una serie de peculiaridades o características que lo hacen diferente del resto de lenguajes de la familia ECMAScript:

- WMLScript no está embebido en las páginas WML tal y como suele hacerlo el más conocido lenguaje de script (Javascript) en HTML. WMLScript solo soporta ser ejecutado estando en un archivo separado, al que se debe hacer referencia desde la página WML.
- WMLScript no es enviado al cliente en formato texto, sino que previamente a su envío es compilado en el servidor y enviado al cliente en formato “bytecode”. De esto se deduce el porque no puede ser introducido el código WMLScript dentro de una página WML.
- WMLScript es un lenguaje débilmente tipificado y que no acepta objetos.
- Un fichero con código fuente WMLScript tendrá extensión **.**wmls, mientras que una vez compilado llevará extensión **.**wmlsc.
- Al programar en WMLScript se hace un uso intensivo de una serie de bibliotecas estándar definidas para este lenguaje. Estas bibliotecas agilizan el trabajo de los programadores dado que están incorporadas en el traductor de *bytecode* instalado el teléfono y son directamente ejecutables por él.

2.10.3 LLAMADAS A FUNCIONES

Es imprescindible conocer a la perfección el modo en que desde WMLScript se hace una llamada a una función, dado que es la única forma en la que podemos hacer uso de este lenguaje desde una página WML.

Al revés que en Javascript, donde se puede poner código en cualquier sitio de la página Web para que se ejecute en cuanto sea leído por el Navegador, en WMLScript se ha de poner todo el código del script dentro de funciones en el archivo.wmls, y para ejecutar ese código se han de hacer llamadas a dichas funciones.

La forma en que se realiza una llamada a una función depende de si la función pertenece a una biblioteca estándar de WMLScript o si pertenece a una biblioteca creada por el usuario o programador.

a. Si el código que se desea ejecutar pertenece a una biblioteca estándar, la llamada a dicha función tendrá la siguiente forma;

   nombreDeLaBiblioteca.nombreDeLaFuncion(parámetros)

   Se ha de tener en cuenta que el punto (.) es el separador del nombre de la biblioteca y el nombre de la función que se desea ejecutar.

b. Sin embargo, si la función pertenece a una biblioteca creada por el usuario, la llamada se hará de la siguiente forma:

   URLdelFichero.wmls#nombreDeLaFuncion(parámetros)

   En este caso, es el carácter almohadilla (#) el que se usa como separador entre el nombre del archivo y el de la fundón.

Las llamadas a funciones en WMLScript se pueden realizar en cualquier sitio en el que podamos colocar una dirección URL. Por ejemplo, se puede hacer que el usuario llame a una función en WMLScript desde cualquier punto de una página WML usando un enlace:

```
<a href="biblioteca.wmls#funcion(parametros)">Pincha aquí </a>
```

O bien, añadiendo una opción al menú de opciones del teléfono con la etiqueta <do> y dentro de ella una etiqueta <go>:

```
<do type="accept" label="Enviar">
    <go href="biblioteca.wmls#mi_alert('Saludos')"/>
</do>
```

O, por último, la acción de ejecutar una función WMLScript puede ser el resultado de que se lance un determinado evento.

2.10.4 SINTAXIS

Al igual que Javascript, WMLScript toma su sintaxis de los lenguajes de programación más conocidos, fundamentalmente C, aunque también algo de Pascal.

- Se pueden añadir comentarios con; // y /* ... */
- Se distingue entre mayúsculas y minúsculas.
- Los nombres de las variables siguen el formato habitual de los lenguajes de programación: empiezan por letra o por "_", y sigue con más letras, o números.
- No soporta objetos.
- No soporta arrays o matrices.
- Aunque es débilmente tipificado (el tipo de una variable puede cambiar fácilmente), es obligatorio declararlas de la siguiente forma:

  var nombreDeLaVariable=valor;

  Aunque la asignación de un valor inicial es opcional. Como se ve, no hay que indicar en ningún momento el tipo de las variables.
- El ámbito de validez de una variable es únicamente la función en la que se ha declarado.
- Se pueden usar variables globales definiendo variables de entorno para la página WML. Estas variables de entorno pueden ser accedidas desde WMLScript mediante la biblioteca estándar WMLBrowser.
- El paso de parámetros a las funciones es siempre por valor; no existe paso por variable o referencia.
- Se pueden usar cinco tipos de datos, que son: Integer, Float, String, Boolean, Invalid.
- Es obligatorio terminar todas las sentencias con punto y coma (;).

Respecto a la sintaxis de las sentencias usadas en este lenguaje, puede considerarse muy similar a la usada en Javascript. Esta sintaxis puede organizarse en varios apartados según el tipo de sentencia:

1. Secuenciales.

La operación de asignación se realiza de la misma forma que en C, es decir:

variable = expresión;

WMLScript no posee una sentencia de escritura en pantalla como suele ser común en este tipo de lenguaje de script, no obstante esta característica debe ser simulada mediante

la introducción de variables de entorno en las cartas WML. Esto es, para cambiar el contenido de la pantalla del dispositivo móvil, crearemos desde WML una variable vacía y la escribiremos en pantalla. Luego se accederá a esa variable y se modificará desde una función WMLScript que tendrá en alguna parte de su código instrucciones como las siguientes:

```
WMLBrowser.setVar( 'nombre', 'valor' ); WMLBrowser.refresh(); ...
```

Es la sentencia “WMLBrowser.refresh();” la que provoca que se actualice la pantalla del navegador WAP del dispositivo móvil, generando la reescritura de la carta visualizada en ese momento, y haciendo que se vea el nuevo valor de la variable deseada.

Por su parte, también la sentencia de lectura es simulada con etiquetas del tipo “input” pertenecientes a WML, o mediante llamadas a la función Dialogs.prompt(), la cual permite pedir un dato al usuario e introducirlo en una variable WMLScript.

2. Condicionales.

WMLScript incorpora la sentencia condicional por excelencia de cualquier lenguaje: IF... THEN... ELSE..., usando el formato establecido en lenguaje C.

```
if ( condición ) { sentencias true; } else { sentencias false; }
```

3. Iterativas.

Al igual que en el apartado anterior, WMLScript incorpora las típicas estructuras iterativas de todo lenguaje de programación, a saber:

```
while( condición ) { sentencias; }
do { sentencias; } while ( condición; )
for( inicialización; condición; incremento ) { sentencias; }
```

4. Incrementos y operaciones con variables.

Como ocurre con C y otros lenguajes, WMLScript aporta formas rápidas de incrementar y decrementar el valor de las variables, así como de hacer asignaciones selectivas:

```
variable++; ++variable; variable--; --variable;
variable+=valor; variable-=valor; variable*=valor; variable/=valor;
variable=(condicion)?valor_true:valor_false;
```

2.10.5 DECLARACIÓN DE FUNCIONES

Antes de especificar cómo se declaran funciones en WMLSCript, se han de tener en cuenta una sería detalles:

- No se pueden declarar funciones anidadas. Esto es, una función no puede ser declarada dentro de otra función.
- El nombre de cada función debe ser único dentro de cada fichero wmls.
- Todos los parámetros son pasados por valor a la función.
- En las llamadas a una función, se le deben pasar el mismo número de valores como parámetros tenga especificados en su declaración.
- Todas las funciones devuelven un valor, que por defecto es la cadena vacía, "".

La declaración de funciones se realiza de la siguiente forma:

```
[extern] function nombre( [parámetros] ) {
    sentencias;
}
```

Donde las palabras entre corchetes ([ ]) son de uso opcional. Así la palabra [extern] indica si la función que se declara puede ser llamada desde el exterior (fuera del propio archivo wmls donde se define esta función) de la biblioteca o no. Como se ve, es necesario el uso de esta palabra reservada para que la función declarada pueda ser llamada desde fuera, como es el caso de ser llamada desde una página WML, u otro archivo de WMLScript. Por defecto no se permite.

Todas las funciones devuelven un valor que, por defecto es la cadena vacía. No obstante esto puede ser modificado mediante la sentencia:

```
return expresión;
```

Que permite devolver cualquier tipo de valor y/o variable.

Como ya se dijo, la forma de realizar una llamada a una función dependerá de dónde esté declarada dicha función. Así, se distinguen llamadas a funciones locales, externas o de la biblioteca estándar.

Para realizar llamadas a funciones locales (es decir, definidas en la misma biblioteca desde donde se realiza la llamada), solo hemos de poner el nombre de la función y los parámetros necesarios.

Para las llamadas a funciones externas (definidas en otra biblioteca), hay que proporcionar un alias a la biblioteca externa y posteriormente usar el carácter almohadilla (#) para llamar a la función. Este es el caso de una llamada desde una página WML a una función de WMLScript, como se vio en el apartado 2.10.3.

Por último, para la llamada a funciones de bibliotecas estándar se debe usar el nombre de la biblioteca, un punto y el nombre de la función deseada.

2.10.6 BIBLIOTECAS ESTÁNDAR

Con la finalidad de agilizar la tarea del programador y de reducir la cantidad de código a enviar al dispositivo móvil cada vez que se ejecuta un programa en WMLScript, se han incorporado en éstos dispositivos una serie de bibliotecas estándar de WMLScript, de cuyas funciones se puede hacer uso.

Función	Descripción
getVar(nombreDeVariable)	Devuelve el valor de la variable WML nombreDevariable. Si dicha variable no existiera, devuelve la cadena vacía.
setVar(nombreDeVariable, valor)	Asigna a la variable WML el valor especificado. Devuelve "true" si la asignación es satisfactoria, "false" en otro caso.
Go(url)	Carga en el dispositivo móvil la página indicada en el parámetro "url". Si "url" es la cadena vacía, no hace nada. Devuelve cadena vacía.
Prev( )	Carga en el dispositivo móvil la página anteriormente visitada a la actual, atendiendo al historial del navegador WAP. Devuelve cadena vacía.
newContext( )	Reinicia el contexto de la página WML activa. El resultado de dicha sentencia es reiniciar el valor de todas las variables WML activas en el navegador.
getCurrentCard( )	Devuelve el URL relativo más pequeño especificando la carta (si es que la hay) que está siendo procesada en ese instante por el teléfono.
refresh( )	Actualiza el contexto del teléfono haciendo que muestre de nuevo la carta que está procesando. Es especialmente útil para mostrar en pantalla el valor de las variables "globales" una vez han sido calculadas.

Tabla 2.6 Funciones de la librería estándar WMLBrowser.

1. WMLBrowser

La biblioteca WMLBrowser es usada para poder acceder al contexto de la página WML, esto es, a sus variables. Las funciones que forman parte de esta biblioteca devolverán el valor "invalid", sólo si el teléfono no soporta llamadas a esta biblioteca. Algunas de las funciones más importantes de esta librería se pueden encontrar en la tabla 2.6.

2. Lang

Esta biblioteca, Lang, contiene un conjunto de funciones directamente relacionadas con la especificación del lenguaje WMLScript. Algunas se muestran en la tabla 2.7.

Función	Descripción
abs(valornúmero)	Devuelve el valor absoluto del número que se le pasa.
min(valor_1número, valor_2número) max(valor_1número, valor_2número)	Devuelve el valor menor/mayor de los dos que se le pasan, respectivamente.
parseInt(valorstring) parseFloat(valorstring)	Devuelve el valor entero o real correspondiente a la cadena que se le ha pasado, parando en el momento en que encuentra cualquier carácter distinto de los dígitos 0 al 9.
isInt(valorcualquier tipo) isFloat(valorcualquier tipo)	Devuelve "true" si el valor pasado como parámetro puede convertirse a entero/real usando parseInt/parseFloat
exit (valorcualquier tipo)	Finaliza la interpretación del código en WMLScript devolviendo el control a la entidad que llamó a la función. Devuelve el valor que se le pasa como parámetro.
abort(valorstring)	Finaliza la interpretación del código en WMLScript devolviendo el control a la entidad que llamó a la función junto a una descripción del error.
random(valornúmero)	Devuelve un número entero pseudo-aleatorio entre 0 y el valor que se le pasa (ambos incluidos).

Tabla 2.7 Funciones de la librería estándar Lang.

3. Float

La biblioteca Float contiene las funciones que más se utilizan a la hora de operar con números reales. Proporciona las típicas funciones de redondeo, raíz, exponencial, etc.

4. String

La biblioteca estándar String aglutina funciones para el tratamiento de cadenas, aunque también puede ser usada para implementar matrices, ya que WMLScript no las soporta de forma nativa.

5. Url

La biblioteca URL contiene funciones que manejan direcciones URL tanto absolutas como relativas.

6. Dialogs

Dialogs es una biblioteca con tres funciones cuyos nombres y funcionalidad serán muy conocidos por los programadores de JavaScript, dada la gran importancia de estas funciones y el amplio uso que se hace en páginas Web. Estas funciones pueden verse en la tabla 2.8.

Función	Descripción
alert(mensajestring)	Muestra el mensaje al usuario y espera hasta que pulsa OK. Devuelve la cadena vacía
confirm(mensajestring, aceptarstring, cancelarstring)	Muestra al usuario un mensaje junto a las alternativas aceptar y cancelar, y espera a que éste elija una de ellas. Devuelve true si el usuario elige la alternativa aceptar, false si elige cancelar.
prompt(mensajestring, valorPorDefectostring)	Muestra al usuario un mensaje y le permite introducir un valor, dándole también un valor por defecto. Devuelve una cadena con el valor introducido por el usuario.

Tabla 2.8 Funciones de la librería estándar Dialogs.

2.11 XHTML

2.11.1 INTRODUCCIÓN

XHTML, acrónimo inglés de *eXtensible Hypertext Markup Langua*ge (lenguaje extensible de marcado de hipertexto), es el lenguaje de marcado pensado para sustituir a HTML como estándar para las páginas Web. XHTML es la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple las especificaciones, más estrictas, de XML. Su objetivo es avanzar en el proyecto del W3C (*World Wide Web Consortiu*m) de lograr una Web semántica, donde la información, y la forma de presentarla estén claramente separadas. En este sentido, XHTML serviría únicamente para transmitir la información que contiene un documento, dejando para hojas de estilo como las CSS y JavaScript su aspecto y diseño en distintos medios (ordenadores, PDAs, teléfonos móviles, impresoras...).

2.11.2 VISTA GENERAL

XHTML es el sucesor de HTML. Es por eso que muchos lo consideran la “versión actual” del HTML, pero es una recomendación aparte y a la vez paralela; la W3C continúa recomendando el uso de XHTML 1.1, XHTML 1.0, y HTML 4.01 para publicar en la Web.

La necesidad de una versión más estricta de HTML se sintió principalmente porque el contenido de la World Wide Web ahora puede visualizarse desde numerosos dispositivos (como móviles) aparte de los ordenadores tradicionales, donde no pueden dedicarse recursos suplementarios para afrontar la complejidad añadida de la sintaxis del HTML.

La mayoría de las versiones recientes de los navegadores Web más populares soportan XHTML adecuadamente, pero algunas versiones más antiguas solo pueden leer el XHTML como si se tratara de HTML. Asimismo casi todos los navegadores que son compatibles con XHTML también leen HTML correctamente. Algunos argumentan que esta compatibilidad ralentiza el cambio de HTML a XHTML.

Una característica especialmente útil del XHTML es que se pueden incorporar elementos de distintos espacios de nombres XML (como *MathML* y *Scalable Vector Graphics*), cosa que no es posible en HTML.

Los cambios de HTML a la primera generación de XHTML (es decir, XHTML 1.x) son menores ya que, principalmente, están destinados a conseguir la conformidad con XML. El cambio más importante es el requisito de que el documento esté bien formado y que todas las etiquetas estén explícitamente cerradas, como se requiere en XML. Como las etiquetas en XML distinguen entre mayúsculas y minúsculas (case-sensitive), la recomendación XHTML ha definido todos los nombres de etiqueta en minúsculas. Lo cual está en contraste directo con las tradiciones extendidas en el tiempo de HTML 2.0, cuando mucha gente prefería las mayúsculas para éstos casos, normalmente para mostrar más fácilmente el contraste de éstas con el contenido.

En XHTML, los valores de los atributos deben encerrarse entre comillas (siempre comillas "dobles"). Al contrario ocurría en SGML y por consecuencia en HTML, donde las comillas podían omitirse en algunos casos.

Todos los elementos han de ser explícitamente cerrados, incluyendo elementos vacíos como pueden ser img o br. Esto puede hacerse añadiendo una barra a la etiqueta: <img.. /> y <br />. La minimización de atributos (como <option selected>) está también prohibida, ya que el atributo “selected” no contiene ningún valor explícito; en lugar de eso, se usa

<option selected="selected">. Hay más diferencias detalladas en la recomendación XHTML 1.0 de la W3C.

2.11.3 VENTAJAS

Las principales ventajas del XHTML sobre otros formatos son:

- Compatibilidad parcial con navegadores antiguos: la información se visualiza, aunque sin formato. Cabe apuntar que el XHTML 1.0 fue diseñado expresamente para ser mostrado en navegadores que soportan HTML de base.
- Un mismo documento puede adoptar diseños radicalmente distintos en diferentes dispositivos, pudiendo incluso escogerse entre varios diseños para un mismo medio.
- Facilidad de edición directa del código y de mantenimiento.
- Formato abierto, compatible con los nuevos estándares que actualmente está desarrollando el W3C como recomendación para futuros agentes de usuario o navegadores.
- Los documentos escritos conforme a XHTML 1.0 pueden potencialmente presentar mejor rendimiento en las actuales herramientas Web que aquellos escritos conforme a HTML.

2.11.4 REGLAS SINTÁCTICAS

La siguiente lista muestra algunas reglas de XHTML 1.0 que lo diferencian de HTML 4.01. Muchas de estas diferencias vienen con el cambio de ser una aplicación SGML a ser una aplicación del más estricto XML:

A. Los elementos vacíos deben cerrarse siempre.

   Incorrecto:

   Correcto:
</br> o <br/> o <br />.

B. Los elementos no vacíos también deben cerrarse siempre.

   Incorrecto: Primer párrafo Segundo párrafo.

   Correcto es: Primer párrafo Segundo párrafo

C. Los elementos anidados deben tener un correcto orden de apertura/cierre (el que se abre último, debe cerrarse primero).

   Incorrecto: Texto

   Correcto: Texto

D. Se debe siempre especificar un texto alternativo para imágenes (atributo "alt" para la etiqueta "img"), que será mostrado cuando no se pueda visualizar la imagen especificada.

   Incorrecto:

   Correcto:

E. No se deben insertar elementos de bloque dentro de elementos de línea.

   Incorrecto: <h2>Título</h2>

   Correcto: <h2>Título</h2>

F. Los valores de los atributos deben siempre ir encerrados entre comillas (simples o dobles).

   Incorrecto: <td rowspan=3>

   Correcto: <td rowspan="3">

   Correcto: <td rowspan='3'>

G. Los nombres de elementos y atributos deben ir en minúsculas.

   Incorrecto: <A HREF="http://www.domname.com">Domname</A>

   Correcto: Domname

H. No está permitida la minimización de atributos (se usa el nombre del atributo como valor).

   Incorrecto: <textarea readonly>Solo-lectura</textarea>

   Correcto: <textarea readonly="readonly">Solo-lectura</textarea>

I. Los atributos desaprobados en HTML 4.01 no forman parte de XHTML.

   Incorrecto: Blue text

   Correcto: Blue text

J. El texto no debe ser insertado directamente en el cuerpo (dentro de la etiqueta "body"), sino que debe ser insertado usando otra etiqueta de texto permitida.

   Incorrecto: <body>Texto plano</body>

   Correcto: <body>Texto plano</body>

2.11.5 XHTML-MP

XHTML MP es el lenguaje de etiquetas definido en WAP 2.0, que es la especificación más reciente para servicios móviles creada por el WAP Forum (actualmente Open Mobile Alliance). Junto a este lenguaje de etiquetas, y dado que lo que se pretende es separar el contenido de la presentación, en WAP 2.0 se define conjuntamente las WAP CSS o

también conocidas como WCSS. WCSS es usado junto con XHTML-MP para cambiar de forma sencilla la presentación de las páginas generadas.

XHTML-MP es un subconjunto de XHTML, el cual es la versión estricta de HTML tal y como se ha visto en el punto anterior. Se puede considerar que XHTML-MP está formado por XHTML Basic (un subconjunto de XHTML formado por las características más básicas de XHTML), más una serie de elementos adicionales y atributos de la versión completa de XHTML para poder completar el rango de características de los dispositivos móviles.

La meta de XHTML-MP es unificar las tecnologías empleadas en la navegación móvil por Internet, con las tecnologías usadas en la World Wide Web. Antes de la aparición de este lenguaje, los desarrolladores de WAP hacían uso de WML y WMLScript para crear contenidos y páginas WAP, mientras que los desarrolladores de páginas Web, hacían uso de HTML/XHTML y CSS.

Con el anuncio de XHTML-MP, los lenguajes de etiquetas tanto de la Web como del mundo inalámbrico, convergieron finalmente. XHTML-MP junto con WCSS aportan un mayor control y mejoras en la presentación de los contenidos, que junto con la enorme ventaja que supone el poder usar la misma tecnología para desarrollar la versión Web como la versión inalámbrica, hacen de XHTML-MP un candidato perfecto.

2.11.5.1 Cambios respecto a WML

El cambio producido al pasar de WML en WAP 1.X a XHTML-MP en WAP 2.0, ha ocasionado una serie de modificaciones a la hora de programar las páginas para los dispositivos móviles. Aunque XHTML-MP ofrece una mayor separación entre contenido y presentación, el cambio de lenguaje siempre conlleva que el nuevo lenguaje ofrezca nuevas funcionalidades o posibilidades que el anterior no tenía, y viceversa. En este apartado se van a mencionar algunas de las características que se han perdido al pasar de WML a XHTML-MP. No obstante en XHTML-MP siempre existen formas de proporcionar las mismas prestaciones, usando para ello alternativas diferentes.

Algunas de las principales características de WML perdidas en XHTML-MP son:

1. XHTML MP no soporta la estructura de “cartas” y “barajas”.
2. XHTML MP no soporta timers.
3. XHTML MP no soporta eventos.

4. XHTML MP no soporta variables.

Éstas y otras muchas características perdidas en XHTML-MP son subsanadas con creces con el uso de las hojas de estilo WCSS y con el uso del lenguaje de script, que en esta versión de WAP 2.0 se denomina ESMP, el cual es mucho más potente que WMLScript.

2.11.5.2 Declaración XML y DOCTYPE

Todos los documentos XHTML-MP son documentos XML al fin y al cabo, por ello deben proporcionar al principio del documento una declaración que especifique la versión de XML que es. Adicionalmente en esta declaración puede definirse el tipo de codificación que se usará.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Incluir está declaración no es obligatoria para XHTML-MP, sin embargo es de uso recomendado ya que ciertos dispositivos móviles pueden presentar errores si se omite.

De la misma forma, todos los documentos XHTML-MP deben poseer una declaración del DOCTYPE, el cual especifica el DTD que será usado, así como la dirección Web donde se puede encontrar. Este DTD contiene información acerca de la sintaxis del lenguaje de etiquetas, elementos, atributos y reglas a seguir para una correcta construcción del documento. Esta declaración debe ser colocada entre la declaración de documento XML y el principio de la página (marcado por el elemento <html>).

La declaración de DOCTYPE para XHTML-MP puede ser la siguiente:

```
<!DOCTYPE html PUBLIC "-//WAPFORUM//DTD XHTML Mobile
1.0//EN" "http://www.wapforum.org/DTD/xhtml-mobile10.dtd">
```

2.11.5.3 Tipos MIME

Para el uso de documentos XHTML-MP pueden ser usados uno de los siguientes tipos MIME:

- application/vnd.wap.xhtml+xml
- application/xhtml+xml
- text/html

El tipo MIME especificado en la OMA para XHTML-MP es el primero de los anteriormente mostrados, y su uso es un requisito de muchos navegadores móviles para poder presentar correctamente los documentos XHTML-MP.

2.12 ECMASCRIPT-MP

2.12.1 INTRODUCCIÓN

ECMAScript Mobile Profile (ESMP) es el lenguaje de script usado en XHTML-MP para proveer de inteligencia el terminal móvil. Al igual que WMLScript era el lenguaje de script soportado en WAP 1.X junto con WML, ESMP está también diseñado para aportar funcionalidad a las páginas WAP teniendo en cuenta las características restrictivas de los dispositivos móviles, así como el menor ancho de banda de las redes inalámbricas.

ESMP es un subconjunto del estándar ECMAScript al igual que lo es WMLScript, en este caso está definido en la versión ECMA262. Dado que tanto ESMP como WMLScript parten del estándar ECMA, ambos comparten gran cantidad de aspectos similares, pero a la vez, también poseen diferencias referentes a los lenguajes de etiquetas para los que fueron diseñados, esto es, WMLScript está diseñado para dar soporte a WML, mientras que ESMP para darlo a XHTML.

2.12.2 CARACTERÍSTICAS

Puede considerarse que ESMP engloba a WMLScript, puesto que aporta toda la funcionalidad de este último, añadiendo nuevas características y posibilidades. ESMP no solo aporta la funcionalidad para hacer dinámicas las páginas XHTML-MP, sino que aporta parte de la funcionalidad perdida en el paso de WML a XHTML-MP como se vio en el apartado anterior.

Añadido a estas ventajas, ESMP es a su vez un subconjunto del famoso JavaScript, lo cual lo hace perfecto a la hora de compatibilizar su funcionalidad con la Web, tal y como pretendía XHTML-MP.

Las principales características de este lenguaje de script son las mismas que las de WMLScript, por lo que no entraremos en detalle de cada una, tan solo se mencionarán aquellas que son nuevas en ESMP:

- Una de las principales características es que ESMP es un lenguaje basado en objetos (es decir, el paradigma de programación es básicamente el de la programación dirigida a objetos, pero con menos restricciones). Ésto es una gran diferencia

respecto a WMLScript que no soporta objetos. Gracias a esta característica, podemos desde ESMP acceder a cada uno de los elementos definidos en la página XHTML-MP, modificarlos, eliminarlos e incluso crear nuevos elementos una vez la página ya ha sido cargada en el navegador.

- ESMP es un lenguaje orientado a eventos, debido por supuesto al tipo de entornos en los que se utiliza, en los que los eventos son fundamentales. Con esta particularidad, se solventa la carencia de manejar eventos que posee XHTML respecto a WML, dejando el lenguaje de etiquetas para el contenido y el de script para manejar eventos y aportar dinamismo. Además la cantidad y variedad de eventos soportados en ESMP es muy superior a la de WML.
- ESMP soporta la declaración de variables de tipo "array" que no era soportado en su antecesor WMLScript.
- El código de ESMP puede incluirse en un fichero externo tal y como lo hace WMLScript, o bien puede estar escrito en la propia página XHTML-MP, usando para ello las etiquetas de apertura y cierre "<script>….</script>". En la figura2.13 se expone está característica.

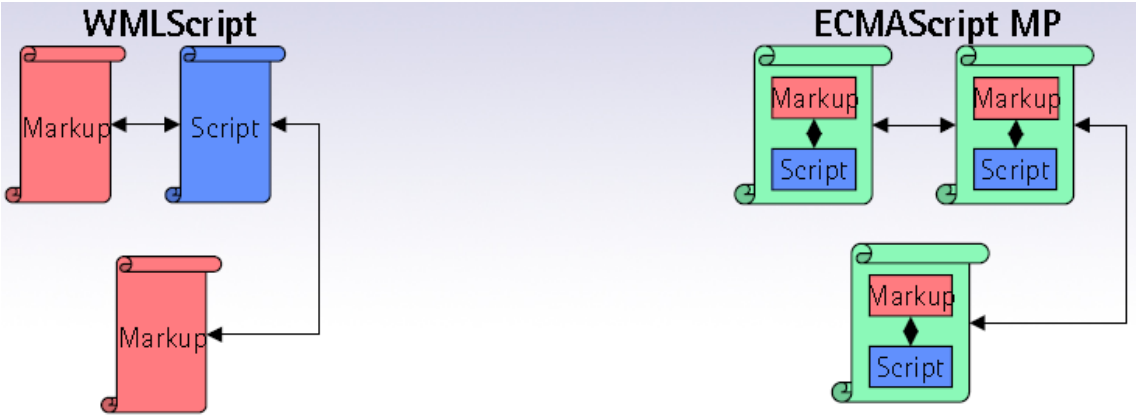

Figura 2.13 WMLScript comparado con ESMP

2.13 HOJAS DE ESTILO WCSS

2.13.1 INTRODUCCIÓN

Las hojas de estilo son conjuntos de instrucciones, a veces en forma de archivo anexo, que se asocian a los archivos de texto y se ocupan de los aspectos de formato y de presentación de los contenidos: tipo, fuente y tamaño de letras, justificación del texto, colores y fondos, etc. Las hojas de estilo permiten liberar la composición del texto de los

aspectos visuales y favorecen que se estructure y anote mediante códigos que permiten un tratamiento más eficaz de los contenidos. Eso se consigue asociando atributos de presentación a cada una de las etiquetas de XHTML o a subclases de éstas.

