PROYECTO FIN DE CARRERA

Título:	Desarrollo de un bot de ayuda en lenguaje AIML con acceso Web y WAP
Autor:	Alejandro Marqués Rodríguez
Tutor:	Carlos Ángel Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Mercedes Garijo Ayestarán
Vocal:	José Carlos González Cristóbal
Secretario:	Carlos Ángel Iglesias Fernández
Suplente:	Pedro Alonso Martín

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESARROLLO DE UN BOT DE AYUDA EN LENGUAJE AIML CON ACCESO WEB Y WAP

Alejandro Marqués Rodríguez

CURSO 2008-2009

RESUMEN

El gran crecimiento que ha experimentado el uso de Internet en los últimos años ha supuesto un aumento de la cantidad de información disponible en la red, y, con ello, un incremento del esfuerzo requerido por parte de los usuarios para filtrar aquella que les pueda resultar útil.

Tanto en las herramientas de búsqueda como dentro de los diferentes sitios Web, la tarea de seleccionar la información recae en el usuario, que tiene que navegar entre grandes volúmenes de datos o hacer uso de opciones avanzadas, y a menudo complejas, para limitar los resultados obtenidos.

El uso del procesamiento del Lenguaje Natural y de las aplicaciones de respuesta a preguntas supone un cambio en el enfoque tradicional de búsqueda de información. El usuario puede abstraerse de las tareas de filtrado, las cuales lleva a cabo la aplicación, debiéndose preocupar únicamente por plantear sus preguntas, tal y como lo haría en una conversación con otra persona.

Este nuevo planteamiento de búsqueda de información en Internet ha supuesto que se empiece a popularizar el uso de diversas aplicaciones de búsqueda, respuesta a preguntas o asistentes virtuales basados en el uso del Lenguaje Natural. Además, estas aplicaciones no sólo mejoran la experiencia del usuario en el proceso de selección de información sino, que en el ámbito comercial, permiten una mejora del servicio ofrecido y a su vez una reducción de costes.

Vistas las ventajas que puede suponer el uso del procesamiento del Lenguaje Natural, este Proyecto de Fin de Carrera tiene como objetivo el desarrollo de un bot de ayuda basado en lenguaje AIML, accesible tanto a través de una interfaz Web como de una interfaz WAP, que sea capaz de responder de manera autónoma a las preguntas formuladas por los usuarios. La aplicación deberá, además, disponer de todos los mecanismos necesarios para el registro de eventos y conversaciones así como de aquellos para la realización de pruebas automatizadas.

Palabras Clave: Lenguaje Natural, Bot Conversacional, Respuesta a Preguntas, AIML, Asistente Virtual, Interfaz Web, Interfaz WAP

Índice

Índice de Ilustraciones

Índice de Tablas

Listado de Acrónimos

AIML	*Artificial Intelligence Markup Language*
ALICE	*Artificial Linguistic Internet Computer Entity*
CSS	*Cascading Style Sheets*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
J2EE	*Java 2 Platform, Enterprise Edition*
JRE	*Java Runtime Environment*
JSP	*Java Server Pages*
JSTL	*Java Server Pages Standard Tag Library*
MVC	*Model-view-controller*
SETL	*SET Language*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WAP	*Wireless Application Protocol*
WAR	*Web Application aRchive*
WML	*Wireless Markup Language*
XML	*Extensible Markup Language*

1 Introducción

Este capítulo, como su propio nombre indica, sirve como introducción del presente Proyecto de Fin de Carrera. Para ello en primer lugar se explicarán las razones que motivan la realización del mismo, para, posteriormente, definir los objetivos que se pretenden alcanzar a su conclusión.

1.1 Motivación

El Lenguaje Natural es una de las diferentes formas que pueden usarse para el diálogo entre un humano y una máquina. Pese a no haberse potenciado demasiado en el ámbito del manejo de aplicaciones, debido a la sencillez de manejo de las interfaces gráficas basadas en iconos y ventanas, puede resultar de gran utilidad en combinación con éstas y, sobretodo, en aquellos campos relacionados con la búsqueda de información.

El auge que ha experimentado en los últimos años el uso de Internet, ha llevado consigo un aumento en la cantidad de información que se encuentra disponible en la red. Sin embargo, no toda esta información resulta útil al usuario y, en la mayoría de los casos, se dan situaciones en las que encontrar lo que se busca puede resultar imposible sin recurrir a aplicaciones complejas o el uso avanzado de los diferentes filtros que proporcionan las herramientas de búsqueda.

El procesamiento del Lenguaje Natural, junto a las aplicaciones de respuesta a preguntas, puede marcar la diferencia en estos casos, ya que abstrae al usuario de las tareas de filtrado del proceso de búsqueda. En vez de navegar entre grandes volúmenes de información o recurrir al manejo de las características más avanzadas, y por ello más complejas, de las que disponen los diferentes motores de búsqueda, el usuario puede encontrar la información deseada, preguntando por ella tal y como lo haría en una conversación con otra persona.

Por estas razones se está empezando a popularizar en Internet el uso de diversas aplicaciones de búsqueda, respuesta a preguntas o asistentes virtuales basados en el uso del Lenguaje Natural.

1.2 Objetivos

El objetivo de este Proyecto de Fin de Carrera consiste en el desarrollo de un bot conversacional basado en lenguaje AIML, que sea capaz de responder de manera autónoma a las preguntas formuladas por los usuarios.

La aplicación debe ser accesible a través de interfaces tanto Web como WAP y debe, asimismo, presentar los mecanismos necesarios para el registro de conversaciones y eventos y la realización de pruebas automatizadas.

Para llevar a cabo este objetivo se realizarán una serie de tareas, las cuales se detallan a continuación:

- **Estudio Preliminar:** Análisis de las diferentes tecnologías relacionadas con el desarrollo de bots conversacionales, haciendo especial hincapié en el lenguaje AIML.
- **Definición de la Base de Conocimiento:** Especificación de las áreas de conocimiento del bot, así como del grado de detalle con el que se tratará cada una de ellas.
- **Análisis y diseño del sistema:** Obtención de la especificación del sistema software a desarrollar. Definición de la arquitectura y diseño de los diferentes componentes que lo forman. Definición de la batería de pruebas (unitarias y de integración) que debe superar el sistema a desarrollar.
- **Implementación e integración:** Desarrollo de los diferentes módulos y herramientas que componen la aplicación, e integración de los mismos.
- **Pruebas unitarias y de Integración:** Pruebas de cada uno de los componentes de la aplicación de manera individual y de la aplicación en su conjunto.

2 Tecnología de Bots

En el presente capítulo se describirá el estado actual de la tecnología en aquellos ámbitos relacionados con la temática del proyecto.

En primer lugar se definirán los diferentes conceptos relacionados con la creación de un bot de ayuda y, posteriormente, se estudiarán ejemplos concretos de tecnologías y aplicaciones.

2.1 Introducción

Un bot de ayuda se basa principalmente en dos conceptos, las aplicaciones de respuesta a preguntas para conseguir la finalidad básica de dar la información que solicita el usuario y la tecnología de bots conversacionales para dotar a la interacción de un mayor realismo.

2.2 Aplicaciones de respuesta a preguntas

Para poder comprender mejor en qué consisten las aplicaciones de Respuesta a Preguntas se deben definir antes ciertos conceptos como Lenguaje Natural, Lenguaje Formal o Procesamiento de Lenguaje Natural.

2.2.1 Lenguaje Natural

El Lenguaje Natural [*Alle94*, *Bate95*] es aquel que utilizamos los humanos de manera cotidiana para comunicarnos entre nosotros. Este tipo de lenguaje surge de la propia necesidad de comunicación y se va enriqueciendo de manera progresiva con la asimilación de nuevas experiencias.

El Lenguaje Natural se caracteriza por la gran riqueza de sus componentes semánticos y el uso de estructuras muy elaboradas, que lo dotan de un gran poder expresivo y permiten la exposición de razonamientos sutiles y análisis de situaciones altamente complejas. Otra propiedad importante del Lenguaje Natural es la polisemántica o, lo que es lo mismo, la posibilidad de variación en el significado de una palabra o una construcción, dependiendo del contexto en el que se encuentre.

Estas características dotan a la comunicación humana, mediante Lenguaje Natural, de una variedad casi infinita y, por ello, dificultan su modelado completo mediante lenguajes formales.

Por lo tanto, el Lenguaje Natural se puede caracterizar por las siguientes propiedades:

1. Riqueza de sus componentes semánticos, que permiten una comunicación muy variada y detallada.
2. Posibilidad de variación del significado de sus componentes según el contexto.
3. Dificultad de una formalización completa.

2.2.2 Lenguaje Formal

El Lenguaje Formal [*Roze97*, *Mate96*] es aquel que se desarrolla de manera específica para definir situaciones que se dan en un ámbito concreto de conocimiento científico. Todas las palabras y oraciones de un Lenguaje Formal están perfectamente definidas, siendo su significado el mismo en cualquier situación, independientemente de su contexto o uso. Otra de las características importantes de este tipo de lenguajes es que no poseen ningún componente semántico fuera de los operadores y relaciones previamente definidos.

Todas estas propiedades hacen del Lenguaje Formal un lenguaje muy preciso y carente de ambigüedad, siendo, por lo tanto, idóneo para modelar cualquier tipo de teoría o conocimiento científico.

En resumen, las características del Lenguaje Formal son las siguientes:

1. Desarrollado específicamente para una teoría preestablecida.
2. Componente semántico mínimo.
3. Preciso y sin ambigüedades.

2.2.3 Lenguaje Natural vs. Lenguaje Formal

Una vez definido qué es Lenguaje Natural y Lenguaje Formal [*Jord92*] se puede determinar el tipo de lenguaje apropiado en diferentes ámbitos de comunicación. En concreto se analizará primero la comunicación humana, posteriormente la comunicación entre máquinas y por último la comunicación humano-máquina.

Por la propia definición de Lenguaje Natural se puede ver que es el tipo de lenguaje apropiado para la comunicación humana, la gran riqueza semántica permite una comunicación muy variada y llena de matices, además tiene la ventaja añadida, a diferencia del Lenguaje Formal, de no requerir un esfuerzo para aprenderlo por parte de los implicados.

El Lenguaje Formal por su parte, es el lenguaje más apropiado para la comunicación entre máquinas, puesto que éstas tienen una comprensión limitada del lenguaje. Por lo tanto, el hecho de que tenga un componente semántico mínimo y que las palabras que lo forman estén perfectamente definidas, lo convierten en el lenguaje idóneo para este ámbito.

Ahora bien, en la comunicación humano-máquina, la elección del tipo de lenguaje no está tan clara, porque las ventajas que cada lenguaje tiene para una de las partes se tornan en inconvenientes para la otra. La gran variedad semántica de los Lenguajes

Naturales supone que sea muy difícil modelarlos para que sean comprensibles por las máquinas, mientras que el conocimiento de un Lenguaje Formal requiere un esfuerzo por parte de los humanos. Hasta el momento, debido al uso extendido de interfaces gráficas basadas en ventanas e iconos, no se había dado demasiada importancia al uso de Lenguaje Natural para esta comunicación, ya que el uso de dichas interfaces puede ser más sencillo y superar la velocidad de escritura de muchos usuarios. Sin embargo, gracias a los recientes avances en el procesamiento del lenguaje oral y a la tecnología de Procesamiento de Lenguaje Natural, parece que una solución más deseable sería el uso de interfaces híbridas de tipo "Gráfico-Lenguaje Natural".

2.2.4 Procesamiento de Lenguaje Natural

El Procesamiento de Lenguaje Natural [*Dale00*, *Carb92*] es una subdisciplina de la Inteligencia Artificial y de la lingüística computacional que se ocupa de la formulación e investigación de mecanismos eficaces computacionalmente para la comunicación entre personas o entre personas y máquinas por medio de lenguajes naturales. El uso de Lenguaje Natural para comunicarse con máquinas facilita la interacción humana y el desarrollo de programas que realicen tareas relacionadas con el propio lenguaje.

Las aplicaciones del Procesamiento de Lenguaje Natural son muy variadas debido a su gran alcance, algunas de dichas aplicaciones son:

- Traducción automática.
- Recuperación de la información.
- Extracción de información y resúmenes.
- Resolución cooperativa de problemas.
- Reconocimiento y procesado de voz.

2.2.5 Respuesta a Preguntas

La Respuesta a Preguntas [*Herm00*, *Hovy00*] es un tipo de aplicación de recuperación de la información. Dada una base de conocimientos, el sistema es capaz de extraer las respuestas adecuadas a preguntas planteadas en Lenguaje Natural. La Respuesta a Preguntas se considera como la evolución en la tecnología de buscadores y es uno de los sistemas de recuperación de información que requiere una tecnología de Procesamiento de Lenguaje Natural más compleja.

Este tipo de aplicaciones, unidas a la tecnología de bots conversacionales, dan lugar a agentes de asistencia técnica muy eficaces, ya que añaden a la utilidad de recuperación de información otros aspectos propios de la comunicación humana que hacen la interacción con el usuario mucho más amigable.

2.3 Bots Conversacionales

Un bot conversacional [*Leon98*, *Maul94*] es un agente *software* diseñado para emular el comportamiento de una persona en una conversación. Para lograrlo se puede recurrir al análisis y comprensión de las estructuras gramaticales o, como es más habitual, al

reconocimiento de palabras clave. Con este último método el bot es capaz de seguir una conversación de forma más o menos lógica, aunque realmente no comprenda el significado de la misma.

Desde el punto de vista de la interfaz de usuario, lo normal es que la conversación se produzca de manera textual, aunque son fácilmente integrables con tecnologías de reconocimiento y síntesis de voz, dando lugar a interfaces mucho más sencillas de usar por parte de un humano.

Actualmente, la dificultad de su programación implica que conseguir un bot conversacional realista suponga una gran inversión de recursos. Sin embargo, en los últimos tiempos se está tomando conciencia de la gran utilidad de estos sistemas a la hora de dar información sobre materias concretas, utilidad que se incrementa aun más en aquellos que incorporan capacidad de aprendizaje. Es por ello que se están destinando muchos esfuerzos en la simplificación de su elaboración, ya sea en mejoras de desarrollo o en la mayor modularidad de las librerías, tanto de vocabulario como de los algoritmos de inteligencia artificial.

2.3.1 Ejemplos de Bots conversacionales

A día de hoy existen múltiples bots conversacionales accesibles vía Web, a través de programas de mensajería, integrados en aplicaciones o como programas independientes. También varía su interfaz, pudiendo ser desde una sencilla línea de comandos hasta los más avanzados con reconocimiento y síntesis de voz y aspectos visuales que les dotan de un mayor realismo. Algunos de estos bots conversacionales más conocidos son:

ELIZA

Eliza *[Weiz65]* es el programa precursor de los bots conversacionales, fue creado en 1966 por Joseph Weizenbaum como una parodia del psicólogo americano Carl Rogers. Al elegir el campo de la psicoterapia evitaba la necesidad de introducir una base de conocimiento, ya que es posible mantener una conversación simplemente pidiendo información adicional sobre la declaración del usuario. Para lograr este efecto, Eliza se basa en el uso de frases tipo, como "ya entiendo", "cuéntame más", etc. y en el procesado de la entrada del usuario sustituyendo palabras clave, reescribiendo dicha entrada de tal forma que se puede crear una pregunta a partir de las interacciones del usuario. Por ejemplo si el usuario dice "Estoy preocupado" Eliza podría responder "¿Por qué dice que está preocupado?" simplemente añadiendo una construcción sencilla y cambiando la persona del verbo.

Realmente Eliza no entiende la entrada del usuario, pero los usuarios asumen de manera subconsciente que las preguntas que realiza Eliza implican interés y preocupación por sus problemas, incluso en aquellos casos en los que los usuarios son conscientes de estar hablando con una máquina. A este efecto, por el cual un usuario asume de manera subconsciente que el comportamiento de una máquina es análogo al de un humano, se le conoce como "Efecto Eliza" en honor al bot conversacional creado por Joseph Weizenbaum.

A.L.I.C.E.

A.L.I.C.E. (*Artificial Linguistic Internet Computer Entity*) es uno de los bots conversacionales más conocidos. Está inspirado en Eliza y su desarrollo comenzó en 1995, siendo el autor original del proyecto el Dr. Richard Wallace. En 1998 se reescribió en Java y, tras la publicación en 2001 de la especificación del lenguaje AIML

(Lenguaje basado en XML y desarrollado específicamente para la creación de A.L.I.C.E.) [*Bush01*, *Ring02*] numerosos desarrolladores han continuado la labor de Richard Wallace. Ha sido designado ganador del Premio Loebner (Concurso anual que premia a los bots conversacionales con un comportamiento más humano) en los años 2000, 2001 y 2004.

A lo largo de los años se han ido utilizando distintas tecnologías en la implementación de A.L.I.C.E. dando lugar a distintas versiones del programa:

- **Program A:** La primera versión de A.L.I.C.E. se creó en 1995 usando SETL, que es un lenguaje poco conocido que hace uso de la teoría de conjuntos y lógica matemática. Aunque el proyecto original estaba disponible como software gratuito, no atrajo apenas colaboradores hasta que se realizo la migración a la plataforma Java en 1998. La primera implementación de A.L.I.C.E. en Java tomó el nombre de "*Program A*".
- **Program B:** El lanzamiento de *Program B* supuso un gran avance en el desarrollo de A.L.I.C.E. con más de 300 desarrolladores participando en su creación. Además el lenguaje AIML evolucionó para convertirse en una gramática totalmente conforme con XML, permitiendo la utilización multitud de editores y herramientas para su desarrollo. *Program B* se convirtió en el primer software AIML gratuito ampliamente adoptado por los programadores de bots conversacionales.
- **Program C:** Jacco Bikker creó en el año 2000 la primera implementación de AIML en C/C++ lo cual supuso el punto de partida para la utilización del motor *Alicebot* en numerosos ámbitos, como scripts CGI, IRC, mensajería instantánea, etc. Esta implementación, tanto del motor *Alicebot* como del propio AIML para C/C++ se pasó a conocer como "*Program C*".
- **Program D:** *Program B* está basado en tecnología previa a Java 2 y, aunque puede ejecutarse en múltiples plataformas, no puede hacer uso de las características más recientes de Java como *Swing* o *Collections*. Así pues, para evitar estas limitaciones, Jon Baer recodificó *Program B* con tecnología Java 2 añadiendo además múltiples funcionalidades adicionales. Este gran cambio, tanto en la interfaz como en el propio núcleo del programa y el hecho de que Jon Baer llamase a su bot "*Danny*", supuso que se conociese a esta versión por la letra "D". Desde su creación en el año 2000 *Program D* [*Bush02*] ha sido la única versión java del motor *Alicebot* que ha contado con un apoyo activo por parte de los desarrolladores.

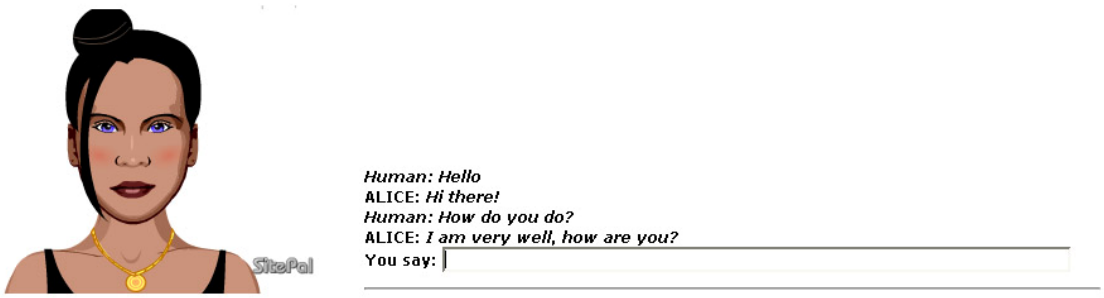

Ilustración 1: Interfaz del bot conversacional A.L.I.C.E.

Otros Bots Conversacionales

Si bien Eliza [*ElizURL*] y A.L.I.C.E. [*AlicURL*] son los bots conversacionales más conocidos, el primero por ser precursor de este tipo de aplicaciones y el segundo por ser posiblemente el bot conversacional más completo hasta la fecha, existen muchos otros, como por ejemplo:

- iGod [*iGodURL*] (Web)
- Mitsuku [*MitsURL*] (Web)
- JabberWacky [*JabbURL*] (Web y mensajería instantánea)

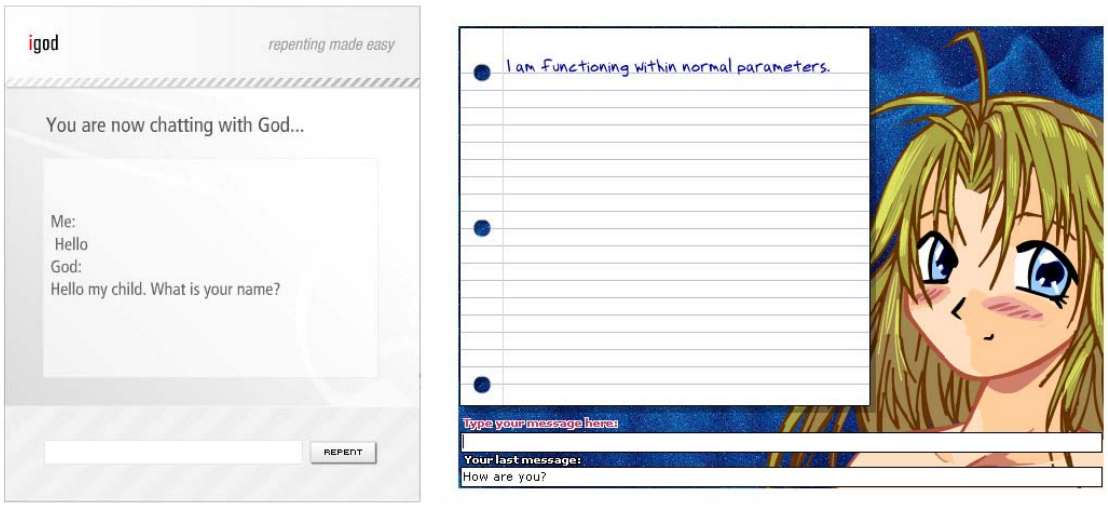

Ilustración 2: Interfaces de los bots conversacionales iGod y Mitsuku

También están empezando a extenderse en el ámbito institucional y comercial como asistentes virtuales, así podemos ver cómo diversas instituciones y compañías los han adoptado para complementar sus páginas de ayuda, algunos ejemplos pueden ser:

- i [*iURL*] (Ministerio de Cultura)
- Bea [*CajaURL*] (Caja Madrid)
- Anna [*IkeaURL*] (Ikea)
- Telefónica Online [*TeleURL*]

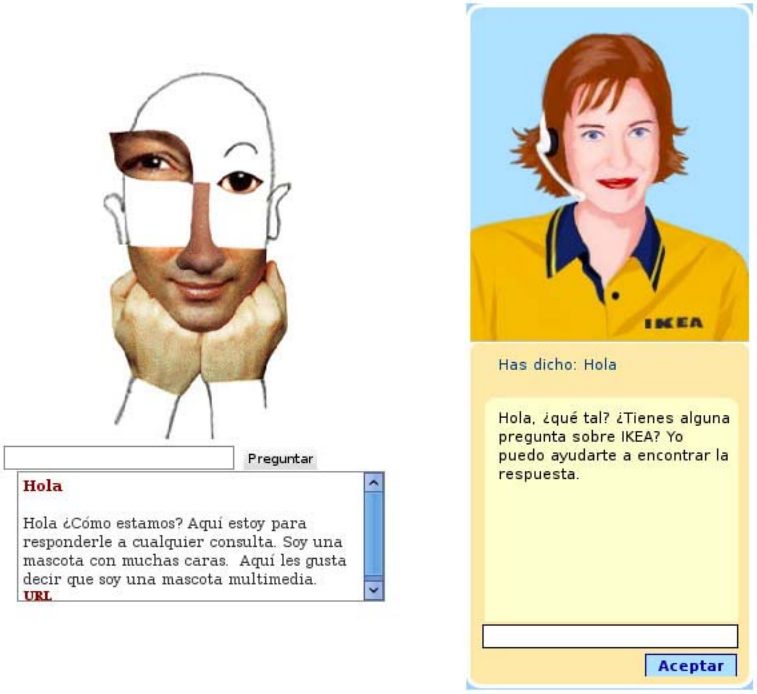

Ilustración 3: Interfaces de los asistentes virtuales del Ministerio de Cultura e Ikea

Ilustración 4: Interfaz del asistente virtual de Telefónica Online

3 Descripción de Tecnologías Usadas

A lo largo de este proyecto se hará referencia a las múltiples tecnologías necesarias para llevarlo a cabo. Para facilitar la comprensión por parte del lector, en el presente capítulo, se darán unas nociones básicas sobre dichas tecnologías, haciendo especial hincapié en aquellas que puedan ser menos conocidas.