2.13.2 HOJAS DE ESTILO WCSS

Las hojas de estilo en cascada inalámbricas o WCSS, son un subconjunto de las CSS2 (hojas de estilo empleadas en la Web), a las que se les ha añadido alguna extensión específica para WAP. Estas extensiones son muy útiles para los terminales móviles y las propiedades que se implementan pueden organizarse en tres categorías:

- Teclas de Acceso: Solo se incluye una propiedad en esta categoría y sirve para asociar una tecla a un elemento XHTML-MP.
- Entrada: incluye propiedades para los campos de texto, si pueden estar vacíos, número de elementos, etc.
- Marquee: contiene las propiedades para generar movimientos de textos en diferentes direcciones y sentidos. Esto es muy útil en dispositivos móviles debido a la limitación en el tamaño de las pantallas.

En esencia, son documentos en los cuales se define, de forma separada a la página a la que estén ligados, las características con las que ésta se va a mostrar; características tales como el tamaño de la fuente, el tipo de letra, el color, adornos en las letras… Otra forma de crear las hojas de estilo es incrustándolas en el propio documento al cual queremos dotar de un estilo, pero esta es la forma menos recomendable, ya que así se pierde una gran ventaja que es la separación entre presentación y contenido.

Son documentos tan sencillos que no tienen ningún editor especializado en este lenguaje, ya que sólo definen unos atributos para un determinado estilo. En una misma hoja WCSS se pueden definir tantos estilos como se quiera para, una vez ligada con cierta página, poder usarlos libremente desde ella tan sólo usando el nombre del estilo.

Es evidente por tanto, que con esta separación de presentación y contenido es muy fácil cambiar completamente el aspecto de una página Web con unos pocos cambios en su hoja de estilos. Justamente esta cualidad es la que posibilita ventajas como las siguientes:

- Fácil mantenimiento y actualización de la presentación de la página.
- Posibilidad de tener varias hojas de estilo para una misma página, según las necesidades del usuario que vaya a verla. En nuestro caso dado que las páginas serán visualizadas en dispositivos móviles, puede servir para adaptar el

contenido a las características físicas de cada terminal, como tamaño de la pantalla, colores soportados, etc.

- Claridad en el documento HTML.
- Accesibilidad. Usuarios con deficiencias visuales, por ejemplo pueden usar fuentes o enlaces más grandes. Es por ello que su uso es obligado en este proyecto si se pretende hacer una aplicación medianamente vistosa, y con posibilidades de realizar un mantenimiento adecuado en ella si fuera necesario.

2.14 SERVIDOR DE APLICACIONES TOMCAT

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) es un contenedor de *servlet* desarrollado bajo el proyecto Jakarta en la *Apache Software Foundation*. Tomcat implementa las especificaciones de los *servlet* y de *JavaServer Pages* (JSP) de Sun Microsystems. Es un servidor Web con soporte de *servlet* y JSP e incluye el compilador Jasper, que compila JSP convirtiéndolas en *servlet*. El motor de *servlet* de Tomcat a menudo se presenta en combinación con el servidor Web Apache.

Tomcat puede funcionar como servidor Web por sí mismo. En sus inicios, existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción, y Tomcat es usado como servidor Web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual. Además, se da la circunstancia de que lo desarrollan y mantienen miembros de la *Apache Software Foundation* y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software License*. Estas características son sin duda la mejor baza para formar parte del presente proyecto.

2.15 EMULADORES

En el desarrollo y depuración de este proyecto, se han usado diferentes emuladores y entornos de pruebas atendiendo a las diferentes partes que componen la aplicación. A continuación se hace una breve descripción de ellos, detallando las características por la que se eligió, así como los problemas o deficiencias que presentan.

- **Nokia Mobile Browser 4.0 (NMB):** Es un SDK (*Software Development Kit*) para el navegador de Internet móvil, con el que pueden visualizarse contenidos desarrollados en XHTML, WML, WMLScript y mensajes WAP Push. Sus características son:
    - Permite ejecutar simultáneamente varias instancias y utilizarlas independientemente.
    - Incluye vistas de diagnósticos para todas las instancias abiertas.
    - Es uno de los emuladores más importantes del mercado, dado la importancia de Nokia en este sector.
    - Como inconveniente este emulador no soporta ECMAScript, con lo que no se pueden emplear scripts para WAP 2.0, lo que impide su uso en fases avanzadas del proyecto. Aunque es cierto que si es válido en una primera aproximación.

- **Nokia WAP Gateway Simulator:** Es un simulador monousuario de una pasarela WAP que permite al SDK del teléfono (en nuestro caso el NMB 4.0) acceder a Internet móvil sin necesidad de una conexión real al WAP Gateway. Solo incluye soporte para clientes locales, y es necesario para poder utilizar el NMB junto con el servidor Apache Tomcat.
- **OpenWave Phone Simulator 7.0**: Es otro SDK que simula teléfonos móviles y que permite la visualización de páginas escritas en lenguaje XHTML y WML. Es el otro gran simulador del mercado junto con el NMB, corresponde a la última versión de los simuladores de OpenWave, y presenta unas características similares al NMB, aunque la presentación final es diferente puesto que implementa un móvil de diferentes características. Para este proyecto presenta las mismas limitaciones que el anterior, puesto que no es capaz de visualizar correctamente el código ECMAScript.

Inicialmente se escogieron éstos emuladores para realizar el diseño de la aplicación, pero dados los problemas que éstos presentan a la hora de interpretar el código ECMAScript, se optó por otros emuladores que si soportaran dicho lenguaje. La tarea de búsqueda de un emulador que soportara ECMAScript, que recordando de apartados anteriores es el lenguaje de script de WAP 2.0, no resultó ser nada trivial, dado que WAP 2.0 está actualmente es expansión y la mayoría de simuladores solo soportan XHTML pero no su correspondiente lenguaje de script.

- **S60 SDK:** Es otro SDK de Nokia, correspondiente a la serie 60 de sus navegadores móviles. Es un emulador que está basado en el sistema operativo Symbian y es una de las plataformas más usadas en la actualidad para dispositivos móviles, siendo usado no solo por Nokia, sino también por compañías como Samsung o LG electronics. Este emulador sobrepasa con diferencia a los anteriores y proporciona características muy superiores a las requeridas por este proyecto, no obstante cumple los requisitos de soportar WAP 2.0 en su totalidad, por lo que ha sido escogido para el desarrollo y las pruebas sobre WAP 2.0. Se puede encontrar mucha información sobre las características de esta plataforma en la página www.s60.com así como en los foros de Nokia. Igualmente en http://www.s60.com/life/s60phones/browseDevices.do se relatan y describen los terminales móviles existentes en el mercado que implementan dicha plataforma. Un problema de este emulador es que en teoría está diseñado para soportar tanto WAP 1 como WAP2, pero en realidad no soporta correctamente las aplicaciones realizadas para WAP 1, por lo que se optó por una versión de simulador más antigua que solo soportara WML y WMLScript para realizar el desarrollo y las pruebas de esta parte de la aplicación.
- **OpenWave Phone Simulator 5.1**: Es una versión antigua del simulador de Openwave. Ha sido elegida para este proyecto, dado que es la versión más reciente que soporta en su totalidad el protocolo WAP 1, es decir da soporte tanto a WML como a WMLScript. Por ello en este simulador se han desarrollado y probado las partes de la aplicación relacionadas con el protocolo WAP 1.

2.16 XMLPAD

XMLPAD es un editor de documentos XML de libre uso como muchos otros del mercado. El uso de este editor en el presente proyecto se ha basado en su cualidad de realizar validaciones de documentos XML usando XML-Schema (XSD). Este software permite en primer lugar comprobar la correcta estructura sintáctica de un documento XML, comprobando errores en la escritura de los elementos, fallos al no cerrarlos, etc. En segundo lugar permite crear documentos XSD que definan las restricciones y cualidaddes que deben poseer determinados documentos XML, permitiendo finalmente aplicar estas

restricciones a los documentos XML mediante el proceso de validación, comprobando que se cumplen las restricciones o señalando las causas de no cumplirlas.

CAPÍTULO 3: DISEÑO DE LA APLICACIÓN

3.1 INTRODUCCIÓN

En el presente capitulo se exponen y explican en detalle cada una de las partes que componen la aplicación desarrollada es este proyecto, para ello el capitulo se divide en tres partes. La primera parte es una visión general de la aplicación y de sus principales componentes. En la segunda parte se explicará el sistema de archivos usado por la aplicación, y la forma en que éste es usado dentro de la arquitectura MVC. En la tercera parte se describirá con detalle el funcionamiento de cada uno de los procesos que forman la aplicación, y como éstos acceden al sistema de archivos. Finalmente se mencionarán y razonarán las consideraciones adicionales sobre el diseño de la aplicación que se han tenido en cuenta a la hora de su realización.

3.2 VISIÓN GENERAL

La aplicación desarrollada es un “motor de videojuegos conversacionales sobre WAP para dispositivos móviles”. Esto implica a grandes rasgos y a modo de orientación, que inicialmente es un motor, es decir una aplicación genérica que produce un efecto cuando se le proporciona una alimentación. La aplicación pretende ser la base sobre la que ejecutar diversos juegos de similares características usando para ello una serie de archivos y procesos estándar de fácil generación y configuración. El segundo aspecto importante es que se trata de un motor para videojuegos conversacionales, es decir el motor implementado no pretende poder ejecutar cualquier tipo de videojuego para los dispositivos móviles, sino que se centra en un tipo de videojuego que posee unas características específicas y una forma determinada de desarrollar la acción. Finalmente es una aplicación que está configurada para comunicarse con usuarios de dispositivos móviles y que utilizan el protocolo WAP en dicha comunicación.

Teniendo en cuenta las características y la finalidad de esta aplicación, se hace necesario disponer de una arquitectura que organice los contenidos y facilite la separación entre las diversas partes que la componen, agilizando de esa forma no solo las operaciones de mantenimiento, sino permitiendo de forma sencilla la introducción de nuevos

videojuegos o soportes. Es por ello que la aplicación se ha desarrollado siguiendo la arquitectura MVC, que separa de forma clara los contenidos de la aplicación en las tres partes que le dan nombre (Modelo, Vista y Control).

Para el caso concreto de esta aplicación, el modelo está formado por el sistema de archivos, encargado de configurar cada uno de los videojuegos implementados y de personalizar éstos a cada uno de los usuarios activos. La vista está compuesta por una serie de archivos JSP que adaptan los contenidos a mostrar a las características específicas de cada usuario permitiendo de esta forma el tener un mismo videojuego para diferentes configuraciones del usuario. Finalmente el control está formado por el conjunto de *servlets* que implementan la inteligencia de esta aplicación. Tal y como indica su nombre, es el encargado de controlar las comunicaciones y el paso de la información entre la vista y el modelo, así como el encargado de hacer las oportunas peticiones de cambio en el modelo, según sea la información que recibe del usuario.

3.3 SISTEMA DE ARCHIVOS

3.3.1 INTRODUCCIÓN

El presente proyecto está organizado siguiendo la ya mencionada arquitectura MVC. Es por ello que se hace necesario un sistema de archivos que organice toda la información correspondiente al modelo en dicha arquitectura.

El sistema de archivos hace referencia a todos los archivos y directorios implicados en la definición y configuración de cada uno de los posibles videojuegos a implementar en la aplicación. Puesto que el motor desarrollado es capaz de soportar diversos videojuegos para una misma vista de usuario de forma simultanea, la arquitectura y organización de los contenidos del modelo deben estar fijadas previamente para una correcta comunicación entre ambas partes, así como para permitir la creación de nuevos videojuegos de forma sencilla y cómoda.

3.3.2 ARQUITECTURA DEL SISTEMA DE ARCHIVOS

En la Figura 3.1 se muestra la arquitectura típica de archivos y directorios para cada videojuego de la aplicación.

Todos los videojuegos implementados para esta aplicación, deben ser creados siguiendo dicha estructura, la cual se fundamenta en un directorio con el nombre de la aplicación

"MotorConv" que es la raíz de toda la arquitectura y desde el que cuelgan el resto de directorios y archivos. Seguidamente la arquitectura está compuesta por un nivel de directorios correspondientes a las diferentes vistas que se implementan para la aplicación (las cuales pueden ampliarse), y una serie de directorios que contienen archivos necesarios para el funcionamiento de la aplicación, pero que no pertenecen al sistema de archivos, por lo que se estudiarán con más detenimiento en posteriores apartados. Dentro del directorio de cada vista se incluyen por un lado los archivos correspondientes a la presentación y adaptación de los contenidos al usuario (archivos JSP), y por otro lado un nivel de directorios con los nombres de los diferentes videojuegos implementados.

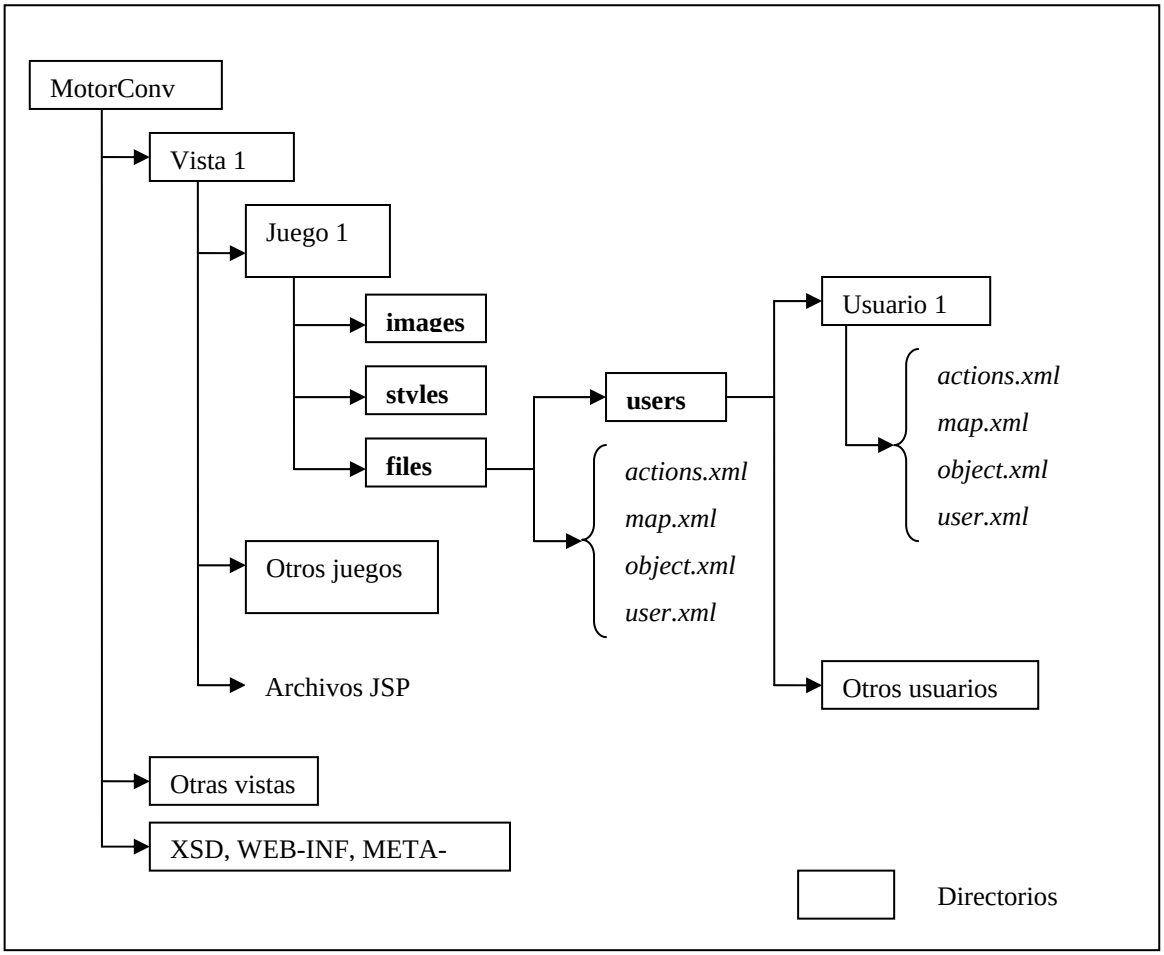

Figura 3.1 Arquitectura del sistema de archivos

Cada directorio correspondiente a un videojuego, sigue a su vez una arquitectura predefinida e independiente a la vista para la que se encuentre definido. De cada directorio de videojuego cuelga un árbol formado por tres directorios: el primero es un directorio en el que se incluyen las imágenes para ese videojuego "images", ya sean de las diferentes localizaciones del juego, o de imágenes usadas para la presentación de los contenidos. El

segundo directorio contiene los diferentes archivos para dar formato gráfico al videojuego "styles", como puede ser el caso de hojas de estilo. Finalmente en el tercer directorio denominado "files" se incluyen un subdirectorio llamado "users" y una serie de archivos de configuración del juego.

El directorio "users" es el lugar donde se guarda la información de los diferentes usuarios de ese videojuego, siguiendo una estructura en árbol como la mostrada en la Figura 3.1 en la que a cada usuario se le asigna un directorio con su nombre.

El conjunto de archivos del directorio "files", formado por *actions.xml*, *map.xml*, *object.xml* y *user.xml* son los encargados de definir el videojuego en cuestión, puesto que en ellos se establecen las acciones, textos y objetos que intervienen en el juego, así como los datos necesarios para conocer el estado del usuario en cada momento. Éstos archivos son los mismos que los usados por cada usuario dentro de su directorio, ya que como se ve en apartados posteriores, al crear una nueva cuenta se copian éstos archivos al directorio del usuario, de forma que al modificar los archivos de cada usuario por el flujo normal del juego, los archivos base del directorio "files" siguen estando intactos para la creación de nuevas cuentas de usuarios.

3.3.3 ARCHIVOS DE CONFIGURACIÓN DE VIDEOJUEGO

Dada la importancia de los diferentes archivos que configuran y definen cada videojuego, en este apartado se realiza un análisis pormenorizado de cada uno de ellos, explicando la información que guardan, así como la arquitectura y la forma en que organizan esos contenidos.

Todos los archivos que configuran el juego poseen formato XML, dada la facilidad de dotar a este tipo de documento de una organización lógica clara, que para este proyecto es esencial, así como por la facilidad de modificación y ampliación de sus contenidos. Adicionalmente incluyen documentación en forma de comentarios tanto al principio del documento, como en cada uno de los elementos que se definen, aclarando y facilitando de esta forma su comprensión para una fácil modificación.

A. user.xml

Es el archivo encargado de guardar la información correspondiente al usuario. Posee un elemento raíz llamado "<config>", del que cuelgan dos elementos hijos: "<screen>" y "<password>", tal y como se aprecia en la Figura 3.2.

El primero de los dos elementos mostrados, almacena en su interior un número correspondiente a la localización en la que actualmente se encuentra el usuario. El segundo de los elementos hijos almacena la contraseña que el usuario proporciona en la creación de la cuenta de usuario, necesaria para la identificación del usuario en posteriores accesos a la aplicación.

```
<config>
        <screen> 1 </screen>
        <password> 1234 </password>
</config>
```

Figura 3.2 Ejemplo del archivo *user.xml*

B. map.xml

Archivo que define el mapa del videojuego, es decir las diferentes localizaciones en las que se divide la aventura implementada. Es el archivo base de cada videojuego puesto que en él se especifican los textos, imágenes, títulos, y direcciones a poder seguir por el usuario en cada momento.

Este archivo configura completamente todos los textos a mostrar en cada una de las localizaciones del videojuego, así como los textos de las fases iniciales de presentación del videojuego o de las fases de creación de cuenta e identificación de usuario como que se verá más adelante en el presente documento. Ésto hace que los textos mostrados al usuario sean totalmente modificables desde este archivo, posibilitando de esta forma un cambio en el idioma del videojuego sin tener necesidad de modificar los archivos propios de la aplicación, para lo cual sí es necesario tener ciertos conocimientos de programación.

El elemento raíz de este documento se denomina “<map>” y del mismo cuelgan una serie de hijos correspondientes a las diferentes localizaciones del juego. De entre los elementos hijos, se puede hacer una división atendiendo al tipo de contenido que presentan:

1 →Elementos con nombre propio de la localización que definen. Estos elementos definen los textos fijos a mostrar en las diferentes fases de la aplicación, de forma que estos textos no dependen del lugar en el que se encuentre el usuario dentro del propio videojuego. Estas fases son: presentación del videojuego “<screenIndex>”, creación de cuenta de usuario “<screenLogin>”, reinicio de la partida “<screenRestart>”, envío y petición de información al usuario

"<screenShowInfo>", final de partida con éxito "<screenEnd>" y final de partida por muerte "<screenDead>".

Cada uno de estos elementos es único y diferente al resto de los elementos contenidos en "<map>". Cada elemento anteriormente mencionado define los textos necesarios atendiendo al contenido que se muestre en esa fase de la aplicación, así por ejemplo en la figura 3.3 puede verse el elemento "<screenIndex>", el cual define el título, el texto a mostrar en la presentación del videojuego, la URL de la imagen que incorpora, y los textos para las diferentes opciones del formulario que aparece en dicha fase de la aplicación, así como para el texto a mostrar en caso de producirse un error debido a no seleccionar ninguna de las opciones del formulario.

```
<screenIndex> <!--Presentación de la aplicación. Se muestra en  "index.jsp"-->
        <title>Presentación</title>
         <text>
            AVENTURA ORIGINAL I, LA BUSQUEDA.
            ¡La gran caverna! Origen de leyendas sobre fabulosas riquezas y
            terribles peligros. ¿Mito o realidad?
            Siguiendo el borroso mapa que te dio un extraño (por cierto, muy rico) a
            quien salvaste de las garras de unos trolls has llegado a este apacible valle.
            Presientes que estas muy cerca, pero de repente el viejo mapa se deshace en
            tus manos mientras una calida brisa te susurra:
           "Solo los valientes que dominen la magia merecen mis tesoros". ¡SUERTE!
        </text>
         <img>./AO/images/ORIGINAL_index.png</img>
        <optionyes>Empezar</optionyes>          <!-- Texto para el formulario
        inicial, indicando SI -->
        <optionno>Salir</optionno>       <!-- Texto para el formulario inicial,
        indicando NO -->
        <alertIndex>Escoge alguna opción. </alertIndex>  <!-- Texto de la alerta en
      index.jsp cuando no escoges ninguna opcion de las 2 dadas -->
</screenIndex>
```

Figura 3.3 Fragmento del archivo map.xml

2→ Elementos que definen los textos, imágenes y alertas a mostrar al usuario dependiendo de la localización del juego en la que este se encuentre. Éstos elementos siguen una notación fija para su nombre, de la forma "<screen id="xx">", donde el atributo id="xx" define el número de la localización al que representa esa etiqueta. Todos los juegos implementados comienzan por la localización número 1 "<screen id='1'>", y poseen un número indefinido de localizaciones a crear dependiendo de lo amplio que sea el mapa del videojuego sobre el que se desarrolle la historia.

Todos estos elementos poseen una estructura predefinida de los elementos que tienen que contener, ya que esta información siempre se muestra en la fase de envío y recepción de datos del usuario. Los elementos que contienen son:

<title>	Indica el texto del título a mostrar.
<text>	Texto que describe esa localización.
<img>	URL relativa de la imagen a mostrar para esa localización.
<direction>	Direcciones posibles que puede tomar el usuario desde la localización en la que se encuentra.

Cada una de las direcciones viene definida por un elemento "<d>" tal y como se muestra en la Figura 3.4, donde se muestra un ejemplo de un elemento de este tipo.

```
<screen id="8">
    <title>Bosque</title>
    <text>
        Aquí el bosque es bastante cerrado, con árboles por todas partes.
     </text>
    <img>../xhtml/AO/images/ORIGINAL_8.png</img>
    <direction>
        <d id="12" activ="si">Norte</d>
        <d id="Dead" activ="si">Sur</d>
        <d id="8" activ="si">Oeste</d>
    </direction>
    <dead>Te caes por el precipicio y te chocas con el suelo, mueres </dead>
 </screen8>
```

Figura 3.4 Estructura de un elemento de localización en *map.xml*

Cada dirección está definida por un elemento "<d>" el cual posee una serie de atributos que complementan la información especificada por este. El elemento "<d>" tan solo almacena el texto que se mostrará al usuario para indicar hacia dónde apunta esa dirección, siendo este texto totalmente configurable y no necesariamente una dirección cardinal, es decir, pueden ser Norte, Sur, Suroeste, Arriba, Bajar, Saltar, Escalar, etc.

Cada elemento dirección contiene dos atributos: el primero de ellos "id" indica la localización a la que se mueve el usuario en caso de que seleccione esta opción, y el segundo atributo "activ" indica si esa dirección está activa o no, es decir, si el valor del atributo es "si" entonces esa dirección podrá ser elegida por el usuario, mientras que si el valor es "no" esa opción no será mostrada al usuario, de forma que no podrá ser escogida.

El atributo “id” suele contener un numero, haciendo referencia a la localización “<screen id=”x”>” a la que se desplaza el usuario, no obstante hay dos tipos de localizaciones especiales que no poseen esa nomenclatura, y son “<screenDead>” y “<screenEnd>” a las que se accede correspondientemente con un valor de “id=End” o “id=Dead”.

Opcionalmente, y solo en las localizaciones del juego en las que el usuario pueda morir al escoger alguna de las opciones de dirección, se añade dentro del elemento “<screen id=”x”>” otro hijo denominado “<dead>” en el que se especifica el texto que describe la causa de la muerte.

C. object.xml

Archivo en el que se definen los diferentes objetos que aparecen en el videojuego. Los objetos son parte fundamental de los videojuegos conversacionales, ya que éstos son los que dan interactividad y abren un abanico de posibilidades al usuario a la hora de manejarlos y utilizarlos para diferentes propósitos. En este archivo se definen todos y cada uno de los objetos que pueden ser usados por el usuario, así como otros objetos que tan solo aparecen en la historia pero que no pueden ser usados, a modo de dificultar un poco la resolución de cada situación planteada.

El elemento raíz de este documento se denomina “<objects>” y del mismo cuelgan todos los elementos objetos “<ob>”, así como una serie de elementos que definen los textos para las diferentes acciones a poder realizar sobre esos objetos, como coger, dejar o mirar. De la misma forma, cuelgan otros elementos que definen los textos en caso de que esas acciones no se puedan realizar sobre esos objetos: errorcoger, errorusar, errorabrir y errorcerrar.

En la Figura 3.5 se muestra un fragmento del archivo *object.xml* perteneciente al videojuego “Aventura Original” adaptado al motor que implementa este proyecto.

Los objetos de un videojuego pueden ser manejados por el usuario, pero no todos pueden ser manejados del mismo modo, algunos pueden ser guardados, otros solo pueden ser examinados, etc. Es por ello que se definen una serie de mensajes tanto para el caso de que una acción tenga éxito, como para el caso de que no la tenga.

Cada objeto puede definir dentro de si mismo un texto específico para cada acción, o para el caso de que esa acción no pueda ser realizada, pero adicionalmente e independientemente a cada objeto, se definen en este documento una serie de textos genéricos para el caso de que un objeto no defina un texto específico para la acción

seleccionada, de forma que en dicho caso se recurrirá a este texto genérico. Con ello se consigue que al usuario siempre se le muestre alguna información que indique si la acción se ha realizado o no.

Cada objeto está representado en este documento *objects.xml* usando un elemento "<ob>", el cual está complementado con una serie de atributos que añaden información sobre el estado del mismo.

```
<objects>
        <!-- Textos por defecto -->
        <coger>Lo guardas en el inventario</coger>
        <!-- texto por defecto para coger, si el objeto no especifica otro para esta accion-->
        <dejar>Lo dejas en el suelo</dejar>
        <!-- texto por defecto para dejar, si el objeto no especifica otro para esta accion-->
        <mirar>Hmm no tiene nada especial</mirar>
        <!-- texto por defecto para mirar, si el objeto no especifica otro para esta accion-->
        <errorcoger>No puedo coger eso</errorcoger>
        <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->
        <errorusar>No puedo usar eso</errorusar>
        <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->
        <errorabrir>No puedo abrir eso</errorabrir>
        <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->
        <errorcerrar>No puedo cerrar eso</errorcerrar>
        <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->

        <ob id="rio" screen="1" cogible="no" usable="no">
                <errorusar>Como no me meta, no se como usarlo.</errorusar>
                <mirarno>Dicen que las aguas de éste río tienen la mágica propiedad de hacer crecer
                        semillas muy viejas.
                </mirarno>
        </ob>

        <ob id="casa" screen="1" cogible="no" usable="no">
                <mirarno>Es una sólida construcción que antaño perteneció a los dueños del valle, hoy
                        está abandonada.
                </mirarno>
        </ob>
```

Figura 3.5 Fragmento del archivo object.xml

Estos atributos son:

"id" → Indica el nombre del objeto. Este nombre actúa como identificador del objeto a lo largo de todo el videojuego, por lo que no pueden existir dos objetos con el mismo nombre.

"screen" → Indica mediante un número, la localización en la que se encuentra el objeto en cada momento. Esto es necesario debido a que los objetos pueden ser cogidos por el usuario y dejados en otra localización. Existen dos casos especiales en los que el valor de este atributo no es el número de la localización en la que se encuentra el objeto: el primero se produce cuando "id=0", lo que indica que el

objeto no está activo, es decir, que ese objeto no se puede usar en el juego o que ya ha sido usado y ha desaparecido. El segundo caso es producido en el momento en que el usuario coge un objeto, este deja de estar en esa localización y pasa a estar en el inventario del jugador, por lo que este atributo toma un valor especial e igual al nombre del usuario que lo ha cogido. Para el caso en que un objeto deba aparecer en el inventario antes de que el usuario lo coja, como es el caso de unir dos objetos que producen un nuevo objeto, se puede utilizar el atributo "id=user" lo cual será interpretado por la aplicación como que pertenece igualmente al inventario.

"cogible" →Es un atributo que indica si el objeto que lo contiene puede ser cogido por el usuario, o no. En el caso de "cogible=si" podrá ser cogido por el usuario, y con "cogible=no" no podrá.

"usable" →Atributo que especifica la usabilidad de un objeto de forma independiente. Este atributo funciona como un seguro previo de las acciones usar, abrir y cerrar, ya que para poder realizar cualquiera de ellas es necesario en primer lugar que dicho atributo esté puesto al valor "si" y en segundo lugar que la acción requerida esté contenida en el archivo de acciones *actions.xml*. Este atributo permite con "usable=si" que un objeto se pueda usar por si solo (es decir, se permite al motor de la aplicación el buscar en *actions.xml* si la acción que pide el usuario está permitida) sin necesidad de usarlo con otro objeto, o caso de "usable=no" que estas acciones no se puedan realizar.

La funcionalidad de este atributo puede observarse en situaciones como la necesidad de abrir una puerta cerrada con llave. Inicialmente en el archivo de acciones *actions.xml* debe existir una acción del tipo abrir, que afecte al objeto puerta, pero que solo se debe poder ejecutar una vez ha sido usada la llave sobre la puerta. Para ello el objeto puerta poseerá inicialmente el atributo "usable=no", de forma que no se puede abrir la puerta (notar que no se llega a buscar la acción en el archivo *actions.xml*), y solo tras realizar la acción del tipo "usar con" entre el objeto llave y el objeto puerta (donde una de las reacciones de esta acción será modificar el objeto puerta, cambiando su atributo "usable=si"), se podrá ejecutar la acción de abrir puerta con un resultado satisfactorio.

D. actions.xml

Archivo en el que se definen las diferentes acciones a realizar sobre los objetos del videojuego, ya sean acciones que solo afectan a un objeto, o acciones que requieren el uso

de varios objetos. Se definen no solo las acciones posibles, sino lo que tal vez es más importante, las consecuencias de esa acción sobre otros objetos o sobre el propio jugador.

Las acciones a poder realizar sobre los objetos del juego son cuatro: usar, abrir, cerrar y usar con. Las tres primeras acciones son aplicadas a un solo objeto, mientras que la última requiere del uso de dos objetos para poder ser llevada a cabo.

```
<acciones>
        <!-- error dado al usar un objeto con otro que no está previsto -->
        <error>No puedo hacer eso.</error>

        <usarcon obj1="llave" obj2="candado">
                <text>Clic!! Abres el candado que mantenía cerrada la reja. AL quitarlo, desaparece
                        de tus manos por arte de magia.
                </text>
                <reacMod obj="reja" atrib="usable" value="si" />
                <!--Pongo el objeto reja como usable, ya podemos abrirla -->
                <reacMod obj="candado" atrib="screen" value="0" />
                <!--Desactivo el objeto candado (desaparece) -->
        </usarcon>

        <abrir obj="reja">
                <text>Abres la reja que da paso a la gran caverna.</text>
                <reacModdir  screen="6"  direction="27"  atrib="activ"  value="si" />
                <!-- permitimos entrar en la cueva -->
        </abrir>
        <cerrar obj="reja">
                <text>Cierras la reja que da paso a la gran caverna.</text>
                <reacModdir  screen="6"  direction="27" atrib="activ" value="no" />
                <!-- NO permitimos entrar en la cueva -->
        </cerrar>
        <usar obj="rama">
                <text>Zarandeas la rama con todas tus fuerzas..... Finalmente el objeto brillante cae
                al suelo.
                </text>
                <reacMod  obj="rama" atrib="usable" value="no"></reac>
                <!--Pongo el objeto rama como NO usable -->
                <reacMod  obj="cosa_brillante" atrib="screen" value="0"></reac>
                <!--Desactivo el objeto brillante (se cae abajo) -->
                <reacMod  obj="llave" atrib="screen" value="16"></reac>
                <!-- Activo la llave en la pantalla 16 -->
        </usar>
```

Figura 3.6 Fragmento del archivo *actions.xml*

Cada acción que pueda ser realizada en la historia del videojuego, debe estar contenida en este documento mediante el uso de un elemento “<usar>, <abrir>, <cerrar> o <usarcon>”. Cada uno de esos elementos hace referencia a una acción de las cuatro posibles a realizar, y su sintaxis puede apreciarse en la Figura 3.6, donde se muestra un fragmento del archivo *actions.xml* perteneciente al videojuego “Aventura Original”.

El elemento raíz es “<acciones>” y de él cuelgan todas las acciones anteriormente definidas. De forma añadida existe un elemento denominado “<error>” en el que se define

el texto de error a mostrar para el caso de que el usuario intente realizar la acción “usar con” con dos objetos que no están previstos en este archivo.

Las acciones “usar”, “abrir” y “cerrar” solo disponen de un atributo llamado “obj”, en el que se especifica el nombre del objeto sobre el que se realiza la acción y que debe coincidir con uno de los objetos definidos previamente en el archivo *objects.xml*. Por su parte la acción “usar con” dispone de dos atributos, uno para cada objeto implicado en la acción, que igualmente almacenan los nombres de los objetos. Otro elemento perteneciente a cada acción posible es el elemento “<text>” en el que se explica con un texto el desarrollo de la acción.

Independientemente de la acción especificada, se pueden añadir una serie de consecuencias o reacciones usando para ello los elementos “<reac>”. Éstos elementos no almacenan información salvo en los atributos que incorporan, los cuales definen diferentes tipos de reacciones o consecuencia. Hay tres tipos básicos de reacciones:

- **<reacIra desp="xx" />** → En este caso la consecuencia o reacción especifica un desplazamiento del usuario (ir a la localización…), y con el atributo “desp” se indica el número de la localización a la que moverse. En este ejemplo el usuario se desplazará a la localización número “xx”. Este tipo de reacción es usado por ejemplo al usar una linterna en una habitación oscura, en ese momento se desplaza al usuario a otra habitación igual a la anterior, pero donde la imagen y la descripción del texto corresponden a esa habitación iluminada y visible por el usuario.
- **<reacModdir screen="x" direction="y" atrib="z" value="w" />** → Esta reacción especifica un cambio en una de las direcciones a poder tomar en esta localización o en otra de las localizaciones del mapa del videojuego. Lo que realiza esta acción es una modificación en uno de los elementos “<d>” del archivo *map.xml* en el que se indican las direcciones que puede seguir el usuario desde esa localización. Los atributos de esta reacción son “screen” que especifica la localización en la que se modificará una dirección, “direction” que especifica de entre todas las direcciones a poder tomar en esa localización, cuál es la que se desea cambiar. El atributo “atrib” indica qué atributo se desea cambiar de la dirección especificada y finalmente con el atributo “value” indicamos el nuevo valor que tomará.

Dada la cantidad de atributos ofrecidos, esta reacción permite realizar multitud de tareas, como puede ser modificar una dirección para que ésta mueva al usuario a otra localización, activar una dirección que antes no estaba disponible, o desactivar una que si lo estaba**.**

- **<reacMod obj="x" atrib="y" value="z" />→** Es la reacción más polivalente de todas, ya que no se adapta a ninguna acción específica, sino que tan solo especifica que modifica algún objeto de los disponibles en el archivo *objects.xml*. Con el atributo “obj” se indica el objeto que se desea modificar, con “atrib” se especifica en qué atributo de ese objeto se está interesado, y finalmente con el atributo “value” se indica el nuevo valor que tomará.

  Hay muchas acciones a poder realizar sobre un objeto, como puede ser activarlo (atrib="screen" value="zz") donde "zz" especifica la localización en la que se introduce el objeto, o “value=user” si se activa el objeto en el inventario del usuario. También se puede desactivar en cuyo caso (atrib="screen" value="0"), donde indicamos que ese objeto dejará de estar disponible en el juego, o incluso podemos modificar la usabilidad de ese objeto (atrib="usable" value="si"), ya que modificamos su atributo “usable” al valor que deseemos “value=si” para indicar que puede usarse, o “value=no” para el caso contrario. De la misma forma puede modificarse el que un objeto pueda ser cogido o no por el usuario en un determinado momento (atrib=”cogible” value=”si/no”).

Para el caso de la acción “usar con” las reacciones a poder realizar son las mismas que para las otras acciones, pero en este caso se deben especificar dos objetos a usar para poder llevar a cabo dicha acción. De esos dos objetos el primero debe ser necesariamente perteneciente al inventario del usuario, mientras que el segundo objeto puede ser del inventario igualmente o simplemente ser un objeto que está en una localización del mapa, y que no ha sido cogido por el usuario.

3.3.3.1 Consecuencias de una acción

Una acción puede, tal y como se ha visto, incorporar una serie de reacciones o consecuencias que alteran el contexto en el que se encuentra el jugador haciendo posible no solo el cumplimiento de esa acción, sino desencadenando posibles eventos ajenos al propio jugador. Estas consecuencias pueden estar formadas por un solo elemento <reac>, o

por más de uno para el caso de consecuencias más complejas y que conllevan el manejo de más de un objeto u atributo.

Seguidamente se muestra una lista de posibles consecuencias que pueden desencadenar las acciones de un videojuego, teniendo en cuenta que tan solo se mencionan las más importantes o comúnmente usadas en este tipo de aventuras, y que queda en mano del administrador o creador de la historia el diseñar nuevas consecuencias adaptadas al contexto del videojuego creado, haciendo uso para ello de los diferentes elementos <reac> de los que se dispone.

- Hacer aparecer un objeto en el inventario del jugador.

```
<reacMod obj="telefono" atrib="screen" value="user"/>
```

  En este ejemplo la reacción, que debe estar incluida dentro de una acción, hace aparecer en el inventario del jugador un objeto llamado “teléfono”. Por ello se ha modificado del objeto “teléfono” el atributo “screen” que indica en qué localización o lugar se encuentra el objeto en cada momento, asignándole el valor “user” que está reservado para indicar el inventario del jugador.

- Hacer desaparecer un objeto del inventario del jugador

```
<reacMod obj="telefono" atrib="screen" value="0"/>
```

  En este ejemplo se modifica el objeto “teléfono” para hacerlo desaparecer del inventario del usuario (el valor actual del atributo “screen” es “user”), desactivándolo (no aparece en el juego). Por ello el valor del atributo “screen” se actualiza a “0”. Igualmente se podría indicar el valor de una localización, haciendo que el objeto desapareciera del inventario y apareciera en dicha localización.

- Hacer aparecer un objeto en una localización determinada del mapa.

```
<reacMod obj="mariposa" atrib="screen" value="4"/>
```

  En este ejemplo se activa el objeto “mariposa” en la localización número 4. No importa dónde estuviera previamente el objeto “mariposa”, desactivado, en el inventario o en otra localización, tras ejecutar dicha reacción el objeto “mariposa” estará en la localización número 4. Algunas acciones que pueden ocasionar ésto pueden ser el accionar una palanca que deje escapar la mariposa,

haciendo que aparezca en el juego o dejar la mariposa que previamente se tenía en el inventario.

- Activar una nueva dirección de movimiento en una localización.

```
<reacModdir screen="6" direction="27" atrib="activ" value="si"/>
```

En este ejemplo se activa una nueva dirección (que previamente estaba desactivada y por tanto no podía usarse), haciendo que en la localización número 6 del juego, aparezca la dirección que apunta a la localización 27. Cabe destacar que la dirección no se crea, sino que se activa, lo que implica que dicha dirección ya debía existir en el archivo *map.xml*, pero con el atributo “activ=no”. Es la típica consecuencia de abrir una puerta o un candado, donde se activa la dirección a través de esa puerta que previamente no estaba disponible.

- Desactivar una dirección de movimiento en una localización.

```
<reacModdir screen="6" direction="27" atrib="activ" value="no"/>
```

Es el caso opuesto al anterior, en el que la dirección de la localización número 6, que apunta a la localización número 27 deja de estar disponible para el jugador, no pudiendo seleccionarla de la lista de direcciones que se le muestra.

- Alterar una dirección de movimiento para que apunte a otra localización.

```
<reacModdir screen="6" direction="27" atrib="id" value="3"/>
```

En este caso una dirección que ya estaba activa se modifica para que apunte a otra localización. En este caso la dirección que apuntaba a la localización número 27 ahora lo hará a la localización número 3. Está reacción puede ser empleada por ejemplo para el uso de un tren, el cual siempre desplaza al jugador una misma localización a excepción de que se modifique la vía (cuya acción ocasiona esta reacción).

- Mover al jugador a una nueva localización.

```
<reacIra desp="4"/>
```

Esta reacción ocasiona un desplazamiento del jugador a otra localización, en este caso a la localización número 4, sin que el usuario seleccione una de las direcciones de movimiento. Esta reacción puede ser muy útil para el caso de una

habitación a oscuras y al accionar un interruptor se deba ver la habitación iluminada. Para poder hacerlo se puede mover al usuario a una nueva localización (habitación iluminada) que sustituye a la anterior, generando el efecto de iluminar. Notar que en este caso se necesitan dos localizaciones, la habitación a oscuras y la habitación iluminada.

- Unir dos objetos entre si, creando un nuevo objeto.

```
<usarcon obj1="pila" obj2="linterna">
    <text>Introduces la pila en la linterna.</text>
    <reacMod obj="linterna" atrib="usable" value="si"/>
    <reacMod obj="pila" atrib="screen" value="0"/>
</usarcon>
```

En este ejemplo se ilustra la acción de introducir una pila en una linterna con la finalidad de que ésta pueda ser usada (encendida). Para ello se necesitan varias reacciones, la primera modifica el objeto “linterna” haciendo que su atributo “usable” este activado, permitiendo que en próximas acciones el usuario la use. En segundo lugar se debe desactivar el objeto “pila”, ya que ha sido usado con éxito y no tiene sentido seguir teniendo una pila en el inventario si ésta está dentro de la linterna, es por ello que su atributo “screen” adquiere el valor “0”.

- Separar un objeto compuesto, en dos objetos simples.

```
<abrir obj="linterna">
    <text>Abres la linterna y sacas la pila</text>
    <reacMod obj="pila" atrib="screen" value="user"/>
    <reacMod obj="linterna" atrib="usable" value="no"/>
</abrir>
```

En este caso se saca la pila de la linterna, por lo que el objeto “pila” regresa al inventario del jugador, y la linterna toma un valor de su atributo “usable” igual a “no”, indicando que esa linterna no puede ser encendida. De la misma forma, se podría haber especificado que al separar el objeto compuesto, éste desapareciera y en su lugar se obtuvieran dos nuevos objetos simples, tal y como se muestra en el segundo ejemplo.

```
<abrir obj="arco">
    <text> Desmontas el antiguo e inservible arco</text>
    <reacMod obj="arco" atrib="screen" value="0"/>
    <reacMod obj="palo" atrib="screen" value="user"/>
    <reacMod obj="cuerda" atrib="screen" value="user"/>
</abrir>
```

- Hacer que un objeto que no podía ser cogido por el usuario, pueda serlo.

```
<reacMod obj="candado" atrib="cogible" value="si"/>
```

Tan solo necesitamos cambiar su atributo “cogible=si”, de forma que desde ese momento la acción “coger candado” sí pueda realizarse. Esta reacción puede usarse en conjunción con otras reacciones dentro de una misma acción.

- Hacer que un objeto pueda usarse, abrirse, o cerrarse, cuando antes no podía. Esta reacción se aplica sobre el atributo “usable” que como ya se comentó es el encargado de dar permiso a esas tres acciones para poder ser ejecutadas, o impedir su ejecución en un momento determinado a pesar de estar definidas en el archivo *actions.xml*.

```
<usarcon obj1="llave" obj2="candado">
    <text>Clic!! Abres el candado que mantenía cerrada la reja. Al
            quitarlo, desaparece de tus manos por arte de magia.
    </text>
    <reacMod obj="reja" atrib="usable" value="si"/>
    <reacMod obj="candado" atrib="screen" value="0"/>
</usarcon>
```

El ejemplo mostrado agrupa varias reacciones como consecuencia de abrir un candado con la llave apropiada. La primera reacción hace que el objeto “reja” que no era usable (es decir no se podían ejecutar sobre él las acciones de abrir, cerrar o usar), pueda serlo, permitiendo una posterior acción de “abrir reja”. La segunda reacción proporciona la desaparición del objeto “candado”, tal y como se muestra en el texto descriptivo de la acción.

- Producir la muerte del jugador en el videojuego.

```
<reacIra desp="Dead"/>
```

Puede darse el caso de que al realizar ciertas acciones, una de las reacciones sea ocasionar la muerte del jugador. Ésto que a simple vista puede parecer complejo no es más que mover al jugador a la localización en la que se le muestra la confirmación del fallecimiento, y en la que se muestra el texto descriptivo explicando las causas.

3.4 FUNCIONAMIENTO DE LA APLICACIÓN

El funcionamiento general de la aplicación puede verse en la Figura 3.7, donde se detalla gráficamente cada una de las fases por las que pasa la aplicación y la relación entre ellas. No obstante esta figura es un diagrama funcional, donde cada una de las fases muestra un estado del funcionamiento de la aplicación, que en la realidad vendrá gestionado por uno o varios archivos.

Los archivos que implementan cada una de las fases de esta organización funcional, tal y como se dijo en el apartado 3.2, están organizados siguiendo la arquitectura MVC que les confiere una alta separación lógica. En la Figura 3.8 se muestra la relación entre los principales archivos de la aplicación y la forma en que éstos están organizados.

Como se aprecia en la Figura 3.8, la vista (encargada de mostrar la información al usuario) está formada por un conjunto de archivos desarrollados usando la tecnología JSP, y éstos son independientes en su totalidad de los archivos que forman el control de la aplicación.

Con ésto se consigue un aislamiento completo y permite el cambio de una vista a otra, sin tener que modificar en absoluto el control o el modelo de esta aplicación.

El modelo de la aplicación está formado por el sistema de archivos detallado en el apartado 3.3, en el que se definen los diferentes videojuegos a ejecutar por la aplicación, así como la información propia de cada usuario en cada una de las vistas disponibles. Este modelo es accedido y consultado tanto por el control de la aplicación, como por la vista, a fin de obtener los valores de textos y mensajes de error que no dependen de las acciones realizadas por el control, facilitando de esa forma el paso de la información.

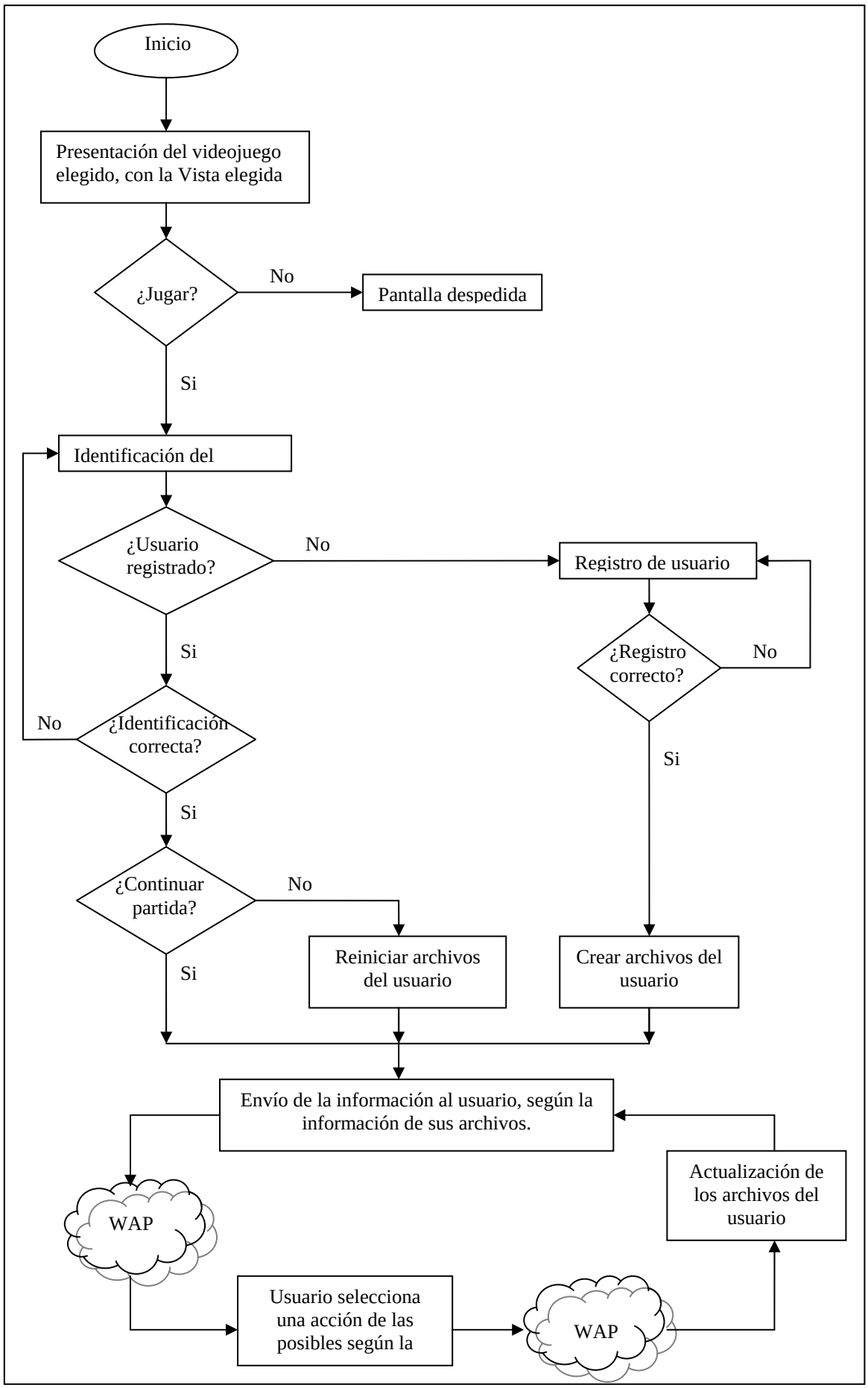

Figura 3.7 Diagrama de funcionamiento de la aplicación

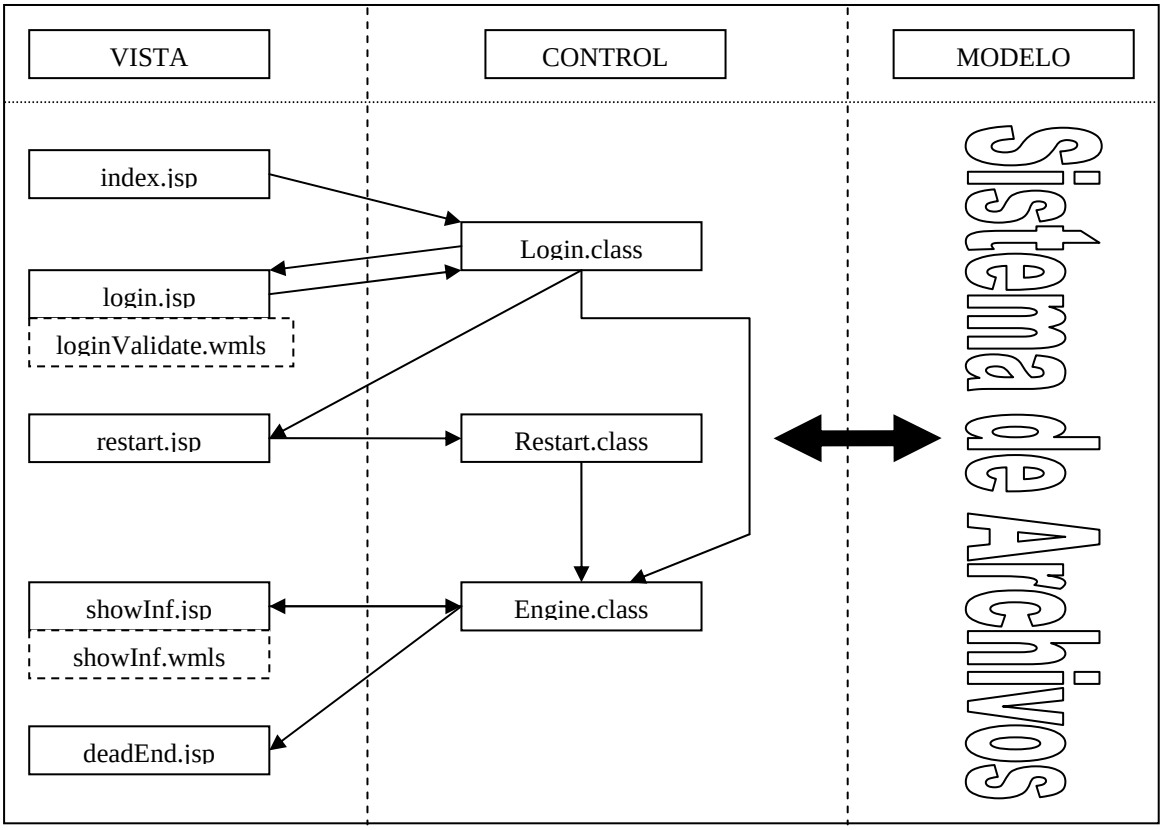

Figura 3.8 Organización de los archivos según la arquitectura MVC

Atendiendo a las Figuras 3.7 y 3.8 puede deducirse de forma sencilla el funcionamiento de la aplicación, la cual se basa en una serie de procesos iniciales como la presentación del videojuego (index.jsp), una fase de registro y autentificación de usuario (login.jsp, login.class y para el caso de WML loginValidate.wmls), posteriormente una fase opcional en la que un usuario puede reiniciar la partida o retomarla (restart.jsp, restart.class), y finalmente las fases en las que se desarrolla el juego en su totalidad basándose en una comunicación entre el usuario y el servidor (engine.class, showInf.jsp y solo para WML showInf.wmls).

3.4.1 CONFIGURABILIDAD DE LA VISTA

Una de las grandes premisas de este proyecto era desarrollar un motor para videojuegos conversacionales que fuera totalmente configurable desde archivos de texto, facilitando de esta forma el cambio en cualquiera de sus presentaciones (vistas) de forma sencilla.

Esto se ha conseguido en gran manera gracias al uso de la arquitectura MVC, ya que las vistas implementadas lo que establecen es la forma en la que el contenido debe organizarse en la pantalla del dispositivo móvil del usuario, pero no el contenido que deben mostrar. Es decir, una vista establece las cabeceras, el tipo de etiquetas según el lenguaje de marcas

soportado por el cliente, las características gráficas y de diseño, pero no establece el contenido a mostrar, sino que realiza llamadas al modelo (o el control realiza esta llamada y posteriormente pasa los datos a la vista) para obtener la información a mostrar en esa fase de la aplicación.

De esta forma se pueden desarrollar diversas vistas usando siempre el mismo modelo, o lo que parece más interesante el poder cambiar de forma sencilla y sin necesidad de tener conocimientos de programación el contenido mostrado al usuario, posibilitando de esta forma por ejemplo una fácil traducción a otro idioma o la generación de nuevos videojuegos.