3.1 Introducción

Este proyecto de fin de carrera requiere del uso de diversas tecnologías para su realización. La mayoría de estas tecnologías están lo suficientemente extendidas como para necesitar únicamente una pequeña introducción, mientras que otras como AIML requieren de una explicación más extensa para poder entender la utilidad y funcionamiento dentro del ámbito del proyecto.

Así pues, a lo largo de este capítulo se nombrarán las diferentes tecnologías usadas, junto con una pequeña descripción que pueda ayudar a comprender la función que desempeñan en el desarrollo del bot de ayuda, incidiendo en el lenguaje AIML el cual se intentará definir con el suficiente grado de detalle como para ser capaz de implementar una base de conocimiento sencilla.

3.2 Java

Java [*Gosl05*, *JavaURL*] es un lenguaje de programación orientado a objetos desarrollado por *Sun Microsystems* y lanzado en 1995 como componente principal de la plataforma java de *Sun Microsystems*. Java toma mucha sintaxis de C y C++ pero posee un modelo de objetos más simple y elimina algunas de las utilidades de bajo nivel como la manipulación directa de punteros y memoria. Las aplicaciones Java suelen compilarse a *bytecode* que se puede ejecutar sobre cualquier máquina virtual de Java (JVM) independientemente de la plataforma.

Los cinco principales objetivos en la creación del lenguaje Java fueron:

- Usar una metodología simple y orientada a objetos.
- Ser robusto y seguro.
- Permitir su ejecución independiente de la plataforma.
- Ejecutarse con alto rendimiento.
- Ser interpretado, permitir hilos y ser dinámico.

3.3 XML

XML [*XmlURL*], siglas de *eXtensible Markup Language* (lenguaje de marcas ampliable), es un metalenguaje de etiquetas extensible desarrollado por el W3C (*World Wide Web Consortium*). Esto significa que no es un lenguaje en particular sino que sirve para definir lenguajes para diferentes necesidades. XML es una tecnología sencilla que permite la compatibilidad entre sistemas para compartir información de una manera segura fiable y fácil.

Las ventajas que proporciona XML son:

- **Extensibilidad:** Una vez creado un XML se puede extender con nuevas etiquetas de tal forma que los clientes con versiones anteriores puedan seguir accediendo sin problemas.
- **Estandarización:** El analizador de XML es estándar por lo que no es necesario crear un analizador específico para cada uno de los lenguajes definidos por XML.
- **Compatibilidad:** Debido a la estandarización un XML es sencillo de procesar por lo que mejora la compatibilidad entre aplicaciones muy diferentes.

3.4 HTML

HTML [*HtmlURL*, *XhtmURL*], siglas de *HyperText Markup Language* (Lenguaje de marcas de hipertexto), es el lenguaje de marcado más utilizado en la actualidad para la construcción de páginas Web. HTML se usa tanto para describir el contenido y la estructura del texto de una página Web (Definiendo elementos como enlaces, párrafos, encabezados, etc.) como para complementar dicho texto con contenidos adicionales como imágenes o animaciones.

HTML es un término genérico que se usa tanto para referirse tanto a los lenguajes HTML que descienden de SGML (HTML 4.01 y anteriores) como a los que lo hacen de XML (XHTML 1.0 y posteriores).

3.5 WML

WML [*WmlURL*] (*Wireless Markup Language* o lenguaje de marcas inalámbrico) es un lenguaje cuyo origen es el XML y que se utiliza para construir páginas Web que puedan visualizarse de manera correcta en dispositivos dotados de la tecnología WAP (*Wireless Application Protocol* o protocolo de aplicaciones inalámbricas), como puedan ser teléfonos móviles y los asistentes personales digitales (PDA). La visualización de la página dependerá del dispositivo que se use y de la forma en que éste interprete el código. WML es un metalenguaje, lo que implica que además de usar etiquetas predefinidas se pueden crear componentes propios.

3.6 CSS

CSS [*CssURL*] (*Cascading Style Sheets* u hojas de estilo en cascada) es el lenguaje formal utilizado para definir la presentación de un documento estructurado en HTML o XML (Y por lo tanto todos los lenguajes derivados de éste como por ejemplo XHTML). Aunque las versiones más antiguas de HTML permitían aplicar formatos, CSS pretende separar la estructura de los documentos de su presentación.

Las ventajas que supone el uso de hojas de estilo son:

- **Control centralizado:** Se puede definir la apariencia de todo un sitio Web de manera centralizada agilizándose de manera considerable la actualización de la misma.
- **Modularidad:** Al disociar el contenido y la presentación se da la posibilidad de aplicar apariencias diferentes a un mismo contenido, siendo posible por ejemplo que una persona con deficiencias visuales use su propia hoja de estilo para visualizar mejor cierto contenido. También esto permite definir diferentes hojas de estilo para adaptar el contenido a los diferentes dispositivos en los que se va a visualizar.
- **Diferenciación:** La separación de contenido y apariencia supone que cada uno de los documentos que los definen estarán más estructurados y serán más sencillos de comprender.

3.7 Javascript

Javascript [*Flan02*, *JscrURL*] es un lenguaje de programación interpretado, utilizado principalmente en páginas Web y con una sintaxis muy parecida a la de los lenguajes Java y C. *Javascript* se utiliza en páginas Web HTML para realizar tareas y operaciones únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Para ello se ejecuta en el agente de usuario (Todos los navegadores modernos interpretan el código *JavaScript* integrado dentro de las páginas Web) al mismo tiempo que las sentencias van descargándose junto con el código HTML.

3.8 JSP y JSTL

JavaServer Pages (JSP) [*JspURL*] es una tecnología desarrollada por la compañía *Sun Microsystems* Java que permite generar contenido dinámico para Web, en forma de documentos HTML, XML o de otro tipo.

Las JSP's permiten la utilización de código Java mediante *scripts* que se ejecutan en el servidor, siendo posible además, utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Librerías de Etiquetas (*TagLibs* o *Tag Libraries*) externas e incluso personalizadas.

Una de esas librerías de etiquetas es la *JavaServer Pages Standard Tag Library* (JSTL) [*JstlURL*] que proporciona utilidades ampliamente utilizadas en el desarrollo de páginas Web dinámicas como pueden ser análisis de expresiones, acceso a bases de datos o dar formato a información.

3.9 AIML

Cómo se ha explicado en el capítulo “Tecnología de Bots”, AIML es un lenguaje de programación basado en XML que define la gramática para la creación de agentes software con lenguaje natural.

En este apartado se explicarán las funcionalidades básicas que permite el lenguaje AIML a través de sus diferentes elementos [*Wall03*, *Sesh05*, *Aiml02*].

3.9.1 Categorías

El elemento básico de un documento AIML es la categoría (<*category*>). La categoría más simple consta de un patrón (<*pattern*>) y una plantilla (<*template*>). Así, por ejemplo, podríamos escribir lo siguiente:

```
<category>
      <pattern>QUE ES AIML</pattern>
      <template>
            AIML es un lenguaje de programación basado en XML.
      </template>
</category>
```

Con esta categoría, cuando un usuario escribiese la frase exacta que aparece en el patrón, en nuestro ejemplo "¿Qué es AIML?", el intérprete le respondería con la frase que se haya definido como plantilla, en este caso "AIML es un lenguaje de programación basado en XML".

Hay que tener en cuenta que AIML hace distinciones entre palabras con tilde y sin ella y reconoce los distintos signos de puntuación, con lo que habría que preprocesar la entrada del usuario eliminando ciertos elementos para no tener que hacer frente a todas las posibles variaciones. Esto, por supuesto, supone una pérdida de información pero si se tiene cuidado a la hora de diseñar la base de conocimientos para minimizar el impacto de esta pérdida se consigue a cambio reducir drásticamente su complejidad.

Cuando el patrón tiene que coincidir de manera exacta se conoce como patrón atómico. Sin embargo un usuario puede hacer una misma pregunta de diferentes maneras, y por ello AIML define los comodines y las reducciones para dotar al bot conversacional de mayor versatilidad.

3.9.2 Comodines

Para definir patrones genéricos se puede hacer uso de los símbolos “*” y “_”. Gracias a ellos podemos definir patrones a los que el intérprete es capaz de responder de manera adecuada aunque coincida sólo una parte de la entrada del usuario,.

Los comodines pueden equivaler a una palabra o a un grupo de palabras, siendo el patrón que sólo consta de un comodín el más genérico ya que cualquier entrada se correspondería con dicho patrón. Un ejemplo de uso de comodines podría ser:

```
<category>
      <pattern>HOLA *</pattern>
      <template>Hola ¿Qué tal?</template>
</category>
```

Con esta categoría cualquier frase que comience por la palabra "Hola" recibirá como respuesta "Hola ¿Qué tal?".

Como complemento del comodín, el lenguaje AIML dispone de la etiqueta <*star/*> que toma el valor de la palabra o palabras a las que sustituye el comodín. Por lo tanto esta etiqueta es muy útil para usar las propias palabras del usuario, bien en la propia respuesta o bien para procesar la entrada del usuario con mayor precisión. En el siguiente código se puede ver un ejemplo básico del uso de dicha etiqueta:

```
<category>
      <pattern>NO ME GUSTA *</pattern>
      <template>¿Por qué no te gusta <star/>?</template>
</category>
```

Con este sencillo patrón se puede devolver una pregunta adecuada a cada entrada del usuario que comience por “No me gusta…”. Así pues si el usuario dice por ejemplo “No me gusta bailar” el bot responderá “¿Por qué no te gusta bailar?” mientras que si el usuario dice “No me gusta el pescado” cambiará la respuesta para responder “¿Por qué no te gusta el pescado?”. Como se puede ver en este ejemplo en el primer caso la etiqueta <*star/*> estaría tomando el valor “bailar” y en el segundo su valor sería “el pescado”.

El intérprete comprueba los patrones con comodines de mayor a menor grado de especificación, así por ejemplo si tenemos los patrones "¿Qué es AIML?", "¿Qué es *?" y "*", comprobaría si la entrada corresponde con el primero, si no probaría con el segundo y si tampoco coincide daría la respuesta contenido en el tercer patrón, ya que como se ha indicado anteriormente cualquier entrada corresponde con "*". Esto resulta muy útil ya que el bot siempre podrá dar una respuesta aunque no tenga cierta entrada en concreto definida en el archivo AIML.

En este caso el código resultante sería:

```
<category>
      <pattern>QUE ES AIML</pattern>
      <template>
            AIML es un lenguaje de programación basado en XML.
      </template>
</category>

<category>
      <pattern>QUE ES *</pattern>
      <template>No sé qué es eso…</template>
</category>

<category>
      <pattern>*</pattern>
      <template>
            No he entendido tu pregunta, ¿podrías formularla de
            otra manera?
      </template>
</category>
```

Así que, si el usuario preguntase "¿Qué es AIML?" el bot respondería "AIML es un lenguaje de programación basado en XML.", si realizase cualquier otra pregunta que empezase por "¿Qué es …?" respondería "No sé qué es eso…" y, para cualquier entrada que no se corresponda con dicha estructura respondería "No he entendido tu pregunta, ¿podrías formularla de otra manera?", con lo que puede que la siguiente entrada sí que case con una de las categorías.

Hay una excepción en el orden de comprobación de los patrones que resulta del uso del símbolo "_" en lugar de "*". Un patrón que contiene el comodín "_" se comprobará siempre antes que un patrón que no lo tiene, por lo tanto, si usamos el siguiente código:

```
<category>
      <pattern>QUE ES AIML</pattern>
      <template>
            AIML es un lenguaje de programación basado en XML
      </template>
</category>

<category>
      <pattern>QUE ES _</pattern>
      <template>No sé qué es eso…</template>
</category>

<category>
      <pattern>*</pattern>
      <template>
            No he entendido tu pregunta, ¿podrías formularla de
            otra manera?
      </template>
</category>
```

Obtendríamos la respuesta “No sé qué es eso…” para cualquier frase que empiece por “Qué es …” incluso en el caso de que la pregunta fuese “¿Qué es AIML?”.

3.9.3 Reducciones

Son unas herramientas muy útiles a la hora de crear un bot conversacional ya que permiten llevar a cabo múltiples acciones para la mejor comprensión de la entrada del usuario por parte del bot.

Las reducciones se llevan a cabo mediante la etiqueta <*srai*> y permiten llevar a cabo las siguientes tareas:

- Simplificar formas gramaticales complejas.
- Dividir la entrada en varias partes y combinar respuestas.
- Definir diferentes formas de decir una misma cosa.
- Aplicar correcciones ortográficas o gramaticales.
- Detectar palabras clave.
- Implementar condicionales.

En el siguiente ejemplo se demuestran algunas de las utilidades de las reducciones:

```
<category>
      <pattern>ME PODRIAS DECIR QUE ES *</pattern>
      <template>
            <srai>QUE ES <star/></srai>
      </template>
</category>

<category>
      <pattern>K ES *</pattern>
      <template>
            <srai>QUE ES <star/></srai>
      </template>
</category>

<category>
      <pattern>QUE ES *</pattern>
      <template>
            Lo siento, no sé qué es <star/>.
      </template>
</category>
```

En la primera categoría se hace uso de la utilidad de simplificación, así si alguien dice "¿Me podrías decir que es AIML?" el intérprete vería que coincide con el primer patrón y que en ella el comodín "*" está tomando el valor "AIML". Así pues compone

la entrada "¿Qué es AIML?" y vuelve a comparar con todas las categorías viendo que coincide con la tercera, siendo la respuesta final "Lo siento, no sé qué es AIML".

En la segunda categoría se utiliza la reducción para corregir la entrada. Si el usuario dice "K es AIML" el intérprete sustituirá esa entrada por "¿Qué es AIML?" y volverá a comprobar si coincide con el patrón de alguna categoría, devolviendo la misma respuesta que en el caso anterior.

Por lo tanto, para tres entradas distintas "¿Me podrías decir que es AIML?", "K es AIML" y "¿Qué es AIML?" la respuesta del intérprete es la misma ya que ha reducido todas ellas a una forma básica "¿Qué es …?" a la que se puede responder fácilmente.

3.9.4 Control Interno

Aparte de las diferentes utilidades de las que disponemos para el reconocimiento de patrones y definición de respuestas asociadas, AIML también proporciona una serie de herramientas para el control de la conversación. Para ello permite definir variables exclusivas para cada usuario, darles valores y llevar a cabo ciertas operaciones con ellas de manera transparente al exterior. La etiqueta principal que permite llevar a cabo todas estas funciones es la etiqueta <*think*> ya que cualquier instrucción que se defina dentro de dicha etiqueta no se mostrará al usuario. Estas instrucciones deben ir asociadas a un patrón por lo que la etiqueta <*think*> se sitúa dentro de la etiqueta <*template*>, la cual a su vez puede contener otras instrucciones que sí que se mostrarán al usuario.

Un ejemplo donde se puede ver las capacidades de esta utilidad es la personalización de conversaciones al guardar ciertos parámetros en memoria para utilización posterior, como el nombre de usuario:

```
<category>
      <pattern>ME LLAMO *</pattern>
      <template>
            <think>
                  <set name="username"><star/></set>
            </think>
            Encantado de conocerte <star/>.
      </template>
</category>
```

En este ejemplo se puede ver como el usuario nos da su nombre, el cual se guarda en una variable de manera transparente para el usuario y se le da una respuesta adecuada. Así si el usuario nos dice "Me llamo Juan", el bot conversacional respondería "Encantado de conocerte Juan", y además le daría a la variable "*username*" el valor "Juan" que podría utilizar posteriormente, por ejemplo así:

```
<category>
      <pattern>RECUERDAS COMO ME LLAMO</pattern>
      <template>
            Por supuesto, te llamas <get name="username">.
      </template>
</category>
```

Por lo que si el usuario anterior pregunta "¿Recuerdas cómo me llamo?" el bot obtendría el nombre de usuario de la variable "*username*" y respondería "Por supuesto, te llamas Juan".

El desarrollador puede usar los nombres de las variables que considere adecuados, sin embargo hay una variable que tiene un uso particular por lo cual tiene un nombre específico y un tratamiento ligeramente diferente al resto de variables, se trata de "*topic*". Esta variable nos permite tener en cuenta el hilo de la conversación y discriminar entre unas respuestas u otras que en condiciones normales podrían llevar a confusión. Para ello según el tema tratado se puede guardar en la variable "*topic*" un descriptor por ejemplo "ordenadores", "coches", "cine", etc. Podemos ver cómo se guarda el tema tratado en el siguiente código:

```
<category>
      <pattern>HABLEMOS DE COCHES</pattern>
      <template>
            <think>
                  <set name="topic">coches</set>
            </think>
            Vale, me parece un tema muy interesante.
      </template>
</category>
```

Así, cuando una entrada de usuario no sea concreta, se puede buscar primero en el tema de conversación tratado hasta el momento. Por ejemplo si un usuario pregunta "¿Qué marca te gusta más?" el bot conversacional no podría saber a qué se refiere a no ser que sepa de qué tema se ha hablado previamente, en cuyo caso podría dar una respuesta adecuada a cada tema, por ejemplo:

```
<topic name="coches">
      <category>
            <pattern>QUE MARCA TE GUSTA MAS</pattern>
            <template>
                  No tengo preferencias de marca, pero siempre
                  me han gustado los coches deportivos.
            </template>
     </category>
</topic>
```

```
<topic name="coches">
      <category>
            <pattern>QUE MARCA TE GUSTA MAS</pattern>
            <template>
                  No tengo preferencias de marca, pero siempre
                  me han gustado los coches deportivos.
            </template>
     </category>
</topic>
```

Así, para un mismo patrón el intérprete dará la respuesta que se corresponda con el tema que se hubiese estado tratando hasta el momento, permitiendo seguir una conversación de manera más adecuada.

En el ejemplo el usuario había dicho en la interacción previa “Hablemos de coches”, así que el intérprete guardó en la variable “*topic"* el valor “coches” por lo tanto al preguntarle después “¿Qué marca te gusta más?” podrá comprobar que le está preguntando por una marca de coches y responderá “No tengo preferencias de marca, pero siempre me han gustado los coches deportivos.”.

3.9.5 Otros

AIML proporciona otras utilidades para dotar de mayor credibilidad a las conversaciones. Algunas de dichas funciones pueden ser dar varias respuestas a un mismo patrón y que el intérprete seleccione una de dichas respuestas de manera aleatoria, definir condicionales para dar una respuesta u otra en función de la entrada del usuario o guardar en memoria la última respuesta para poder seguir interacciones consecutivas.

Ejemplo de respuesta aleatoria:

```
<category>
      <pattern>ADIOS</pattern>
      <template>
            <random>
                  <li>Adiós.</li>
                  <li>Hasta otra.</li>
                  <li>Hasta pronto.</li>
                  <li>Que tengas un buen día.</li>
            </random>
      </template>
</category>
```

Si un usuario se despide diciendo "Adiós" el intérprete elige una de las respuestas posibles de manera aleatoria, con lo que la conversación es más realista.

Ejemplo de memoria de respuesta:

```
<category>
      <pattern>HOLA *</pattern>
      <template>
            Hola ¿Es la primera vez que entras aquí?
      </template>
</category>

<category>
      <pattern>SI</pattern>
      <that>Hola ¿Es la primera vez que entras aquí?</that>
      <template>Bienvenido, ¿Cómo te llamas?</template>
</category>

<category>
      <pattern>NO</pattern>
      <that>Hola ¿Es la primera vez que entras aquí?</that>
      <template>
            Pues no recuerdo tu cara ¿Cómo te llamas?
      </template>
</category>
```

Los patrones "Sí" y "No" son muy generales, así que si no se sabe a qué está respondiendo el usuario no se puede dar una respuesta adecuada, sin embargo, gracias a la etiqueta <*that*> el intérprete comprueba no sólo si la entrada coincide con el patrón si no además qué fue lo último que preguntó el bot, siendo capaz de responder en consecuencia.

3.9.6 Conclusiones

Como puede observarse por las funcionalidades que aporta AIML, un bot conversacional basado en este lenguaje no comprende la conversación en la que toma parte, depende del programador aportar la inteligencia del programa al crear una base de conocimiento lo suficientemente amplia y versátil como para poder seguir de manera fluida las conversaciones que se le puedan presentar

4 Análisis de Requisitos

En este capítulo se dará una visión general del sistema a implementar y, sobre dicha visión, se obtendrá una especificación detallada de los requisitos que debe cumplir la aplicación para satisfacer las necesidades del usuario.