En la Figura 3.9 se muestra la organización de los contenidos en el servidor, atendiendo a la arquitectura MVC y a la posibilidad de tener en el mismo servidor, diferentes vistas y diferentes modelos (diferentes videojuegos).

Sólo hay un grupo de archivos dedicados al control, y éstos son comunes a todas las vistas y a todos los videojuegos implementados en la aplicación. Sin embargo, a pesar de que se podría haber realizado un único modelo (conjunto de archivos que definen el videojuego) válido para diversas vistas, ésto no se ha implementado, y se ha optado por realizar para cada vista un modelo propio del videojuego. Esto ha sido así dada la necesidad de adaptar ciertos aspectos propios del videojuego a la vista usada, como puede ser el tipo de imágenes que éste soporta. Es decir, aunque en su mayor parte el modelo de un videojuego es prácticamente igual para diferentes vistas, hay pequeñas diferencias que imposibilitan su uso común.

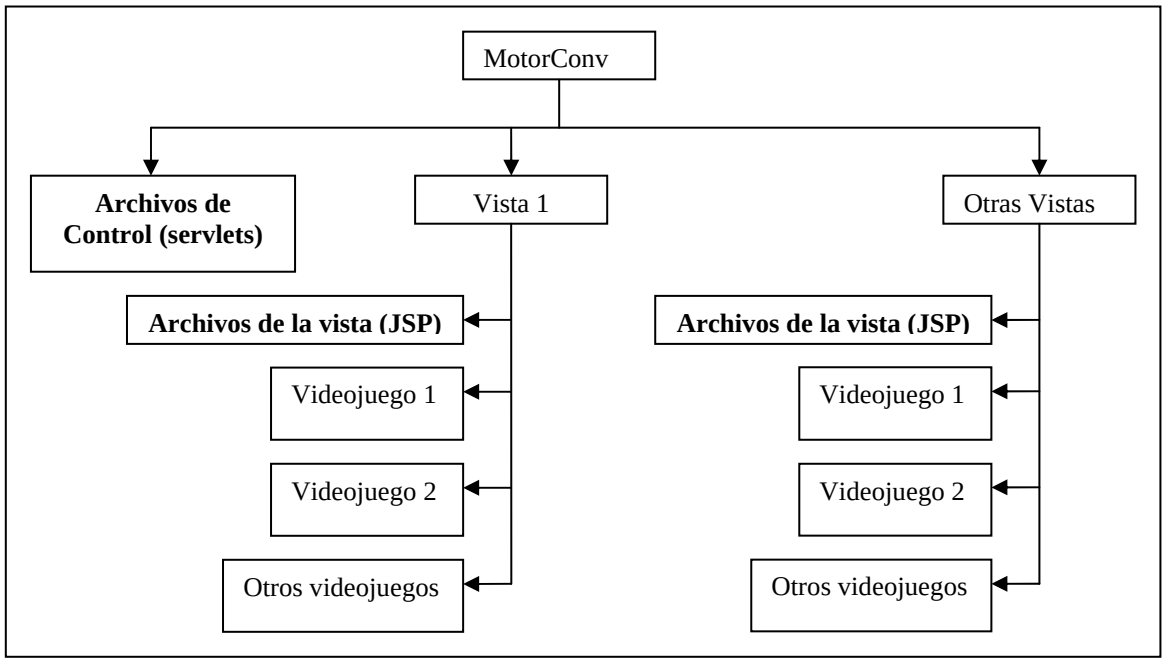

Figura 3.9 Organización de los contenidos en el servidor

Teniendo en cuenta la forma en la que se organizan los contenidos en el servidor, un usuario deberá realizar una llamada a la aplicación indicando el tipo de vista que desea y el videojuego en el que está interesado.

3.4.2 PROCESOS DE LA APLICACIÓN

En este apartado se describen con detalle cada uno de los procesos de la aplicación, los cuales se basan principalmente en una pareja formada por un archivo de control y la correspondiente vista generada al usuario. Hay que tener en cuenta que los archivos de la parte de control son únicos y comunes para todas las vistas posibles y para todos los videojuegos, mientras que los archivos de la vista son comunes a todos los videojuegos, pero diferentes a los archivos de otras vistas. En el caso de este proyecto se han realizado dos vistas atendiendo a las dos versiones del protocolo WAP, permitiendo de esta forma dar cabida a los móviles más y menos modernos.

3.4.2.1 Presentación

El proceso de presentación es el proceso inicial en el que se realiza una presentación del videojuego seleccionado. Según la vista seleccionada, la forma en que el contenido es mostrado al usuario varía, adaptándose a las características del protocolo de acceso y de las del terminal del usuario.

Es un proceso muy simple que muestra una imagen y un texto introductorio del videojuego. Finalmente se introduce un pequeño formulario que permite al usuario confirmar si desea jugar a ese videojuego o salir de la aplicación, en cuyo caso se mostrará un pequeño texto de despedida que es obtenido de los propios archivos de configuración del juego, y que por lo tanto es configurable para cada videojuego, como lo es el texto mostrado en esta presentación y las opciones del formulario.

La presentación de los contenidos viene dada por el archivo *index.jsp* correspondiente a la vista seleccionada, al cual hay que proporcionar un parámetro (“*room*”) en la llamada indicándole cuál es el videojuego que se desea lanzar (ver figura 3.9). Este parámetro debe proporcionarse por el método GET o POST, aunque de forma natural se realiza mediante el método GET, ya que se introduciría en la propia URL que llama a la aplicación.

Estructura de la llamada a la aplicación:

http://**direcionServidor:[puerto]**/MotorConv/**VISTA**/index.jsp?room=**nombre_juego**

Ejemplos de llamadas:

http://localhost:8080/MotorConv/wml/index.jsp?room=AO

http://localhost:8080/MotorConv/wml/index.jsp?room=escapada

http://localhost:8080/MotorConv/xhtml/index.jsp?room=AO

http://localhost:8080/MotorConv/xhtml/index.jsp?room=escapada

Figura 3.10 Diferentes llamadas al motor

En la Figura 3.10 se pueden apreciar diferentes llamadas a la aplicación, especificando diferentes vistas (XML o XHTML) y diferentes videojuegos (Aventura Original, escapada, etc.), usando para ello el parámetro "*room*".

Antes de enviar la información del pequeño formulario a *login.class*, que será el archivo perteneciente al control encargado de gestionar esa información, se realiza una pequeña comprobación para verificar que alguna de las dos opciones del formulario ha sido marcada, usando para ello un lenguaje de script si fuera necesario. Tras ésto, se envía la respuesta al *servlet login.class* que la procesa y muestra en caso negativo un texto de despedida, o en caso afirmativo un formulario en el que se pide al usuario se identifique como usuario registrado.

3.4.2.2 Registro de usuario

Dado que cada juego se personaliza para cada usuario, localización del mapa por la que está jugando, objetos que posee en el inventario, acciones que ya ha realizado, etc. es necesario para la aplicación tener constancia de los diferentes jugadores y de su estado. Es por ello que se hace necesario un registro de los usuarios antes de comenzar a jugar.

El registro de los usuarios está gestionado por los archivos *login.jsp* y *login.class*. El primero de ellos, perteneciente a la vista, es la presentación de un formulario donde se pide

información sobre el nombre y la contraseña del usuario para la creación de una cuenta en dicho juego, y el segundo archivo, perteneciente al control, es el que realmente realiza el proceso de la creación de la cuenta según los datos obtenidos del primero.

A la hora de crear la cuenta de usuario, hay que tener en cuenta varios aspectos:

1. Antes de enviar la información del formulario al *servlet*, es necesario comprobar que:
    a. Ninguno de los campos del formulario está vacío.
    b. La contraseña se pide dos veces para evitar posibles errores. Por ello, los dos campos de contraseña del formulario deben coincidir.

    Estas comprobaciones se pueden hacer usando un lenguaje de script, o usando las características propias del lenguaje de marcas, si este las ofrece.
2. El nombre de usuario especificado en el formulario para la creación de la cuenta no puede existir previamente para ese juego. Es decir, cada usuario va a ser identificado unívocamente por su nombre, por lo que este debe ser único, y no se puede crear un nuevo usuario con el nombre de otro que ya existe. Es por ello que si este problema se diera, el *servlet* envía un mensaje de error indicando que ese nombre ya está en uso.
3. Cada usuario es local al juego en que se crea la cuenta. Es decir, al crear una nueva cuenta con un nombre y contraseña facilitados por el usuario, esta no da acceso a otros videojuegos usando los mismos datos, sino que esos datos solo son válidos para ese videojuego. Con ello se permite que un mismo nombre de usuario pueda ser usado en diferentes juegos por diferentes usuarios, aumentando notablemente la posibilidad de elección de los nombres.

Todos los posibles errores producidos en este proceso son mostrados al usuario usando alertas, de forma que se visualice el mensaje de error y al cerrar la ventana de la alerta se vuelve a tener el formulario para subsanar ese error y volver a intentar el proceso. Una vez las comprobaciones se han realizado con éxito, el proceso de creación de una cuenta de usuario se basa en crear un directorio con el nombre de ese usuario, y dentro de este se colocan los archivos de configuración del juego. Éstos archivos se ven en detenimiento en el apartado 3.3 dedicado al Sistema de Archivos, pero a grandes rasgos la información que se guarda para cada usuario es el mapa del juego que contiene las diferentes localizaciones, un indicador de la localización en la que se encuentra el usuario, la contraseña que

especifica, los objetos que posee en el inventario y las acciones a realizar y que ya ha realizado.

Una vez realizado con éxito el proceso de registro del usuario, no es necesaria la identificación del mismo, sino que automáticamente se pasa a la ejecución del videojuego comenzando por la localización inicial del mismo.

3.4.2.3 Identificación de usuario

Una vez se ha confirmado que se desea jugar al videojuego seleccionado, el usuario debe identificarse como un usuario registrado para poder acceder al mismo (ya debe estar registrado previamente). Es por ello que este proceso necesita tan solo de la información que proveen los campos de nombre y contraseña que se le muestran al usuario en un formulario también incluido en el archivo *login.jsp*.

Tal y como indica su extensión, *login.jsp* forma parte de la vista, con lo que los contenidos se adaptan a cada cliente según sus características. No obstante e independientemente a la vista, este archivo contiene dos formulario en si mismo, uno para el registro de los usuario y otro formulario un poco más simple para la identificación de los mismos. El mostrar uno u otro formulario depende de la forma en que se llame a *login.jsp* y de los parámetros que se le pasen, aspecto que es totalmente transparente al usuario.

En el formulario de identificación del usuario, tan solo hay un campo para el nombre y otro para la contraseña, e igualmente al formulario del registro se comprueba antes de enviar la información que los campos no están vacíos, usando para ello un lenguaje de script como en el caso anterior.

El archivo *login.class* es el encargado de comprobar que ese usuario existe en el registro y que la contraseña facilitada por el usuario es la correcta. Hay varios aspectos que se deben comprobar para una correcta identificación:

1. El usuario con el nombre especificado debe existir. Lo primero que se comprueba es que para ese videojuego existe un usuario cuyo nombre coincide con el especificado. Teniendo en cuenta que los usuarios registrados disponen de un directorio con su nombre dentro del sistema de archivos, se debe comprobar únicamente que ese directorio existe para el videojuego seleccionado.
2. Caso de que el usuario sí exista, es necesario comprobar que la contraseña facilitada a través del formulario es la correcta para ese usuario en ese juego. Para ello se accede al archivo de configuración de usuario (user.xml), contenido

dentro del directorio que lleva su nombre, y se comprueba si las contraseñas en ambas partes coinciden.

Si por algún motivo la identificación no es correcta, el *servlet* enviará un mensaje de error al usuario indicándole la causa del fallo. Este error, al igual que en el proceso de registro, se realiza usando alertas.

Una vez el proceso ha concluido con éxito (el usuario se ha identificado correctamente para el videojuego seleccionado), se le ofrece al usuario la posibilidad de continuar con la partida actual (lo que retomará la partida según la dejó la ultima vez que jugó, manteniendo los objetos, las acciones realizadas, etc.), o comenzar de nuevo (lo cual reinicia todos los archivos de ese usuario y vuelve a dejarlas como si se iniciara el juego por primera vez), mediante los archivos *restart.jsp* y *restart.class*. Para el caso de elegir la segunda de las opciones, se realiza una llamada al proceso de reinicio, el cual borra todos los archivos de configuración del usuario y vuelve a crearlos como la primera vez que se realiza el registro de usuario, de forma que se reinicia el juego. En ambas opciones se termina por pasar el control al *servlet "engine.class"* que ejecuta el juego propiamente dicho, y que se detalla en el siguiente apartado.

Tanto para el caso de registro de un nuevo usuario, como para una posterior identificación del mismo, la aplicación crea una serie de variables de sesión en el servidor para almacenar información necesaria para la aplicación. Estas variables de sesión almacenan el nombre del usuario (facilitado en la identificación del mismo), la vista que está usando (para este proyecto puede ser WML o XHTML), y el videojuego en el que está (room=AO, por ejemplo). Al tratarse de variables de sesión, esta información permanecerá disponible mientras el usuario no cierre el navegador, o si no permanece más de 30 minutos de inactividad. Si la información de estas variables se pierde, será necesario reiniciar la aplicación y volver a identificarse, recobrando de esa forma el contexto en el que se dejó la aplicación.

3.4.2.4 Ejecución del videojuego

Este proceso se lleva a cabo por una pareja formada por los archivos *showInf.jsp* y *engine.class*. El primero es un archivo perteneciente a la vista y encargado de adaptar los contenidos que le envía el servidor a las características del terminal del usuario, y el segundo pertenece al control de la aplicación y es el encargado de realizar los cambios oportunos en los archivos del usuario para llevar a cabo las acciones que éste le pide.

Desde este momento, el flujo de la aplicación bascula entre éstos dos archivos, tal y como muestra la parte inferior de la Figura 3.8, en la que al usuario se le muestra un contexto del juego y se espera a que este realice una petición de alguna acción al servidor, éste reaccionará modificando el modelo si fuera necesario, y volverá a enviar al usuario los datos correspondientes.

Son sin duda los dos archivos principales de la aplicación, y por ello se detalla su funcionamiento con más de detenimiento.

a) showInf.jsp

Tal y como se ha comentado, este archivo es el encargado de adaptar el contenido a mostrar en cada uno de los posibles contextos del videojuego, a las características del usuario teniendo en cuenta la vista seleccionada.

Este archivo puede separarse en dos partes:

1. Parte estática: corresponde a la parte que siempre es la misma sin importar en qué localización del juego se encuentre el usuario. Aquí se incluyen las diferentes acciones a poder realizar, el nombre de los diferentes formularios, y los mensajes de error a mostrar en caso de ser necesario. Esta información, dado que es fija, se incluye en un archivo de configuración denominado *map.xml*, bajo el elemento “<screenShowInfo>” que se encuentra definido para cada videojuego, y que contiene para cada localización del juego los elementos estáticos.
2. Parte dinámica: la cual está formada por las diferentes opciones que acompañan a las acciones, los textos e imágenes según la localización del juego en la que se encuentre el usuario. Esta información varía según la acción requerida por el usuario y de la localización en la que se encuentre, es por ello que dicha información se buscará en los archivos de configuración propios de cada usuario, atendiendo a la localización en la que se encuentre.

El flujo de tareas a realizar por este archivo para cumplir con su cometido, es el siguiente:

a. La primera tarea que realiza este archivo es leer de las variables de sesión el nombre del usuario y el nombre del juego en el que está participando.
b. Seguidamente del archivo de configuración *map.xml*, adquiere los textos y palabras fijos a mostrar al usuario, los cuales no varían de una localización a

otra, y que se encuentran bajo el elemento “<screenShowInfo>”. Éstos son el nombre de las acciones a realizar (coger, usar, mirar, etc.), los títulos de los formularios, y los mensajes de error por dejar vacío algún formulario

c. Posteriormente se accede a la información dinámica que depende de la localización del juego en la que se encuentre el usuario, y de la acción que realizó en última instancia. A esta información no accede mediante la lectura directa del archivo *map.xml*, sino que esa información ya ha sido leída y tratada por el *servlet engine.class*, el cual tras terminar su trabajo envía dicha información a la vista usando variables internas del servidor, agilizando de esa forma el proceso. Esta información incluye el titulo de la localización, el texto que la describe, la imagen que se muestra como acompañamiento, así como los diferentes objetos que se pueden usar en cada una de las acciones disponibles. Finalmente se incorpora un mensaje de texto que se mostrará al usuario previo al resto de la información en forma de alerta, y en la que se describirá el transcurso de la última acción requerida, caso de que esta exista.

d. Finalmente y con toda la información de la que dispone tras los pasos anteriores, adapta éstos datos al usuario según la vista (WML, XHTML, etc.), haciendo uso del lenguaje de etiquetas indicado y de los posibles lenguajes de script soportados. Independientemente de la vista, al usuario se le muestra la imagen y el texto descriptivo de la situación, así como un formulario en el que se le ofrecen las diferentes acciones a poder realizar en cada momento (ir a, mirar, coger, usar, usar con, abrir, cerrar, dejar e inventario), así como las diferentes opciones sobre las que puede realizar esas acciones. Dicho formulario debe ser confeccionado por el usuario eligiendo una acción y una opción, las cuales serán enviadas al *servlet engine.class* para que realice dicha operación.

e. Un caso excepcional de este archivo, se produce cuando el usuario ha terminado el juego o ha realizado alguna acción que le lleva a la muerte. En ambos casos no se muestra el formulario de *showInf.jsp*, sino que se traspasa el control a un archivo denominado *deadEnd.jsp*, donde se muestra la causa de la muerte o se felicita por haber terminado con éxito el juego. En este caso al usuario se le ofrece la oportunidad de comenzar de nuevo el juego mediante un enlace que llama de nuevo al *servlet* encargado de reiniciar los archivos del jugador (*restart.class*, como en el caso producido tras la correcta identificación del usuario).

b) engine.class

Este archivo es el corazón de la aplicación, ya que en él se realizan los principales procesos y tareas que el usuario requiere en cada momento. Es el encargado de realizar todas las acciones disponibles, así como de modificar el modelo de cada usuario para ir adaptándolo a las situaciones mostradas. En la Figura 3.11 puede verse el diagrama de flujo de las acciones que lleva a cabo.

De forma detallada las acciones que lleva a cabo este archivo son:

a. Accede a las variables de sesión del usuario que le llamó, accediendo a su nombre, al videojuego en que está inmerso y a la vista que está usando. Igualmente recoge la información del formulario *showInfo.java* que ha cumplimentado el usuario, recogiendo la acción y la opción sobre la que se aplica.

b. Tras recoger esa información, intenta ejecutar la acción que le pide el usuario. Para ello accede a los archivos de configuración del usuario, mirando si la acción que éste requiere es posible realizarla. Ésto se realiza llamando a un método de dicho *servlet* denominado “doAction”. En caso de que sí pueda ser realizada, la ejecuta modificando de la manera oportuna los archivos correspondientes, es decir, si el usuario desea moverse de localización y ésto es posible, el *servlet* modificará el archivo *user.xml* para indicar la nueva localización en la que se encuentra el usuario. Por el contrario si el usuario deseaba usar un objeto, se mostrarán las diferentes reacciones que éste conlleva, activando o desactivando otros objetos como se detalla en el apartado 3.3 del presente documento. Para el caso de que la acción no se pueda realizar, este método retorna un mensaje de error que será mostrado al usuario mediante una alerta, y sin realizar ningún cambio sobre los archivos del usuario.

c. Finalmente, se prepara la nueva información a enviar al usuario. Para ello y atendiendo a si ha podido o no realizar la acción requerida por el usuario, lee de los archivos de configuración la información que debe enviar de nuevo a *showInf.jsp*, completando de esa forma el ciclo en que está inmerso el desarrollo del juego.

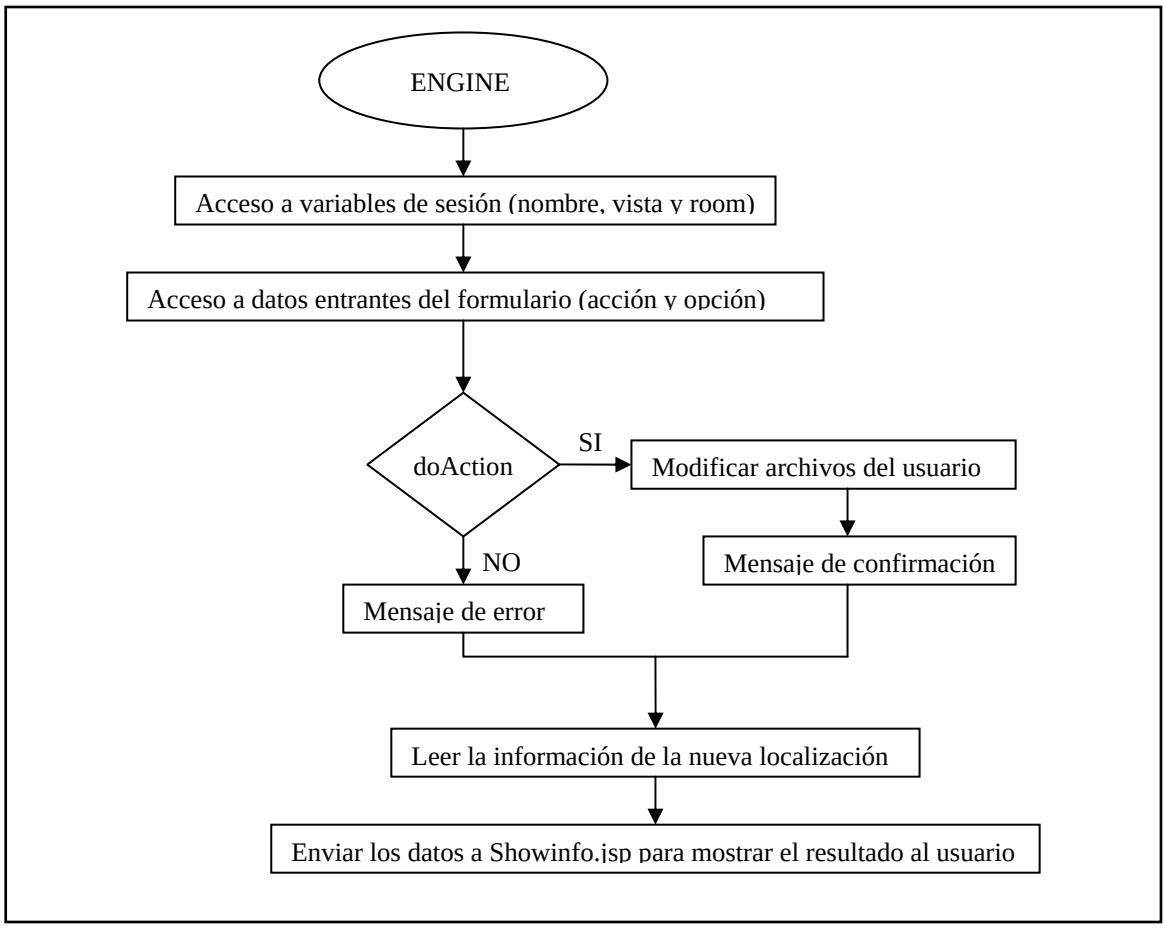

Figura 3.11 Diagrama de flujo de engine.class

3.5 CONSIDERACIONES ADICIONALES SOBRE EL DISEÑO

A la hora de diseñar la aplicación surgieron distintos problemas o posibilidades, que se explicarán a continuación junto con la solución adoptada ante cada uno de ellos.

3.5.1 CREACIÓN DE SESIÓN

Una vez que se identifica el usuario se debe mantener información sobre un cliente a lo largo de un proceso que implica múltiples conexiones. Se puede optar por tres soluciones tradicionales:

· cookies.

· seguimiento de sesiones.

· reescritura de URL

La tercera opción queda descartada, ya que la reescritura no es un proceso transparente al cliente en el navegador, y además porque la información que se puede trasportar en una URL es limitada.

Las cookies permiten que la información se pueda acceder/modificar en cliente y servidor, lo que permite distribuir el trabajo entre ellos. No obstante esta opción ha sido descartada debido a las variadas y diferentes características de los terminales móviles, lo que no garantiza que todos ellos sean capaces de soportar las cookies, o de manejarlas de la misma forma.

El seguimiento de sesiones lo implementa el *servlet* gracias a la clase HTTPSession. Consiste en una estructura de datos creada en servidor cuyo identificador lleva información de la dirección IP del cliente, hora de la conexión, etc.

Para la aplicación que se ha diseñado se ha hecho uso de esta última opción, a la que se le ha añadido una serie de variables de sesión adicionales, porque la sesión implementada por los *servlets* no consiste más que en una tabla sencilla, mientras que la estructura de datos que se necesita es más compleja. Además se desea que el identificador de la sesión sea el identificador del usuario y no un identificador de la conexión realizada, ya que solo se permite una conexión simultánea con el identificador de usuario, y en el caso de que cualquiera de los dos equipos tuviera IP dinámica y cambiara en el transcurso de la sesión, se perdería la referencia a los datos.

El nuevo modelo de sesión creado se basa en varias variables (identificador de usuario, vista en la que se encuentra e identificador del videojuego seleccionado).

3.5.2 PRESENTACIÓN DE LOS CONTENIDOS

Una vez el usuario ha elegido el juego en el que está interesado y comienza a jugar, los contenidos del videojuego se adaptan al usuario en función de la vista escogida por el mismo, lo cual depende del protocolo inalámbrico que soporte su terminal móvil.

No obstante y para todos los casos de vistas posibles, se debe tener en cuenta las limitaciones a la hora de presentar los contenidos y sobre todo las limitaciones a la hora de recibir información de los usuarios. Ésto se debe mayoritariamente a las limitaciones en los teclados de los dispositivos móviles, que dificultan en gran manera la escritura de textos de forma asidua.

Es por ello que la presentación de los contenidos se ha realizado tanto para la versión WML, como para la versión XHTML, haciendo uso de elementos que limiten la necesidad de que el usuario escriba textos o palabras usando el teclado. En la aplicación, las

instrucciones que el usuario puede enviar al motor son limitadas y tan solo pueden escogerse de una lista previamente configurada por el motor. Esto se aprecia a la hora de elegir la acción a realizar, y la opción que le acompaña en cada caso.

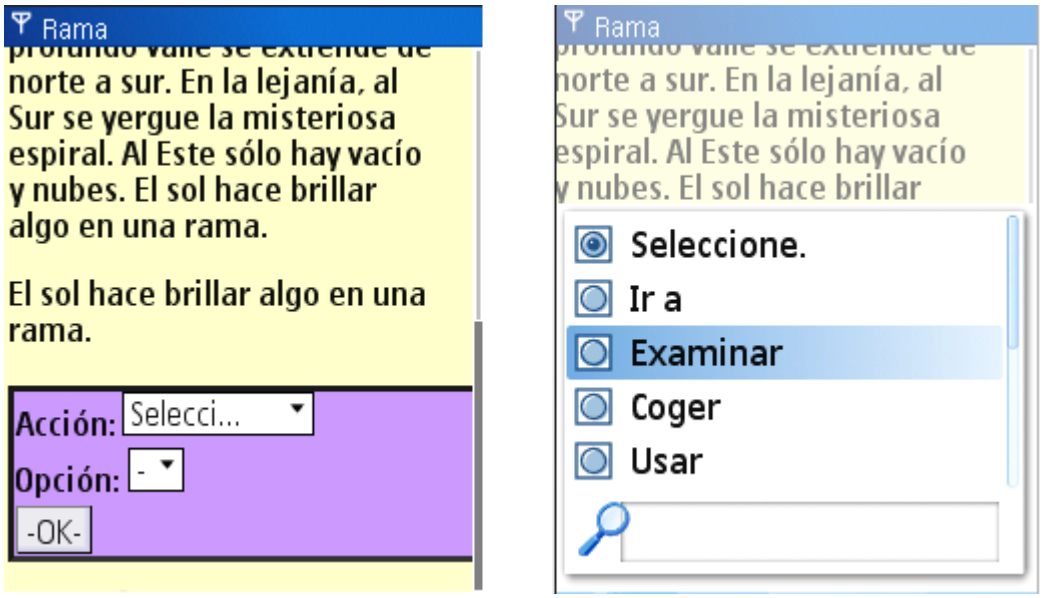

Figura 3.12 Menús y listas desplegables en XHTML

Tal y como se muestra en las Figuras 3.12 y 3.13, se ha escogido el uso de listas desplegables “select” o de menús en los que se muestran todas las opciones posibles. De esas opciones solo algunas serán las correctas a realizar en un momento determinado, mientras que el resto son proporcionadas para dar mayor complejidad y dinamismo a la historia.

El uso de un tipo de elemento u otro depende tan solo de la vista usada y de las características del lenguaje de marcas que soporte, siendo totalmente independiente del motor de la aplicación.

Figura 3.13 Menús y listas desplegables en WML.

Esta característica limita la variedad de combinaciones acción-opción que se le pudieran ocurrir al usuario si dispusiera de una entrada de texto en la que poder expresar sus ideas en cada momento, pero teniendo en cuenta las limitaciones de la mayoría de los teclados de los actuales dispositivos móviles, se ha considerado como la mejor opción, ya que se pierde variedad de acciones a realizar pero se gana en velocidad y dinamismo a la hora de jugar.

3.5.3 SOPORTE PARA IMÁGENES

Tal y como se ha desarrollado la aplicación, cada videojuego puede ser altamente configurable para que siguiendo una misma arquitectura se puedan crear diferentes historias. No obstante, todos deben seguir esa estructura prefijada la cual impone el uso de una imagen por cada localización o escena del videojuego.

En esencia ésto no presentaba ningún problema, pero surgió la posibilidad de crear una aventura conversacional que solo dispusiera de texto. En primer lugar para ofrecer mayor variedad a la hora de crear aventuras para esta plataforma, y en segundo lugar para mejorar la velocidad del juego para aquellos terminales que no dispongan de las últimas tecnologías.

Por este motivo se modificó ligeramente el código de las vistas usadas, de forma que si al leer la URL de una imagen que debe ser mostrada del archivo “map.xml” correspondiente al usuario, esta está vacía, entonces no se muestra nada al usuario, ni una imagen errónea, ni un texto alternativo, tan solo se muestra el texto asociado a esa localización, consiguiendo de esa forma que una misma vista soporte imágenes y texto, o solo texto.

En las figuras 3.14 y 3.15 se muestran varios ejemplos de aventuras desarrolladas en este proyecto, tanto para el caso de WML como XHTML, así como para el uso de imágenes o su omisión.

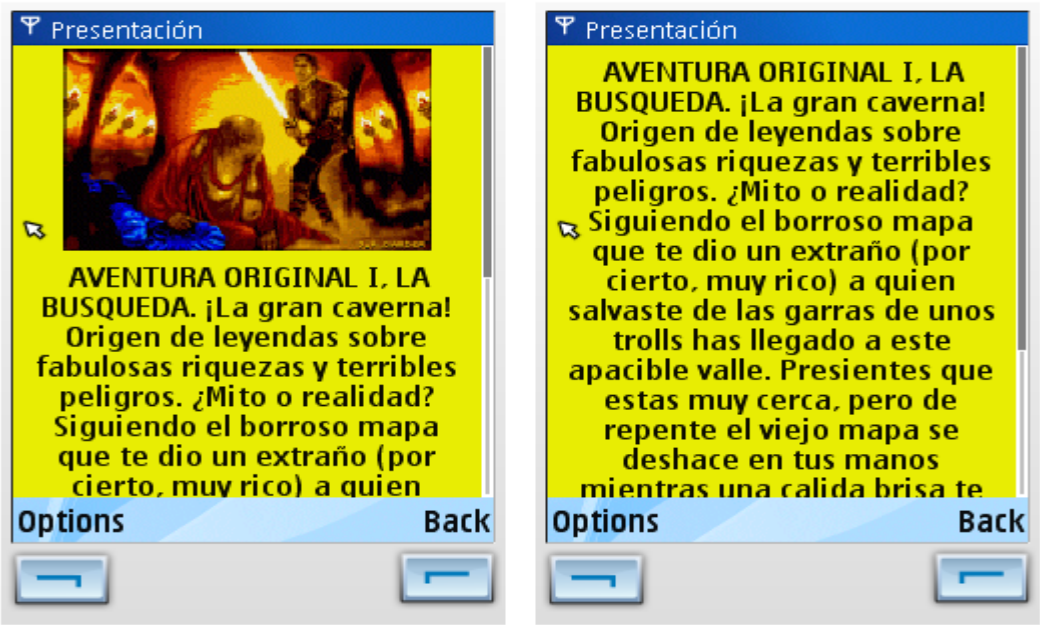

Figura 3.14 Imágenes de la Aventura Original para la vista XHTML

Figura 3.15 Imágenes de la Aventura Original para la vista WML

Es por ello que para crear una aventura en la que no se muestren imágenes, tan solo hay que dejar vacío el elemento que define la URL de la misma en el archivo de configuración "map.xml", tal y como se muestra en la figura 3.16.

```
<screen2>
    <title>Cabaña</title>
    <text>
        Estás en una habitación bastante guarra. Las paredes desconchadas y el suelo roto indican
        que hace tiempo fue abandonada. Ves la oscura boca de un pozo por la que baja una
        herrumbrosa escalera.
    </text>
    <img></img>
   <direction>
        <d id="1" activ="si">Salir</d>
        <d id="3" activ="si">Bajar</d>
  </direction>
 </screen2>
```

Figura 3.16 Fragmento del archivo map.xml para el caso de no usar imágenes.

CAPÍTULO 4: MANUAL DE USUARIO

En este capitulo se hace referencia al manual del usuario, estando dividido principalmente en dos apartados. El primero de ellos enfocado a un usuario administrador del sistema, haciendo referencia a como se deben gestionar y crear los archivos necesarios para la creación de nuevos videojuegos, y el segundo apartado enfocado al usuario jugador, definiendo las características básicas para poder hacer uso del entorno proporcionado en función de las diferentes visas desarrolladas.

4.1 MANUAL DE USUARIO ADMINISTRADOR

Lo que se pretende con este apartado es proporcionar una guía básica a un usuario administrador sobre la confección de nuevos videojuegos para la aplicación. Esta guía definirá los pasos y acciones a llevar a cabo para confeccionarlos de forma ordenada y clara.

4.1.1 ESPECIFICACIÓN DEL VIDEOJUEGO

El primer paso a contemplar es la confección de la idea del videojuego. Se recomienda realizar un boceto del videojuego en el que se defina a grandes rasgos la situación en la que se desarrollará la acción del videojuego, las diferentes localizaciones en las que dividirá la aventura, los objetos que podrían ser necesarios, etc.

Esto es indispensable a la hora de realizar un desarrollo claro y ordenado de un videojuego para esta aplicación, dada la variedad de vistas y archivos que entran en juego. Igualmente es recomendable tener conocimientos básicos del lenguaje XML, aunque no es un requisito, dado la sencillez de los archivos empleados.

4.1.2 ELECCIÓN DE LA VISTA

El segundo paso es decidir para qué vista de todas las posibles se desea realizar el videojuego. En caso de ser una vista ya implementada en la aplicación (WML o XHTML), no habrá más que hacer uso de los archivos JSP de que esta está provista, pero en caso de utilizar un vista que no esté implementada (supóngase el caso de que se desea realizar un videojuego para una futura versión del protocolo WAP, o para otro protocolo similar), es necesario desarrollar esos archivos JSP que especifican como se adaptarán los contenidos al terminal móvil atendiendo a las características que este soporte.

El desarrollo de los archivos JSP para la creación de una nueva vista de usuario escapa de la intención de este apartado. No obstante se hace necesario unos mínimos conocimientos de programación Java, así como del lenguaje de marcas soportado por el protocolo elegido. A grandes rasgos y sin entrar en detalles, los archivos JSP son muy similares de una vista a otra, dado que el código puramente Java es siempre el mismo, y tan solo hay que variar las partes del lenguaje de marcas empleado y la forma en la que se introducen los valores de las variables Java en ese lenguaje de marcas.

Claro está la posibilidad de desarrollar un videojuego que tenga soporte para varias vistas (como en el caso de este proyecto que se han desarrollado tres aventuras y todas soportadas tanto en WAP 1 como en WAP 2).

Una vez elegida la vista deseada, se debe crear un directorio con el nombre del videojuego (o un nombre que le haga referencia) dentro del directorio cuyo nombre sea la vista escogida. En la Figura 4.1 puede verse un ejemplo en el que se han creado dos videojuegos para la vista XHTML, por ello dentro del directorio MotorConv se ha escogido el directorio XHTML, y dentro de este se han creado los directorios con los nombres de los videojuegos, en este caso “AO” que hace referencia a la versión adaptada de la “Aventura Original” y el segundo videojuego denominado “escapada” de creación propia para este proyecto.

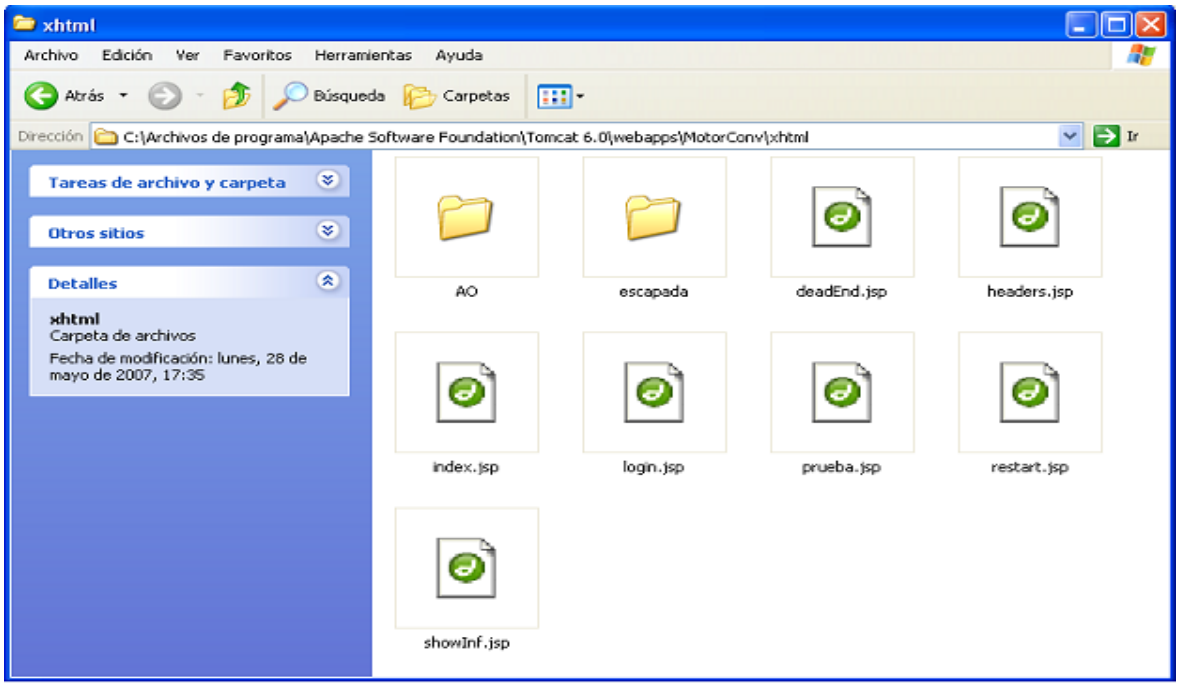

Figura 4.1 Estructura de directorios para nuevos videojuegos

Tal y como se aprecia en la Figura 4.1, los archivos JSP están incluidos en la vista, por lo que no es necesario volver a crearlos, tan solo será necesario crear los contenidos específicos del videojuego deseado.

4.1.3 ESTRUCTURA Y CONTENIDOS.

Para cada videojuego a desarrollar, y dentro del directorio que lleva su nombre se ha de crear una estructura de directorios como la mostrada en la Figura 4.2. Se deben crear esos tres directorios con esos nombres (teniendo cuidado con las mayúsculas y minúsculas). En el directorio “files” se crearán los archivos que configuran el videojuego, tal y como se vio en el apartado 3.3. En el directorio “images” se introducirán las imágenes necesarias a lo largo de todo el videojuego, y finalmente en el directorio “styles” se podrán introducir de forma opcional, archivos que configuren el aspecto gráfico, tal y como son las hojas de estilo.

Una vez creada la estructura de directorios, solo falta crear cada uno de los archivos necesarios para especificar las localizaciones, textos, nombre de las imágenes y cómo se reparten en el juego, objetos a usar, acciones, etc. que se detallaron en el apartado 3.3 del presente documento, teniendo en cuenta que el servidor debe ser capaz no solo de leer y servir dichos archivos, sino de escribir y modificar parte de sus contenidos, por lo que se deben tener activos los permisos de lectura y escritura en el directorio creado.

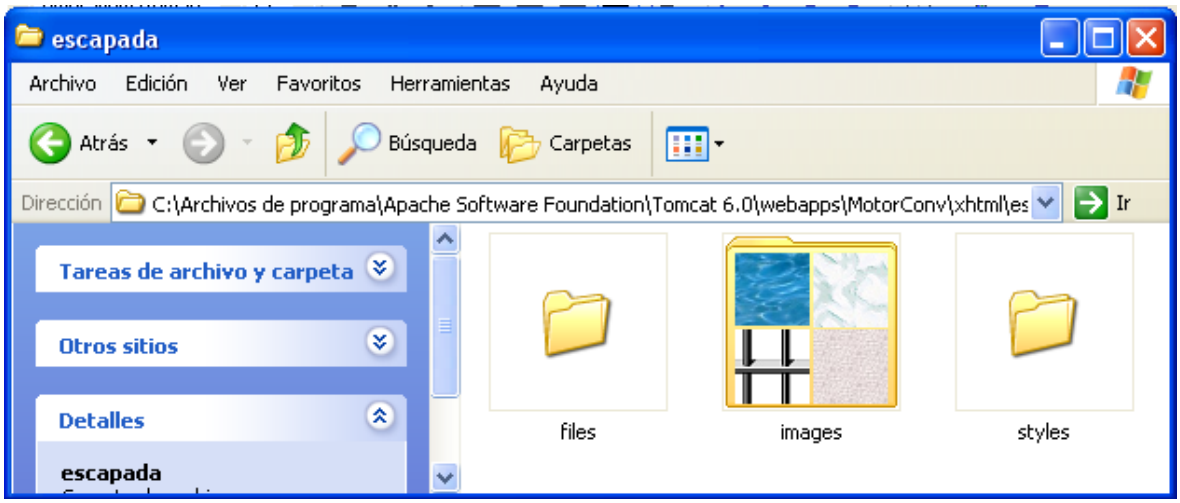

Figura 4.2 Estructura de directorios perteneciente a cada videojuego

Para realizar esto, se recomienda seguir una serie de pasos:

1. Confeccionar un mapa de forma esquemática conteniendo todas las localizaciones o escenas que se le podrán presentar al usuario, indicando para cada una de ellas

el nombre de la foto que se mostrará, el texto descriptivo de esa localización y sobre todo las posibles direcciones que podrá tomar el usuario, indicando a que localización le llevarán. Se recomienda realizar este esquema dibujándolo sobre papel o haciendo uso de un entorno gráfico que posibilite el entendimiento claro y total del mapa de la historia.

2. Una vez se tenga ese mapa de forma esquemática, se hace necesario pasar dicha información a un formato que sea legible a la aplicación. Esto se realiza creando o modificando el archivo “map.xml” perteneciente al directorio “files” del videojuego a desarrollar. En ese archivo se creará un elemento con el nombre de cada una de las localizaciones que participan en la aplicación: tanto las iniciales “<screenIndex>”,”<screenLogin>”, etc. como las específicas para cada una de las localizaciones en las que se podrá mover el usuario dentro de la historia planteada. Para cada una de las localizaciones “<screen id=’x’>” se incluirá el texto descriptivo que acompaña a la imagen, la URL de la imagen que se desee mostrar (que estará contenida en el directorio “images”), y las direcciones posibles a tomar desde esa localización, usando para ello el elemento “<direction>” y los sub-elementos “<d id=”x” activ=”si/no”> texto </d>” que especifican cada una de las direcciones.

3. El siguiente paso corresponde a la definición de los diferentes objetos de la aventura. Al igual que en el punto anterior, se recomienda colocar esos objetos sobre el mapa esquemático del videojuego, especificando dónde puede encontrar el usuario dicho objeto inicialmente, y adicionalmente algunas características del mismo, como puede ser el si puede cogerse o no.

4. Para hacer que los objetos sean accesibles desde la aplicación, es necesario pasar dicha información a formato electrónico, creando para ello el archivo “*object.xml*”, en el que no solo se definen los objetos y su situación inicial, sino una serie de atributos de los mismos que se deben especificar de la forma más detallada posible. Se recomienda a modo de organizar los contenidos de este archivo no definir los objetos que intervienen en la historia siguiendo el orden en que serán usados (ya que eso depende de las acciones que lleve a cabo el usuario, y como es lógico no todos los usuarios realizarán las mismas acciones en el

mismo orden), sino organizando los objetos según en la localización del juego en que aparecen por primera vez.

No obstante eso no varia el desarrollo del juego, tan solo facilita la posibilidad de modificar dichos objetos en caso de ser necesario. Así, para cada objeto a crear en la historia se deben especificar todos y cada uno de los atributos de los que consta el elemento “<ob id=’w’ screen=’x’ cogible=’y’ usable=’z’>, los cuales ya se estudiaron en detalle en el apartado 3.3.3.

Adicionalmente, se podían añadir una serie de textos descriptivos de las diferentes acciones que se podía realizar sobre dicho objeto, así como una descripción propia del objeto que se mostraba al usuario al llegar a la localización del juego en la que se encuentra el objeto tratado.

5. Finalmente, y no por ello menos importante, se deben especificar las diferentes acciones a llevar a cabo en la aventura para poder avanzar (archivo *actions.xml*). Se pueden especificar todas las acciones que se deseen, y se recomienda encuadrarlas o agruparlas no por el tipo de acción a realizar, sino por la localización en la que se hace necesario su uso, facilitando de esa forma la comprensión lógica de las acciones a lo largo del juego. Se puede pensar que ordenarlas de forma cronológica sería la mejor opción, pero dado que en este tipo de aventuras la cronología de las acciones no es fija, sino que depende de las acciones y preferencias del usuario, ésta no es una opción recomendable.

   Inicialmente puede servir de ayuda el realizar una descripción de cómo podría solucionarse toda la aventura, a forma de posible solución del creador. De esta forma se evita el olvidar algún objeto u acción necesaria en el transcurso de la historia. Una vez hecho esto, se separan los objetos y las acciones en sus correspondientes archivos, y se organizan en función de las localizaciones en las que se hacen necesarias.

   Para definir las acciones que puede llevar a cabo un usuario, se recomienda leer el apartado 3.3 en el que se detalla la estructura y el comportamiento de cada uno de los archivos y elementos.

   Se recomienda especial precaución a la hora de definir una nueva acción, el especificar de forma correcta el objeto o los objetos sobre los que se realizará, teniendo en cuenta que en todo momento los nombres de los objetos deben haber sido definidos con anterioridad en el archivo *object.xml.*

6. Seguramente en la definición de las acciones se hace necesario modificar objetos que ya habían sido definidos (alterando alguno de sus atributos o redefiniendo alguno de sus textos), o incluso crear objetos que no se tuvieron en cuenta en el paso 3. Es por eso que el desarrollo de un videojuego no es tan sencillo ni secuencial, sino que se hace necesario seguir varias veces éstos pasos, hasta que el videojuego quede completamente confeccionado.

7. Una vez se ha completado la creación de los archivos implicados en la definición del videojuego, y antes de probarlo, es recomendable revisar la estructura y la correcta sintaxis de estos documentos. Para ello, tal y como se especifica detalladamente en el apéndice A se puede hacer uso de un validador de documentos XML y hacer uso de documentos que restrinjan su arquitectura, como son los DTD (*Document Type Definition*) o los XSD (*XML Schema Definition*).

4.2 MANUAL DE USUARIO JUGADOR

Para poder hacer uso de cualquier videojuego de la aplicación por parte de un usuario, tan solo es necesario que éste haya accedido mediante una conexión de Internet al servidor que los contenga. Una vez se haya accedido a la aplicación, la forma de jugar depende en gran medida de la vista que el usuario haya escogido, ya que la forma de presentar los contenidos varia mucho de una vista a otra.

En este apartado se van a tratar las características de las dos vistas generadas en este proyecto: XHTML y WML.

4.2.1 MANUAL DE USUARIO PARA VISTA XHTML

Una vez se accede a la aplicación y se elige el juego deseado dentro de la vista XHTML, se muestra una presentación del juego y se pide una simple confirmación, tal y como muestra la figura 4.3.(a). Tras confirmar que se desea jugar a esa aventura, se muestra un formulario en el que se debe introducir nombre y contraseña a modo de identificación. Para el caso de que no se esté registrado previamente para esta aventura, será necesario acceder a la fase de registro haciendo uso del enlace mostrado en la parte inferior de este formulario, tal y como se muestra en la figura 4.3.(b). El nuevo formulario mostrado en la figura 4.3.(c) debe ser completado solo una vez, ya que en próximos accesos se podrá hacer uso del formulario de la figura 4.3.(b).

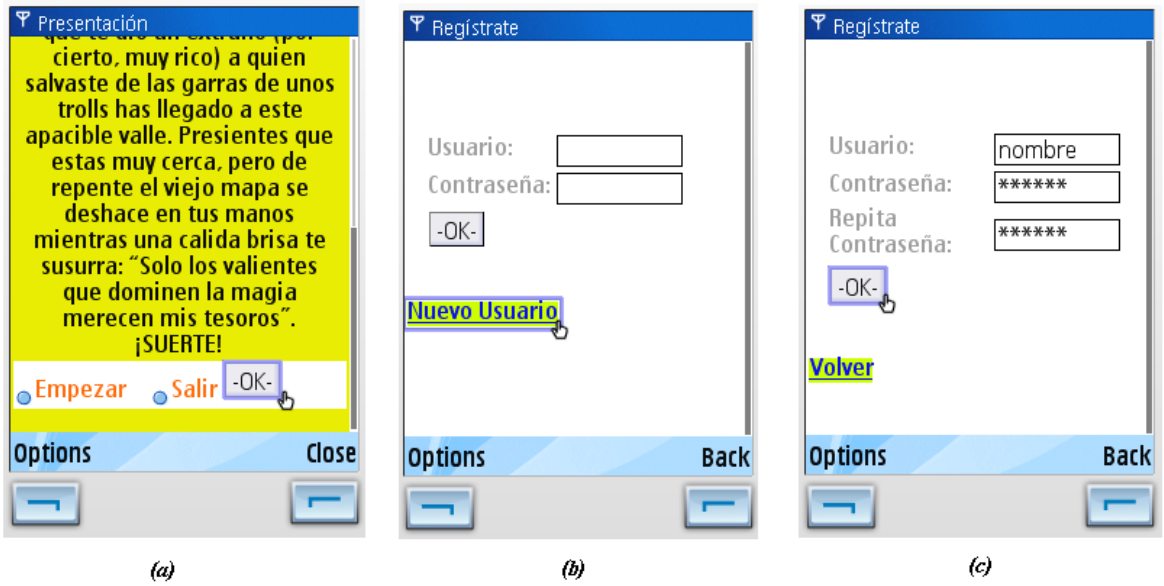

Figura 4.3 Acceso e identificación de nuevo usuario para la vista XHTML

Una vez la identificación ha sido realizada correctamente, se muestran dos enlaces para dar la posibilidad de retomar la aventura desde el principio, o continuar por donde se dejó la última vez (esta opción no está disponible la primera vez que se accede, ya que carece de sentido).

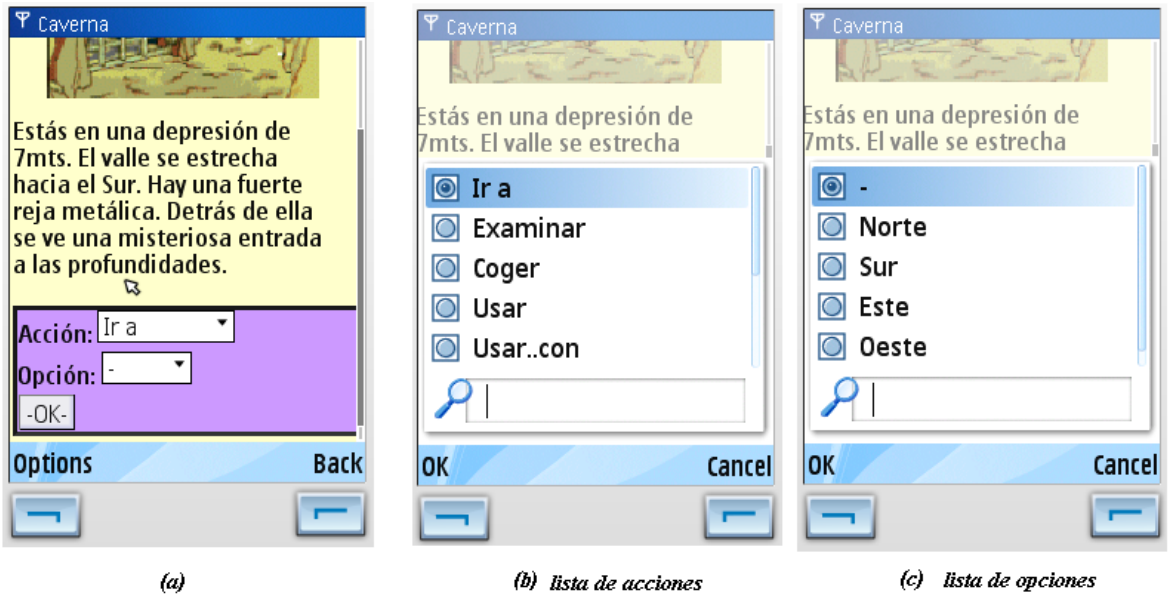

Figura 4.4 Información y formularios enviados al usuario según localización

Tras escoger la opción deseada se accede a la aventura escogida previamente, mostrándose la información de la localización en la que nos encontremos. Los contenidos son mostrados siempre siguiendo la estructura mostrada en la figura 4.4.(a) donde se muestra una imagen en la parte superior, seguida del texto descriptivo y a continuación el

formulario en la que escoger la acción y la opción. Tal y como se ve en las figuras 4.4.(b) y 4.4.(c) se debe escoger primeramente una acción, para luego escoger una de las opciones sobre la que realizar esa acción, las cuales cambian de forma automática al variar la acción seleccionada.

Una vez escogida la acción deseada de entre la lista mostrada y su correspondiente opción, se podrá pulsar el botón de confirmación (OK), enviando dicha petición al motor de la aplicación, el cual nos responderá con una nueva localización (o la misma si la acción no conlleva un desplazamiento), junto con una posible alerta en la que se nos mostrará información adicional acerca de la acción que se ordenó realizar, indicando si pudo o no realizarse, así como información acerca de algún objeto que pueda ser de ayuda en el planteamiento de la historia.

4.2.2 MANUAL DE USUARIO PARA VISTA WML

Al igual que para la vista XHTML, en esta vista se muestran las mismas opciones a la hora de confirmar que se desea jugar al juego que se presenta, las mismas opciones para la identificación y creación de la cuenta de usuario, así como elección de continuar la partida o comenzar de nuevo la aventura. La única diferencia es la forma de presentar los contenidos debido al uso de lenguajes de marcas diferentes y a las inferiores características gráficas que suelen tener los dispositivos que soportan tan solo la primera versión del protocolo WAP, tal y como se aprecia en la figura 4.5.

Figura 4.5 Acceso e identificación de nuevo usuario para la vista WML

Una vez se ha accedido al juego deseado, se ofrece al usuario la misma información que en el caso de la vista XHTML, pero la forma en la que el usuario puede escoger la acción y

la opción, varia considerablemente a la anterior vista debido a las limitaciones de los lenguajes WML y WMLScript, los cuales no pueden variar de forma automática las opciones a escoger en función de la acción elegida, tal y como ocurría en la vista XHTML.

Figura 4.6 Información y formulario para la vista WML

Por ello, tal y como se muestra en la figura 4.6 se dispone de una lista en la que se debe seleccionar la acción deseada, y a continuación se debe pulsar el botón (ver opciones) que genera las opciones correspondientes a dicha acción, ofreciéndolas en una nueva pantalla (una nueva carta WML). Dado que ahora no se generan automáticamente las opciones de cada acción, y que por ello no se tiene la posibilidad de modificar la acción una vez que se ha pulsado el botón para ver las diferentes opciones, se dispone al final de todas las opciones de un enlace que regresa a la pantalla en la que se muestra el formulario inicial, a forma de poder rectificar la acción elegida.

En esta vista no se dispone de un botón de confirmación, sino que al escoger una de las opciones mostradas, la petición se envía directamente al motor de la aplicación, el cual, tal y como sucedía con la vista XHTML enviará la información de la nueva localización, así como una alerta con la información adicional sobre el desarrollo de la acción requerida.