4.1 Introducción

Una vez se poseen los conocimientos teóricos necesarios para la creación de un bot de ayuda se pueden sentar las bases del desarrollo del proyecto.

Se determinarán cuáles son los requisitos funcionales y no funcionales de la aplicación, lo cual permitirá definir el alcance del trabajo que hay que llevar a cabo. También se darán una serie de casos de uso que ilustrarán con mayor detalle las funcionalidades que se esperan de la aplicación.

4.2 Visión del Sistema

En el escenario básico uno o más usuarios se conectan, bien mediante un navegador a la interfaz Web o bien a través de un móvil a la interfaz WAP de la aplicación, la cual está alojada en un servidor.

Una vez conectado, el usuario realiza sus consultas al bot de ayuda mediante una línea de texto y éste, tras consultar en la base de conocimiento, le devuelve una respuesta adecuada a través de la misma interfaz mediante la cual accedió el usuario.

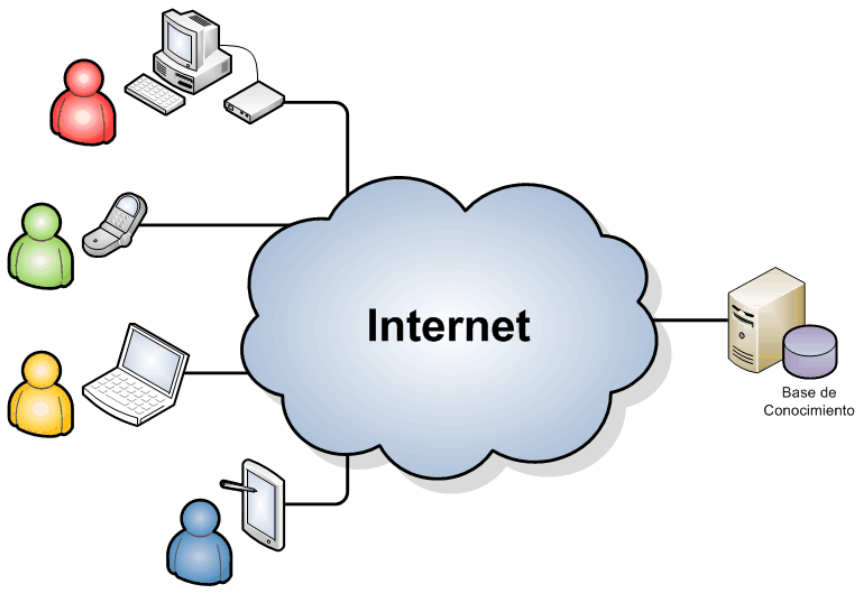

Ilustración 5: Esquema de Escenario Básico

4.3 Requisitos

Partiendo de la visión que se ha dado en el apartado anterior, se pasa a detallar los requisitos del sistema. En el ámbito de este documento se van a distinguir dos tipos de requisitos:

1. **Requisitos funcionales:** Funciones que lleva a cabo el sistema.
2. **Requisitos no funcionales:** Limitaciones que condicionan el funcionamiento del sistema expuesto mediante los requisitos anteriores.

4.3.1 Requisitos Funcionales

- **Respuesta a preguntas:** El sistema debe ser capaz de realizar el análisis de la entrada del usuario y responder consecuentemente, bien con una respuesta adecuada en caso de estar contemplada la entrada en la base de conocimiento o bien indicando al usuario que no dispone de la información requerida en caso contrario.

- **Base de conocimiento conforme a las necesidades del cliente:** La base de conocimiento debe abarcar todos los temas solicitados por el cliente, es decir la sección de ayuda de la página de Orange. Asimismo debe incluir temas de conocimiento adicionales para dotar de mayor realismo a las conversaciones.

- **Redirección a página Web:** En los casos en los que exista una página Web adecuada, con información ampliada sobre el tema que ha solicitado el usuario, debe ser posible ponerla a disposición de éste de manera adicional a la respuesta del asistente.

- **Emociones dependiendo de la pregunta:** Para dotar de una mayor expresividad al bot, en aquellas interfaces en las que sea posible, debe darse información complementaria a la respuesta del asistente, indicando las emociones que pueden suponerse como consecuencia de la interacción con el usuario.

- **Logs de conversaciones:** Todas las conversaciones deben ser grabadas y almacenadas, de tal forma que sean útiles en caso de querer detectar posibles funcionamientos anómalos o ampliar la base de conocimiento con aquellos temas para los que el asistente no haya sido capaz de dar una respuesta.

- **Pruebas automatizadas:** El sistema debe disponer de un mecanismo de pruebas automatizadas que permitan controlar su correcto funcionamiento de manera rápida y segura.

4.3.2 Requisitos no Funcionales

- **Sistema multiusuario:** El sistema debe ser capaz de dar servicio a múltiples usuarios de manera simultánea, permitiendo llevar cada conversación y la información asociada a la misma de manera independiente.

- **Escalabilidad:** El sistema debe ser capaz de soportar cargas de trabajo elevadas, permitiendo el uso por parte de un gran número de usuarios sin que ello suponga una disminución de la calidad del servicio.

- **Extensibilidad:** El sistema debe ofrecer una interfaz cómoda y común para el desarrollo de nuevos componentes que extiendan su funcionalidad o su interfaz gráfica y que puedan acoplarse fácilmente a él. Además su base de conocimiento debe poder ser modificada o ampliada de manera sencilla, tanto en el número de temas tratados como en la información disponible para dichos temas.

- **Amigable e intuitivo:** Es deseable que el sistema sea accesible para cualquier usuario y que no requiera de conocimientos técnicos excesivos para su uso. Además es recomendable que las interfaces sean agradables para los usuarios, tanto en manejo como en aspecto.

- **Compatibilidad:** Sería conveniente que el sistema pudiera ejecutarse en la mayor variedad de plataformas posible, de tal manera que su instalación sea sencilla independientemente de los recursos disponibles.

- **Seguridad:** El sistema debe disponer de protección frente a posibles ataques que se puedan realizar a través de él al servidor en el que esté alojado.

4.4 Casos de Uso

En los casos de uso se describe qué es lo que debe hacer la aplicación dependiendo de las distintas interacciones de los actores implicados con la misma. En el caso del bot de ayuda se distinguirán dos tipos de casos de uso, aquellos en los que interviene un usuario y aquellos en los que interviene un administrador.z

4.4.1 Casos de Uso de Usuario

En estos casos de uso interviene un usuario que puede ser bien un humano o un programa externo que pretende obtener una respuesta por parte del bot de ayuda a cierta entrada.

Acceso a través de interfaz Web

El usuario accede a la aplicación mediante un navegador Web. A través de la interfaz introduce una petición de ayuda mediante una cadena de texto. La entrada de texto es procesada por el intérprete de AIML y éste devuelve una respuesta compuesta por texto y, de manera opcional, por una URL y un indicador del estado de ánimo. La respuesta se le muestra al usuario a través de la interfaz Web.

Caso de uso 1	**Acceso a través de interfaz Web**
Descripción	La aplicación debe responder a un usuario que se conecta mediante un navegador Web.
Actores	Usuario
Precondiciones	El usuario se conecta a la aplicación mediante un navegador Web.
Secuencia normal	1. El usuario introduce la entrada a través de la interfaz Web. 2. El intérprete recibe la entrada del usuario, un identificador del mismo y un identificador del bot al que se envía la petición. 3. El intérprete analiza la entrada, la procesa y devuelve una respuesta compuesta por una cadena de texto y, de manera opcional, una URL y un indicador del estado de ánimo. 4. Se muestra la respuesta de texto y la posible información adicional al usuario a través de la interfaz Web. 5. Se graba en un registro tanto la entrada del usuario como la respuesta asociada.
Poscondiciones	La interacción con el usuario queda almacenada en el registro de conversaciones.

Tabla 1: Acceso a través de interfaz Web

Acceso a través de interfaz WAP

El usuario accede a la aplicación mediante un móvil con soporte WAP. A través de la interfaz introduce una petición de ayuda mediante una cadena de texto. La entrada de texto es procesada por el intérprete de AIML y éste devuelve una respuesta compuesta por texto y, de manera opcional, por una URL y un indicador del estado de ánimo. La respuesta se le muestra al usuario a través de la interfaz WAP.

Caso de uso 2	**Acceso a través de interfaz WAP**
Descripción	La aplicación debe responder a un usuario que se conecta a través de un móvil WAP.
Actores	Usuario
Precondiciones	El usuario se conecta a la aplicación mediante un móvil a la interfaz WAP.
Secuencia normal	1. El usuario introduce la entrada a través de la interfaz WAP. 2. El intérprete recibe la entrada del usuario, un identificador del mismo y un identificador del bot al que se envía la petición. 3. El intérprete analiza la entrada, la procesa y devuelve una respuesta compuesta por una cadena de texto y, de manera opcional, una URL y un indicador del estado de ánimo. 4. Se muestra la respuesta de texto y la posible información adicional al usuario a través de la interfaz WAP. 5. Se graba en un registro tanto la entrada del usuario como la respuesta asociada.
Poscondiciones	La interacción con el usuario queda almacenada en el registro de conversaciones.

Tabla 2: Acceso a través de interfaz WAP

Acceso a través de servlet

El usuario se conecta directamente al *servlet* de la aplicación introduciendo en la propia dirección URL el texto de la petición y el identificador del bot de ayuda al que va dirigido. La entrada de texto es procesada por el intérprete de AIML y éste devuelve una respuesta compuesta por texto y, de manera opcional, por una URL y un indicador del estado de ánimo. La respuesta se devuelve en forma de una HTML simple con una estructura fija.

Caso de uso 3	**Acceso a través de servlet**
Descripción	La aplicación debe responder a un usuario que se conecta directamente al *servlet* de la aplicación.
Actores	Usuario
Precondiciones	El usuario se conecta a la aplicación a través del *servlet*.

Secuencia normal	1. El usuario introduce la URL del servicio junto con la entrada de texto y el identificador del bot al que le quiere enviar dicha entrada. 2. El intérprete recibe la entrada del usuario, un identificador del mismo y el identificador del bot. 3. El intérprete analiza la entrada, la procesa y devuelve una respuesta compuesta por una cadena de texto y, de manera opcional, una URL y un indicador del estado de ánimo. 4. Se devuelve la respuesta de texto y la posible información adicional al usuario en forma de una página HTML simple. 5. Se graba en un registro tanto la entrada del usuario como la respuesta asociada.
Poscondiciones	La interacción con el usuario queda almacenada en el registro de conversaciones.

Tabla 3: Acceso a través de servlet

4.4.2 Casos de Uso de Administrador

En estos casos de uso interviene un usuario de tipo administrador que no pretende obtener una respuesta del bot sino variar su comportamiento o analizar su funcionamiento para actuar en consecuencia.

Arranque de aplicación

El administrador despliega la aplicación sobre un servidor. Todas las incidencias y registros de actividad que se den durante el proceso de arranque quedan almacenados en el archivo de incidencias.

Caso de uso 4	**Arranque de aplicación**
Descripción	El administrador despliega la aplicación sobre un servidor.
Actores	Administrador
Precondiciones	El servidor sobre el que se desplegará la aplicación debe encontrarse correctamente instalado.
Secuencia normal	1. El administrador despliega el archivo con la aplicación en un servidor. 2. Toda la información del arranque de la aplicación, así como cualquier incidencia que pudiera surgir al iniciarse la misma, queda registrada en el archivo de incidencias.

Poscondiciones	La aplicación se encuentra arrancada y todos los *logs* de arranque e incidencias quedan registrados en un archivo.

Tabla 4: Arranque de aplicación

Configuración de la base de conocimiento

El administrador abre el archivo de configuración de la base de datos y modifica, añade o borra la entrada o las entradas correspondientes a los archivos que poseen los temas de conocimiento que desee. Al reiniciar el sistema debe cargarse la base de conocimiento con los cambios que ha introducido el administrador.

Caso de uso 5	**Configuración de la base de conocimiento**
Descripción	La base de conocimiento debe ser configurable.
Actores	Administrador
Precondiciones	Ninguna
Secuencia normal	1. El administrador abre el archivo de configuración de la base de conocimiento. 2. El administrador modifica, borra o añade la entrada o las entradas correspondientes a los archivos de conocimiento que desee. 3. El administrador guarda la configuración y reinicia la aplicación.
Poscondiciones	La nueva base de conocimiento se encuentra activa.

Tabla 5: Configuración de la base de conocimiento

Configuración de logs

El administrador abre el archivo de configuración de *logs* y modifica la ruta de en la que se almacenarán los archivos de registro o el formato de los mismos. Al reiniciar el sistema deben hacerse efectivos los cambios en la configuración.

Caso de uso 6	**Configuración de logs**
Descripción	La ruta y el formato de los *logs* debe ser configurable.
Actores	Administrador

Precondiciones	Ninguna
Secuencia normal	1. El administrador abre el archivo de configuración de *logs*. 2. El administrador modifica la ruta en la que se guardan los archivos de *log* o el formato con el que se almacenan las conversaciones e incidencias en dichos archivos. 3. El administrador guarda la configuración y reinicia la aplicación.
Poscondiciones	La nueva configuración de *logs* se encuentra activa.

Tabla 6: Configuración de logs

Ejecución de pruebas automatizadas

El administrador define una serie de pruebas que deberá pasar la aplicación. Estas pruebas se ejecutan de manera automática y devuelven un resultado con la tasa de acierto en dichas pruebas.

Caso de uso 7	**Ejecución de pruebas automatizadas**
Descripción	El administrador debe poder realizar pruebas automatizadas para comprobar el grado de acierto de la aplicación.
Actores	Administrador
Precondiciones	La aplicación debe encontrarse arrancada.
Secuencia normal	1. El administrador define una serie de pruebas que deberá pasar la aplicación. 2. El administrador ejecuta las pruebas automatizadas sobre una aplicación previamente arrancada. 3. Se genera un informe especificando el grado de acierto de la aplicación así como detalles sobre aquellas pruebas no superadas.
Poscondiciones	Se genera un informe de pruebas.

Tabla 7: Ejecución de pruebas automatizadas

Análisis estadístico

El administrador accede a los *logs* de conversaciones. Analizando las interacciones con el usuario puede detectar posibles problemas o realizar análisis de las preferencias de los usuarios para poder mejorar la base de conocimiento u obtener datos útiles para mejorar las estrategias de marketing.

Caso de uso 8	**Análisis estadístico**
Descripción	El administrador debe ser capaz de acceder a los *logs* de conversaciones para analizarlos.
Actores	Administrador
Precondiciones	Deben haberse producido una cantidad suficiente de conversaciones entre la aplicación y los usuarios como para poder obtener estadísticas fiables.
Secuencia normal	1. El administrador abre el archivo de *logs* en el que se encuentran registradas las conversaciones con los usuarios. 2. Se analizan las preferencias de los usuarios, los temas tratados más a menudo, en qué fechas y a qué horas se producen las diferentes conversaciones. 3. Con esa información se elaboran estrategias de marketing y se mejora la base de conocimiento en aquellos temas que lo requieran.
Poscondiciones	Mejoras en estrategias de marketing y en la base de conocimiento.

Tabla 8: Análisis estadístico

5 Arquitectura

En el presente capítulo se definirá la arquitectura general de la aplicación, los componentes que la forman y las diferentes relaciones que existen entre ellos. Una vez definida la arquitectura general de la aplicación se detallará asimismo la estructura de cada uno de dichos componentes.

Al mismo tiempo que se define la arquitectura de la aplicación y de cada uno de los componentes que la forman, se explicarán las diferentes decisiones de diseño que se han tomado durante el proceso de creación del bot de ayuda.

5.1 Introducción

Para facilitar la implementación del agente inteligente se recurre a la aplicación de patrones de arquitectura [*Bush96*]. Estos patrones proporcionan esquemas de organización de la estructura fundamental de un sistema software, definidos por los diferentes subsistemas que los componen, sus funciones y responsabilidades.

En aplicaciones complejas, en las que se proporcionan al usuario grandes cantidades de información, es deseable separar la interfaz de usuario de la lógica, de tal forma que se consiga la modularidad del sistema, es decir, que sea posible realizar cambios en la presentación sin que esto afecte al manejo de la información y viceversa. Básicamente, el objetivo que se persigue con estas arquitecturas es la evolución y el crecimiento adecuado del sistema de información. El patrón MVC (Modelo vista controlador) soluciona este problema separando el acceso a la base de datos y la lógica de negocio de la presentación de la información y la interacción de usuario mediante la introducción de un componente intermedio, el controlador.

Así pues el patrón MVC consta de modelo, vista y controlador, realizando cada una de estas capas las siguientes funciones:

- **Modelo:**
  - Definir la funcionalidad del sistema y procesar la información, es decir, la lógica de negocio.
  - Acceder a la capa de almacenamiento de datos. Es recomendable que sea independiente del sistema de almacenamiento.
  - Llevar un registro de las vistas y los controladores del sistema.

- **Vista:**
  - Recibir los datos del modelo y presentarlos de manera adecuada para la interacción del usuario. Pueden existir diferentes vistas para un único modelo.

- **Controlador:**
  - Recibir los eventos de entrada.
  - Gestionar los distintos eventos.
  - Responder a los eventos de manera adecuada.

Dadas las características del patrón MVC se puede comprobar que es el más adecuado para la realización del bot de ayuda, ya que no sólo proporciona un grado de modularidad que permite dividir una aplicación compleja en partes más manejables, sino que además cumple las necesidades de tener varias vistas sobre una misma lógica de negocio y permite futuras modificaciones de las diferentes partes sin necesidad de alterar el resto.

5.2 Descripción de la Arquitectura

De la aplicación del patrón de arquitectura MVC al bot de ayuda se obtienen las siguientes capas:

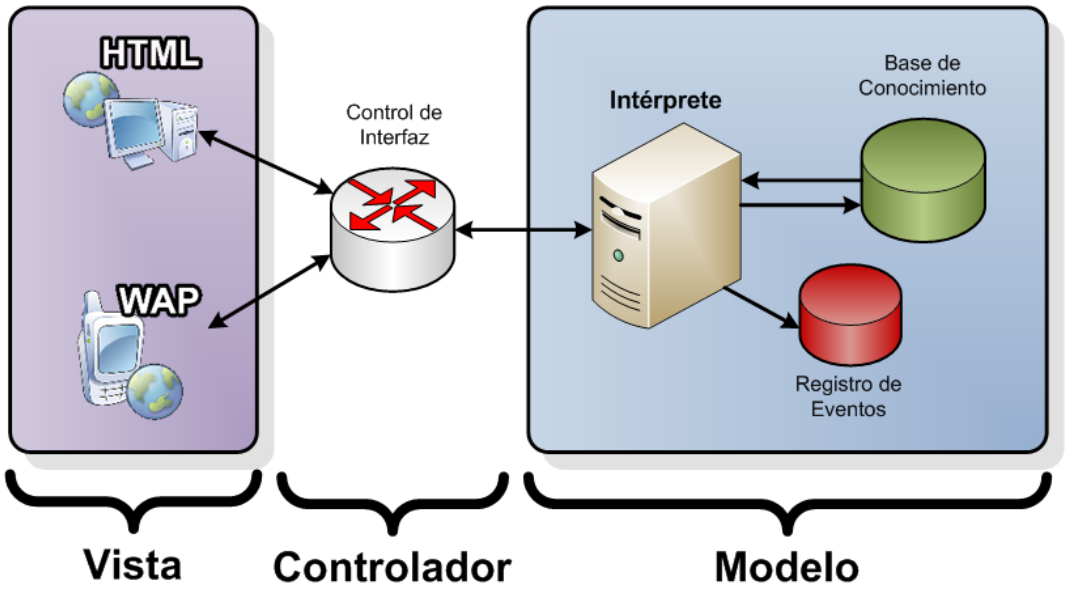

Ilustración 6: Esquema de patrón MVC aplicado al Bot de Ayuda

- **Modelo:** Es el intérprete de AIML, se encarga de iniciar los bots con las bases de conocimiento, procesar estas bases de conocimiento ante una consulta, gestionar las conversaciones de los usuarios y realizar tareas de análisis de las entradas. Además incluye un registro de eventos y de conversaciones.

- **Vista:** Consiste en una interfaz adecuada para cada tipo de dispositivo, desde la cual el usuario pueda acceder al bot de ayuda.

- **Controlador:** Es un *servlet* y un jsp sencillo de control, que gestionan las distintas vistas y conectan las consultas recibidas con el bot correspondiente.

Así pues, siguiendo esta arquitectura la interacción con el sistema se haría de la siguiente manera:

1. El usuario introduce una consulta a un bot a través de una de las interfaces disponibles dependiendo del dispositivo.
2. La consulta pasa al controlador que se encarga de sanearla y adecuarla a las necesidades del modelo.
3. El modelo recibe la petición, la analiza, procesa las bases de conocimiento y devuelve la respuesta al controlador quedando a su vez registrada la interacción en un archivo de *log*.
4. El controlador devuelve la respuesta a la interfaz correspondiente a la consulta.
5. El usuario recibe la respuesta a su consulta a través de la interfaz elegida para la petición.

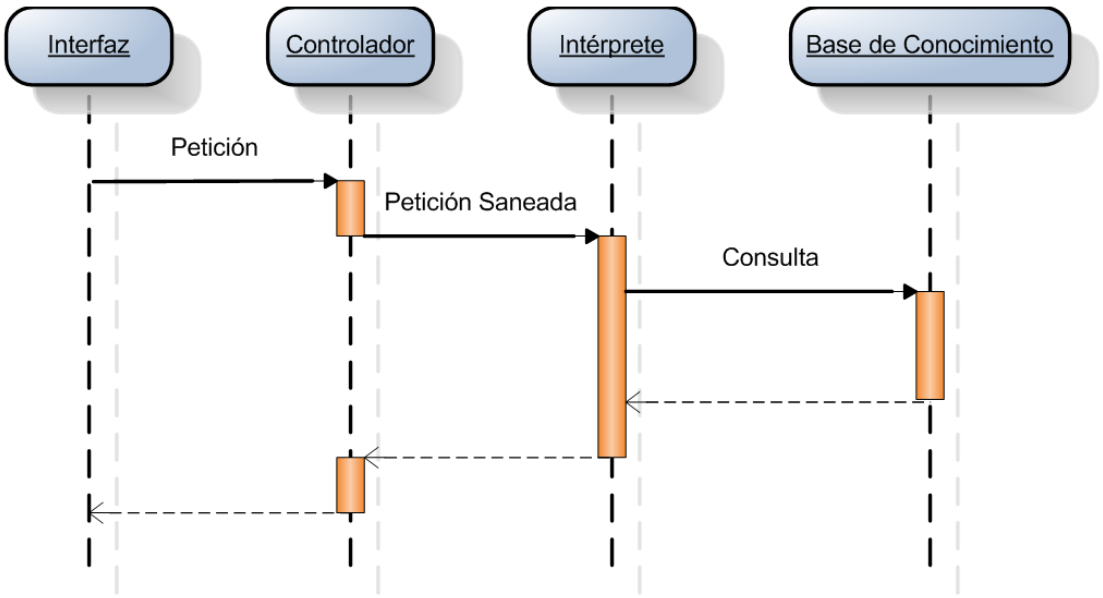

Ilustración 7: Diagrama de Secuencia de la Aplicación

5.3 Modelo

El modelo engloba el núcleo de la lógica de la aplicación, debe recoger la consulta, acceder a la base de conocimiento, procesar la entrada comparándola con los patrones de la base de conocimiento y devolver una respuesta.