CAPÍTULO 5: PLAN DE PRUEBAS

En este capitulo se detallan las pruebas realizadas a la aplicación para comprobar y verificar su correcto funcionamiento una vez ésta ha sido completada. Igualmente y en adición a las pruebas de funcionamiento, se detallan algunas otras medidas realizadas a modo de aportar información relevante para el uso de la aplicación en un contexto real.

5.1 MEDICIONES DE TRÁFICO GENERADO

5.1.1 CONSIDERACIONES PREVIAS

Dado que la finalidad de esta aplicación es proveer de un motor de videojuegos conversacionales sobre WAP, y que a día de hoy las comunicaciones móviles por WAP son tarificadas por el tráfico generado y recibido por el terminal, se hace muy necesario e interesante poder estimar a priori el coste que le generaría a un usuario hacer uso de esta aplicación desde su terminal.

No obstante, estas mediciones poseen a su vez una gran importancia a la hora de estimar aspectos como el tiempo de transferencia medio de la formación correspondiente a cada localización.

Para realizar las medidas del tráfico generado por la aplicación, se ha hecho uso de dos ordenadores del departamento que se encuentran conectados por la red interna de la escuela, y que posibilitan de esa forma las mediciones de tráfico a través del interfaz de red utilizado.

En uno de los ordenadores se instaló el servidor de aplicaciones Tomcat, mientras que en el otro (que funcionaba como cliente) se instalaron los diferentes emuladores necesarios para poder realizar las medidas de tráfico en las diferentes vistas soportadas. Es por ello que se instaló el emulador para la versión XHTML (Emulador Nokia Serie 60), y el emulador necesario para WML (Openwave SDK 5.1).

El cómo se elegir que ordenador funciona como cliente y cual como servidor, es un dato poco significativo si se tiene en cuenta que en la práctica, las comunicaciones por el medio radio son mucho más determinantes que la diferencia en la capacidad de procesamiento de los ordenadores usados en las pruebas. Además las pruebas son realizadas en la cantidad del tráfico y no en la velocidad en que este es servido.

Para las mediciones de tráfico se utilizó el analizador de protocolos de red Ethereal, el cual, como todos los programas usados en este proyecto es software libre.

5.1.2 MEDICIONES REALIZADAS

Para estimar el tráfico generado por la aplicación de una forma objetiva y transparente a las características de cada usuario (sus gustos, preferencias por unas acciones u otras, conocimientos previos de este tipo de videojuegos, soltura en el manejo del dispositivo móvil, etc.), se han realizado medidas atendiendo a las diferentes acciones posibles, así como a las consecuencias que éstas puedan traer.

Al realizar las medidas con Ethereal, se ha hecho uso de un filtro para desestimar todo el tráfico que pudiera aparecer durante las medidas que no tuviera que ver exactamente con el de la aplicación. Por eso se utilizó un filtro que solo dejaba pasar el tráfico proveniente de HTTP sobre TCP.

Se realizó un plan de mediciones atendiendo primeramente a las diferentes vistas soportadas por la aplicación, ya que el uso de una vista u otra modifica en gran manera no solo la forma en la que la información es presentada (lo cual varia la cantidad de información), sino también la forma en que ese tráfico es enviado por el medio de transmisión.

Para cada vista se detalló un plan de mediciones correspondiente a las diferentes acciones que el usuario puede realizar en el juego. Para cada tipo de medición se tomaron entre 5 y 10 medidas, de forma que la información obtenida fuera concisa y suficiente para realizar un promedio estadístico correcto.

No obstante, para todas las mediciones realizadas se debe tener en cuenta que las condiciones en las que realizaron no son las reales, dado que en las medidas no se dispone de las características del medio radio, el cual por ejemplo, comprime los datos antes de enviarlos reduciendo el tráfico. Es por ello que las medidas que a continuación se detallan son meramente datos orientativos y en ningún caso pretenden ser un fiel reflejo de la realidad.

5.1.2.1 Mediciones para XHTML

En las mediciones de la versión XHTML, se hizo uso del emulador de la Serie 60 de Nokia funcionando como cliente del servidor Tomcat instalado en otro ordenador. Se realizaron las medidas con el videojuego “AO”, adaptación de “La Aventura Original” el cual consta de un conjunto de imágenes cuyo peso medio es de 10 KBytes. El peso de las imágenes es muy importante y debe ser el menor posible, garantizando eso si una cierta calidad gráfica.

En la Tabla 5.1 se muestrean las diferentes medidas realizadas y el valor promedio obtenido del tráfico generado para las diferentes acciones y situaciones mostradas al usuario.

Tipo de acción	**Tráfico promedio (KBytes)**
Acceso a la aplicación (index)	45
Identificación del usuario (login)	7
Acceso a la primera localización (reiniciando o continuando por donde se dejó)	30
Ir A (una nueva localización)	24
Ir A (localización conocida, acierto caché)	13,2
Coger …	13,2
Dejar …	13,2
Examinar …	13,2
Usar …	13,3
Usar … con …	13,4
Morir (acción que produce la muerte)	9,3
Morir en el mismo lugar (acierto caché)	3,2

Tabla 5.1 Medidas de tráfico para la vista XHTML

Tal y como se aprecia en la tabla, hay que tener en cuenta la caché del dispositivo móvil a la hora de realizar las medidas de tráfico. En este caso, y dado que el emulador también dispone de caché, se han realizado diferentes medidas, algunas reiniciando la caché del emulador y otras permitiéndole que acceda a los contenidos ya descargados sin tener que descargarlos de nuevo del servidor, reduciendo notablemente el tráfico por la interfaz.

Esto se aprecia por ejemplo en el movimiento del usuario por las diferentes localizaciones del juego, donde si la localización a la que accede es nueva para él, entonces tendrá que descargar del servidor tanto la descripción y la imagen del lugar, como los datos anexados a la localización como son los diferentes objetos que posee o que aparecen en esa localización. Por el contrario, si al moverse con el comando “IR A”, el usuario es desplazado a una localización que había visitado previamente (y que aún se encuentra en la caché del dispositivo), se produce un acierto en caché de la imagen mostrada que no tendrá que ser descargada de nuevo del servidor, por lo que se reduce el tráfico generado. No

obstante el resto de información sí es necesaria que se descargue de nuevo del servidor, dado que los contenidos son dinámicos y cambian constantemente, por lo que aunque se acceda a una localización ya visitada los datos de objetos o acciones realizadas pueden ser diferentes.

Tal y como se comprueba en la tabla hay una diferencia de aproximadamente 10KB en el tráfico medido atendiendo si hubo o no acierto en caché al moverse de localización, cantidad que corresponde en su mayoría al tamaño de la imagen.

El resto de las acciones suelen tener asociado un tráfico similar a un acierto en caché, dado que esas acciones realizan modificaciones en los archivos de configuración del usuario, y retornan a la misma localización tan solo añadiendo una alerta en forma textual explicando las posibles consecuencias de esa acción. Dado que esa alerta es en formato texto, apenas modifica el tráfico que viaja por la interfaz, mientras que al tratarse de la misma localización, se consigue un 100% de acierto en caché, reduciendo el tráfico a los aproximadamente 13KB.

Estas medidas no son del todo correctas, teniendo en cuenta que se pueden dar casos en los que al realizar una acción no solo se añada una alerta explicativa, sino que se produzca un desplazamiento del usuario a otra localización del juego, en cuyo caso no se produciría forzosamente un acierto en caché, y por lo tanto el tráfico aumentaría aproximadamente a los 24KB de media, dado que sería necesario descargar una nueva imagen.

Es por ello que se pueden agrupar las medidas de tráfico en aquellas que corresponden a acciones que implican un cambio de localización (siendo ésta una localización no conocida previamente), y acciones que no implican el cambio de la localización, siendo estas ultimas medidas del tráfico inferiores a las primeras en un valor aproximado al peso de las imágenes usadas.

Se han realizado medidas separadas para las acciones que conllevan la muerte del jugador, debido a que la localización que muestra la muerte es diferente al resto de localizaciones del videojuego, lo cual se aprecia a simple vista del tráfico que se genera. Al igual que en casos anteriores, el acierto en caché reduce el tráfico generado, en este caso la imagen mostrada solo ocupaba 5Kbytes.

5.1.2.2 Mediciones para WML

En este caso se hizo uso del emulador contenido en el SDK 5.1 de Openwave funcionando como cliente de la aplicación. Se realizaron las mismas medidas de tráfico que en el caso de XHTML, para lo que se usó igualmente el videojuego “AO” pero en su

versión de WML, lo que implica tamaños de archivos diferentes y a su vez imágenes más pequeñas, ya que al ser en formato WBMP ocupan tan solo 2KBytes de media.

En la tabla 5.2 se pueden apreciar las diferentes medidas realizadas, que tal y como se comentó al inicio de este capitulo son promedios de las diferentes repeticiones realizadas a cada una de las acciones mostradas.

Tipo de acción	**Tráfico promedio (KBytes)**
Acceso a la aplicación (index)	8
Identificación del usuario (login)	8
Acceso a la primera localización (reiniciando o continuando por donde se dejó)	15
Ir A (una nueva localización)	31
Ir A (localización conocida, acierto caché)	31
Coger …	34,4
Dejar …	34,4
Examinar …	37
Usar …	37,2
Usar … con …	37

Tabla 5.2 Medidas de tráfico para la vista WML

El tráfico generado por la aplicación para la versión WML es en la mayoría de los casos muy superior a la de XHTML, mientras que las imágenes en formato WBMP ocupan mucho menos que las de formato PNG usadas en XHTML. El motivo de este aumento del tráfico se debe al mal funcionamiento del emulador de Openwave al trabajar como cliente de un servidor externo a la propia maquina. Este error se ha detectado gracias al analizador de protocolos, en el que estudiando los paquetes recibidos y enviados entre cliente y servidor, se detectan varios fallos o incongruencias:

1. El sistema de caché del emulador no funciona correctamente, dado que se producen un número excesivo de fallos a caché en situaciones en las que no deberían producirse. Esto es el motivo por el cual el tráfico generado cuando se produce un acierto en caché (supuestamente) es muy parecido a cuando no se produce dicho acierto. Según el emulador el sistema de caché funciona correctamente, pudiendo visualizar desde el propio emulador los archivos que

éste contiene en su memoria caché, pero al acceder a ellos nuevamente, el emulador produce un fallo a caché y éstos vuelven a descargarse del servidor, aumentando de forma considerable el tráfico generado.

2. Al acceder a una carta dentro de la misma baraja, se produce un nuevo fallo a caché y el emulador da la orden de descargar de nuevo el archivo completo. Recordemos que una página WML está compuesta por una baraja con una o varias cartas dentro de ella, y que al descargar la página el navegador solo presenta una de esas cartas en cada momento. No obstante un funcionamiento correcto es el poder navegar entre las diferentes cartas de una misma baraja de forma local al navegador del dispositivo móvil, lo cual no genera ningún tipo de tráfico entre el cliente y el servidor. Usando el emulador mencionado, el navegar entre las diferentes cartas de una misma baraja sí produce tráfico, con lo que el tráfico generado total es en todos los casos muy superior al que le corresponde. En la tabla 5.3 se describen algunas medidas realizadas sobre este error, describiendo de forma aproximada el tráfico generado al moverse entre cartas de una misma baraja.

Tipo de acción	Tráfico promedio (KBytes)
Ver opciones de una acción	11,1
Volver a la primera carta de la baraja	12,2

Tabla 5.3 Medidas de tráfico generado al moverse dentro de una baraja

Estos errores del emulador producen que al acceder a una localización nueva, nunca se tenga en caché ni el archivo de script, ni la imagen a mostrar, lo cual aumenta el tráfico. Pero lo que es más grave, el realizar una acción produce una serie de pasos que generan mucho tráfico indebido:

- Petición de la página WML deseada, y muestra de la primera carta. (correcto).
- Petición del archivo WMLScript. (incorrecto salvo para la primera vez).
- Petición nuevamente de la página WML para poder mostrar las opciones a la acción requerida. (incorrecto puesto que se trata de mostrar una carta de la misma baraja).
- Para el caso de una acción que no implique el movimiento del jugador a otra localización, se realiza una nueva petición de la página WML a mostrar, sobre la

que se superpone la alerta con el texto explicativo de esa acción. (incorrecto puesto que la página debería estar en caché).

- Para el caso de elegir una acción inicialmente, y antes de elegir la opción regresar para cambiar la acción, se realiza una nueva petición de la página WML. (incorrecto puesto que es otra carta de la misma baraja).

Todos estos pasos generan un tráfico a la vista muy superior al correspondiente, y es por ello que las medidas obtenidas no son significativas para un entorno en el que la memoria caché del dispositivo funcione correctamente, así como la capacidad de navegar entre cartas de una misma baraja de su navegador.

Adicionalmente a todos éstos problemas del emulador, se debe tener en cuenta que al realizarse las medidas con un sistema basado en ordenadores que simulan el cliente y el servidor, se omite el uso de la pasarela WAP, la cual tal y como se mencionó en el capitulo 2 del presente documento, realiza tareas de compresión al acceder al medio radio, reduciendo el tráfico.

5.2 PLAN DE PRUEBAS DETALLADO

Las pruebas que a continuación se detallan, son las pruebas realizadas a la aplicación para verificar el correcto funcionamiento de la misma ante los diferentes escenarios planteados. No obstante las pruebas descritas en este apartado son tan solo un resumen de todas las pruebas realizadas durante la fase de diseño de la aplicación.

5.2.1 PRUEBAS CON EMULADOR Y SERVIDOR LOCAL

Las primeras pruebas realizadas se basaron en comprobar el funcionamiento general de la aplicación en el entorno en el que ésta fue desarrollada, es decir, usando un emulador para visualizar los resultados y el servidor Tomcat, ambos de forma local en la misma maquina. Las pruebas realizadas en esta fase se mencionan a continuación, y todas fueron realizadas para las dos vistas creadas en este proyecto (WML y XHTML):

- Correcto acceso a la aplicación desde el emulador.
- Acceso al índice de la aplicación.
- Correcta creación de un usuario nuevo (registro), generando la carpeta con su nombre y copiando los archivos necesarios.

- Envío del mensaje de error al intentar registrar un usuario con un nombre que ya estaba registrado.
- Correcta creación de un nuevo usuario usando el mismo nombre que el de otro usuario, pero en diferentes juegos.
- Envío del mensaje de error correspondiente al dejar vacío un campo del formulario cuando éste debe estar relleno.
- Identificación de usuario registrado.
- Envío del mensaje de error al intentar identificarse como usuario registrado cuando no es cierto.
- Acceso al videojuego retomando la partida por donde se dejó la última vez que se jugó.
- Acceso al juego tras reiniciar los archivos de configuración, para poder comenzar de nuevo desde el principio.
- Correcta presentación de los diferentes contenidos, imágenes, textos, formularios, etc.
- Funcionamiento de los diferentes formularios presentados, permitiendo al usuario poder visualizar todas las opciones.
- Correcto funcionamiento de los scripts que modifican el formulario de las opciones en función del valor del formulario de las acciones.
- Movimiento del usuario por las diferentes localizaciones del mapa de un videojuego usando el comando “IR A”.
- Coger y dejar objetos que se muestran al usuario, modificando correctamente el inventario.
- Coger un objeto y dejarlo en otra localización diferente, mostrando de forma correcta el texto descriptivo del mismo en la nueva localización.
- Imposibilidad de coger objetos cuyo atributo “cogible=no”.
- Examinar un objeto, mostrando su texto en una alerta del navegador.
- Abrir y cerrar objetos que permitan dicha acción, posibilitando la acción correspondiente. (Ejemplo, al abrir una puerta, la posibilidad de pasar por ella).
- Usar objetos unos con otros siempre y cuando esté especificado en los archivos de configuración.
- Imposibilidad de usar objetos cuyo atributo “usable=no”.
- Correcta generación de consecuencias ante las diferentes acciones que las posibilitan.

- Correcta modificación de los archivos de configuración del usuario ante una acción u movimiento a otra localización.
- Muerte del usuario al realizar una acción cuya finalidad es ésta.
- Presentación de la localización que indica la muerte del jugador.
- Presentación de la localización de fin de la ventura.
- Correcto reinicio del juego tras la muerte o fin de la aventura, si el usuario lo desea.

5.2.2 PRUEBAS CON EMULADOR Y SERVIDOR EXTERNO

Tras comprobar el correcto funcionamiento de la aplicación usando el servidor y el emulador en la misma máquina, el siguiente paso realizado fue el comprobar el funcionamiento para el caso en el que el servidor Tomcat se encontrara en otra máquina diferente a la del emulador.

De esta forma se comprobaba el acceso desde un emulador a un servidor, sin importar dónde estuviera este, usando para ello la dirección IP y el puerto del servidor si fuera necesario.

Las pruebas realizadas fueron las mismas que en el caso anterior, a las que se le añadieron una serie de pruebas que solo tienen sentido en este nuevo contexto.

A continuación se nombran estas pruebas añadidas:

- Correcto acceso a la aplicación, usando la IP conocida del servidor.
- Correcto acceso a las diferentes vistas del juego, recibiendo los tipos MIME correctos.
- Funcionamiento con 2 usuarios de la misma vista y mismo juego. Esto es 2 o más jugadores jugando a un mismo videojuego en una misma vista de forma simultanea.
- Funcionamiento correcto con 2 usuarios de la misma vista en diferentes videojuegos.
- Funcionamiento correcto de 2 usuarios de diferentes vistas.

5.2.3 PRUEBAS CON DISPOSITIVOS MÓVILES REALES

Finalmente y tras haber pasado satisfactoriamente las pruebas del emulador, se realizaron una serie de pruebas en móviles reales. El planteamiento inicial era buscar

terminales móviles que cumplieran los requisitos necesarios para soportar cada una de las dos vistas generadas para este proyecto (WML y XHTML). Dada la dificultad de encontrar un móvil real que soportara WML junto con su homónimo WMLSCRIPT, se optó por realizar las pruebas solo para la versión de XHTML, teniendo en cuenta que la única diferencia respecto a WML es la forma de presentar los contenidos, ya que el motor de la aplicación es el mismo en ambos casos.

Para estas pruebas se usó un terminal móvil del departamento de Tecnología Electrónica, concretamente un Nokia N93 perteneciente a la serie S60 de Nokia que da soporte tanto para XHTML como para WML. El problema está en que solo soporta ECMASCRIPT y no WMLSCRIPT, por lo que no es un terminal válido para realizar las pruebas para la vista WML, siendo tan solo usado para el desarrollo de las pruebas de la vista XHTML.

En dicho terminal se realizaron las pruebas descritas en el apartado 5.2.1, usando como servidor en el que alojar el motor de la aplicación el servidor de la escuela.

CAPÍTULO 6: CONCLUSIONES Y LÍNEAS FUTURAS

El objetivo de este proyecto fin de carrera ha sido el diseñar e implementar un motor destinado a ejecutar videojuegos conversacionales para dispositivos móviles, así como el establecer una estructura interna del motor que posibilite de forma sencilla la creación de nuevos videojuegos o su modificación.

El motor desarrollado está basado en el servidor de aplicaciones Tomcat, sobre el que se monta la estructura de directorios y archivos que conforman la aplicación. Para conseguir los objetivos planteados ha sido necesario desarrollar varias alternativas, que a lo largo del desarrollo se han ido descartando o modificando según los inconvenientes o ventajas que planteaban.

Finalmente se ha conseguido desarrollar una aplicación que cumple las siguientes características:

- Aplicación totalmente independiente a los videojuegos que proporciona. Se ha desarrollado el motor de tal forma que los archivos (basados en *servlets*) que ejecutan los videojuegos creados, son totalmente independientes a estos, lo que posibilita la creación de nuevos videojuegos sin modificar en absoluto la estructura interna.
- Capacidad de soportar la ejecución de varios videojuegos conversacionales de forma simultánea, posibilitando de esa forma tener varios jugadores para cada una de las aventuras activas.
- Capacidad para adaptarse a las características de los terminales móviles ofreciendo la posibilidad de elegir entre varias vistas de un mismo videojuego. Esto es que cada vista adapta los contenidos del videojuego de una forma diferente, atendiendo a las características del lenguaje de marcas y de las restricciones del terminal al que se destina. En el presente proyecto se han desarrollado a modo de ejemplo dos vistas correspondientes a las dos versiones del protocolo WAP: WML y XHTML. No obstante la aplicación es capaz de soportar un número indefinido de vistas diferentes, dejando abierta la posibilidad de añadir nuevas vistas que adapten los contenidos a futuros protocolos inalámbricos.

- Facilidad para la generación de nuevos videojuegos conversacionales. Cada videojuego conversacional de esta aplicación está basado en tan solo cuatro archivos XML que definen de forma sencilla y estructurada toda la información necesaria para su ejecución. Al tratarse de un lenguaje sencillo, no es necesario tener conocimientos de programación para la creación de nuevas aventuras, ampliando de esta forma el rango de personas que pudieran estar interesadas en este propósito.
- Alta configurabilidad del idioma, formato de las imágenes y formato de la presentación, lo cual se consigue basando la presentación de los contenidos en un acceso previo a la información que almacenan los diferentes archivos XML que configuran cada videojuego, y a las diferentes hojas de estilo de la vista seleccionada. Consiguiendo de esta forma que modificando un simple archivo XML se pueda obtener toda la aplicación en otro idioma, o presentada de forma muy diferente a la desarrollada en este proyecto.
- Posibilidad de validación de los diferentes archivos XML, evitando gran cantidad de errores a la hora de su creación y modificación. La validación de cada uno de los archivos XML que conforman un videojuego se realiza usando XML-Schema (XSD.

Todas las aventuras conversacionales desarrolladas para ser ejecutadas en este motor comparten a su vez una serie de características, basadas en las posibilidades que la plataforma ofrece:

- Cada videojuego conversacional está formado por cuatro archivos en formato XML:
    - user.xml → Contiene información del usuario y la localización del mismo en el mapa del juego en cada momento.
    - map.xml → Contiene información sobre las diferentes localizaciones en las que se desarrolla la aventura, indicando para cada una de ellas los textos, imágenes y posibles direcciones a tomar.
    - objetc.xml → Descripción de las características de los diferentes objetos que entran en juego en el desarrollo de la aventura.
    - actions.xml → Especifica las diferentes acciones que el usuario puede realizar, y sobre que objetos puede realizarlas, encauzando la historia según el creador de la misma.

- Compuestos por varias localizaciones comunicadas entre si formando un mapa.
- Cada localización está compuesta a su vez por una imagen, un texto descriptivo, y un formulario en el que el jugador ordenará nuevas acciones especificando la acción que desea realizar y el objeto sobre el cual se realizará. Adicionalmente existe una alerta (texto que se superpone momentáneamente a la información anterior), para el caso de ser necesaria en la explicación de algún suceso acontecido. La inclusión de la imagen es opcional, posibilitando de esta forma la creación de aventuras conversacionales de solo texto.
- En todas las aventuras, el jugador dispone de 9 acciones a poder elegir en cada momento: Ir a, Examinar, Abrir, Cerrar, Usar, Usar con, Coger, Dejar e Inventario. Estas acciones siempre están disponibles, mientras que los objetos sobre las que realizarlas varían según el transcurso de la aventura.
- Cuando un jugador hace una petición al motor indicándole una acción a ejecutar y uno o varios objetos sobre la que realizarla, el motor intenta ejecutarla, y en caso de que pueda ser llevada a cabo se producen una serie de consecuencias que modifican el estado del juego. Estas consecuencias pueden ser de tres tipos:
    - reacIra → Mueve al jugador a una nueva localización.
    - reacMod → Modifica el estado de uno de los objetos del juego, permitiendo crear nuevos objetos, hacer desaparecer alguno que ya existía, cambiando sus propiedades, etc.
    - reacModdir → Modifica una de las direcciones que puede tomar el usuario para moverse por el mapa del videojuego.
- Entre todas las consecuencias posibles, el motor ofrece la posibilidad de hacer que termine la aventura por la muerte del jugador, al realizar una acción que no era la adecuada, como usar una pistola o bajar por un precipicio.

No obstante, a pesar de haber cumplido todos los requisitos planteados, a lo largo del desarrollo de este proyecto han quedado puertas abiertas a posibles expansiones o mejoras del sistema:

- Desarrollar un entorno gráfico para la confección de nuevas aventuras conversacionales, evitando de esa forma que el usuario tenga que manejar los archivos XML que conforman cada videojuego. El usuario administrador tan solo debería ejecutar la aplicación del entorno gráfico para poder realizar la confección de nuevas aventuras conversacionales, de tal forma, que

introduciendo la información necesaria en ese entorno gráfico, ésta generara de forma automática los archivos XML necesarios para que el videojuego funcione en el motor desarrollado.

- Aumentar el número de acciones a poder realizar por el jugador en cada momento, añadiendo acciones más complejas que posibiliten el aumentar la diversificación de las historias creadas.
- Incluir NPC (Non-Playable Character) en las aventuras. Ésto es introducir personajes pásivos que sean manejados por el propio motor, ofreciendo la posibilidad de que el usuario interaccione con ellos para poder resolver la aventura, y posibilitando nuevas acciones como “hablar” con los NPC, “comprar, cambiar u ofrecer” objetos que aparecen en la aventura, etc.

Bibliografía

[1] http://www.adese.es/pdf/Datos-industria-2006.pdf. Informe de resultado anual correspondiente al año 2006 sobre la industria de ocio audiovisual, realizado por la aDesE.

[2] http://www.elmundo.es/suplementos/ariadna/2006/273/1143473736.html. Noticia del periódico El Mundo del 26 de Marzo de 2006 titulada "atados al móvil".

[3] http://es.wikipedia.org/wiki/Aventura_conversacional. Artículo relacionado con las aventuras conversacionales del portal Web de la Wikipedia.

[4] http://www.caad.es. Club de Aventuras AD - Página de la comunidad hispana de Aventuras Conversacionales y Relatos Interactivos.

[5] "Aventuras Conversacionales". Micromanía, Nº 9, Febrero 1989.

[6] J. García de Jalón, J.I Rodríguez, I Mingo, A Imaz, A Brazales, A Larzabal, J Calleja, J García, "Aprenda JAVA como si estuviera en primero", www.tecnun.es/asignaturas/Informat1/ayudainf/aprendainf/Java/Java2.pdf.

[7] http://www.javahispano.org. Portal Web dedicado al lenguaje de programación JAVA, con artículos y manuales en castellano.

[8] J. García de Jalón, J.I Rodríguez, A Imaz "Aprenda Servlets de Java como si estuviera en primero", www.tecnun.es/asignaturas/Informat1/ayudainf/aprendainf/JavaServlets/servlets.pdf.

[9] http://www.javahispano.org/articles.article.action?id=50. Artículo dedicado al manejo de documentos XML mediante JAVA.

[10] http://jdom.org. Portal Web dedicado a la librería JDOM para el manejo de documentos XML desde JAVA.

[11] http://es.wikipedia.org/wiki/MVC. Artículo dedicado a la arquitectura Modelo-Vista-Controlador del portal Web de la Wikipedia.

[12] Wap Forum, “Wireless Application Protocol White Paper”, June 2000

[13] Wap Forum,”Wireless Application Protocol WAP 2.0 Technical White Paper”, Enero 2002.

[14] http://www.openmobilealliance.org. Protal Web de la Open Mobile Alliance.

[15] Wap Forum, “Wireless Markup Language Version 2.0”, Junio 2001.

[16] NOKIA, “WML Script Reference Versión 1.1”, Septiembre 1999

[17] http://www.wmlclub.com/tutoriales. Portal Web dedicado al lenguaje de etiquetas WML, artículos, manuales, ejemplos, etc.

[18] http://www.developershome.com. Portal Web dedicado a la difusión de diversas tecnologías móviles: WAP/WAP 2.0, WML, WMLScript, XHTML-MP, WCSS.

[19] Wap Forum, “XHTML Mobile Profile”, Octubre 2001.

[20] http://tomcat.apache.org. Página Web del servidor de aplicaciones Tomcat.

[21] http://www.w3c.org. Página Web del organismo W3C.

[22] http://developer.openwave.com/dvl/. Portal Web de la red de desarrollo de Openwave.

[23] http://www.forum.nokia.com. Portal Web del Forum de Nokia.

[24] http://www.s60.com/life, Portal Web de la serie S60 de Nokia.

APÉNDICE A: FICHEROS XML PARA LA AVENTURA ORIGINAL.

A continuación se muestra el código fuente de los cuatro archivos que configuran la adaptación del videojuego La Aventura Original para el motor desarrollado en este proyecto fin de carrera. El estudio de cada elemento, así como su función se estudia en el apartado3.3 del presente documento.