Además debe gestionar el acceso múltiple de usuarios, guardar un registro de estado para cada uno de dichos usuarios y almacenar en forma de archivos de *log* las distintas conversaciones e incidencias.

Por lo tanto, viendo las funciones del modelo, se pueden diferenciar dos partes, por una el elemento puramente lógico, es decir, el intérprete de AIML y por otra un elemento que actúa como recurso sin llevar a cabo ninguna función lógica, la base de conocimiento.

5.3.1 Intérprete de AIML

Para el desarrollo del bot de ayuda se optará por la utilización de uno de los intérpretes de AIML existentes a día de hoy en vez de implementar uno desde cero. Como consecuencia de esta decisión la arquitectura del intérprete no podrá ser definida durante esta fase, sino que quedará condicionada por dicha elección.

Por lo tanto se deberá elegir un intérprete que se adapte a los requisitos que se han definido previamente y modificarlo, si fuese necesario, de tal forma que cumpla todas las funcionalidades que se esperan de la aplicación.

Elección del Intérprete de AIML

La primera decisión que se tomó a la hora de elegir el intérprete del bot de ayuda fue que estuviese desarrollado en Java. Esta elección se debe a dos razones fundamentales, por un lado es un lenguaje conocido y fácil de usar, lo cuál simplifica la labor de modificar el código en caso de ser necesario, y por otro lado permite cumplir el requisito no funcional de ser capaz de funcionar en la mayor variedad de plataformas posibles, ya que el lenguaje Java tiene como una de sus principales características ser independiente de la plataforma en la que se ejecute.

Tras esta primera elección se seleccionaron aquellos intérpretes desarrollados en Java que contaban con versiones más recientes y un mejor soporte, quedando dos opciones entre las que elegir, *Chatterbean* [***ChatURL***] y *Program D* [***ProDURL***].

Una vez se analizaron ambos intérpretes se llegó a la conclusión de que el más adecuado para implementar el bot de ayuda era *Program D*, puesto que *Chatterbean* no era conforme a la especificación de AIML en varios puntos fundamentales:

- **Tratamiento de Espacios:** Al utilizar la herramienta de reducción de AIML *Chatterbean* no trata de manera correcta los espacios ya que los elimina. Por ejemplo en el caso de tener la entrada “información sobre tarifas” y aplicar una reducción para eliminar la palabra “sobre” *Chatterbean* devolvería “informacióntarifas” en vez de “información tarifas” como cabría esperar. Este fallo hace que resulte imposible utilizar las reducciones de AIML siendo éstas un elemento básico para un bot conversacional.

- **Asignación de cadenas vacías a Comodines:** *Chatterbean* permite asignar una cadena vacía a un comodín, lo cual hace que se pierdan ciertas funcionalidades del lenguaje AIML haciéndolo menos versátil. Por ejemplo, si tenemos el patrón "Hola *" debería aplicarse únicamente a las entradas cuya primera palabra sea "Hola" pero que incluyan al menos una palabra después, es decir, el comodín debe corresponderse con una cadena de texto no vacía. Este comportamiento permite diferenciar la situación en la que el usuario dice únicamente "Hola", de aquella en la que empieza por "Hola" pero que luego dice algo más. En el primer caso bastaría con devolver el saludo mientras que en el segundo habría que procesar el texto siguiente, ya que podría encadenar una petición.

Así pues, para el desarrollo del proyecto se partirá del intérprete de AIML *Program D* y se modificará en aquellos puntos que no coincidan con los requisitos del bot de ayuda o se añadirán nuevas funcionalidades si fuese necesario.

5.3.2 Base de Conocimiento

La base de conocimiento determina los temas que podrá reconocer el bot de ayuda. En este caso se diferenciarán dos tipos de temas, por una parte los temas de ayuda relativos a los servicios de Orange y por otro a los temas de conversación generales.

Al estructurar los temas de esta manera no sólo se logra una mayor organización si no que se hace una diferenciación de aquellos temas que pueden ser reutilizados para otros bots conversacionales y aquellos otros que sólo tienen interés en el ámbito de un asistente de Orange.

A su vez los temas específicos dentro de esos dos grupos genéricos se modelarán en forma de archivos independientes, de tal manera que cualquier modificación o mejora en la base de conocimiento se pueda realizar de forma modular sin afectar al resto.

Así pues la base de conocimiento constará de los siguientes archivos:

- **Temas Generales:**
  - **auxiliar.aiml**: En este archivo se incluyen las aquellas transformaciones auxiliares que procesan la entrada para facilitar la comprensión por parte del intérprete. Ejemplos de estas transformaciones serían cambiar el número y el género a las palabras que lo permitan o sustituir por una única palabra todos los sinónimos, de tal manera que no haya diferentes palabras que definan un único concepto.
  - **base.aiml:** Este archivo contiene todos los temas relativos al propio bot como pueden ser el nombre, la edad, etc. Además se incluyen respuestas a palabras o expresiones comunes ("gracias", "perdón", "nada", etc.) y funciones auxiliares como el cambio de cara.
  - **conversacion.aiml:** En este archivo se encuentran los temas generales que pueden surgir en una conversación y que no están relacionados con Orange, como pueden ser deportes, cine, etc.
  - **despedidas.aiml:** Archivo que contiene respuestas a las diferentes despedidas por parte del usuario.

- **educacion.aiml:** En este archivo se incluye el reconocimiento de palabras groseras e insultos por parte del usuario.
- **reducciones.aiml:** Gracias a este archivo se procesa la entrada del usuario eliminando aquellas palabras que no son necesarias para comprender la consulta, con lo cual se facilita la tarea de reconocimiento de la entrada por parte del intérprete.
- **saludos.aiml:** En este archivo se encuentran los posibles saludos por parte del usuario y las respuestas adecuadas a cada tipo de saludo.
- **verbos.aiml:** Al igual que en el caso del archivo "auxiliar.aiml" en el que se transformaban las palabras para que todas las que hiciesen referencia a un mismo concepto se convirtiesen en una única palabra reconocible por el intérprete, en este caso lo que se modifican son todos los verbos para eliminar la información relativa al tiempo, persona y número dejando únicamente el infinitivo del verbo.

- **Temas Específicos de Orange:**
  - **auxiliarOrange.aiml:** Este archivo es equivalente a "auxiliar.aiml" pero se encarga de transformaciones que sólo son aplicables en el ámbito de un asistente de Orange.
  - **fijo.aiml:** En este archivo se incluyen todos los temas relacionados con el servicio de telefonía fija de Orange.
  - **general.aiml:** En este archivo se encuentran todos aquellos temas que son compartidos por más de un servicio de Orange. Así pues si un usuario pregunta por uno de dichos temas se le preguntaría a qué servicio en concreto se refiere y se redireccionaría consecuentemente.
  - **internet.aiml:** Archivo que incluye todos aquellos temas relacionados con el servicio de Internet de Orange, ya se trate de línea básica o de ADSL.
  - **movil.aiml:** Este archivo contiene los temas relacionados con el servicio de telefonía móvil.
  - **television.aiml:** En este archivo se encuentran todos aquellos temas acerca del servicio de televisión de Orange.

Así pues con este listado quedan definidos no sólo los temas que va a ser capaz de reconocer el bot de ayuda si no también cómo se estructurarán en la fase de diseño.

5.4 Vista

Como se ha comentado en la descripción del patrón MVC su uso nos permite, no sólo disociar la parte de presentación de la parte lógica, sino que además permite la creación de varias interfaces asociadas a una única lógica de negocio. En el caso de la vista del bot conversacional se dispone de dos tipos de interfaces asociadas al intérprete de AIML, una interfaz Web y otra WAP cada una de las cuales tiene que ser capaz de mostrar la aplicación de manera atractiva al usuario en cada tipo de terminal, tomar la

entrada del usuario, entregársela al controlador, recibir la respuesta de éste y mostrársela al usuario.

Además de la funcionalidad básica de presentación y captura de entrada de usuario, las vistas tienen también una parte activa, ya que se encargan de obtener información relativa a la sesión, mostrando al usuario un mensaje de bienvenida en caso de que acabe de iniciar sesión o realizando su funcionamiento habitual en caso contrario. Es decir, tienen que hacer frente a dos situaciones y actuar en consecuencia en cada uno de los casos:

- Si es la primera interacción del usuario debe mostrar un mensaje de bienvenida que le guíe en el uso de la aplicación.
- Si no es la primera interacción debe tomar la entrada del usuario y enviarla al controlador junto con la información necesaria para que éste último sepa a quién tiene que devolver la respuesta correspondiente.

En el caso de la interfaz Web también debe ser capaz de recopilar información acerca de las capacidades ofrecidas por el navegador que está utilizando el usuario y ofrecer unas posibilidades u otras en cada caso:

- Si el usuario soporta *javascript* la interfaz Web hace uso de animaciones flash y apertura de enlaces de manera automática.
- En caso de no soportar *javascript* las animaciones se sustituyen por imágenes fijas y se da el hipervínculo al usuario para que pueda acceder a la página referenciada.

Hay que tener en cuenta que las interfaces deberán no sólo cumplir las funciones que se especifiquen sino que además deberán ser conformes a las especificaciones de las tecnologías utilizadas.

Una vez se han definido las funcionalidades básicas de cada tipo de interfaz se pasará a definir un esquema básico de implementación para cada una de ellas.

5.4.1 Interfaz Web

La interfaz Web constará de tres partes diferenciadas, cada una de las cuales tiene una función y unas características específicas:

- **Página de Inicio:** Es la página a través de la cual se accede a la aplicación, consta de un único enlace que abre un *popup* con la aplicación. Además de su función de puerta de entrada al bot de ayuda, se utilizará la ventana en la que se abra como soporte donde abrir cualquier URL que se le ofrezca al usuario.

  Debido a la sencilla función de la página de inicio se puede implementar mediante una página estática HTML, teniendo como únicas características importantes que sea atractiva para el usuario, que abra la

aplicación en un *popup* y que la ventana en la que se abre tenga algún identificador al que pueda hacer referencia la aplicación a la hora de abrir cualquier URL.

- **Página del Bot de Ayuda:** Es el núcleo de la interfaz Web, no sólo es la encargada de tomar la interacción del usuario y devolverle una respuesta, si no que se encargará de la interacción con el controlador por medio de parámetros de sesión y realizará ciertas labores auxiliares para mejorar la experiencia del usuario.

  La página del bot de ayuda deberá mostrar un mensaje de bienvenida al usuario durante la primera interacción, recoger la entrada del usuario mediante un formulario y mandársela al controlador. Posteriormente recibiría la información a través de los parámetros de sesión y mostraría tanto el mensaje de respuesta como cualquier información adicional asociada como, pudiera ser el estado de ánimo que refleje la respuesta.

  Aparte de las funciones básicas debe de tener un aspecto agradable al usuario, permitir variaciones en la presentación dependiendo del equipo usado por el mismo y permitir el acceso a una página en la que se explique el funcionamiento de la aplicación.

  Dado que esta página debe llevar a cabo ciertas operaciones, como acceso a variables de sesión, cambios en la presentación o variaciones dinámicas del contenido, se hará uso de una página JSP y ciertos elementos de *javascript*.

- **Página de Ayuda al Usuario:** Esta página estará accesible en todo momento a través de la página del bot de ayuda. En ella se explicará al usuario en qué consiste el bot y cuál es su forma de uso.

  Al igual que en el caso de la página de inicio, la función de esta página es muy sencilla y no se requiere ningún elemento dinámico, únicamente es necesario que siga la misma estética del resto de la aplicación y que muestre de forma correcta el texto de ayuda, por lo tanto también se modelará mediante una página *HTML*.

5.4.2 Interfaz WAP

La interfaz WAP presenta una estructura parecida a la de la interfaz Web, pero prescinde la página de inicio. El usuario entrará directamente a la página del bot de ayuda, ya que no es necesario presentarla en forma de *popup*. Por lo tanto, la interfaz WAP consistirá en dos partes:

- **Página del Bot de Ayuda:** Al igual que en el caso de la interfaz Web, esta página es la parte principal de la interfaz WAP, siendo la encargada de llevar a cabo todas las funciones básicas.

Estas funciones son, mostrar un mensaje de bienvenida si es la primera interacción con el usuario, recoger la entrada de éste y devolverle la respuesta que dé el intérprete. Además se encargará de mostrar al usuario la información adicional que se pueda entregar junto con la respuesta de texto, como pueden ser URL's y estados de ánimo y de permitir el acceso a la página de ayuda.

Al necesitarse variar el contenido de la página de manera dinámica, al igual que en el caso de la interfaz Web, se utilizará una página JSP.

- **Página de Ayuda al Usuario:** En la página de ayuda se da, en formato de texto, la información sobre el bot de ayuda y su uso. No necesita ningún tipo de funcionalidad aparte de mostrar texto en pantalla, exceptuando la posibilidad de volver a la página del asistente.

  Dada la sencillez de la página, no se requiere ningún elemento dinámico y, por tanto, se implementará como una página WML.

5.5 Controlador

El control de la aplicación se divide en dos partes, en primer lugar existe un JSP que se encarga del control de las vistas y, en segundo lugar, un *servlet* que se encarga de la gestión de entrada de usuario, redirigiendo las consultas hacia el bot correspondiente y recogiendo la respuesta de dicho bot. Además el *servlet* se encarga de ciertas labores adicionales de seguridad y control de sesión.

5.5.1 Control de Vistas

El control de vistas actúa como un distribuidor, se encarga de tomar la entrada del usuario y redirigirla al *servlet*, para, una vez que recibe la respuesta, devolvérsela a la interfaz desde la que se solicitó.

Para llevar a cabo el control de vistas, se hará uso de una variable que servirá para identificar la interfaz que recibió la petición, de esta forma, será sencillo añadir interfaces adicionales con sólo añadir el código correspondiente en el JSP.

Se definirá la interfaz Web como interfaz por defecto, así que, si el valor de la variable de control no se corresponde con una interfaz de las definidas en el JSP, se dará por supuesto que la petición se realizó desde la interfaz Web.

Como función adicional el control de vistas se encargará de añadir la variable "*timestamp"* a la URL de la interfaz, para evitar que se produzcan problemas de caché y se devuelvan respuestas previas almacenadas por el navegador o un *proxy* intermedio en vez de dar la nueva información.

5.5.2 Gestor de Entrada de Usuario

En la gestión de entrada de usuario es donde se encuentra la mayor complejidad de la capa de control de la aplicación, ya que es donde se realiza la conexión con el intérprete. En esta conexión el *servlet* de control debe identificar el bot al que se le quiere enviar la petición y el usuario que la envía. Con estos datos realizará la consulta

al intérprete y obtendrá la correspondiente respuesta e información adicional, como estado de ánimo, URL y la variable auxiliar "*test*". Toda esa información se guardará en variables de sesión para que sea fácilmente accesible por la capa de vistas.

Aparte de enviar la información que recibe del intérprete a la interfaz correspondiente el gestor añade cierta lógica adicional para simplificar la labor de las interfaces y proteger al intérprete de ciertas entradas. Las funciones auxiliares que lleva a cabo el gestor de entrada de usuario son:

- **Interfaz Genérica:** Aparte de actuar como enlace con las vistas, el gestor actúa como una interfaz genérica, devolviendo el resultado de la petición como una página HTML plana con un formato determinado. De esta forma se podrán conectar a la aplicación agentes externos sin necesidad de interactuar con las vistas.
- **Creación de Variables Auxiliares:** Una vez reciba la información por parte del intérprete, el gestor creará las variables auxiliares que necesiten las interfaces, ya sea modificando la información recibida, bien creando las variables de cero.
- **Reinicio de Variables:** Algunas de las variables que envía el intérprete han de ser reiniciadas para el correcto funcionamiento de la aplicación. El gestor deberá enviar la orden de reinicio al intérprete una vez haya obtenido los valores de dichas variables.
- **Saneamiento de Entrada de Usuario:** Los usuarios podrían aprovechar la introducción de texto de la aplicación para enviar código malicioso a la aplicación. El gestor deberá analizar la entrada y eliminar aquellos símbolos que presenten algún tipo de peligro.
- **Comprobación de Entrada Vacía:** Para evitar sobrecargar al intérprete de manera innecesaria, se comprobará si la entrada del usuario está vacía, en cuyo caso se devolverá el control a la interfaz desde la que se realizó la petición, volviendo al punto en el que ésta se realizó.

6 Diseño

Una vez se ha definido la arquitectura del bot de ayuda, se puede pasar a la fase de desarrollo de la aplicación.

En este capítulo se detallará la implementación concreta de todos los elementos que componen la aplicación, esto es, intérprete, base de conocimiento, interfaz y controlador.

6.1 Introducción

En los capítulos anteriores se ha definido en qué consistía el proyecto y como se pretendía llevar a cabo. En este capítulo se describirá el proceso de llevar toda esa planificación a la práctica.

Ya que en el capítulo anterior se definió la arquitectura del proyecto, en este tema se seguirá la misma estructura, detallando, en primer lugar, la implementación del modelo, compuesto por intérprete y base de conocimiento, en segundo lugar, la vista, tanto la interfaz Web como la WAP, y, por último, el controlador.

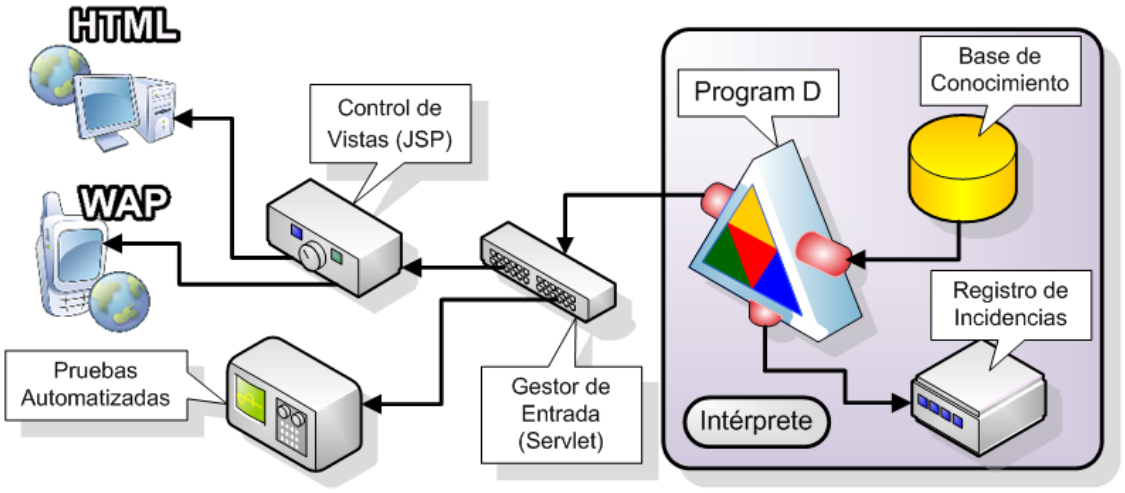

Ilustración 8: Esquema del Diseño del Bot de Ayuda

En todos los apartados se hará hincapié en las diferentes dificultades de implementación y las decisiones de diseño que se han tomado a lo largo del desarrollo. Asimismo se detallará el código resultante en aquellos casos en los que se considere necesario para la correcta comprensión de la implementación.

6.2 Diseño del Intérprete de AIML

Como se ha explicado en el tema de arquitectura de la aplicación, se va a utilizar *Program D* como intérprete de AIML del bot de ayuda, añadiéndose únicamente aquellas funcionalidades que se requieran y no proporcione dicho intérprete, o modificando aquellas que no se adecuen a las necesidades del proyecto.

Así pues, en este apartado se comenzará haciendo un análisis de la implementación de *Program D* para, posteriormente, pasar a detallar las modificaciones que se han realizado sobre el código original.

6.2.1 Características

Aparte de las funcionalidades básicas propias de un intérprete de AIML, como son el análisis de los archivos que contienen la base de conocimiento y el procesado de la entrada de usuario, *Program D* cuenta con una serie de funcionalidades adicionales:

- **Configuración mediante archivos XML:** La configuración de la aplicación se realiza por medio de archivos XML en los que se definen los diferentes parámetros. Para el análisis de estos archivos de configuración se reutiliza el módulo encargado de analizar la base de conocimientos, siendo un ejemplo claro de la utilidad de la modularidad del diseño del intérprete.
- **Multiplexación de Usuarios:** *Program D* permite atender a múltiples usuarios de forma simultánea. Esta funcionalidad se lleva a cabo de manera transparente para el usuario, guardando la información necesaria a lo largo de toda la sesión. Además, se puede usar ese mismo mecanismo no sólo para almacenar la información de sesión, si no también información adicional sobre el propio usuario. Esta característica abre una serie de posibilidades muy interesantes a la hora de hacer más realistas las conversaciones, ya que el intérprete podría guardar ciertos datos acerca del usuario, como puede ser el nombre o la edad y utilizarlos posteriormente para dotar al bot de unas características más humanas.
- **Soporte para Múltiples Bots:** El intérprete da la posibilidad de configurar diferentes bots sobre la misma aplicación, cada uno de ellos con una base de conocimiento asociada. Esta característica resulta especialmente útil en caso de querer ampliar la aplicación para dar soporte a varios temas diferentes o incorporar diferentes personalidades para los bots de ayuda, puesto que permite tenerlos todos desplegados sobre una única aplicación.
- **Preprocesado de la Entrada de Usuario:** Antes del procesado de la entrada de usuario para devolver una respuesta adecuada, el intérprete permite realizar una serie de transformaciones previas sobre dicha entrada. Esta característica permite no sólo añadir un mecanismo de protección a la aplicación mediante la eliminación de código potencialmente peligroso, si no que dota de una mayor versatilidad al propio intérprete, al dar la posibilidad de modificar la entrada del usuario para, posteriormente, procesarla más fácilmente.