- **user.xml**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- ********************************************************************************** -->
<!-- DESARROLLO DE UN MOTOR DE VIDEOJUEGOS CONVERSACIONALES SOBRE WAP PARA DISPOSITIVOS
MÓVILES                                                                                -->
<!-- Autor: JAVIER GONZÁLEZ MONROY                                                      -->
<!-- USER.XML V4  PARA XHTML                                                            -->
<!-- ********************************************************************************** -->

<!--ELEMENTOS DEFINIDOS:
        SCREEN -> Localización actual del juego en la que se encuentra el usuario (está inicializada a 1)
        PASSWORD -> Contraseña definida por el usuario en el momento de la creación de la cuenta

-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../../XSD/user.xsd">
        <screen>1</screen>
        <password></password>
</config>
```

- **map.xml**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- ********************************************************************************************** -->
<!-- DESARROLLO DE UN MOTOR DE VIDEOJUEGOS CONVERSACIONALES SOBRE WAP PARA DISPOSITIVOS
MÓVILES -->
<!-- Autor: JAVIER GONZÁLEZ MONROY                                    -->
<!-- MAP.XML V4  PARA XHTML                                        -->
<!-- ********************************************************************************************** -->

<!-- Este archivo contiene los textos y palabras a mostrar durante toda la aplicación, de forma que puede cambiarse de idioma
completamente modificando
        únicamente este archivo. Igualmente atiende a las rutas de las imágenes que se van a mostrar en cada una de las localización
del juego -->

<!--CONTENIDOS
        Los elementos de cada pantalla se engloban dentro de las etiquetas que lo definen <nombre_pantalla> ... </nombre_pantalla>
        Dentro de esta se definen una serie de elementos que hacen referencia al texto que mostrarán por pantalla.

        title -> Titulo de la pantalla a mostrar en el navegador del dispositivo móvil.
        text -> Texto a mostrar en esa pantalla, información, preguntas, etc.
        img -> Url relativa de la imagen a mostrar en esa pantalla. Hay que tener en cuenta la estructura de directorios para saber
desde donde se llama a cada imagen y así poder definir correctamente su URL relativa. Por norma general la ruta es
../xhtml/NOMBRE_ROOM/images/nombre_imagen.ext
        d (direction) ->Son las posibles direcciones a tomar por el usuario desde esa pantalla
                ID representa el valor de la nueva pantalla a la que se desplazará (numero de la pantalla o "End/Dead" para
                        terminar al juego)
                ACTIV..Indica si esa dirección está activa (puede usarse) o no
                TEXTO representa el texto que se mostrará por pantalla al usuario como indicativo de esa dirección
        dead... Texto mostrado cuando algunas de las direcciones de esa pantalla llevan a la muerte
-->

<map xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../../XSD/map.xsd">
        <!-- PANTALLAS DE LA INTRODUCCIÓN AL JUEGO -->
```

```
<screenIndex> <!--Presentación de la aplicación. Se muestra en  "index.jsp"-->
  <title>Presentación</title>
  <text>
            AVENTURA ORIGINAL I, LA BUSQUEDA.
            ¡La gran caverna! Origen de leyendas sobre fabulosas riquezas y terribles peligros. ¿Mito o realidad?
            Siguiendo el borroso mapa que te dio un extraño (por cierto, muy rico) a quien salvaste de las garras de unos trolls has
            llegado a este apacible valle.
            Presientes que estas muy cerca, pero de repente el viejo mapa se deshace en tus manos mientras una calida brisa te susurra:
            “Solo los valientes que dominen  la magia merecen mis tesoros”. ¡SUERTE!
  </text>
  <img>./AO/images/ORIGINAL_index.png</img>
  <!-- única imagen que empieza con ./ puesto que se lee desde un JSP y no desde un servlet -->
  <optionyes>Empezar</optionyes>
  <!-- Texto para el formulario inicial, indicando SI -->
  <optionno>Salir</optionno>
  <!-- Texto para el formulario inicial, indicando NO -->
  <alertIndex>Escoge alguna opción. </alertIndex>
  <!-- Texto de la alerta en index.jsp cuando no escoges ninguna opcion de las 2 dadas -->
            </screenIndex>
<!-- Pantalla del LOGIN  login.jsp-->
<screenLogin>
  <title>Regístrate</title>
  <!-- Titulo de esta pantalla -->
  <textUser>Usuario: </textUser>
  <!--Texto para pedir el nombre del usuario en el formulario -->
  <textPassword>Contraseña: </textPassword>
  <!--Texto para pedir la contraseña del usuario en el formulario  -->
  <textRepassword>Repita Contraseña:</textRepassword>
  <!--Texto para pedir por 2º vez la contraseña del usuario en el formulario  -->
  <textNewUser>Nuevo Usuario</textNewUser>
  <!--Texto del enelace a nuevo usuario -->
  <textBack>Volver</textBack>
  <!-- Texto mostrado para retornar del formulario de nuevo usuario, al formulario normal de User y Password -->
  <textBye>            <!--texto mostrado cuando en index.jso elegimos la opcion de NO jugar -->
            Cantidad de tesoros y aventuras te esperan en esta aventur. No lo olvides.
  </textBye>
  <alertEmpty>Debes rellenar los campos del formulario.</alertEmpty>
  <!--Alerta mostrada si no rellenamos todos los campos del formulario -->
  <alertEquals>Los passwords deben coincidir</alertEquals>
  <!--Alerta mostrada cuando los password en NUEVOUSUARIO no coinciden -->
  <alertUserExist>Ese nombre de usuario ya existe, por favor escoge otro diferente.</alertUserExist>
  <alertNotFound>No se encuentra ese nombre de usuario, o el password es incorrecto.</alertNotFound>
  <alertMkdir>Se ha producido un error al crear los archivos necesarios. Pruebe con otro nombre.</alertMkdir>
</screenLogin>
<!-- PANTALLA DE RESTART.JSP -->
<screenRestart>
  <title>Empezar de Nuevo?</title>
  <!-- Titulo de esta pantalla -->
  <textQuestion>¿Qué deseas hacer </textQuestion>
  <!-- Pregunta realizada en restart.jsp para saber si continuar la partida o no -->
  <textContinue>Continuar con la partida actual</textContinue>
  <!-- Texto del enelace para continuar la partida -->
  <textRestart>Comenzar de nuevo</textRestart>
  <!-- Texto del enlace para empezar de nuevo la partida -->
            </screenRestart>
<!-- PANTALLA SHOWINFO.JSP, AQUI PONEMOS LOS TEXTOS FIJOS QUE SALEN EN ESA PANTALLA -->
<screenShowInfo>
  <alertOption>Escoge alguna opción.</alertOption>
  <!-- Alerta mostrada si no escoges accion -->
  <alertAction>Escoge alguna acción</alertAction>
  <!-- Alerta mostrada si no escoges opcion -->
  <titleForm1>Acción: </titleForm1>
  <!-- Titulo del primer formulario para escoger una acción -->
  <form1>
    <opt0>Seleccione.</opt0>
    <!--Opcion por defecto "Seleccione.." -->
    <opt1>Ir a</opt1>
    <!--Texto para la opción "Ir A " -->
    <opt2>Coger</opt2>
    <!--Texto para la opción "COGER"-->
    <opt3>Dejar</opt3>
    <!--Texto para la opción "DEJAR"-->
    <opt4>Examinar</opt4>
    <!--Texto para la opción "MIRAR"-->
    <opt5>Usar</opt5>
    <!--Texto para la opción "USAR"-->
```

```
    <opt6>Abrir</opt6>
    <!--Texto para la opción "ABRIR"-->
    <opt7>Cerrar</opt7>
    <!--Texto para la opción "CERRAR"-->
    <opt8>Usar..con</opt8>
    <!--Texto para la opción "USAR..CON"-->
    <opt9>Inventario</opt9>
    <!-- Texto para la opcion de ver el inventario (objetos que has cogido) -->
  </form1>
   <titleForm2>Opción: </titleForm2>
   <!-- Titulo del segundo formulario para escoger una opcion -->
   <titleForm3>Con: </titleForm3>
   <!-- Titulo del tercer formulario para escoger el segundo objeto al USARCON -->
   <object1>Objeto 1:</object1>
   <!-- SOLO WML: Texto para el primer objeto-->
   <object2>Objeto 2:</object2>
   <!-- SOLO WML: Texto para el segundo objeto -->
   <textBack>Volver</textBack>
   <!-- SOLO WML: Texto para el enlace que vuelve hacia atrás-->
</screenShowInfo>
  <!-- Pantalla mostrada cuando teminas con exito el juego -->
  <screenEnd>
   <title>The End</title>
   <text>
            AVENTURA ORIGINAL I, LA BUSQUEDA.
            Adaptación para version WAP.
            Autor - Javier González Monroy
            Tutor - Francisco Javier González Cañete
            Proyecto Fin de Carrera (ETSIT Málaga)
            2007
   </text>
   <textLink>Volver a jugar!!</textLink>
   <!-- Texto mostrado en el enlace final, para invitar a empezar de nuevo a jugar -->
   <img>../xhtml/AO/images/ORIGINAL_index.png</img>
  </screenEnd>
  <!-- Pantalla mostrada cuando haces una accion que te lleva a la muerte -->
  <screenDead>
   <title>RIP</title>
   <text>   ESTAS MUERTO.    </text>
   <textLink>Volver a intentarlo!!</textLink>
   <!-- Texto mostrado en el enlace final, para invitar a empezar de nuevo a jugar -->
   <img>../xhtml/AO/images/ORIGINAL_dead.png</img>
  </screenDead>

<!-- ..................................................................................................................-->
<!-- ..................................................... PANTALLAS DEL JUEGO.................................... -->
<!-- ..................................................................................................................-->

  <screen id="1">       <!--Primera pantalla del juego propiamente dicha -->
   <title>Frontera</title>
   <text>
            Estás justo en la frontera entre las tierras áridas del Norte y la zona húmeda y boscosa del Sur.
            Muy cerca hacia el Este hay una cabaña de ladrillos rodeada por un frondoso bosque. Un río poco caudaloso fluye hacia el
            Sur.
   </text>
   <img>../xhtml/AO/images/ORIGINAL_1.png</img>
   <direction>
     <d id="19" activ="si">Norte</d>
     <d id="4" activ="si">Sur</d>
     <d id="2" activ="si">Este</d>
     <d id="2" activ="si">Entrar</d>
   </direction>
  </screen>
  <screen id="2">
   <title>Cabaña</title>
   <text>
            Estás en una habitación bastante guarra. Las paredes desconchadas y el suelo roto indican que hace tiempo fue abandonada.
            Ves la oscura boca de un pozo por la que baja una herrumbrosa escalera.
   </text>
   <img>../xhtml/AO/images/ORIGINAL_2.png</img>
   <direction>
     <d id="1" activ="si">Salir</d>
     <d id="3" activ="si">Bajar</d>
   </direction>
  </screen>
  <screen id="3">
```

```
  <title>Pozo</title>
  <text>Este sitio apesta, y está muy oscuro. </text>
  <img>../xhtml/AO/images/ORIGINAL_3.png</img>
  <direction>
    <d id="2" activ="si">Subir</d>
  </direction>
</screen>
<screen id="4">
  <title>Rio</title>
  <text>
            Estás al comienzo del valle, al lado de un alegre arroyuelo que juguetea sobre el lecho rocoso en su camino hacia el Sur. Hay
            bosque a ambos lados.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_4.png</img>
  <direction>
    <d id="1" activ="si">Norte</d>
    <d id="5" activ="si">Sur</d>
    <d id="13" activ="si">Este</d>
    <d id="8" activ="si">Oeste</d>
  </direction>
</screen>
<screen id="5">
  <title>Valle</title>
  <text>
            El valle se hace mas estrecho y profundo. El arroyo es cada vez menos caudaloso, y ahora no tan alegre,
            porque poco a poco se va filtrando por las numerosas grietas de rocoso lecho.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_5.png</img>
  <direction>
    <d id="4" activ="si">Norte</d>
    <d id="6" activ="si">Sur</d>
    <d id="15" activ="si">Este</d>
    <d id="14" activ="si">Oeste</d>
  </direction>
</screen>
<screen id="6">
  <title>Caverna</title>
  <text>
            Estás en una depresión de 7mts. El valle se estrecha hacia el Sur. Hay una fuerte reja metálica.
            Detrás de ella se ve una misteriosa entrada a las profundidades.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_6.png</img>
  <direction>
    <d id="5" activ="si">Norte</d>
    <d id="7" activ="si">Sur</d>
    <d id="8" activ="si">Este</d>
    <d id="8" activ="si">Oeste</d>
    <d id="27" activ="no">Entrar</d>
  </direction>
</screen>
<screen id="7">
  <title>Precipicio</title>
  <text>
            El cañón se abre en lo que fue antes una cascada y hoy solo un precipicio rocoso. Un enorme cortado parte el paisaje de este
            a oeste, y desde tus 500mts.
            De altura tienes una vista maravillosa de las tierras del Sur. Desde el Norte llega el viento a través de la garganta rocosa.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_7.png</img>
  <direction>
    <d id="6" activ="si">Norte</d>
    <d id="Dead" activ="si">Sur</d>
    <d id="8" activ="si">Este</d>
    <d id="8" activ="si">Oeste</d>
  </direction>
  <dead>Con un graciosillo salto emprendes el vuelo...Tremendo batacazo en el fondo del abismo...Pareces estar un poco
            muerto
.</dead>
</screen>
<screen id="8">
  <title>Bosque</title>
  <text>
            Aquí el bosque es bastante cerrado, con árboles por todas partes.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_8.png</img>
  <direction>
    <d id="12" activ="si">Norte</d>
```

```
    <d id="9" activ="si">Sur</d>
    <d id="8" activ="si">Oeste</d>
  </direction>
</screen>
<screen id="9">
  <title>Inicio Rampa</title>
  <text>
            Estás al comienzo de una maravillosa rampa artificial fabricada en medio del bosque, no se sabe hace cuantos años ni por
            quien.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_9.png</img>
  <direction>
    <d id="8" activ="si">Norte</d>
    <d id="10" activ="si">Sur</d>
  </direction>
</screen>
<screen id="10">
  <title>Rampa</title>
  <text>
            Sigues en la rampa artificial construida en medio del bosque. Frente a ti ves la base de una extraña, pero bellísima estructura.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_10.png</img>
  <direction>
    <d id="9" activ="si">Norte</d>
    <d id="11" activ="si">Sur</d>
  </direction>
</screen>
<screen id="11">
  <title>Espiral</title>
  <img>../xhtml/AO/images/ORIGINAL_11.png</img>
  <text>
            La rampa te conduce hasta un majestuoso claro. Ante ti se eleva la espiral, pulida, brillante y sobre todo impresionante.
            Detrás el suelo cae bruscamente.
 </text>

  <direction>
    <d id="10" activ="si">Norte</d>
    <d id="Dead" activ="si">Sur</d>
  </direction>
  <dead>Con un graciosillo salto emprendes el vuelo...Tremendo batacazo en el fondo del abismo...Pareces estar un poco
            muerto.
 </dead>
</screen>
<screen id="12">
  <title>Bosquecillo</title>
  <text>
            Este es un bosquecito muy agradable al lado Este del valle. Oyes el murmullo de una corriente de agua cerca.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_12.png</img>
  <direction>
    <d id="13" activ="si">Oeste</d>
    <d id="8" activ="si">Sur</d>
  </direction>
</screen>
<screen id="13">
  <title>Picnic</title>
  <text>
            En ésta zona los enanos hacían sus picnics (aunque las malas lenguas dicen que eran verdaderas orgías).
            Hay varias sillas junto a una mesa y un árbol con un curioso letrero.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_13.png</img>
  <direction>
    <d id="12" activ="si">Este</d>
    <d id="4" activ="si">Oeste</d>
    <d id="1" activ="si">NorOeste</d>
  </direction>
</screen>
<screen id="14">
  <title>Arboles</title>
  <text>
            Esta parte del bosque es muy densa, los árboles te impiden orientarte bien y son todos muy parecidos.
            Vagas entre la maleza buscando tu rumbo pero pronto te das cuenta de que estás algo despistadillo
  </text>
  <img>../xhtml/AO/images/ORIGINAL_14.png</img>
  <direction>
    <d id="14" activ="si">Norte</d>
```

```
    <d id="16" activ="si">Sur</d>
    <d id="14" activ="si">Este</d>
    <d id="15" activ="si">Oeste</d>
    <d id="14" activ="si">NorEste</d>
    <d id="14" activ="si">NorOeste</d>
    <d id="14" activ="si">SurEste</d>
    <d id="14" activ="si">SurOeste</d>
  </direction>
</screen>
<screen id="15">
  <title>Arboles</title>
  <text>
          Esta parte del bosque es muy densa, los árboles te impiden orientarte bien y son todos muy parecidos.
          Vagas entre la maleza buscando tu rumbo pero pronto te das cuenta de que estás algo despistadillo
  </text>
  <img>../xhtml/AO/images/ORIGINAL_14.png</img>
  <direction>
    <d id="15" activ="si">Norte</d>
    <d id="15" activ="si">Sur</d>
    <d id="14" activ="si">Este</d>
    <d id="15" activ="si">Oeste</d>
  </direction>
</screen>
<screen id="16">
  <title>Arbol Enorme</title>
  <text>
          Te encuentras en lo profundo del bosque rodeado por densa vegetación y junto a un enorme árbol con desgajadas ramas que
          llegan al suelo.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_16.png</img>
  <direction>
    <d id="17" activ="si">Norte</d>
    <d id="15" activ="si">Sur</d>
    <d id="15" activ="si">Este</d>
    <d id="15" activ="si">Oeste</d>
    <d id="18" activ="si">Subir</d>
  </direction>
</screen>
<screen id="17">
  <title>Arboles</title>
  <text>
          Esta parte del bosque es muy densa, los árboles te impiden orientarte bien y son todos muy parecidos.
          Vagas entre la maleza buscando tu rumbo pero pronto te das cuenta de que estás algo despistadillo.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_14.png</img>
  <direction>
    <d id="15" activ="si">Norte</d>
    <d id="13" activ="si">Sur</d>
    <d id="17" activ="si">Este</d>
    <d id="15" activ="si">Oeste</d>
    <d id="16" activ="si">SurEste</d>
  </direction>
</screen>
<screen id="18">
  <title>Rama</title>
  <text>
          Ya estás sentado entre las ramas del enorme árbol. Tienes una esplendida vista de toda la región.
          Al Norte queda el fardón volcán. Rodeada de bosque está la casita. Hacia el Oeste un profundo valle se extiende de norte a
          sur.
          En la lejanía, al Sur se yergue la misteriosa espiral. Al Este sólo hay vacío y nubes.
  </text>
  <img>../xhtml/AO/images/ORIGINAL_18.png</img>
  <direction>
    <d id="Dead" activ="si">Norte</d>
    <d id="Dead" activ="si">Sur</d>
    <d id="Dead" activ="si">Este</d>
    <d id="Dead" activ="si">Oeste</d>
    <d id="16" activ="si">Bajar</d>
  </direction>
  <dead>Con un graciosillo salto emprendes el vuelo...Te rrompes el hocico contra la raiz del arbol...Pareces estar un poco
muerto.</dead>
</screen>
<screen id="19">
  <title>Campo</title>
  <text>
          En campo abierto, rodeado por escasa vegetación. Un camino va hacia la cabaña y hacia el Este te llega una peste a azufre.
```

```
    </text>
    <img>../xhtml/AO/images/ORIGINAL_19.png</img>
    <direction>
      <d id="19" activ="si">Norte</d>
      <d id="1" activ="si">Sur</d>
      <d id="20" activ="si">Este</d>
      <d id="19" activ="si">Oeste</d>
    </direction>
  </screen>
  <screen id="20">
    <title>Arida</title>
    <text>
            Estás en una zona muy árida donde solo crecen malezas espinosas. La fragancia azufrada ya empieza a molestar.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_20.png</img>
    <direction>
      <d id="20" activ="si">Norte</d>
      <d id="21" activ="si">Este</d>
      <d id="19" activ="si">Oeste</d>
    </direction>
  </screen>
  <screen id="21">
    <title>Tapizado</title>
    <text>
            Estás en un terreno tapizado de lava vomitada por el activo volcán norteño. Hacia el Sur hay una colinita que se eleva
            suavemente.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_21.png</img>
    <direction>
      <d id="25" activ="si">Norte</d>
      <d id="24" activ="si">Sur</d>
      <d id="22" activ="si">Este</d>
      <d id="20" activ="si">Oeste</d>
    </direction>
  </screen>
  <screen id="22">
    <title>Alucinante</title>
    <text>
            En éste lugar el paisaje es alucinante, lleno de rocas erosionadas y partidas,
            mudos testigos de anteriores erupciones producidas por el furioso volcán del Norte.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_22.png</img>
    <direction>
      <d id="25" activ="si">Norte</d>
      <d id="23" activ="si">Sur</d>
      <d id="21" activ="si">Oeste</d>
    </direction>
  </screen>
  <screen id="23">
    <title>Precipicio</title>
    <text>
            ¡Cuidado! Estás al borde del precipicio donde antaño se suicidaban los enanos celosos.
            Aquí terminan (aparte de las pasiones desenfrenadas) tanto la senda del Oeste como el camino del Norte.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_23.png</img>
    <direction>
      <d id="22" activ="si">Norte</d>
      <d id="Dead" activ="si">Sur</d>
      <d id="Dead" activ="si">Este</d>
      <d id="24" activ="si">Oeste</d>
      <d id="Dead" activ="si">Bajar</d>
    </direction>
    <dead>Con un graciosillo salto emprendes el vuelo...Tremendo batacazo en el fondo del abismo...Pareces estar un poco
muerto.</dead>
  </screen>
  <screen id="24">
    <title>Colina</title>
    <text>
            Desde la colina contemplas el fiero espectáculo de fuego, humo y cenizas producido por el volcán.
            Hacia el Oeste, a lo lejos, y rodeada de denso bosque, se ve una casita.
            Una senda parte hacia el Este, y al Sur queda el bosque, tras el cual ves una extraña torre en espiral rodeada de nubes.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_24.png</img>
    <direction>
      <d id="21" activ="si">Norte</d>
      <d id="14" activ="si">Sur</d>
```

```
      <d id="23" activ="si">Este</d>
    </direction>
  </screen>
  <screen id="25">
    <title>Volcan</title>
    <text>
            El calor que se desprende de la burbujeante lava es asfixiante.
            Un enorme penacho de cenizas se desprende flotando perezosamente del ardiente magma, dejando una peste cosa mala.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_25.png</img>
    <direction>
      <d id="Dead" activ="si">Norte</d>
      <d id="22" activ="si">Sur</d>
      <d id="25" activ="si">Este</d>
      <d id="Dead" activ="si">Bajar</d>
    </direction>
    <dead>La lava te reduce a cenizas, pareces estar un poco muerto.</dead>
  </screen>
  <screen id="26">
    <title>Pozo</title>
    <text>
            Puaf, El fondo del pozo está casi seco (por suerte para ti, curiosón).
            Solo quedan algunos charcos malolientes de agua estancada.
            La estrechez de las mohosas paredes y el enorme silencio que reina hacen que tengas por única compañía el latido desbocado
            de tu corazón.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_26.png</img>
    <direction>
      <d id="2" activ="si">Subir</d>
    </direction>
  </screen>
  <screen id="27">
    <title>Caverna</title>
    <text>
            Estás en una depresión de 7mts. El valle se estrecha hacia el Sur. Tienes ante ti la entrada a la Gran caverna.
            No deberias pasar si no tienes : Una moneda, Una tortilla, Una linterna con pilas, una llave y una botella.
    </text>
    <img>../xhtml/AO/images/ORIGINAL_27.png</img>
    <direction>
      <d id="End" activ="si">Entrar</d>
      <d id="6" activ="si">Salir</d>
    </direction>
  </screen>
</map>
```

- **object.xml**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- <!DOCTYPE objects SYSTEM "../../DTD/object.dtd"> -->


<!-- ********************************************************************************************** -->
<!-- DESARROLLO DE UN MOTOR DE VIDEOJUEGOS CONVERSACIONALES SOBRE WAP PARA DISPOSITIVOS MÓVILES -->
<!-- Autor: JAVIER GONZÁLEZ MONROY                                                                 -->
<!-- OBJECT.XML V4  PARA XHTML                                                                      -->
<!-- ********************************************************************************************** -->

<!-- En este documento se definen los diferentes objetos con los que el usuario puede interaccionar a lo largo de todo el juego.
            Igualmente se definen los textos producidos al manipularlos (cogerlos, mirarlos, etc) -->

<!--DESCRIPCIÓN DEL DOCUMENTO
            En este archivo se encuentran todos los objetos del juego en cuestión, tanto los activos inicialmente, como los que aparecen
            por el resultado de alguna acción.
            Cada objeto puede manipularse por el usuario y por ello cada objeto tiene una serie de atributos

            ID... Donde indicamos el nombre del objeto.
            screen.... Atributo  que indica en la pantalla en la que se encuentra (numero de la pantalla).
                        Si el valor es el nombre del usuario indica que el objeto está en el inventario.
                        SI  screen=0  indica que ese objeto está inactivo por el momento, no se muestra al usuario.
            Cogible.... Atributo que indica si ese objeto puede cogerse. (si/no)
                                cogible=si .. Puede cogerse.
                                cogible=no .. No puede cogerse.
            Usable.... Atributo que indica si un objeto puede ser utilizado por si solo ( esto incluye varias acciones USAR, ABRIR,
```

```
                    CERRAR...)
                    usable=no .. Indica que no puede usarse en ese momento
                    usable=si .. Indica que si puede realizarse la acción y hay que buscarla en el archivo de acciones (actions.xml)

          Además de los objetos, se muestran una serie de textos que se mostrarán al usuario como consecuencia de una acción.
                    ERRORCOGER...Es el texto que se muestra cuando se intenta coger un objeto que no puede cogerse en ese
                                  momento
                    ERRORUSAR..Texto mostrado cuando la accion USAR no puede realizarse (usable="no" o usable="si" pero la
                                acción USAR no está permitida, es decir no esta en actions.xml)
                    EERORABRIR.. TExto mostrado cuando la acccion ABRIR no puede realizarse (usable="no" o usable="si" pero
                                la accion ABRIR no está permitida)
                    EERORCERRAR.. TExto mostrado cuando la acccion CERRAR no puede realizarse (usable="no" o usable="si"
                                pero la accion CERRAR no está permitida)
                    TEXTO..Descripcion del objeto (se muestra el texto en la pantalla cuando el objeto está en ella)
                    MIRAR..Texto que se usa para describir con más detalle el objeto. Se muestra con la accion mirar objeto.
                              MIRARno ..Texto mostrado al mirar el objeto cuando no se puede usar (usable=no)
                              MIRARsi ..Texto mostrado al mirar el objeto cuando si se puede usar (usable=si)
                    COGER..Indica el texto mostrado cuando COGER un objeto tiene exito.
                    DEJAR..Indica el texto mostrado cuando DEJAR un objeto tiene exito.
-->

<objects xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../../XSD/object.xsd">
          <!-- Textos por defecto -->
<coger>Lo guardas en el inventario</coger>          <!-- texto por defecto para coger, si el objeto no especifica otro para esta accion-->
<dejar>Lo dejas en el suelo</dejar>          <!-- texto por defecto para dejar, si el objeto no especifica otro para esta accion-->
<mirar>Hmm no tiene nada especial</mirar>          <!-- texto por defecto para mirar, si el objeto no especifica otro para esta accion-->
<errorcoger>No puedo coger eso</errorcoger>          <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->
<errorusar>No puedo usar eso</errorusar>          <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->
<errorabrir>No puedo abrir eso</errorabrir>          <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->
<errorcerrar>No puedo cerrar eso</errorcerrar>          <!-- error dado por defecto, si el objeto no especifica otro error para esta accion-->

          <!-- SCREEN 1 -->
          <ob id="rio" screen="1" cogible="no" usable="no">
                    <errorusar>Como no me meta, no se como usarlo.</errorusar>
                    <mirarno>Dicen que las aguas de éste río tienen la mágica propiedad de hacer crecer semillas muy
                              viejas
                    </mirarno>
           </ob>

          <ob id="casa" screen="1" cogible="no" usable="no">
                    <mirarno>Es una sólida construcción que antaño perteneció a los dueños del valle, hoy está
                              abandonada.
                    </mirarno>
          </ob>

          <ob id="bosque" screen="1" cogible="no" usable="no">
                    <mirarno>No estás en clase de botánica</mirarno>
          </ob>