- **Pruebas Automatizadas:** *Program D* dispone de un sencillo mecanismo de pruebas automatizadas que permiten comprobar el correcto funcionamiento de la aplicación de manera rápida.
- **Registro de Conversaciones e Incidencias:** El mecanismo de registro de eventos de la aplicación se realiza mediante la librería *Log4J*. Esta librería es ampliamente configurable, tanto a nivel de grado de detalle de los eventos como a nivel de almacenamiento de los registros, ya permite mostrar la salida por consola o guardarla en un archivo de texto o incluso en una base de datos. El caso del almacenamiento en base de datos es especialmente interesante para el registro de conversaciones, dado que permite procesar la información de manera ordenada, pudiéndose estudiar de manera sencilla para mejorar el conocimiento del bot.
- **Interfaces y Listeners adicionales:** Se puede interactuar con el intérprete, bien de manera local a través de consola, bien de manera remota por medio de Web o mensajería instantánea. Para permitir el uso de diferentes interfaces sobre una misma instancia del intérprete, *Program D* proporciona una interfaz de aplicación común para todos esos accesos. Además, en el caso del acceso a través de un programa de mensajería instantánea, la aplicación incluye una serie de *listeners* que permiten la conexión con algunos de los sistemas de mensajería más usados a día de hoy.

6.2.2 Arquitectura del Intérprete

Program D está estructurado en gran cantidad de módulos, siguiendo el patrón MVC. Cada módulo es suficientemente independiente como para poder utilizarse en diferentes tareas dentro de la misma aplicación o en otras aplicaciones y ser modificado sin incurrir en la necesidad de llevar a cabo cambios en las demás clases. Es por tanto, un diseño muy modular que permite realizar modificaciones y añadir nuevas funcionalidades sin que los cambios afecten a más de uno o dos paquetes del proyecto.

La aplicación tiene un diseño centralizado en el que existe una clase principal, *Core*, en torno a la cual se añaden todas las funcionalidades de presentación, multiplexación de usuarios, trazas y gestión de recursos necesarios.

El diagrama UML siguiente muestra las clases más importantes del modelo de la arquitectura:

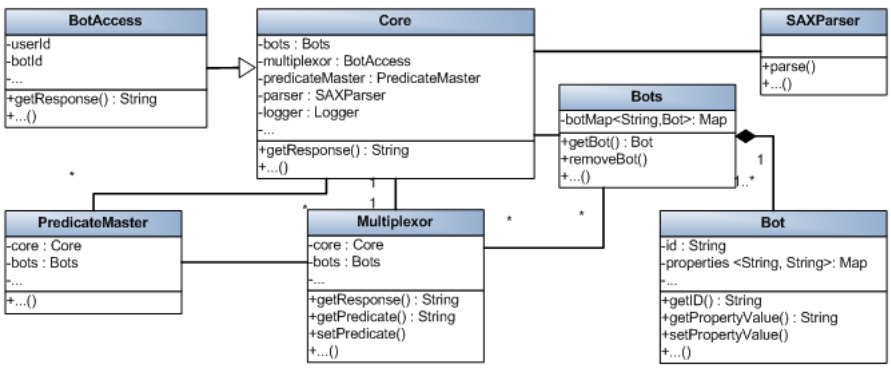

Ilustración 9: Diagrama UML del intérprete

6.2.3 Diseño

Cómo puede verse en el diagrama de clases, *Program D* gira en torno a la clase *Core*, que se encarga de gestionar los diferentes módulos que componen el intérprete y que además contiene el método *getResponse*, mediante el cual un usuario puede solicitar una respuesta a una entrada determinada.

Sin embargo no se permite llamar a los métodos de *Core* de manera directa, sino que se hace uso de *BotAccess,* que es la clase que publica la forma en la que se debe acceder al intérprete. Por lo tanto, *BotAccess* es el punto de acceso a *Program D* sirviendo como interfaz para cualquier usuario o aplicación que quiera hacer uso del intérprete.

La clase *Core* se encarga de la gestión de la aplicación, pero necesita ser capaz de acceder a la base de conocimiento para poder dar una respuesta adecuada a la entrada del usuario. Esta base de conocimiento se encuentra almacenada en archivos AIML, siendo, por lo tanto, necesario un módulo que sea capaz de interpretar dichos archivos. Esta función de análisis de los archivos AIML la lleva a cabo la clase *SAXParser*.

Estas tres clases forman el núcleo de la aplicación, ya que cubren las necesidades básicas de un intérprete. Se tiene una clase central que se encarga de gestionar la aplicación, una interfaz con la cual interactuar y un enlace con la base de conocimiento de la cual se extraerán las respuestas. Sin embargo, dado el ámbito del intérprete en el que múltiples usuarios podrían acceder de manera simultánea, este núcleo tiene una funcionalidad limitada, debido a que sólo es capaz de seguir una única conversación.

Para lograr la gestión de múltiples conversaciones de manera simultánea, *Program D* hace uso de un módulo adicional, el cual está compuesto por las clases *PredicateMaster* y Multiplexor. La primera se encarga de guardar información sobre una conversación determinada, tanto en lo relativo al tema tratado en ese momento o a la última respuesta del usuario, como a datos que se puedan haber obtenido del propio usuario (Nombre, edad, etc.). Por su parte la clase se Multiplexor se encarga de gestionar toda esa información, asociándola con la conversación a la que pertenece y, por lo tanto, permitiendo la recuperación posterior de dicha información de manera correcta. Esta multiplexación de usuarios se realiza sin necesidad de manejar hebras, ya que se realiza añadiendo, a cada atributo, un identificador de la conversación de la cuál se ha extraído.

Program D además proporciona dos propiedades adicionales para dotar de una mayor versatilidad al intérprete de AIML, una de ellas es la posibilidad de gestionar varios bots de manera simultánea con diferentes bases de conocimiento y la otra es la adición de diferentes interfaces diseñadas específicamente para dar soporte a diferentes servicios Web y de mensajería instantánea.

Para poder usar varios bots diferentes sobre el mismo intérprete *Program D* hace uso de las clases Bot y Bots. Por cada bot diferente se crea un objeto de la clase Bot que define qué archivos de la base de conocimiento utilizará (Varios bots pueden compartir un mismo archivo de la base de conocimiento) y cuáles son sus propiedades, como por ejemplo su nombre. Todos estos objetos son gestionados por la clase Bots y, junto a la clase Multiplexor, permiten asociar una conversación y la información asociada a la misma a un bot u otro.

En el caso de las diferentes interfaces se hace uso de las clases *TalkToBotServlet* y *Listener*. La primera se usa como punto de entrada para cualquier aplicación Web, mientras que la segunda es una interfaz que debe ser implementada por los distintos *listeners* de protocolos de mensajería instantánea.

6.2.4 Modificaciones

Como ha podido verse en el apartado de diseño, *Program D* permite el seguimiento de múltiples conversaciones de manera simultánea, gracias al almacenamiento de diversas propiedades de cada una de las conversaciones. Este proceso se lleva a cabo creando un objeto predicado, el cual actúa como soporte para dicha información. Este objeto debe mantener la información al menos durante el tiempo que dura la conversación, por lo que *Program D* dispone de dos alternativas de almacenamiento, la primera consiste en almacenar dichos datos en forma de archivos dentro de un directorio del sistema y la segunda consiste en almacenarlos en una base de datos.

El uso de una base de datos entra en conflicto con el requisito no funcional de la compatibilidad, ya que se pretende que el bot de ayuda sea independiente de los recursos del sistema y, por lo tanto, no debería ligarse a un elemento externo para su correcto funcionamiento.

Por su parte, el volcado de datos a disco presenta problemas de escalabilidad, que es otro de los requisitos del proyecto, puesto que el espacio de disco ocupado crecería de manera lineal con el número de usuarios. Esto podría solucionarse con el borrado periódico de los datos almacenados, pero podría provocar inconsistencias en las conversaciones que se estuviesen manteniendo durante el proceso de borrado.

Debido a que ninguna de las alternativas que provee *Program D* resultaba satisfactoria para el cumplimiento de los requisitos impuestos, se decidió implementar una utilidad adicional de almacenamiento de propiedades de las conversaciones.

Dados los requisitos de la aplicación y el ámbito de la misma, la solución más eficaz es la gestión de sesiones. Cada conversación está asociada a una sesión, por lo tanto, si la sesión expira, la información sobre la conversación ya no es necesaria y puede borrarse. Esto supone una ventaja sobre las alternativas que propone *Program D* ya que no es necesario realizar borrados periódicos para mantener la escalabilidad de la aplicación, con las posibles inconsistencias que pudieran provocar, sino que simplemente debe ponerse un disparador que cada vez que detecte que una sesión ha expirado, borre el predicado asociado a la misma. Esto no sólo permite abstraerse del borrado periódico de la información sino que al resultar en una menor cantidad de predicados almacenados de manera simultánea (únicamente se almacenarán tantos predicados como conversaciones hay activas), hace posible guardar esos objetos de manera local, sin necesidad de hacer uso de escrituras en disco o en base de datos, independizando aun más la aplicación del sistema en el que se vaya a usar.

Para la gestión de sesiones se hace uso de un *listener*, que es capaz de disparar una serie de acciones, tanto en el momento de creación de una sesión como en el de destrucción de la misma. El código que lleva a cabo estas dos funciones es el siguiente:

```
public void sessionCreated(HttpSessionEvent se) {
   HttpSession session = se.getSession();
   // Establece el tiempo en segundos permitido entre dos peticiones
   // de una sesión antes de invalidar la misma.
   session.setMaxInactiveInterval(300);
}
```

```
public void sessionDestroyed(HttpSessionEvent se) {
   HttpSession session = se.getSession();
   // Obtiene los atributos de session "Core", "userId" y "botId".
   Core coreToUse = (Core)
        session.getServletContext().getAttribute("core");
   if (session.getId()!=null) {userId = session.getId();}
   if (session.getAttribute("bot")!=null) {botId = (String)
        session.getAttribute("bot");}

    // Elimina el predicado del usuario cuya session ha expirado.
   if (coreToUse != null){
        Map predicateCache =
            coreToUse.getBot(botId).getPredicateCache();
        if (predicateCache.containsKey(userId)){
            predicateCache.remove(userId);
        }
   }
}
```

Por lo tanto, en el caso de que se cree una nueva sesión, simplemente se define el tiempo durante el que dicha conversación se considerará activa aunque no reciba peticiones, mientras que si una sesión expira, se obtienen los identificadores de dicha sesión, y en caso de que haya un predicado asociado a la misma, este se borra.

6.3 Diseño de la Base de Conocimiento

La base de conocimiento está estructurada en varios archivos AIML que pueden contener diferentes tipos de plantillas. Según su función se pueden diferenciar dos grupos de plantillas, por una parte están aquellas que contienen las palabras y las construcciones que es capaz de reconocer el bot de ayuda y, por otra, aquellas que se encargan de procesar la entrada del usuario para conseguir una respuesta coherente.

6.3.1 Temas de Conversación

Dentro de las plantillas de temas de conversación se distinguen dos situaciones que requieren plantillas diferentes. En la primera, la entrada del usuario puede ser aplicable a varios ámbitos, con lo cual será necesario concretar más la entrada para poder dar una respuesta adecuada, mientras que en la segunda la entrada está perfectamente definida, con lo que se le podrá dar una respuesta de manera directa.

Tema General

La plantilla para temas generales se aplica en aquellos casos en los que si bien la entrada se reconoce, ésta puede estar haciendo referencia a varios temas. Por ejemplo, si un usuario pide información sobre el “buzón de voz”, puede referirse al del teléfono móvil o al del fijo, así que habrá que pedirle que concrete el ámbito de la entrada.

Por lo tanto, una vez que se reconoce una entrada como aplicable a distintos temas debe preguntarse al usuario cuál de dichos temas es el más adecuado. Una vez se concreta el tema al que se hace referencia, se pasa a añadir esa información a la entrada inicial y se procesa de nuevo para obtener la respuesta correspondiente.

Un ejemplo de plantilla para temas generales podría ser la siguiente:

```
<!-- Selección de entradas compuestas únicamente por el patrón -->
<category>
      <pattern>BUZON VOZ</pattern>
      <template>
            <think><set name="topic">PREGUNTA BUZON</set></think>
            ¿De qué tipo de buzón de voz necesita información de fijo
            o de móvil?
      </template>
</category>

<!-- Selección de entradas que contienen el patrón. El asterisco
equivale a una o más palabras, correspondiéndose cada una de las
categorías con una de las posibles posiciones dentro de la frase -->
<category>
      <pattern>* BUZON VOZ *</pattern>
      <template><srai>BUZON VOZ</srai></template>
</category>
<category>
      <pattern>BUZON VOZ *</pattern>
      <template><srai>BUZON VOZ</srai></template>
</category>
<category>
      <pattern>* BUZON VOZ</pattern>
      <template><srai>BUZON VOZ</srai></template>
</category>

<topic name="PREGUNTA BUZON">
<category>
      <pattern>FIJO</pattern>
      <template>
            <think><set name="topic">FIJO</set></think>
            <srai>BUZON VOZ FIJO</srai>
      </template>
</category>

<!-- Selección de entradas que contienen el patrón “fijo”-->
[…]

<category>
      <pattern>MOVIL</pattern>
      <template>
            <think><set name="topic">MOVIL</set></think>
            <srai>BUZON VOZ MOVIL</srai>
      </template>
</category>

<!-- Selección de entradas que contienen el patrón “móvil”-->
[…]

<category>
   <pattern>*</pattern>
   <template>
      <think><set name="topic">*</set></think>
      Lo siento mucho pero me temo que o no he entendido
      correctamente el servicio para el que solicita información
      acerca del buzón de voz o el servicio solicitado no dispone de
      buzón de voz. Si quiere puede probar de nuevo y a lo mejor
      logro entenderle si formula su pregunta de otra manera.
   </template>
</category>
</topic>
```

Tema concreto

En este caso el tema de la entrada del usuario queda perfectamente definido, así que únicamente habrá que devolver una respuesta adecuada y guardar los valores de las variables auxiliares correspondientes a dicha respuesta.

```
<!-- Selección de entradas compuestas únicamente por “móvil”-->
<category>
        <pattern>MOVIL</pattern>
        <template>
            <think>
                  <set name="topic">MOVIL</set>
                  <set name="url">http://ayuda.orange.es/movil/</set>
            </think>
            Acabo de abrir la página de ayuda de telefonía móvil de
            Orange. Puede preguntarme por el tema que desee en
            concreto o bien buscar en la página de ayuda el tema en
            el que está interesado.
        </template>
      </category>

<!-- Selección de entradas que contienen el patrón “móvil”-->
<category>
      <pattern>* MOVIL *</pattern>
      <template><srai>MOVIL</srai></template>
</category>

<category>
      <pattern>MOVIL *</pattern>
      <template><srai>MOVIL</srai></template>
</category>

<category>
      <pattern>* MOVIL</pattern>
      <template><srai>MOVIL</srai></template>
</category>
```

6.3.2 Transformaciones Auxiliares

Las plantillas de transformaciones auxiliares no dan ninguna respuesta al usuario, si no que procesan la entrada para que la aplicación de las plantillas de temas de conversación se realice de una manera más eficiente.

Reducciones

Las plantillas de reducción se encargan de eliminar aquellas palabras que no son necesarias para la correcta comprensión de la entrada, de manera que la frase quede con el menor número de palabras posible, manteniendo intacto su significado.

Para estas plantillas se hace uso de los comodines de AIML, de tal forma que se procesen todas las entradas que tengan la palabra o palabras que se quieran eliminar. Una vez que se activa la plantilla, lo que hace es volver a procesar toda la entrada, menos las palabras que han encajado con el patrón.

```
<!-- Reducción de entradas que contienen el patrón “por favor”-->
<category>
      <pattern>_ POR FAVOR</pattern>
      <template><srai><star/></srai></template>
</category>

<category>
      <pattern>POR FAVOR _</pattern>
      <template><srai><star/></srai></template>
</category>

<category>
      <pattern>_ POR FAVOR _</pattern>
      <template>
            <srai><star index="1"/> <star index="2"/></srai>
      </template>
</category>
```

Simplificaciones

Al igual que las plantillas de reducción se encargan de eliminar aquellas palabras que no son necesarias para comprender el significado de la entrada, las plantillas de simplificación se encargan de la misma labor, pero a nivel de palabra, eliminando las partes que no afectan al significado.

En el caso de los verbos se reducirá cualquier forma verbal al infinitivo del verbo, y en el resto de palabras, siempre que sea aplicable, se pasará a masculino y singular.

Para la realización de estas plantillas también se hace uso de comodines, poniendo como patrón todas aquellas entradas que contienen la palabra que se quiere simplificar y devolviendo una entrada que tiene el mismo contenido que la original, pero sustituyendo dicha palabra por la simplificación correspondiente.

```
<!-- Reducción de entradas compuestas únicamente por “básicos”-->
<category>
      <pattern>BASICOS</pattern>
      <template><srai>BASICO</srai></template>
</category>

<!-- Reducción de entradas que contienen el patrón “básicos”-->
<category>
      <pattern>BASICOS _</pattern>
      <template><srai>BASICO <star/></srai></template>
</category>

<category>
      <pattern>_ BASICOS</pattern>
      <template><srai><star/> BASICO</srai></template>
</category>
```

```
<category>
      <pattern>_ BASICOS _</pattern>
      <template>
            <srai><star index="1"/> BASICO <star index="2"/></srai>
      </template>
</category>


<!-- Reducción de entradas compuestas únicamente por “básica”-->
<category>
      <pattern>BASICA</pattern>
      <template><srai>BASICO</srai></template>
</category>

<!-- Reducción de entradas que contienen el patrón “básica”-->
[…]

<!-- Reducción de entradas compuestas únicamente por “básicas”-->
<category><pattern>BASICAS</pattern>
      <template><srai>BASICO</srai></template></category>
<category>

<!-- Reducción de entradas que contienen el patrón “básicas”-->
[…]
```

Sinónimos

Las plantillas de sinónimos tienen como función el sustituir todas las palabras que sean sinónimos por uno sólo de dichos sinónimos, de tal manera que el significado se mantenga inalterado pero el número de patrones a tener en cuenta en las plantillas de temas de conversación se reduzca.

La estructura de la plantilla es similar a la del caso de las simplificaciones, salvo que en vez de sustituir una palabra por su forma simplificada se sustituye por el sinónimo que se haya decidido utilizar.

```
<category>
   <pattern>SMS</pattern>
   <template>
      <srai>MENSAJE CORTO</srai>
   </template>
</category>

<category>
   <pattern>SMS _</pattern>
   <template>
      <srai>MENSAJE CORTO <star/></srai>
   </template>
</category>
```

```
<category>
   <pattern>_ SMS</pattern>
   <template>
      <srai><star/> MENSAJE CORTO</srai>
   </template>
</category>

<category>
   <pattern>_ SMS _</pattern>
   <template>
      <srai><star index="1"/> MENSAJE CORTO <star index="2"/></srai>
   </template>
</category>
```

Funciones

Estas plantillas no transforman la entrada de usuario, si no que le añaden algún tipo de información. Un ejemplo claro es el cambio de la variable que define el estado de ánimo dependiendo de la entrada del usuario.

Una vez que se va a dar una respuesta a una entrada de usuario se puede llamar a una de estas funciones (Normalmente dentro de una etiqueta <*think*> para que la llamada sea transparente al usuario), de tal forma que aparte de la respuesta de texto se pueda llevar a cabo alguna acción adicional.

En el siguiente ejemplo puede verse como si la entrada del usuario es el nombre del bot, este responderá "¿Sí?" pero además invocará a la función que fija el estado de ánimo como alegre, pudiendo obtener uno de los cuatro estados asociados de manera aleatoria.

```
<category>
   <pattern>PON CARA ALEGRE</pattern>
   <template>
      <think>
         <random>
            <li><set name="state">acierto1</set></li>
            <li><set name="state">acierto2</set></li>
            <li><set name="state">acierto3</set></li>
            <li><set name="state">acierto4</set></li>
         </random>
      </think>
      ¡Estoy contenta!
   </template>
</category>
```

```
<category>
   <pattern><bot name="name"/></pattern>
   <template>
      <think>
         <set name="test">General Conversacion Nombre</set>
         <set name="topic">*</set>
         <srai>PON CARA ALEGRE</srai>
      </think>
      ¿Sí?
   </template>
</category>
```

6.4 Diseño de la Interfaz de Usuario

Como se ha explicado en los temas anteriores, el proyecto debe tener dos interfaces diferenciadas, por una parte una interfaz Web que puedan utilizar todos aquellos dispositivos que posean un navegador Web y, por otra, una interfaz WAP para dispositivos móviles que no posean dicha capacidad.

Cada una de estas interfaces es totalmente independiente de la otra y, aunque tienen la misma función, tomar las interacciones del usuario y presentar en pantalla la respuesta del bot, en cada uno de los casos se debe realizar de manera óptima para el dispositivo en el cual se va a utilizar.

6.4.1 Diseño de la Interfaz Web

En el tema anterior se han definido los diferentes componentes de la interfaz Web, así como las diferentes funciones que debían llevar a cabo cada uno de ellos. En este apartado se dará la implementación concreta que permitirá a cada componente realizar dichas funciones.

Para una mayor modularidad se optará por separar la parte lógica de la interfaz de la apariencia de la misma, haciendo uso de las hojas de estilo en cascada (CSS). Esto permitirá además abordar el diseño de cada uno de los componentes de la interfaz por separado para, posteriormente, dotar al conjunto de una apariencia sólida y homogénea.

Por lo tanto se pasará a explicar el diseño concreto de cada uno de los componentes, es decir, página de inicio, página del bot de ayuda y página de ayuda y, después, se definirá el aspecto de los mismos mediante CSS.

Página de Inicio

La funcionalidad básica de esta página es servir como punto de entrada a la aplicación, para ello se utilizará una imagen que servirá de link y que al pulsarla abrirá un *popup* con la aplicación, para ello se utiliza el siguiente código:

```
<h1>Asistencia técnica: Pregúntale a Erika</h1>

<div>
      <a href = "./pages/Erika.jsp" onclick = "return
      openPopup('./pages/Erika.jsp')">
            <img src = "./images/Erika.gif" alt = "Bot de Asistencia
            técnica: Erika"/>
      </a>
</div>

<p>* Pulse sobre la imagen para acceder al asistente.</p>
```

En el código se puede ver como, al pulsar en el enlace, se llama a una función *javascript* llamada *openPopup* cuya función es abrir el bot de ayuda en una ventana emergente de un tamaño variable según la resolución de pantalla del usuario:

```
// Abre la ventana de PopUp con la forma adecuada dependiendo de la
resolución del usuario
function openPopup(url){
      if (screen.width==800){
         window.open(url,'Erika','width=170, height=510, location=no,
             left=0, top=0');
      }
      else {
         window.open(url,'Erika','width=220, height=510, location=no,
             left=0');
      }
      return false;
}
```

Este código sin embargo presenta un problema en caso de que el usuario no tenga activado *javascript*, ya que no funcionaría el enlace. Para solucionarlo se añade un código adicional que sólo se mostrará en caso de que se de esa situación, avisando al usuario y mostrándole un enlace alternativo:

```
<p>
   * Pulse sobre la imagen para acceder al asistente.

    <noscript>
        <br />
        (Tiene javascript desactivado, si no se abre el asistente
        pulse en este
        <a target = "Erika" href = "./pages/Erika.jsp">enlace</a>)
    </noscript>
</p>
```

De esta forma si el usuario tiene *javascript* activado en su navegador podrá pulsar en la imagen para acceder a la aplicación en forma de *popup* y en caso contrario se le mostrará un mensaje indicándole que no lo tiene activado y se le dará un enlace alternativo para poder abrir la aplicación de manera normal. Ambas posibilidades se pueden ver en la siguiente imagen;

Ilustración 10: Página de inicio (Sin CSS) visualizada con y sin Javascript

Página del Bot de Ayuda

La tarea principal de la página del bot de ayuda es tomar la entrada del usuario y pasársela al controlador para posteriormente presentar por pantalla la respuesta del intérprete. Además de dicha respuesta se mostrará la entrada que la produjo para lograr de esta manera una mayor realimentación hacia el usuario.

Para llevar a cabo esta tarea se hará uso de un formulario, en el cual el usuario introducirá su entrada y un cuadro de texto en el cuál se mostrará la respuesta del bot y la entrada que produjo dicha respuesta.

Aparte de la petición del usuario el intérprete necesita ciertos datos adicionales que se enviarán junto a la entrada de texto y que son:

- **Bot:** Identificador del bot al que va dirigida la consulta. Aunque en este proyecto hay un único bot de ayuda se deja la puerta abierta a la inclusión de múltiples bots con un único intérprete. Gracias a este identificador se podría acceder a uno u otro dependiendo de las necesidades.

- **Timestamp:** Marca de tiempo que se adjunta con el mensaje para evitar que se produzcan problemas de caché y se devuelvan respuestas anteriores almacenadas previamente por el navegador o un *proxy* intermedio. La marca la genera el controlador y se la pasa a la página mediante una variable de sesión.

Al formulario se le añadirá, además del botón de envío, un botón adicional que servirá de enlace a la página de ayuda en la que se explicará en qué consiste el bot y cuál es su modo de uso, siendo el código necesario para llevar a cabo estas funciones:

```
<c:set var = "timeStamp" value = "${sessionScope.timeStamp}"/>

<div>
      <div>
            […]
      </div>

      <form method = "get" action = "./Controller.jsp">
        <fieldset>
          <label for = "request"></label>
          <input type = "text" name = "q" value = "" />
          <input type = "hidden" name = "bot" value = "OrangeBot" />
          <input type = "hidden" name = "timeStamp" value =
            "${timeStamp}" />
          <input type = "image" alt = "Botón de enviar" src =
            "../images/SendButton.gif" value = "Enviar" accesskey =
            "e"/>
          <a id = "helpLink" href = "./help.html" accesskey = "a">
              <img id = "help" alt = "Botón de ayuda" src =
                 "../images/HelpButton.gif" />
          </a>
        </fieldset>
      </form>
</div>
```

Como puede apreciarse en el código, si bien se envía la entrada y el resto de valores necesarios al controlador a través del formulario, no se muestra todavía la respuesta, ya que el <*div*> donde debería incluirse está vacío. Para poder darle una respuesta adecuada al usuario tenemos que tomar los valores que introduce el controlador en las variables de sesión a raíz de la entrada de usuario y una variable del contexto de la página. Estas variables son:

- **userRequest:** *String* con la última entrada del usuario que se ha enviado al intérprete.
- **botResponse:** *String* con la respuesta a la última entrada de usuario que se ha enviado al intérprete.

- **isNew:** *Boleaan* que tiene el valor *true* si el usuario acaba de iniciar una sesión y *false* en caso contrario.

Y se obtienen mediante las siguientes líneas de código:

```
<c:set var = "isNew" value = "${pageContext.session.new}"/>
<c:set var = "userRequest" value = "${sessionScope.q}"/>
<c:set var = "botResponse" value = "${sessionScope.response}"/>
```

Gracias a estas variables podemos presentarle un texto adecuado al usuario, bien mostrándole la entrada que había introducido y la respuesta del intérprete o bien un mensaje de bienvenida en el caso de que esté iniciando la sesión.

Para saber si acaba de iniciar la sesión se comprueban dos variables, "*isNew*" y "*userRequest*". Si "*isNew*" es *true* significa que acaba de iniciar sesión y por lo tanto deberemos mostrarle el mensaje de bienvenida, pero podría ocurrir que el usuario recargue la página de bienvenida sin haber enviado ninguna entrada, con lo cual se interpretaría como que no es una sesión nueva pero no habría ningún tipo de mensaje en el cuadro de texto. Así pues, con la variable "*userRequest*" podemos hacer la comprobación adicional de si el usuario había mandado alguna entrada y en caso de no ser así se mostraría el mensaje de bienvenida. El código que realiza esta tarea es el siguiente:

```
<div>
      <c:choose>
         <c:when test = "${isNew || empty userRequest}">
              Bienvenido al sistema de asistencia técnica de orange,
              para hablar con su asistente escriba en el cuadro de
              texto de la parte inferior:
         </c:when>

         <c:otherwise>
            Usuario: <c:out value = "${userRequest}" />

            <br />

            Erika: <c:out value = "${botResponse}" escapeXml =
               "false"/>
         </c:otherwise>
      </c:choose>
</div>
```

Con este código y el anterior se cumple la funcionalidad básica de la página del bot de ayuda, es decir, recibe una entrada del usuario y le da una respuesta. Sin embargo se necesitan otra serie de funcionalidades adicionales, ya que el intérprete no sólo devuelve una respuesta de texto si no que es posible que dé un estado de ánimo asociado a dicha

respuesta o una URL con datos adicionales. Dicha información también la introduce el controlador en las variables de sesión:

- **state:** *String* con estado de ánimo correspondiente a una respuesta determinada.
- **url:** *String* con la URL donde se puede encontrar información adicional sobre el tema tratado.
- **flashSrc:** *String* auxiliar con la ruta de la animación correspondiente a un estado de ánimo determinado.
- **urlSrc:** *String* auxiliar que contiene el código *javascript* para abrir una URL determinada o una cadena vacía en caso de que no haya ninguna URL asociadas.

Estas variables se obtienen de un modo análogo a las variables anteriores, salvo en el caso de *flashSrc*. Esto se debe a que los valores de todas las variables se introducen después de la primera interacción con el usuario, pero *flashSrc* se usa nada más iniciarse la sesión, por lo tanto habrá que darle un valor por defecto:

```
<c:set var = "state" value = "${ sessionScope.state}"/>
<c:set var = "url" value = "${sessionScope.url}"/>
<c:set var = "urlSrc" value = "${sessionScope.urlSrc}"/>
<c:choose>
   <c:when test = "${isNew || empty userRequest}">
      <c:set var = "flashSrc" value = "../flash/espera.swf"/>
   </c:when>

   <c:otherwise>
      <c:set var = "flashSrc" value = "${sessionScope.flashSrc}"/>
   </c:otherwise>
</c:choose>
```

Una vez están iniciadas las variables correctamente se utilizarán de dos formas diferentes dependiendo de si el usuario tiene *javascript* activado o no:

- **Con *javascript*:** Se mostrará el estado de ánimo del bot mediante una animación *flash* utilizando la variable "*flashSrc*" y el *plugin jMedia* [*jMedURL*] y en caso de tener información adicional en forma de URL se utilizará la variable "*urlSrc*" para abrir dicha URL en la ventana de la página de inicio.
- **Sin *javascript*:** Se mostrará el estado de ánimo del bot mediante una imagen haciendo uso de la variable "*state*" y se añadirá un texto a la respuesta del bot con la variable "*url*" en caso de ser una cadena no vacía.

Para mostrar el estado de ánimo tanto en el caso de usar *javascript* como en el caso de que no esté activado se utilizará el siguiente código:

```
<body onload = "loadAnimation('<c:out value="${flashSrc}" />');">

<c:choose>
   <c:when test = "${isNew || empty userRequest}">
      <div id = "faceframe" class = "hiddenframe">
         <img id = "face" alt = "Imagen que muestra el estado de
              ánimo del bot" src = "../images/espera.jpg" />
      </div>
   </c:when>

   <c:otherwise>
      <div id = "faceframe" class = "hiddenframe">
         <img id = "face" alt = "Imagen que muestra el estado de
              ánimo del bot" src = "../images/<c:out value="${state}"
              />.jpg" />
      </div>
   </c:otherwise>
</c:choose>
```

En el código se puede ver como se carga la imagen de espera en caso de ser inicio de sesión o la imagen correspondiente al estado de ánimo del bot en caso de no estar activado *javascript*, si se da el caso contrario se activaría el método "*loadAnimation*" que se ejecuta al cargar la página y sustituiría dichas imágenes por la animación flash correspondiente. Siendo el código *javascript* que realiza dicha función el siguiente:

```
// Carga animación (jMedia)
function loadAnimation(flash){
      $("#faceframe").jmedia({version:"6,0"},
                  {src:flash, width:134,  height:228, quality:"best",
                        scale:"noborder", wmode:"transparent"});
}
```

En el caso de que, aparte de la respuesta, el intérprete proporcione información adicional por medio de una URL, esta se mostrará, bien como enlace, bien abriendo la dirección en una ventana mediante el siguiente código:

```
<body onload = "<c:out value = "${urlSrc}" />">
   Usuario: <c:out value = "${userRequest}" />
   <br />
   Erika: <c:out value = "${botResponse}" escapeXml = "false"/>
```

```
<noscript>
   <c:if test = "${!(empty url)}">
        Tiene javascript desactivado, para abrir la ventana de
        información pulse en este <a ref = "${url}">enlace</a>)
   </c:if>
</noscript>
```

En este caso si *javascript* está activado al cargar la página se cargará el valor de *urlSrc* que es una función *javascript* que abre en la ventana de la página de inicio de la aplicación la URL correspondiente. Si por el contrario no está activado *javascript*, una vez se presenta la respuesta del bot por pantalla se comprueba si hay información adicional en forma de URL y en caso afirmativo se le presenta al usuario en forma de enlace.

La función *javascript* que abre la URL en la ventana de la página de inicio o en una nueva en caso de que dicha ventana se haya cerrado es la siguiente:

```
// Abre la URL en una ventana parent y si esta no existe abre una
nueva ventana con el identificador indicado ("Orange")

function openWindow(url){
      if (window.opener && !window.opener.closed){
            window.opener.location = url;
            }
      else {window.open(url, 'Orange')};
}
```

Por último, una vez se han realizado las funciones básicas y las funciones adicionales faltan por completar una serie de funciones auxiliares, para mejorar la usabilidad de la página. Estas funciones se realizarán mediante *javascript* y por lo tanto sólo estarán disponibles para aquellos usuarios que lo tengan activado:

- **Validación de Entrada no Nula:** Esta función permite comprobar si al darle al botón de enviar o pulsar *enter* la entrada introducida es una cadena vacía, es decir, no se ha introducido ninguna entrada. En este caso la función evita que se envíe la información al intérprete, con lo cual no tiene que procesar la entrada de manera innecesaria.
- **Enfoque de Formulario:** Esta función permite que cada vez que se recarga la página se seleccione automáticamente la caja de entrada del formulario, con lo que el usuario no tiene que seleccionarla para poder escribir en ella.

Estas funciones se ejecutan al enviar una entrada y al cargar la página respectivamente y su código es el siguiente:

```
// Pone el foco de la página en el campo de entrada del formulario
function focusInput(){
	document.getElementById('requestInput').focus();
}


// Comprueba que la entrada del usuario no esté vacía, si lo está no
envía el formulario
function validate() {
	valor = document.getElementById('requestInput').value;
	if (valor == null || valor == "" || valor == '') {
		return false;
	}
	else {
		return true;
	}
}
```

Una vez se han combinado los códigos de las diferentes funciones la página del bot de ayuda tiene el siguiente aspecto para una captura de inicio de sesión y otra de conversación:

Ilustración 11: Interfaz del bot de ayuda (Sin CSS)

Página de Ayuda

La página de ayuda es la más simple de las tres que componen la interfaz Web ya que su única función es presentar un texto en pantalla y permitir al usuario la posibilidad de volver a la página del asistente. Así pues, lo único que necesitará es un texto con cierta estructura y un enlace. El código para conseguirlo es el siguiente:

```
<h1>Ayuda</h1>

<div>
      <h2>¿Qué es Erika?</h2>
      <p>Erika es un agente conversacional que trata de imitar el
          comportamiento de un asistente técnico.</p>
</div>

<div id = "helpButton">
      <a href = "./Erika.jsp" accesskey = "v">
          <img name = "Volver" alt = "Botón de volver" src =
              "../images/BackButton.gif" />
      </a>
</div>
```

En el código se puede ver una de las entradas de la ayuda y el botón que permite volver a la página del bot de ayuda. Con la ayuda completa, la página mostraría el siguiente aspecto:

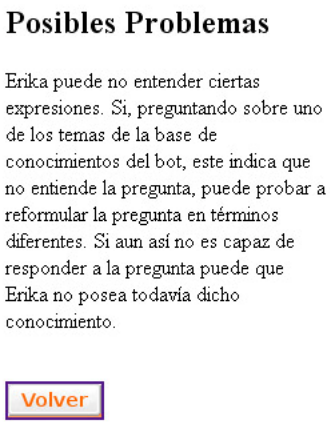

Ilustración 12: Página de ayuda (Sin CSS)

Apariencia

Una vez se ha implementado la parte lógica de la interfaz queda definir una apariencia para que sea más agradable para el usuario y además para darle una mayor coherencia a todas las partes que forman la interfaz.

El primer paso será decidir cuál va a ser el estilo que se le quiera dar a la aplicación, para ello, ya que debería integrarse en la página Web de Orange, se optará por seguir la línea de dicha página. La apariencia tendrá por lo tanto un estilo sencillo que girará en torno a tres colores, naranja, blanco y negro. Además de la parte visual el aspecto también tiene una parte funcional, ya que debe realizarse de tal forma que al mostrarse la aplicación como ventana *popup* permita a su vez ver la ventana raíz y la información que se pueda mostrar en ella.

Con estos conceptos en mente queda bastante definido el aspecto general de la aplicación tanto en tamaño y forma como en colores y líneas generales, pudiendo pasar por tanto a la tarea de aplicar dichas ideas a la apariencia de la aplicación.

Como se ha explicado anteriormente se va a hacer uso de hojas de estilo para implementar el aspecto, por lo tanto el primer paso debe ser identificar los elementos a los que se les quiere variar la apariencia marcándolos con los atributos "*class*" e "*id*". Tomando por ejemplo el código de la página de inicio se puede ver el código de marcado:

```
<body>
  <div id = "ErikaFrame" class = "frame">
   <h1 id = "ErikaHeader">Asistencia técnica: Pregúntale a Erika</h1>
   <div id = "ErikaPicture" class = "hiddenframe">
       <a href = "./pages/Erika.jsp" onclick = "return
           openPopup('./pages/Erika.jsp')" id = "linkErika">
              <img src = "./images/Erika.gif" alt = "Bot de
              Asistencia técnica: Erika" id = "Erika"/>
       </a>
   </div>

   <p id = "ErikaComment">
      * Pulse sobre la imagen para acceder al asistente.
      <noscript>
         <br />
         (Tiene javascript desactivado, si no se abre el asistente
         pulse en este
         <a id = "enlaceAlternativo" target = "Erika" href =
         "./pages/Erika.jsp">
            enlace
         </a>
         )
      </noscript>
   </p>
  </div>
</body>
```

Una vez se han marcado los elementos a los que se les quiere aplicar las hojas de estilo podemos hacer referencia a ellos en un archivo CSS para modificar su aspecto. Por ejemplo para el caso del código anterior tendríamos el siguiente código:

```
body {
      background-color: white; color: black;
      font-family: Arial, Helvetica, sans-serif; font-weight: normal;
}

div.frame {
      background-color: #FFEFDE;
      border-color: #FF6600;  border-style: solid; border-width: 1px;
      font-family: Arial, Helvetica, sans-serif;
}
```

```
div#ErikaFrame {
	width: 200px; border-width: 2px; text-align: center;
}

div#ErikaPicture {
	text-align: center;
}

h1 {
	background-color: #FF6600; color: white; text-align: center;
	font-family: Arial, Helvetica, sans-serif;
	font-size: 14px; font-weight: Bold;
}

h1#ErikaHeader {
	margin: 5px 5px 5px 5px;
	padding: 2px 5px 2px 5px;
}

p#ErikaComment {
	display: block;
	font-size: 10px;
}

img {
	border: none;
}
```

Tras marcar las diferentes páginas que componen la interfaz Web y aplicarles los estilos correspondientes podemos ver como las diferentes partes de la aplicación adquieren una mayor cohesión y son más agradables para el usuario:

Ilustración 13: Páginas de inicio, bot de ayuda y ayuda (Con CSS)

Por último se han implementado dos funciones avanzadas relativas al aspecto, por una parte se han definido variaciones en la hoja de estilos dependiendo de la resolución para el caso de la página del bot de asistencia y la de ayuda, de tal forma que la aplicación se adapte al tamaño del *popup* que la contiene y, por otra parte, se han añadido bordes redondeados al título de la página del bot de ayuda. Estas dos características se han implementado mediante sendas funciones *javascript*:

```
// Cambia el archivo css dependiendo de la resolución del usuario
function changeCSS(){
      if (screen.width==800){
            document.write('<link rel="stylesheet" type="text/css"
            href="../css/style800.css"/>');
      }
}


// Bordes redondeados (jquery.corner)
function roundCorner(){
      $("#mainHeader1").corner("5px");
}
```

La función de bordes redondeados como se puede apreciar en el código hace uso del *plugin jquery.corner* [*jQueURL*].

6.4.2 Diseño de la Interfaz WAP

La interfaz WAP es básicamente una interfaz Web simplificada de tal manera que sea posible mostrarse en terminales que tienen capacidades de presentación y computación limitadas.

En el caso de la interfaz WAP se mantienen las funciones básicas pero desaparecen ciertas funcionalidades avanzadas y auxiliares. También desaparece la capa de presentación como tal, ya que las únicas modificaciones al aspecto serán las que soporte el propio código, el cual no permite el grado de personalización que dan las hojas de estilo. La última diferencia con respecto a la interfaz Web es la eliminación de la página de inicio, ya que su función era presentar la aplicación en forma de *popup* y en la interfaz WAP no existe esa posibilidad, por lo que no es necesario mantenerla.

Tras definir cuáles son las diferencias con respecto a la interfaz Web se pasará a detallar la implementación de los elementos que componen la interfaz WAP, es decir, la pagina del bot de ayuda y la página de ayuda.

Página del Bot de Ayuda

Cómo en el caso de la interfaz Web la tarea principal de la página del bot de ayuda es tomar la entrada del usuario, y una vez el intérprete da una respuesta mostrarla por pantalla. También se mostrará el estado de ánimo y URL's con información adicional sobre el tema tratado. Sin embargo, dadas las limitaciones de los terminales en las que

se va a visualizar se intentará reducir la cantidad de información mostrada por pantalla. Con esta finalidad se tomarán dos medidas:

- **Reducción de Respuestas:** Todas aquellas respuestas que no sean autocontenidas, es decir, aquellas que se puedan explicar en una página Web cuya URL viene asociada con la respuesta, se reducirán para que únicamente devuelvan un mensaje indicando la URL en la que se puede encontrar dicha información.
- **Menor Carga Visual:** A la hora de mostrar el estado de ánimo se suprimirán por completo las animaciones *flash* y se dejarán únicamente las imágenes correspondientes, además estas serán de menor tamaño que en el caso de la interfaz Web.

Para la implementación de estas funciones se seguirá una estructura análoga a la interfaz Web. Se utilizará un formulario para recoger la información del usuario y se presentará una respuesta de texto adecuada para cada caso.

Las variables de las que se hace uso en la interfaz WAP siguen siendo las mismas salvo por dos excepciones, junto con el texto de entrada el formulario manda la variable "*mode*" y para mostrar la respuesta se hace uso de la variable "*test*":

- **Mode:** Identificador del tipo de interfaz que ha enviado la información. En el caso de la interfaz WAP tendrá como valor la cadena de texto "WAP" y sirve para que el controlador sepa dónde tiene que devolver la respuesta. En el caso de la interfaz Web no se utiliza porque es la interfaz por defecto, así que si no se le indica lo contrario al controlador se supondrá que se está pidiendo información desde la interfaz Web.
- **Test:** Es una variable auxiliar que introduce el intérprete junto con la respuesta y consiste en un código que identifica el tipo de respuesta que se ha dado. Este código es muy útil para operaciones internas tanto de pruebas como de funcionalidades adicionales, ya que es más sencillo de analizar que una respuesta completa en lenguaje natural.

Hay que tener en cuenta que la forma de enlazar diferentes páginas WAP es diferente de la que se usa en HTML, por lo que cada página deberá tener un identificador al que puedan hacer referencia el resto de páginas desde las cuales se quiere enlazar.