          <!-- SCREEN 2 -->
          <ob id="pozo" screen="2" cogible="no" usable="no">
                    <mirarno>La boca del pozo está muy oscura</mirarno>
          </ob>

          <ob id="escalera" screen="2" cogible="no" usable="no">
                    <mirarno>Baja p'abajo</mirarno>
          </ob>

          <!-- SCREEN 3 -->
          <ob id="oscuridad" screen="3" cogible="no" usable="no">
                    <mirarno>Pues eso, que no se ve nada.</mirarno>
                    <errorcoger>Hmm aun no estoy tan mal para hacer esas cosas.</errorcoger>
          </ob>
          <!-- SCREEN 4 -->
          <ob id="arroyuelo" screen="4" cogible="no" usable="no">
                    <mirarno>Dicen que las aguas de éste río tienen la mágica propiedad de hacer crecer semillas muy viejas
          </mirarno>
          </ob>

          <!-- SCREEN 5 -->
          <ob id="grietas" screen="5" cogible="no" usable="no">
                    <mirarno>son ranuras estrechas por las que se cuela el agua del río</mirarno>
          </ob>

          <!-- SCREEN 6 -->
          <ob id="candado" screen="6" cogible="no" usable="no">
```

```
                <mirarno>Es muy fuerte, y está cerrado. Impide la entrada.</mirarno>
                 <errorcoger>No puedo cogerlo, está cerrado</errorcoger>
                <errorabrir>Necesito algo para abrilo.</errorabrir>

        </ob>

        <ob id="reja" screen="6" cogible="no" usable="no">
                <mirarsi>Una fuerte reja metálica.</mirarsi>
                <mirarno>Una fuerte reja metálica cerrada con un candado.</mirarno>
                <errorabrir>No puedo abrirla, el candado me lo impide.</errorabrir>
        </ob>

        <!-- SCREEN 7 -->
        <ob id="botella" screen="7" cogible="si" usable="no">Puedes ver la botella.
                <coger>Has cogido la botella</coger>
                <dejar>Has dejado la botella</dejar>
                <errorusar>Esta vacia... </errorusar>
                <mirarno>Es de vidrio y está vacia.</mirarno>
        </ob>

        <ob id="cascada" screen="7" cogible="no" usable="no">
                <mirarno>Es la gran cascada del jefe de los enanos.</mirarno>
        </ob>

        <!-- SCREEN 8 -->
        <ob id="figura" screen="8" cogible="no" usable="no">Oyes un débil llanto y con dificultad ves una figura traslúcida.
                <mirarno>No hay suficiente luz.</mirarno>
        </ob>

        <!-- SCREEN 9 -->
        <ob id="rampa" screen="9" cogible="no" usable="no">
                <mirarno>es una sólida y bella construcción.</mirarno>
        </ob>

        <!-- SCREEN 10 -->
        <ob id="Rampa" screen="10" cogible="no" usable="no">
                <mirarno>es una sólida y bella construcción.</mirarno>
        </ob>

        <ob id="estrctura" screen="10" cogible="no" usable="no">
                <mirarno>No estás suficientemente cerca.</mirarno>
        </ob>

        <!-- SCREEN 11 -->
        <ob id="botas" screen="11" cogible="si" usable="no"> Puedes ver una pareja de botas.
                <mirarno>Estas botas tienen un tacón muy alto, pero no son de tu número.</mirarno>
                <coger>Coges las botas por lo que pueda suceder</coger>
                <dejar>Dejas las botas en el suelo.</dejar>
                <errorusar>Las botas son pequeñas para tus pies</errorusar>
        </ob>

        <ob id="espiral" screen="11" cogible="no" usable="no">
                <mirarno>Es una construcción altísima y de un brillo metálico, pero de superficie tan resbaladiza que la hace
                        inexpugnable.
                 </mirarno>
        </ob>

        <!-- SCREEN 12 -->

        <!-- SCREEN 13 -->
        <ob id="letrero" screen="13" cogible="no" usable="no">
                <mirarno>Pone: cuidado con tus objetos</mirarno>
        </ob>

        <ob id="mesa" screen="13" cogible="no" usable="no">
                <mirarno>De hechura rústica y buena madera. Muy usada por los enanos del bosque.</mirarno>
        </ob>

        <ob id="silla" screen="13" cogible="no" usable="si">
                <mirarsi>De hechura rústica y buena madera. Muy usada por los enanos del bosque.</mirarsi>
        </ob>

        <ob id="tortilla" screen="0" cogible="si" usable="no"> Ves la tortilla.
                <mirarno>dejaría satisfecho hasta a un oso.</mirarno>
                <coger>Coges las fantastica tortilla que te dejó el elfo.</coger>
                <dejar>Con pesadez y desgana, te deshaces de la fantastica tortilla elfica.</dejar>
```

```
            <errorusar>Si me comiera esa tortilla estaría horas sin poder moverme.</errorusar>
</ob>

<!-- SCREEN 16 -->
<ob id="Arbol" screen="16" cogible="no" usable="no">
            <mirarno>por su posición y solidez proporcionan una fácil subida</mirarno>
</ob>

<ob id="llave" screen="0" cogible="si" usable="no">Puedes ver la llave.
            <mirarno>Lleva la insignia del Rey Elfo</mirarno>
            <coger>Has cogido la llave.</coger>
            <dejar>Dejas la llave en el suelo.</dejar>
            <errorusar>Una llave por si sola no vale para nada. Necesito algo que abrir.</errorusar>
</ob>

<!-- SCREEN 18 -->
<ob id="rama" screen="18" cogible="no" usable="si">
             <mirarsi>Una frondosa rama con una cosa que brilla con el sol.</mirarsi>
              <mirarno>Una frondosa rama sin más interes.</mirarno>
</ob>

<ob id="cosa_brillante" screen="18" cogible="no" usable="no">El sol hace brillar algo en una rama.
            <mirarno>Parece una llave.</mirarno>
            <errorcoger>Está demasiado lejos.</errorcoger>
            <errorusar>No puedo usarla si aun no la tengo.</errorusar>
</ob>


<!-- SCREEN 21 -->
<ob id="volcan" screen="21" cogible="no" usable="no">
            <mirarno> un bujero quecha humo y pedorreta.</mirarno>
</ob>

<!-- SCREEN 23 -->
<ob id="precipicio" screen="23" cogible="no" usable="no">
            <mirarno>NO TE PASES!!!</mirarno>
</ob>

<!-- SCREEN 24 -->
<ob id="pila" screen="24" cogible="si" usable="no">Puedes ver la pila eléctrica.
            <mirarno>pone: SEMPER LUX (no lo creas)</mirarno>
            <coger>Has cogido la pila.</coger>
            <dejar>Dejas la pila en el suelo.</dejar>
            <errorusar>Chupas la pila y... Calambrito Calambrito.</errorusar>
</ob>

<!-- SCREEN 25 -->
<!-- Enano enfadado -->
<ob id="enano" screen="25" cogible="no" usable="no">Imprudentemente estas haciendo el payaso en el peligroso borde del
             volcán junto a un enano que rabia de amargura.
            <mirarno>Como los del circo, parece tener muy mala uva, y está enojado porque se siente muy bajo y desearía ser
                        más alto.
            </mirarno>
</ob>
<!-- Enano contento -->
<ob id="Enano" screen="0" cogible="no" usable="no">Imprudentemente estas haciendo el payaso en el peligroso borde del
            volcán junto a un enano que vacila de su altura.
            <mirarno>Como los del circo, ahora parece estar contento puesto que con las botas se siente más alto.</mirarno>
</ob>

<ob id="linterna" screen="0" cogible="si" usable="no">Puedes ver la linterna.            <!-- Linterna apagada-->
            <mirarno>Está apagada</mirarno>
            <mirarsi>Está apagada</mirarsi>
            <coger>Has cogido la linterna.</coger>
            <dejar>Dejas la linterna en el suelo.</dejar>
            <errorusar>No tiene pilas.</errorusar>
</ob>

<ob id="Linterna" screen="0" cogible="si" usable="si">Puedes ver la linterna.            <!-- Linterna encendida-->
            <mirarno>Está encendida y lista para alumbrar</mirarno>
            <mirarsi>Está encendida y lista para alumbrar</mirarsi>
            <coger>Has cogido la linterna.</coger>
            <dejar>Dejas la linterna en el suelo.</dejar>
</ob>

<!-- SCREEN 26 -->
```

```
        <ob id="charco" screen="26" cogible="no" usable="si">
                <mirarsi>Asquerosos charcos llenos de babosidades, y de escasa profundidad.</mirarsi>
        </ob>
        <ob id="moneda" screen="0" cogible="si" usable="no">Puedes ver la moneda.        <!-- Linterna encendida-->
                <mirarno>Ésta antigua moneda perteneció a los Elfos</mirarno>
                <coger>Has cogido la moneda.</coger>
                <dejar>Dejas la moneda en el suelo.</dejar>
        </ob>
</objects>
```

- **actions.xml**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- ********************************************************************************************** -->
<!-- DESARROLLO DE UN MOTOR DE VIDEOJUEGOS CONVERSACIONALES SOBRE WAP PARA DISPOSITIVOS
MÓVILES -->
<!-- Autor: JAVIER GONZÁLEZ MONROY                                                                -->
<!-- ACTIONS.XML V4  PARA XHTML                                                                   -->
<!-- ********************************************************************************************** -->

<!-- En este archivo se detallan las posibles acciones que pueden ser realizadas en el juego, así como las consecuencias que conllevan
cada una de ellas sobre los diferentes objetos o desplazamientos del usuario -->

<!-- ACCIONES A REALIZAR EN EL JUEGO
        Las acciones a poder realizar son: USAR,ABRIR,CERAR,USARCON

        USAR /ABRIR/CERRAR
        Establecemos las acciones a poder realizar sobre los diferentes objetos del juego. Recordar que para que puedan ser usados
        los objetos, en el archivo object.xml
        deben tener el atributo usable="si" (usable).

        Cada una de las acciones detalladas, tiene una serie de atributos que se aplican a su etiqueta <usar>,<abrir>, <cerrar>
        respectivamente:
                OBJ... Atributo que indica el nombre del objeto sobre el que se realiza la accion
                TEXTO..Texto que se muestra en una alerta al realizar la accion
                REAC.., Una consecuencia o reaccion de realizar esa acccion. Hay 3 tipos básicos de reacciones:
                        reacIra DESP="numero" ... Indica que se produce un desplazamiento de pantalla y en DESP indicamos
                                                el número de la pantalla a la que se moverá al usuario
                        reacMod OBJ="nombre" atrib="xx" value="yy"... Indica que se modifica un objeto,para ello indicamos
                                                el nombre del objeto, el atributo a modificar y su valor.
                        ACTIVAR OBJETO (atrib="screen" value="yy") ..donde "yy" es la pantalla en la que se introduce el
                                                objeto o (user) si se introduce en el inventario del     usuario
                        DESACTIVAR OBJETO (atrib="screen" value="0")... Indica que se desactiva el objeto con el nombre
                                                especificado (s="0" del objeto en object.xml)
                        USABLE (atrib="usable" value="si") ..Indica que desde ese momento el objeto especificado puede ser
                                                usado por si solo (al abrir una puerta, se peude
                                                                                        usar)
                        reacModdir direction="xx" screen="yy" atrib="zz" value="pp"..Accion que modifica la direccion "xx"
                                                de la pantalla "yy" en map.xml, cambiando el atributo con ese valor.

        USARCON
        Aparte de las 3 acciones, que solo actuan sobre un objeto, la ccion usar..con requiere dos elementos a usar, y es por ello que
        se separa del resto de acciones.
        De los dos objetos necesarios, el primero siempre es perteneciente al inventario, mientras que el segundo puede ser del
        inventario o un objeto de la pantalla
        Las reacciones son las mismas que en el caso anterior.
-->
<acciones xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="../../../XSD/actions.xsd">
        <error>No puedo hacer eso.</error>          <!-- error dado al usar un objeto con otro que no está previsto -->

        <!-- SCREEN 3 -->
        <usarcon obj1="linterna" obj2="oscuridad">
                <text>Una linterna apagada no alumbra mucho que digamos.</text>
        </usarcon>

        <usarcon obj1="Linterna" obj2="oscuridad">
                <text>Alumbras a tu alrededor.</text>
                <reacIra desp="26"/>            <!--Nos movemos a la pantalla del pozo con luz -->

        </usarcon>

        <!-- SCREEN 6 -->
        <usarcon obj1="llave" obj2="candado">
                <text>Clic!! Abres el candado que mantenia cerrada la reja. AL quitarlo, desaparece de tus manos por arte de
```

```
                    magia.
            </text>
            <reacMod obj="reja" atrib="usable" value="si"/>     <!--Pongo el objeto reja como usable, ya podemos abrirla -->
            <reacMod obj="candado" atrib="screen" value="0"/>      <!--Desactivo el objeto candado (desaparece) -->
</usarcon>

<abrir obj="reja">
            <text>Abres la reja que da paso a la gran caverna.</text>
            <reacModdir screen="6" direction="27" atrib="activ" value="si"/>       <!-- permitimos entrar en la cueva -->
</abrir>
<cerrar obj="reja">
            <text>Cierras la reja que da paso a la gran caverna.</text>
            <reacModdir screen="6" direction="27" atrib="activ" value="no"/>      <!-- NO permitimos entrar en la cueva -->
</cerrar>

<!-- SCREEN 8 -->
<usarcon obj1="linterna" obj2="figura">
            <text>Una linterna apagada no alumbra mucho que digamos.</text>
</usarcon>

<usarcon obj1="Linterna" obj2="figura">
            <text>Alumbras a la figura. Un elfo llora porque está perdido.Al ver la luz de tu linterna corre hacia tí.Te dice:
                    sobre la mesa está la recompensa por ayudarme.
            </text>
            <reacMod obj="figura" atrib="screen" value="0"/>                    <!--El elfo desaparece -->
            <reacMod obj="tortilla" atrib="screen" value="13"/>   <!--Activo la tortilla sobre la mesa de la pantalla 13 -->

</usarcon>

<!--SCREEN 13 -->
<usar obj="silla">
            <text>te sientas y seguidamente te levantas para seguir jugando.. ¡TATE pa la faena!</text>
</usar>

<!-- SCREEN 18 -->
<usar obj="rama">
            <text>Zarandeas la rama con todas tus fuerzas..... Finalemnte el objeto brillante cae al suelo.</text>
            <reacMod obj="rama" atrib="usable" value="no"/>                <!--Pongo el objeto rama como NO usable -->
            <reacMod obj="cosa_brillante" atrib="screen" value="0"/>       <!--Desactivo el objeto brillante (se cae abajo) -->
            <reacMod obj="llave" atrib="screen" value="16"/>               <!-- Activo la llave en la pantalla 16 -->
</usar>


<!-- SCREEN 25 -->
<usarcon obj1="botas" obj2="enano">
            <text>Amablemente cedes las botas que encontraste en la base de la espiral al enano. Este se las calza y saltando
                    de alegría te regala una linterna.
            </text>
            <reacMod obj="linterna" atrib="screen" value="user"/>              <!--Pongo el objeto linterna en el inventario -->
            <reacMod obj="botas" atrib="screen" value="0"/>      <!--Desactivo el objeto botas -->
            <reacMod obj="enano" atrib="screen" value="0"/>      <!--Desactivo el objeto enano para que al mirarlo muestre
                                                                 al enano contento y no a este -->
            <reacMod obj="Enano" atrib="screen" value="25"/>     <!--Activo el objeto enano contencto para que al mirarlo
                                                                 muestre otro texto -->
</usarcon>

<usarcon obj1="pila" obj2="linterna">
            <text>Introduces la pila en la linterna.</text>
            <reacMod obj="linterna" atrib="usable" value="si"/>                <!--Pongo el objeto linterna como usable -->
            <reacMod obj="pila" atrib="screen" value="0"/>                     <!--Desactivo el objeto pila -->

</usarcon>

<abrir obj="linterna">
            <text>Abres la linterna y sacas la pila</text>
            <reacMod obj="pila" atrib="screen" value="user"/>     <!-- ponemos la pila en el inventario -->
            <reacMod obj="linterna" atrib="usable" value="no"/>             <!--Pongo el objeto linterna como NO usable -->
</abrir>

<usar obj="linterna">
            <text>Enciendes la linterna.</text>
            <reacMod obj="linterna" atrib="screen" value="0"/>                 <!--Desactivo la linterna apagada -->
            <reacMod obj="Linterna" atrib="screen" value="user"/>              <!--Activo la linterna encendida -->

</usar>
```

```
        <usar obj="Linterna">
                <text>Apagas la linterna.</text>
                <reacMod obj="linterna" atrib="screen" value="user"/>                <!--Activo la linterna apagada -->
                <reacMod obj="Linterna" atrib="screen" value="0"/>  <!--Desactivo la linterna encendida -->
        </usar>

        <abrir obj="Linterna">
                <text>Abres la linterna y sacas la pila</text>
                <reacMod obj="pila" atrib="screen" value="user"/>            <!-- ponemos la pila en el inventario -->
                <reacMod obj="linterna" atrib="usable" value="no"/>          <!--Pongo el objeto linterna como NO usable -->
                <reacMod obj="linterna" atrib="screen" value="user"/>        <!--Activo la linterna apagada -->
                <reacMod obj="Linterna" atrib="screen" value="0"/>           <!--Desactivo la linterna encendida -->
        </abrir>

        <!-- SCREEN 26 -->
        <usar obj="charco">
                <text>chapoteas tocando gelatinosas babosidades pero de pronto descubres que hay una moneda.</text>
                <reacMod obj="charco" atrib="usable" value="no"/>            <!--Dejop de poder usar el char, ya a aparecido
                                                                             la moneda -->
                <reacMod obj="moneda" atrib="screen" value="26"/> <!--Activo la moneda en esa pantalla -->
        </usar>
</acciones>
```

APÉNDICE B: VALIDACIÓN DE LOS DOCUMENTOS XML.

Antes de probar un videojuego desarrollado para esta aplicación sobre un terminal móvil o incluso sobre un emulador, es recomendable realizar una validación de los documentos XML que definen y configuran al mismo. El proceso de validación de documentos XML puede ser llevado a cabo por multitud de programas y aplicaciones, incluso en entornos Web se ofrecen espacios en los que se pueden realizar validaciones de archivos XML o fragmentos de los mismos. En este proyecto se ha hecho uso del programa XMLPad 3.0.1.0, el cual al igual que el resto de programas usados en el desarrollo de este proyecto es de libre distribución.

El proceso de validación puede dividirse en dos pasos fundamentales:

1. El primer paso revisa el documento para comprobar que es un documento bien formado, esto implica que los elementos definidos en el mismo se adaptan a la sintaxis especificada para los documentos XML, tal y como coincidencia de mayúsculas y minúsculas, el cierre correcto de los elementos, el uso de comillas para la definición de atributos, etc.

   Este primer paso o revisión puede detectar fallos de escritura en el documento, como el olvidar cerrar unas comillas, el no cerrar un elemento en el lugar correcto, etc.
2. El segundo paso es comprobar que se cumplen las normas o restricciones impuestas por un documento que define estos aspectos. Hay varios tipos de documentos que anexados a un documento XML definen su estructura, de los cuales DTD y XSD son los más usados.

El primer paso del proceso de validación se realiza de forma automática por el programa usado en la validación, en este caso XMLPad. Para realizar el segundo paso de la validación, es necesario primeramente decantarse por uno de los diversos formatos de definir las normas o restricciones, en este caso se ha elegido XSD, dado que se puede considerar de alguna forma una evolución de los DTD, cumpliendo una serie de características que lo hacen más propicio:

- Lenguaje más complejo y más potente que el de los DTD.

- Utiliza la misma sintaxis que los documentos XML, lo cual le permite definir de forma más detallada los tipos de datos, y arquitecturas necesarias.
- Es más fácil su modificación y extensión dado que se trata de un documento XML.

Haciendo uso de XSD se han desarrollado una serie de archivos que, anexados a los documentos XML que definen un videojuego, ayudan a su correcta validación, evitando así la mayoría de los errores en su definición. Estos archivos se encuentran en el directorio *XSD* bajo el directorio *MotorConv* de la aplicación, y pueden ser usados tanto para los videojuegos de XHTML y WML, así como para cualquier otra vista implementada, dado que la correcta validación de un documento XML no depende del contenido del mismo.

Los archivos XSD creados son: *user.xsd, map.xsd, object.xsd y action.xsd* los cuales validan la estructura del archivo XML a cuyo nombre representan, esto es *user.xml, map.xml, objectx.ml y action.xml* respectivamente. A continuación se muestra el código fuente de cada uno de esos archivos:

- **user.xsd**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wmh="http://www.wmhelp.com/2003/eGenerator" elementFormDefault="qualified">
  <xs:element name="config">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="screen"/>
        <xs:element ref="password"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="screen" type="xs:string"/>
  <xs:element name="password" type="xs:string"/>
</xs:schema>
```

- **map.xsd**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wmh="http://www.wmhelp.com/2003/eGenerator" elementFormDefault="qualified">
  <xs:element name="map">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="screenIndex"/>
        <xs:element ref="screenLogin"/>
        <xs:element ref="screenRestart"/>
        <xs:element ref="screenShowInfo"/>
        <xs:element ref="screenEnd"/>
        <xs:element ref="screenDead"/>
```

```
      <xs:element ref="screen" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="screenIndex">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="text"/>
      <xs:element ref="img"/>
      <xs:element ref="optionyes"/>
      <xs:element ref="optionno"/>
      <xs:element ref="alertIndex"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="title" type="xs:string"/>
<xs:element name="text" type="xs:string"/>
<xs:element name="img" type="xs:string"/>
<xs:element name="optionyes" type="xs:string"/>
<xs:element name="optionno" type="xs:string"/>
<xs:element name="alertIndex" type="xs:string"/>

<xs:element name="screenLogin">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="textUser"/>
      <xs:element ref="textPassword"/>
      <xs:element ref="textRepassword"/>
      <xs:element ref="textNewUser"/>
      <xs:element ref="textBack"/>
      <xs:element ref="textBye"/>
      <xs:element ref="alertEmpty"/>
      <xs:element ref="alertEquals"/>
      <xs:element ref="alertUserExist"/>
      <xs:element ref="alertNotFound"/>
      <xs:element ref="alertMkdir"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="textUser" type="xs:string"/>
<xs:element name="textPassword" type="xs:string"/>
<xs:element name="textRepassword" type="xs:string"/>
<xs:element name="textNewUser" type="xs:string"/>
<xs:element name="textBack" type="xs:string"/>
<xs:element name="textBye" type="xs:string"/>
<xs:element name="alertEmpty" type="xs:string"/>
<xs:element name="alertEquals" type="xs:string"/>
<xs:element name="alertUserExist" type="xs:string"/>
<xs:element name="alertNotFound" type="xs:string"/>
<xs:element name="alertMkdir" type="xs:string"/>

<xs:element name="screenRestart">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="textQuestion"/>
      <xs:element ref="textContinue"/>
      <xs:element ref="textRestart"/>
```

```
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="textQuestion" type="xs:string"/>
<xs:element name="textContinue" type="xs:string"/>
<xs:element name="textRestart" type="xs:string"/>

<xs:element name="screenShowInfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="alertOption"/>
      <xs:element ref="alertAction"/>
      <xs:element ref="titleForm1"/>
      <xs:element ref="form1"/>
      <xs:element ref="titleForm2"/>
      <xs:element ref="titleForm3"/>
      <xs:element ref="object1"/>
      <xs:element ref="object2"/>
      <xs:element ref="textBack"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="alertOption" type="xs:string"/>
<xs:element name="alertAction" type="xs:string"/>
<xs:element name="titleForm1" type="xs:string"/>

<xs:element name="form1">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="opt0"/>
      <xs:element ref="opt1"/>
      <xs:element ref="opt2"/>
      <xs:element ref="opt3"/>
      <xs:element ref="opt4"/>
      <xs:element ref="opt5"/>
      <xs:element ref="opt6"/>
      <xs:element ref="opt7"/>
      <xs:element ref="opt8"/>
      <xs:element ref="opt9"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="opt0" type="xs:string"/>
<xs:element name="opt1" type="xs:string"/>
<xs:element name="opt2" type="xs:string"/>
<xs:element name="opt3" type="xs:string"/>
<xs:element name="opt4" type="xs:string"/>
<xs:element name="opt5" type="xs:string"/>
<xs:element name="opt6" type="xs:string"/>
<xs:element name="opt7" type="xs:string"/>
<xs:element name="opt8" type="xs:string"/>
<xs:element name="opt9" type="xs:string"/>
<xs:element name="titleForm2" type="xs:string"/>
<xs:element name="titleForm3" type="xs:string"/>
<xs:element name="object1" type="xs:string"/>
<xs:element name="object2" type="xs:string"/>

<xs:element name="screenEnd">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
```

```
      <xs:element ref="text"/>
      <xs:element ref="textLink"/>
      <xs:element ref="img"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="textLink" type="xs:string"/>

<xs:element name="screenDead">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="text"/>
      <xs:element ref="textLink"/>
      <xs:element ref="img"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="screen">
  <xs:complexType>
    <xs:all>
      <xs:element ref="title"/>
      <xs:element ref="text"/>
      <xs:element ref="img"/>
      <xs:element ref="direction"/>
      <xs:element ref="dead" minOccurs="0"/>
    </xs:all>
    <xs:attribute name="id" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="dead" type="xs:string"/>

<xs:element name="direction">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="d" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="d">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="id" type="xs:string" use="required"/>
        <xs:attribute name="activ" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

</xs:schema>
```

- **object.xsd**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wmh="http://www.wmhelp.com/2003/eGenerator" elementFormDefault="qualified">
  <xs:element name="objects">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="coger"/>
        <xs:element ref="dejar"/>
        <xs:element ref="mirar"/>
        <xs:element ref="errorcoger"/>
        <xs:element ref="errorusar"/>
        <xs:element ref="errorabrir"/>
        <xs:element ref="errorcerrar"/>
        <xs:element ref="ob" maxOccurs="unbounded" minOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="coger" type="xs:string"/>
  <xs:element name="dejar" type="xs:string"/>
  <xs:element name="mirar" type="xs:string"/>
  <xs:element name="errorcoger" type="xs:string"/>
  <xs:element name="errorusar" type="xs:string"/>
  <xs:element name="errorabrir" type="xs:string"/>
  <xs:element name="errorcerrar" type="xs:string"/>

  <xs:element name="ob">
    <xs:complexType mixed="true">
      <xs:all>
        <xs:element ref="mirarno" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="errorusar" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="errorcoger" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="errorabrir" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="errorcerrar" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="mirarsi" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="coger" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="dejar" maxOccurs="1" minOccurs="0"/>
      </xs:all>
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="screen" type="xs:string" use="required"/>
      <xs:attribute name="cogible" type="xs:string" use="required"/>
      <xs:attribute name="usable" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="mirarno" type="xs:string"/>
  <xs:element name="mirarsi" type="xs:string"/>
</xs:schema>
```

- **actions.xsd**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:wmh="http://www.wmhelp.com/2003/eGenerator" elementFormDefault="qualified">
  <xs:element name="acciones">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="error"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="usar"/>
```

```
        <xs:element ref="usarcon"/>
        <xs:element ref="abrir"/>
        <xs:element ref="cerrar"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="error" type="xs:string"/>

<xs:element name="usarcon">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="text"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="reacIra"/>
        <xs:element ref="reacMod"/>
        <xs:element ref="reacModdir"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="obj1" type="xs:string" use="required"/>
    <xs:attribute name="obj2" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="text" type="xs:string"/>

<xs:element name="reacIra">
  <xs:complexType>
    <xs:attribute name="desp" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="reacMod">
  <xs:complexType>
    <xs:attribute name="obj" type="xs:string" use="required"/>
    <xs:attribute name="atrib" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="usable"/>
          <xs:enumeration value="cogible"/>
          <xs:enumeration value="screen"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="reacModdir">
  <xs:complexType>
    <xs:attribute name="atrib" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="activ"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="value" type="xs:string" use="required"/>
    <xs:attribute name="screen" type="xs:integer" use="required"/>
    <xs:attribute name="direction" type="xs:string" use="required"/>
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name="abrir">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="text"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="reacIra"/>
          <xs:element ref="reacMod"/>
          <xs:element ref="reacModdir"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="obj" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="cerrar">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="text"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="reacIra"/>
          <xs:element ref="reacMod"/>
          <xs:element ref="reacModdir"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="obj" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="usar">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="text"/>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="reacIra"/>
          <xs:element ref="reacMod"/>
          <xs:element ref="reacModdir"/>
        </xs:choice>
      </xs:sequence>
      <xs:attribute name="obj" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```