Al formulario se le añadirá, además del botón de envío, un botón adicional que servirá de enlace a la página de ayuda en la que se explicará en qué consiste el bot y cual es su modo de uso. El código necesario para llevar a cabo todas estas funciones es el siguiente:

```
<template>
   <do name = "menu_2" type = "options" label = "Ayuda">
      <go href = "./help.wml"/>
   </do>
</template>

<card id = "erika_main" title = "Erika">
   <c:choose>
      <c:when test = "${isNew || empty userRequest}">
         <img src = "../images/espera_movil.jpg" />
         Erika, asistencia técnica de Orange:
      </c:when>

      <c:otherwise>
         <img src = "../images/<c:out value = "${state}"
             />_movil.jpg" />
         <c:choose>
            <c:when test = "${empty url || fn:startsWith(test,
            'Fallo')}">
                <strong>Usuario: </strong><c:out value =
                    "${userRequest}" />
                <br />
                <strong>Erika: </strong><c:out value =
                    "${botResponse}" escapeXml = "false"/>
            </c:when>
            <c:otherwise>
                <strong>Usuario: </strong><c:out value =
                    "${userRequest}" />
                <br />
                <strong>Erika: </strong>Puede ver la información que
                    solicita en este
                <a href = "<c:out value = "${url}" />">enlace</a>.
            </c:otherwise>
         </c:choose>
      </c:otherwise>
   </c:choose>

   <p align = "center">
      <input name = "question" value = ""/>
      <anchor>
         <go method = "get" href = "./Controller.jsp">
            <postfield name = "q" value = "$(question)"/>
            <postfield name = "bot" value = "OrangeBot"/>
            <postfield name = "mode" value = "WAP"/>
         </go>
         Enviar
      </anchor>
   </p>
</card>
```

Este código da lugar a la siguiente interfaz (Para el caso de inicio de sesión y para una interacción por parte del usuario):

Ilustración 14: Página del bot de ayuda para la interfaz WAP

Página de Ayuda

Al igual que en la interfaz Web la página de ayuda es mucho más sencilla que la página del bot de ayuda, por lo que requerirá un código menos complejo. La página consta de un texto de ayuda y un enlace a la página principal, por lo que la única dificultad que presenta su implementación es realizar el enlace de manera correcta para una página WML. El código que se utilizará por lo tanto es el siguiente:

```
<wml>
   <card id="erika_ayuda" title="Ayuda">
      <do name="menu_1" type="accept" label="Erika">
         <go href="./ErikaWap.jsp"/>
      </do>

      <p><b>¿Qué es Erika?</b></p>
      <p>
         Erika es un agente conversacional que trata de imitar el
         comportamiento de un asistente técnico.
      </p>
   </card>
</wml>
```

La página de ayuda quedaría pues de la siguiente manera:

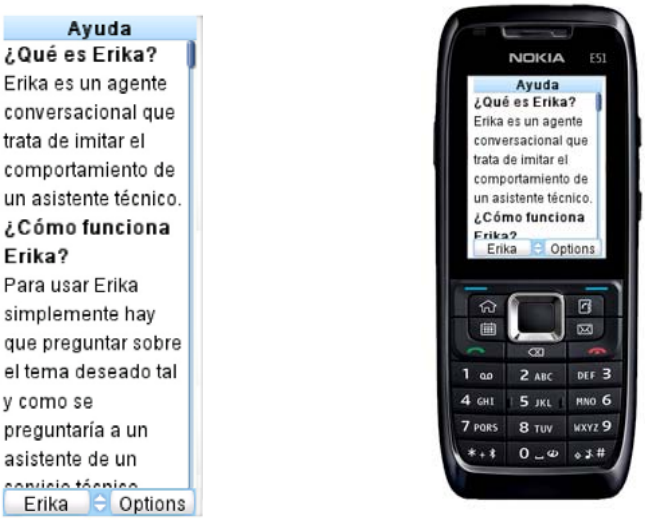

Ilustración 15: Página de ayuda para la interfaz WAP

6.5 Diseño del Controlador

Como se ha explicado en el tema de arquitectura de la aplicación el controlador consta de dos partes el control de vistas y el gestor de entrada de usuario, siendo este último el que realiza la mayor parte de las labores de control.

6.5.1 Control de Vistas

El control de vistas tiene como principal función distribuir las respuestas de manera correcta, es decir, debe tomar la respuesta del intérprete y redirigirla a la interfaz desde la que se solicitó.

Dada su función, el controlador de vistas se implementará como una página JSP a la que se llamará desde las interfaces. En dicha página se importará el *servlet*, con lo que rellenarán las variables de sesión con los valores resultantes de la llamada al intérprete para posteriormente volver a la interfaz que invocó al controlador.

Para realizar la vuelta a la interfaz correcta se hará uso de la variable "*mode*", cuyo valor se rellenará con un valor que identifique a la interfaz que ha invocado al control de vistas.

Por lo tanto el código del JSP que realiza esta función es:

```
<html xmlns = "http://www.w3.org/1999/xhtml" xml:lang = "es" lang =
    "es">
    <head>
        <meta http-equiv = "Content-Type" content =
        "text/html;charset=ISO-8859-15"/>
        <title>Controller</title>
    </head>
```

```
    <body>
     <c:import url = "../TalkToBot" />

     <c:choose>
      <c:when test="${param.mode eq 'WAP'}">
            <c:redirect url = "./ErikaWap.jsp" />
      </c:when>
      <c:otherwise>
            <c:redirect url = "./Erika.jsp" />
      </c:otherwise>
     </c:choose>
    </body>
</html>
```

En el código puede verse como después de importar el resultado de la llamada al *servlet* se comprueba el parámetro *mode*. Si este tiene como valor "WAP" quiere decir que la llamada al controlador se realizó desde la interfaz WAP y por lo tanto se redirige a la página "ErikaWap.jsp" que es la página del bot de ayuda en su versión WAP. En el caso de que la variable "*mode*" tuviese cualquier otro valor se redirigirá a la página "Erika.jsp" que se corresponde con la interfaz Web.

En caso de añadirse otra interfaz simplemente habría que añadir otra comprobación con el identificador correspondiente, y redirigir a dicha interfaz en caso de que se cumpliese la condición.

Como puede apreciarse en el código no se pasa ningún parámetro entre el intérprete y la interfaz a través de la URL, sino que se pasan todos mediante variables de sesión. Esto puede dar lugar a problemas de cacheos de información en los navegadores Web o *proxys*, ya que al comprobar la URL que tiene que cargar puede ver que es igual que la cargada anteriormente y mostraría la página que tuviese guardada en memoria en vez de cargar una nueva página con la información actualizada. Para evitar este problema se adjuntará una marca a la URL de la interfaz Web que varíe en cada interacción, con lo cual los navegadores y *proxys* considerarán que se trata de una página nueva y forzarán la carga de la nueva información.

La marca que se le introduce es la variable "*timestamp*" que se rellena en el control de vistas y que se guarda en sesión, por lo tanto para añadirla se utilizará el siguiente código:

```
<c:set var="timeStamp" value = "${sessionScope.timeStamp}"/>

<c:choose>
     <c:when test = "${param.mode eq 'WAP'}">
           <c:redirect url = "./ErikaWap.jsp" />
     </c:when>
     <c:otherwise>
           <c:redirect url = "./Erika.jsp?timeStamp=${timeStamp}" />
     </c:otherwise>
</c:choose>
```

6.5.2 Gestión de Entrada de Usuario

La función principal del gestor de entrada del usuario es tomar la entrada del usuario, identificar a qué bot va dirigida la petición y qué usuario la realiza y con esos datos solicitar la respuesta del intérprete. Una vez obtenida la respuesta debe almacenar los valores correspondientes a dicha respuesta en las variables de sesión para que las interfaces puedan mostrar dicha respuesta.

Para llevar a cabo dicha tarea se utiliza el siguiente código:

```
// Obtiene la referencia al intérprete y a la sesión
Core coreToUse = (Core) getServletContext().getAttribute("core");
HttpSession session = request.getSession(true);

// Determina el identificador del usuario que realiza la petición
Principal principal = request.getUserPrincipal();
String userId = (principal == null) ? session.getId() :
principal.getName();

// Obtiene el identificador del bot al que va dirigida la petición
// En caso de no encontrarlo pone el valor por defecto
String botId = request.getParameter(botidRequestName);
if(botId == null || botId.equals("null")){botId = orangeBot;}

// Obtiene la respuesta del intérprete y la información adicional
String userRequest = request.getParameter(questionRequestName);
String botResponse = coreToUse.getResponse(userRequest, userId,
      botId);
PredicateMaster predicate = coreToUse.getPredicateMaster();
String state = predicate.get(stateAimlName, userId, botId);
String url = predicate.get(urlAimlName, userId, botId);
String test = predicate.get(testAimlName, userId, botId);

// Guarda la información del intérprete en variables de sesión
session.setAttribute(sessionResponseName, userId);
session.setAttribute(botidResponseName, botId);
session.setAttribute(questionResponseName,userRequest);
session.setAttribute(responseResponseName, botResponse);
session.setAttribute(stateResponseName, state);
session.setAttribute(testResponseName, test);
session.setAttribute(urlResponseName, url);
```

Como puede verse en el código el primer paso es obtener una referencia al intérprete, para ello se obtiene un objeto de tipo *Core* que está almacenado en el contexto del *servlet* y que se inicializa al arrancar el intérprete. Posteriormente se obtienen las referencias al usuario que realiza la petición y al bot al que va dirigida, dando un bot por defecto en el caso de no encontrarse dicho valor.

Con la entrada del usuario y los identificadores de usuario y bot se puede realizar una petición al intérprete, que devolverá la respuesta como una cadena de texto y la información adicional la almacenará como predicados accesibles desde el objeto *Core*.

Por último toda la información devuelta por el intérprete, ya sea directamente o bien accediendo a los predicados se guarda como variables de sesión para su posterior uso.

Además de esta función principal el gestor realiza una serie de funciones adicionales:

Interfaz Genérica

El gestor de control no sólo tiene que dar la información en forma de variables de sesión si no que debe dar la información como una página Web simple a la que pueda acceder cualquier aplicación externa. La información contenida en dicha página debe darse con el siguiente formato:

```
<html>
      <head>
            <title>Servlet DummyBotServlet</title>
      </head>

      <body>
            <p>Session</p>
            <p id = "sessionid">CB074F11E5FF20A08AC9DA4913C2C55F</p>
            <p>Bot id</p>
            <p id = "bot">Erika</p>
            <p>Question</p>
            <p id = "q">Hola, ¿Qué puedo hacer por usted?</p>
            <p>Response:</p>
            <p id = "response">response</p>
            <p>Test:</p>
            <p id = "test">General Saludo</p>
      </body>
</html>
```

Por lo que si se accediese directamente al *servlet* mediante la siguiente URL:

```
http://<url-raíz>/TalkToBot?q=Hola&bot=Erika
```

Donde “url-raíz” es la dirección en la que se ha desplegado la aplicación y le pasamos al *servlet* los parámetros “q” con la entrada para la que queremos una respuesta y “bot” con el identificador del bot al que va dirigida la entrada, obtendríamos la siguiente respuesta:

```
Session

CB074F11E5FF20A08AC9DA4913C2C55F
```

```
Bot id

Erika

Question

hola

Response:

Hola, ¿Qué puedo hacer por usted?

Test:

General Saludo
```

Para conseguir esto, simplemente hay que tomar las variables correspondientes e imprimirlas junto con el código HTML adecuado:

```
PrintWriter out = response.getWriter();

out.println("<html>");
out.println("<head>");
out.println("<title>Servlet DummyBotServlet</title>");
out.println("</head>");
out.println("<body>");
out.println("<p>Session </p>");
out.println("<p id=\"" + sessionResponseName + "\">" +
    session.getId() +  "</p>");
out.println("<p>Bot id</p>");
out.println("<p id=\"" + botidResponseName + "\">" + botId + "</p>");
out.println("<p>Question</p>" );
out.println("<p id=\"" + questionResponseName + "\">" + userRequest +
    "</p>");
out.println("<p>Response:</p>");
out.println("<p id=\"" + responseResponseName + "\">" + botResponse +
    "</p>");
out.println("<p>Test:</p>");
out.println("<p id=\"" + testResponseName + "\">" + test + "</p>");
out.println("</body>");
out.println("</html>");

out.close();
```

Creación de Variables Auxiliares

Como se ha explicado en el diseño de las interfaces, éstas hacen uso de ciertas variables adicionales a las que proporciona el intérprete, es decir a "botResponse", "*url*", "*state*" y "*test*". El gestor de entrada de usuario debe por lo tanto crear esas variables adicionales a partir de las que proporciona el intérprete o creándolas desde cero. Las variables que crea el gestor de entrada de usuario son:

- **urlSrc:** Es una variable de texto que utiliza la interfaz Web. Consiste en el código necesario para abrir la URL que da como respuesta el intérprete en una ventana de tipo *popup*.
- **flashSrc:** Al igual que en el caso anterior es una variable de texto que se utiliza en la interfaz Web. Proporciona la URL con la animación flash asociada a un estado de ánimo determinado.
- **timeStamp:** Es una variable de tipo long que expresa un momento de tiempo. Se utiliza como marca de tiempo en el control de vistas.

Para la obtención de estas variables el gestor utiliza el siguiente código:

```
String urlSrc = textProcessor.getUrlSrc(url);
String flashSrc = textProcessor.getFlashSrc(state);
long timeStamp = Calendar.getInstance().getTime().getTime();

// Creación de la variable urlSrc
// Si la url está vacía devuelve una cadena vacía
// Si hay url devuelve una cadena de tipo “openWindow(‘url’);”
public String getUrlSrc(String url) {
      String urlSrc = "";
      if (url!=null && !url.equals("")){urlSrc = "openWindow('" + url
            + "');";}
      return urlSrc;
}

// Creación de la variable flashSrc
// Si no hay ningún estado se pone el valor por defecto “espera”
// Todos los estados de acierto y de fallo se reducen a “acierto” o
// “fallo” respectivamente
// Se devuelve una cadena de tipo “../flash/estado.swf”
public String getFlashSrc(String state) {
      if (state==null){state = "espera";}
      if (state.startsWith("acierto")){state="acierto";}
      if (state.startsWith("fallo")){state="fallo";}
      String flashSrc = "../flash/" + state + ".swf";
      return flashSrc;
}
```

Reinicio de Variables

Algunas variables que envía el intérprete es necesario que se reinicien entre interacción e interacción para un correcto funcionamiento ya que el intérprete las almacena y sólo las cambia si tiene que sobrescribirlas.

Un ejemplo de esta situación es la variable “*url*”. Supongamos que el usuario introduce una entrada que tiene asociada una respuesta compuesta, es decir cadena de texto y URL. La URL se guarda en los predicados y hasta que se sobrescriba con otra URL. Ahora supongamos que inmediatamente después el usuario introduce otra entrada cuya respuesta no tiene asociada ninguna URL. El controlador miraría en los predicados

y descubriría la URL anterior devolviéndosela al usuario. Por lo tanto cada vez que se obtenga el valor de la variable "*url*" se deberá reiniciar al valor de cadena vacía para que no se de esta situación.

El código para reiniciar la variable "*url*" es el siguiente:

```
predicate.set(urlAimlName, 1, "", userId, botId);
```

Saneamiento de Entrada de Usuario

Al tratarse de una aplicación que permite la introducción de texto por parte del usuario está sujeta a ciertos peligros, ya que el usuario puede introducir código que de alguna forma pueda ejecutarse en el servidor.

Para minimizar estos riesgos, antes de enviar la entrada de usuario al intérprete y procesarla esta se preprocesa eliminando todos aquellos símbolos potencialmente peligrosos como pueden ser "<", ">", "/", "\", "&", etc. Y que no afectan a la correcta comprensión de una entrada no dañina.

Aparte se aprovecha este preprocesado para limpiar la entrada de todos aquellos espacios en blanco innecesarios.

El código para procesar la entrada del usuario es el siguiente:

```
public String processRequest(String request) {
  if (request != null){
    removeSpaces(request);
    // Elimina símbolos potencialmente peligrosos
    request = request.replaceAll("<", " ").replaceAll(">", " ");
    request = request.replaceAll("/", " ").replaceAll("\\\\", " ");
    request = request.replaceAll("\\{", " ").replaceAll("\\}", " ");
    request = request.replaceAll("#", " ");
    request = request.replaceAll("&", " ");
    request = request.replaceAll("$", " ");
    request = request.replaceAll("%", " ");
  }
  return request;
}

private String removeSpaces (String text){
  // Elimina retornos de carro
  text = text.replaceAll("\\n", "").replaceAll("\\t", "");
  // Elimina espacios múltiples
  text = text.replaceAll("\\b\\s{2,}\\b", " ").replaceAll(" \\s{2,}
", " ");
  // Elimina espacios iniciales
  if (text.startsWith(" ")||text.startsWith("\\b")){
    text = text.substring(1);
  }
  return text;
}
```

Comprobación de Entrada Vacía

Es posible que un usuario envíe una entrada vacía desde la interfaz de la aplicación. Al no corresponderse con ninguno de los patrones de la base de conocimiento el intérprete devolvería una respuesta vacía lo cual no es recomendable. Para evitar este funcionamiento y además evitar sobrecargar al intérprete con peticiones innecesarias el *servlet* comprobará que la entrada no está vacía antes de enviarla al intérprete.

En caso de tratarse de una entrada vacía el *servlet* actuará como si hubiese mandado la petición pero devolverá los valores de las variables asociados a la última interacción con el usuario.

```
if (!(userRequest==null || userRequest.equals(""))){

      […]

}

else {
      PrintWriter out = response.getWriter();

      out.println("<html>");
      out.println("<head>");
      out.println("<title>Servlet DummyBotServlet</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<p>Session </p>");
      out.println("<p id=\"" + sessionResponseName + "\">"+
            session.getId() +  "</p>");
      out.println("<p>Bot id</p>");
      out.println("<p id=\"" + botidResponseName + "\">" + botId +
            "</p>");
      out.println("<p>Question</p>");
      out.println("<p id=\"" + questionResponseName + "\">" + null +
            "</p>");
      out.println("<p>Response:</p>");
      out.println("<p id=\"" + responseResponseName + "\">" + null +
            "</p>");
      out.println("<p>Test:</p>");
      out.println("<p id=\"" + testResponseName + "\">" + null +
            "</p>");
      out.println("</body>");
      out.println("</html>");

      out.close();
}
```

7 Registro de Eventos

En el presente capítulo se explicará la captura y registro de eventos por parte de la aplicación y se detallará la implementación y configuración de un sistema de registro de conversaciones e incidencias que permita almacenar dicha información de manera permanente.

7.1 Introducción

El registro de eventos tiene una importancia fundamental en el desarrollo de aplicaciones ya que permite conocer las posibles incidencias que se producen durante el funcionamiento de una aplicación y obtener información útil a la hora de modificarla o ampliar el alcance de la misma.

Para el registro de eventos del bot de ayuda se hará uso de *Log4j* y se crearán dos archivos de *log*, uno en el que se almacenarán todos los mensajes de información de progreso e incidencias de la aplicación y otro en el que se guardarán todas las conversaciones mantenidas entre la aplicación y los usuarios.

7.2 Log4j

Log4j [*Lg4jURL*] es una biblioteca *open source* desarrollada en Java por *Apache Software Foundation* [*ApacURL*] que permite la captura y registro de eventos en tiempo de ejecución.

La configuración de salida de los mensajes de *log* se realiza mediante archivos de configuración externos que permiten definir tanto la presentación como la granularidad de los mensajes mediante el establecimiento de niveles de prioridad.

7.2.1 Niveles de prioridad

Log4j [*Al4jURL*] viene configurado por defecto con 6 niveles de prioridad siendo estos por orden de mayor a menor prioridad:

FATAL	Errores críticos del sistema que causan la terminación de la aplicación.
ERROR	Errores que pueden permitir que la aplicación siga funcionando.
WARN	Mensajes que informan de situaciones potencialmente perjudiciales.
INFO	Mensajes de información que señalan el progreso de la aplicación.
DEBUG	Mensajes de eventos que pueden resultar útiles para depurar la aplicación.
TRACE	Mensajes para depuración mayor grado de detalle que DEBUG.

Tabla 9: Niveles de prioridad de Log4j

7.2.2 Configuración de Log4j

La configuración de *Log4j* se puede realizar mediante un archivo de propiedades o un archivo XML. Para llevar a cabo la configuración se definen tres componentes:

- **Logger:** Son los objetos a los que llama la aplicación para generar los diferentes *logs*. Cada *logger* puede ser definido y configurado de manera independiente.
- **Appender:** Es la salida a la que se envía uno o más *logs*. Existen varios *appenders* disponibles por defecto como pueden ser salida de mensajes redirigida a un fichero de texto (*FileAppender* o *RollingFileAppender*), a un servidor remoto (*SocketAppender*) o a una base de datos (*JDBCAppender*).
- **Layout:** Es el responsable de dar un formato de presentación a los mensajes bien como archivo de texto (*SimpleLayout* o *PatternLayout*), tabla HTML (*HTMLLayout*) o archivo XML (*XMLLayout*).

7.2.3 Uso de Log4j

Para usar *Log4j* para la captura y registro de eventos en una aplicación deben seguirse los siguientes pasos:

1. **Carga de Clases:** El primer paso es importar las clases necesarias de *Log4j* en el código de la clase en la cual se quiere usar.

```
import org.apache.log4j.LogManager;
import org.apache.log4j.Logger;
```

2. **Definición del logger:** Se crea un objeto de tipo *Logger* a partir del nombre de la clase en la que se va a usar o se obtiene una instancia mediante un identificador de tipo *String*.

```
// Creación del objeto de tipo Logger
static Logger logger = Logger.getLogger("ClassName.class");

// Obtención de instancia del objeto de tipo Logger
private Logger logger = LogManager.getLogger("name");
```

3. **Configuración del logger:** Se configuran los *loggers*, *appenders* y *layouts* en un archivo externo de tipo *properties* o xml.

7.3 Archivos de Log

En el caso del bot de ayuda se definirán dos archivos de *log*, por una parte el archivo de incidencias, en el que se registrará toda la información relativa al funcionamiento de la aplicación y las posibles incidencias que se puedan producir y por otra parte un archivo de conversaciones en el que quedarán registradas las distintas entradas de los usuarios junto con las respuestas que dichas entradas producen.

7.3.1 Log de Incidencias

El *log* de incidencias es la referencia para determinar las causas de cualquier funcionamiento anómalo de la aplicación. En este *log* se recoge toda la información del progreso de la aplicación y se detallan todos los posibles problemas de funcionamiento.

El archivo de *log* en el que se almacena la información es "*activity.log*" y los mensajes que registra tienen el siguiente formato:

```
[Fecha Hora] Prioridad: Mensaje
```

Además se limitará el tamaño máximo del archivo generado a 10 MB, por lo tanto una vez definido el *logger* en las clases correspondientes se configurará en un archivo XML mediante el siguiente código:

```
<appender name = "activitylog" class =
      "org.apache.log4j.RollingFileAppender">
      <param name = "File" value =
            "/var/log/orangebot/activity.log"/>
      <param name = "MaxFileSize" value = "10MB"/>
      <param name = "MaxBackupIndex" value = "10"/>
      <layout class = "org.apache.log4j.PatternLayout">
            <param name = "ConversionPattern" value = "[%d{ISO8601}]
                  %p: %m%n"/>
      </layout>
</appender>
```

Esto da lugar a archivos de *log* con la siguiente estructura:

```
[2008-08-22 18:19:45,140] INFO: Loading
      jndi:/localhost/Erika/aiml/comun/general.aiml....
[2008-08-22 18:19:45,171] DEBUG: Adding watch file "
      jndi:/localhost/Erika/aiml/comun/general.aiml".
[2008-08-22 18:19:45,171] ERROR: File not found
      java.io.FileNotFoundException: Couldn't find
      jndi:/localhost/Erika/aiml/movil.aiml".
```

7.3.2 Log de Conversaciones

En el *log* de conversaciones se recogen todas las interacciones entre usuario y bot de ayuda. Este registro permite descubrir problemas en la base de conocimiento ya que detalla aquellas entradas de usuario para las cuales no se ha encontrado un patrón. Un análisis detallado de este *log* nos permite, por lo tanto, tomar medidas a la hora de ampliar o mejorar la base de conocimiento y obtener estadísticas de las preferencias de los usuarios que pueden ser útiles para definir estrategias de marketing.

El archivo en el que se almacena esta información es "*statistics.log*" y el *logger* que lo produce tiene la peculiaridad de ir almacenando de manera automática en diferentes archivos las conversaciones que se han mantenido cada día. Así pues al final de cada día se guardará una copia de "*statistics.log*" con el nombre "*statistics.log.YYYY-MM-DD*" (Siendo YYYY el año, MM el mes y DD el día en el que se produjo dicho archivo de *log*) y se creará un nuevo archivo "*statistics.log*" vacío. El formato de los mensajes será:

```
[Fecha Hora] PRIORIDAD: IdUsuario>"Entrada"; Erika>"Respuesta";
      test>MensajeTest
```

- **IdUsuario:** Identificador de sesión del usuario que realiza la petición.
- **Entrada:** Entrada de texto que introduce el usuario.
- **Respuesta:** Respuesta del bot de ayuda asociada a la entrada del usuario.
- **MensajeTest:** Mensaje auxiliar correspondiente a la respuesta del bot.

Siendo la prioridad "INFO" en caso de que el patrón correspondiente a la entrada se encuentre en la base de conocimiento y "WARN" en caso contrario.

Por lo tanto una vez se ha inicializado el *log* en la clase correspondiente y se han definido los diferentes mensajes con el formato deseado se configura el *logger* mediante el siguiente código:

```
<appender name = "statisticslog" class =
      "org.apache.log4j.DailyRollingFileAppender">
      <param name = "File" value =
            "/var/log/orangebot/statistics.log"/>
      <param name = "DatePattern" value = "'.'yyyy-MM-dd"/>
      <layout class="org.apache.log4j.PatternLayout">
            <param name = "ConversionPattern" value = "[%d{ISO8601}]
                  %p: %m%n"/>
      </layout>
</appender>
```

Dando lugar a archivos de *log* de tipo:

```
[2008-08-26 15:05:22,234] INFO:
      6E7588C095F4D122F41B3DCE582D653D>"Hola "; Erika>"Hola, ¿Cómo
      puedo ayudarle? "; test:General Saludo
[2008-08-26 15:05:41,828] INFO:
      6E7588C095F4D122F41B3DCE582D653D>"Me gustaría conocer las
      últimas ofertas de telefonía móvil "; Erika>" Puede ver todas
      las ofertas disponibles actualmente en la Tienda Orange. ";
      test:Orange Ofertas
[2008-08-26 15:06:14,765] WARN:
      6E7588C095F4D122F41B3DCE582D653D>"Podrías darme información
      sobre la blackberry? "; Erika>" Lo siento, me temo que no
      dispongo de información sobre el tema que me está preguntando.
      Si se trata de información sobre Orange puede probar buscando
      en la web de Orange o a lo mejor logro entenderle si formula su
      pregunta de otra manera. "; test:Fallo
[2008-08-26 15:06:55,671] INFO:
      6E7588C095F4D122F41B3DCE582D653D>"Gracias "; Erika>" De nada,
      me alegro de que le haya podido ayudar. "; test:General
      Conversacion Gracias
[2008-08-26 15:07:14,109] INFO:
      6E7588C095F4D122F41B3DCE582D653D>"Hasta luego "; Erika>"Hasta
      luego, gracias por utilizar nuestro servicio. "; test:General
      Despedida
```

8 Fase de Pruebas

La fase de pruebas permite evaluar la calidad de un sistema software. Es por lo tanto, una fase de suma importancia para comprobar si se han alcanzado los objetivos iniciales del proyecto.

En este capítulo se detallarán las pruebas realizadas para verificar el correcto funcionamiento de la aplicación tanto a nivel lógico como de presentación, así como el cumplimiento de los requisitos previamente estipulados.

8.1 Introducción

Dentro de las pruebas a realizar sobre la aplicación se pueden distinguir dos ámbitos bien diferenciados. Por una parte están las pruebas de funcionamiento encargadas de verificar la lógica de la aplicación y por otra parte las pruebas de presentación para comprobar la correcta visualización de las interfaces.

Dada la naturaleza tan diferente de los elementos a comprobar la metodología de pruebas será distinta para cada uno de los casos, utilizándose pruebas automatizadas mediante *JUnit* [*JUniURL*] y *HttpUnit* [*HUniURL*] para comprobar el correcto funcionamiento de la aplicación y pruebas manuales para verificar la presentación de la misma.

8.2 JUnit

JUnit es un conjunto de librerías utilizadas para la realización de pruebas unitarias en Java. Para lograr dicha funcionalidad *JUnit* permite la ejecución controlada de clases Java de tal manera que pueda evaluar si las clases probadas se comportan como se espera. Es decir, definiendo una entrada se comprueba si el valor devuelto se corresponde con el valor esperado, considerándose superada la prueba en caso afirmativo y fallada en caso contrario.

JUnit no sólo es útil a la hora de comprobar el correcto funcionamiento de una clase si no que también sirve como método de controlar las pruebas de regresión, es decir, para comprobar que una vez se ha modificado parte de la aplicación, esta sigue cumpliendo con los requisitos anteriores.

Existen dos conceptos fundamentales en las herramientas de tipo *XUnit* entre las que se engloba *JUnit,* los casos de prueba y las suites de prueba. Un caso de prueba es un módulo que es capaz de probar los métodos de una clase o un módulo concreto, mientras que una suite es un conjunto de casos de prueba que evalúan el correcto funcionamiento de módulos relacionados funcionalmente.

Así pues, gracias a *JUnit* se pueden construir programas capaces de probar los distintos módulos de una aplicación y que se pueden ejecutar de forma automática, simplificando notablemente el proceso de realización de pruebas de funcionamiento de dicha aplicación.

8.3 HttpUnit

HttpUnit es un conjunto de librerías Java construidas sobre *JUnit* que permiten la realización de pruebas en un entorno Web sin necesidad de un navegador. Para ello *HttpUnit* soporta el envío de formularios HTML, *javascript*, autenticación, redirección automática y gestión de sesiones mediante *cookies*. Además, *HttpUnit* permite procesar las páginas Web devueltas por las aplicaciones a probar, ya sea en forma de texto plano o XML.

El elemento principal de *HttpUnit* es la clase *WebConversation* que hace las veces de navegador a la hora de conectarse a un sitio Web, siendo además responsable del mantenimiento de sesión.

```
WebConversation webConversation = new WebConversation();
WebRequest webRequest = new GetMethodWebRequest(testUrl);
WebResponse response = webConversation.getResponse(webRequest);
```

En el código anterior puede verse cómo se crea un objeto de tipo *WebConversation* para, posteriormente, obtener la respuesta asociada a una URL determinada. Una vez obtenida esa respuesta se puede procesar para ser utilizada en los diferentes casos de prueba con los cuales se comprobará el correcto funcionamiento de la aplicación Web.

8.4 Pruebas de Funcionamiento

Dentro de las pruebas de funcionamiento se engloban tanto aquellas que permiten verificar que el comportamiento de la aplicación se corresponde con el esperado para aquellos casos contemplados, como aquellas que comprueban el grado de cumplimiento de requisitos de la base de conocimiento.

La naturaleza de las pruebas de funcionamiento permite su implementación como pruebas automatizadas, ya que se pueden valorar de forma objetiva. Para saber si una prueba se supera o no simplemente se debe definir un caso concreto y comprobar si los resultados obtenidos se corresponden con los esperados.

La automatización de estas pruebas permite no sólo comprobar de forma rápida si la aplicación final funciona de manera correcta y conforme a los requisitos si no que es una herramienta muy útil en caso de querer ampliar o modificar la aplicación ya que permite saber si las modificaciones han afectado de alguna manera al funcionamiento previo.

Dentro de las baterías de prueba que se han definido se contemplan los siguientes casos implementados usando las librerías *JUnit* y *HttpUnit*:

8.4.1 Comprobación de Petición Básica

La prueba de funcionamiento básica consiste en enviar una entrada de texto a un bot activo. Si la aplicación funciona correctamente debería devolver los parámetros "*response*" y "*test*" con un valor que dependerá de la entrada de texto pero que deberá

ser diferente de "*null*" y "q" deberá tener el valor de la entrada de texto saneada por lo tanto tampoco podrá tener el valor "*null*".

```
public void testBotCliente() throws Exception{

	String testUrl = url + "?q=prueba&bot=Erika";

	WebConversation webConversation = new WebConversation();
	WebRequest webRequest = new GetMethodWebRequest(testUrl);
	WebResponse response = webConversation.getResponse(webRequest);

	assertFalse(response.getElementWithID("response").getText()
		.equals("null"));
	assertFalse(response.getElementWithID("test").getText()
		.equals("null"));
	assertFalse(response.getElementWithID("q").getText()
		.equals("null"));
}
```

8.4.2 Comprobación de Petición sin Entrada de Usuario

Durante la prueba de petición sin entrada de usuario se envía únicamente como parámetro un bot en activo pero no se envía ninguna entrada de texto. Si la aplicación funciona correctamente los parámetros "*response*", "*test*" y "q" deberían encontrarse vacíos y por lo tanto su valor en el HTML de respuesta será "*null*".

```
public void testBotCliente() throws Exception{

	String testUrl = url + "?bot=Erika";

	WebConversation webConversation = new WebConversation();
	WebRequest webRequest = new GetMethodWebRequest(testUrl);
	WebResponse response = webConversation.getResponse(webRequest);

	assertEquals("null",response.getElementWithID("response")
		.getText());
	assertEquals("null",response.getElementWithID("test")
		.getText());
	assertEquals("null",response.getElementWithID("q")
		.getText());
}
```

8.4.3 Comprobación de Petición sin Identificador de Bot

En el caso de la comprobación de petición sin identificador de bot se envía una entrada de texto pero no se especifica ningún bot al que enviar la petición. Si la aplicación funciona correctamente debería asignar un bot por defecto a la petición y

procesarla de manera normal, siendo por lo tanto el funcionamiento esperado igual al del caso de una petición básica, con los parámetros "*response*", "*test*" y "q" distintos del valor "*null*".

```
public void testBotCliente() throws Exception{

	String testUrl = url + "?q=prueba";

	WebConversation webConversation = new WebConversation();
	WebRequest webRequest = new GetMethodWebRequest(testUrl);
	WebResponse response = webConversation.getResponse(webRequest);

	assertFalse(response.getElementWithID("response").getText()
		.equals("null"));
	assertFalse(response.getElementWithID("test").getText()
		.equals("null"));
	assertFalse(response.getElementWithID("q").getText()
		.equals("null"));
}
```

8.4.4 Comprobación de Petición sin Parámetros

En la comprobación de petición sin parámetros se introduce únicamente la dirección del servicio pero no se especifica ni una entrada de texto ni un bot activo. Si se da el funcionamiento normal de la aplicación se asignaría un bot por defecto a la petición pero la cadena de texto sería vacía dándose un caso análogo al de petición sin entrada de usuario, y por lo tanto se espera el mismo comportamiento, es decir los parámetros "*response*", "*test*" y "q" deberían tomar el valor "*null*".

```
public void testBotCliente() throws Exception{

	String testUrl = url;

	WebConversation webConversation = new WebConversation();
	WebRequest webRequest = new GetMethodWebRequest(testUrl);
	WebResponse response = webConversation.getResponse(webRequest);

	assertEquals("null",response.getElementWithID("response")
		.getText());
	assertEquals("null",response.getElementWithID("test")
		.getText());
	assertEquals("null",response.getElementWithID("q")
		.getText());
}
```

8.4.5 Comprobación de Mantenimiento de Sesión

Durante la prueba de mantenimiento de sesión se envían dos peticiones básicas con el mismo usuario. En caso de funcionamiento correcto la aplicación debería reconocer

ambas peticiones como pertenecientes al mismo usuario y por lo tanto el identificador de sesión ha de ser el mismo en los dos casos.

```
public void testBotCliente() throws Exception{

   String testUrl = url + "?q=prueba&bot=Erika";

   WebConversation webConversation = new WebConversation();

   WebRequest webRequest1 = new GetMethodWebRequest(testUrl);
   WebResponse response1 = webConversation.getResponse(webRequest1);

   WebRequest webRequest2 = new GetMethodWebRequest(testUrl);
   WebResponse response2 = webConversation.getResponse(webRequest2);

   String id1 = "Id1";
   String id2 = "Id2";

   if (response1.getElementWithID("sessionid")!=null){
      id1 = response1.getElementWithID("sessionid").getText().
          toString();
   }

   if (response2.getElementWithID("sessionid")!=null){
      id2 = response2.getElementWithID("sessionid").getText().
          toString();
   }

assertEquals(id1, id2);
}
```

8.4.6 Comprobación de Sesiones Múltiples

En la comprobación de sesiones múltiples se envían dos peticiones básicas, cada una como un usuario diferente. Si la aplicación funciona correctamente debería reconocer ambas peticiones como pertenecientes a diferentes usuarios y por lo tanto el parámetro "*sessionid*" deberá ser diferente en cada una de las respuestas.

```
public void testBotCliente() throws Exception{

   String testUrl = url + "?q=prueba&bot=Erika";

   WebConversation webConversation1 = new WebConversation();
   WebRequest webRequest1 = new GetMethodWebRequest(testUrl);
   WebResponse response1 = webConversation1.getResponse(webRequest1);
```

```
    WebConversation webConversation2 = new WebConversation();
    WebRequest webRequest2 = new GetMethodWebRequest(testUrl);
    WebResponse response2 = webConversation2.getResponse(webRequest2);

    String id1 = "Id1";
    String id2 = "Id2";

    if (response1.getElementWithID("sessionid")!=null){
       id1 = response1.getElementWithID("sessionid").getText().
            toString();
    }

    if (response2.getElementWithID("sessionid")!=null){
       id2 = response2.getElementWithID("sessionid").getText().
            toString();
    }

    assertFalse(id1.equals(id2));
}
```

8.4.7 Comprobación de la Base de Conocimiento

Para la comprobación de la base de conocimiento se utilizará un fichero con posibles entradas de usuario y las respuestas que debería dar la aplicación haciendo uso de la variable auxiliar "*test*".

Las diferentes entradas deben cubrir los requisitos de la base de conocimiento, y la prueba consistirá en enviar cada una de las entradas incluidas en el archivo a la aplicación y comprobar que la respuesta que devuelve el bot de ayuda se corresponde con el resultado esperado.

Una vez se han pasado todas las pruebas de la base de conocimiento el programa de pruebas devuelve un informe con los resultados tanto en formato de texto plano como en HTML.

En el siguiente cuadro se puede ver el resultado de las pruebas de la base de conocimiento de la versión final de Erika con una batería de pruebas de 653 entradas de texto (se ha eliminado parte de la información del informe para que resulte legible).

Total tests made: 653
Total tests failed: 0 (0.0%)
Total tests passed: 653

Succesfull tests:

User Input
Orange
¿Orange opera sólo en España?

¿Orange es más barata que el resto de compañías?
¿Qué es Orange?
…
¿Se pueden grabar las películas que compro?
¿Podré grabar contenidos?
¿Cómo se pueden grabar contenidos?

Failed tests:

User Input	Expected Answer	Wrong Answer

A la vista de los resultados se puede comprobar que responde de manera correcta a las 653 entradas que se han definido en la batería de pruebas.

8.4.8 Resultados de las Pruebas de Funcionamiento

La versión final del bot de ayuda superó todas las pruebas descritas en los apartados anteriores como se puede comprobar en el siguiente informe de pruebas:

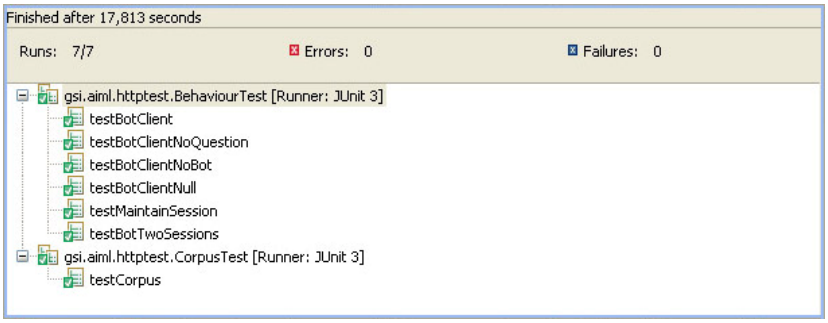

Ilustración 16: Informe de pruebas realizadas sobre la aplicación

Guardando, los nombres que aparecen en el informe, la siguiente correspondencia con los casos de prueba de los apartados anteriores:

- **testBotClient:** Comprobación de petición básica.
- **testBotClientNoQuestion:** Comprobación de petición sin entrada de usuario.
- **testBotClientNoBot:** Comprobación de petición sin identificador de bot.
- **testBotClientNull:** Comprobación de petición sin parámetros.

- **testMaintainSession:** Comprobación de mantenimiento de sesión.
- **testBotTwoSessions:** Comprobación de sesiones múltiples.
- **testCorpus:** Comprobación de la base de conocimiento.

8.5 Pruebas de Visualización

Durante las pruebas de visualización se comprueba si la interfaz de la aplicación se muestra de manera correcta al usuario. Al contrario que en el caso de las pruebas de funcionamiento, estas pruebas no se pueden automatizar de manera sencilla, ya que no dependen de datos objetivos como el valor de una variable concreta.

Debido a las características especiales de las pruebas de visualización estas se realizarán de forma práctica comprobando que la aplicación se presenta al usuario de manera correcta en distintas situaciones. Hay que tener en cuenta, en cualquier caso, que sería inviable comprobar la correcta presentación de la aplicación para todos los dispositivos y navegadores Web soportados, así que se optará por realizar las pruebas para un número reducido de casos que se pretende que sean representativos de un gran número de sistemas.

8.5.1 Interfaz Web

Para la comprobación de la interfaz Web se visualizará la aplicación sobre los dos navegadores con mayor número de usuarios a día de hoy, *Internet Explorer* y *Firefox* (Concretamente la versión 7 de *Internet Explorer* y las versiones 2 y 3 de *Firefox*).

Ilustración 17: Visualización de interfaz Web en Firefox 2, Firefox 3 e Internet Explorer 7

Como se puede comprobar en las imágenes, las diferencias de presentación entre las diferentes versiones son mínimas y en todas ellas se visualiza de manera correcta la aplicación.

8.5.2 Interfaz WAP

En el caso de la interfaz WAP se ha optado por dos métodos para comprobar su correcto funcionamiento. Por un lado se han realizado pruebas sobre un navegador Web con un *plugin* específico para visualizar interfaces WAP y por otro lado sobre un emulador de móvil genérico.

Ilustración 18: Visualización de interfaz WAP en navegador Web (1) y emulador de móvil (2)

De nuevo en las imágenes se puede ver que, aun habiendo ligeras diferencias entre ambas pruebas, la aplicación se visualiza de manera correcta en los dos casos.

9 Conclusiones y Trabajos Futuros

Una vez finalizado el proyecto se puede analizar desde otra perspectiva el grado de consecución alcanzado en el mismo, así como los diferentes factores que han influido en su desarrollo.

En este capítulo se intentará no sólo valorar el trabajo que se ha llevado a cabo y las dificultades que han surgido durante todo el proceso, sino establecer las bases para posibles trabajos futuros.

9.1 Introducción

Al comienzo del proyecto no sólo se definió en qué consistía y cuál era la finalidad del mismo, sino que estos puntos se concretaron en una serie de requisitos que se debían cumplir una vez hubiera concluido.

A lo largo de este capítulo se analizará el trabajo realizado durante el desarrollo del bot de ayuda, comprobando si su versión final satisface todos los requisitos exigidos. Además se definirán las dificultades que han ido apareciendo a lo largo de todo el proceso y por último se darán algunas ideas relativas a trabajos de mejora o ampliación que se podrían realizar tomando como base este proyecto.

9.2 Trabajo Desarrollado

La forma más objetiva de realizar una valoración del trabajo realizado a lo largo un proyecto es comprobar si se han cumplido los requisitos especificados al comienzo del mismo. Así pues en este apartado se tratará de valorar el grado de satisfacción tanto de los requisitos funcionales como de los no funcionales.

9.2.1 Requisitos Funcionales

- **Respuesta a preguntas:** Este requisito es el punto fundamental del proyecto, se ha conseguido desarrollar una aplicación capaz de analizar la entrada de un usuario, comprobar su base de conocimiento y responder consecuentemente, bien con una respuesta adecuada en caso de estar contemplada en la base de conocimiento, bien informando al usuario en caso contrario.
- **Base de conocimiento conforme a las necesidades del cliente:** Como se ha especificado en el requisito previo habrá entradas que estén contempladas en la base de datos y otras que no lo estén. Esto se debe a que la base de conocimiento es limitada y podrá mejorarse y ampliarse de manera continua. Dada la naturaleza de la base de conocimiento debe establecerse un límite que permita definir cuando se ha completado la base de conocimiento. En el

bot de ayuda este límite está definido por las necesidades del cliente y más concretamente por el conjunto de casos de uso para los cuales debe obtenerse una respuesta adecuada. Una vez terminado el proyecto se ha conseguido reconocer el 100% de las entradas que definen las entradas del usuario, así como múltiples entradas no contempladas inicialmente.

- **Redirección a página Web:** El bot de ayuda no sólo proporciona una respuesta textual adecuada a la entrada del usuario, si no que es capaz de proporcionar en aquellos casos en los que sea aplicable información adicional en forma de dirección de una página Web donde puede encontrarse información adicional sobre el tema solicitado. Esta información puede manejarse de diferentes maneras dependiendo de la interfaz utilizada, en el caso de las interfaces implementadas esta información se muestra de forma textual en el caso de la interfaz WAP o mediante la apertura automática de una ventana con la página Web en el caso de la interfaz Web.
- **Emociones dependiendo de la pregunta:** De la misma forma que se proporcionan las direcciones de páginas Web con la respuesta el bot de ayuda es capaz de proporcionar un indicador del estado de ánimo relacionado con la entrada del usuario. La visualización del mismo depende de la interfaz, siendo en el caso de la interfaz WAP mediante imágenes y en la interfaz Web mediante animaciones (O imágenes en aquellos casos en los que no se soporten dichas animaciones).
- **Logs de conversaciones:** Como se ha detallado en el tema "Registro de Eventos" el bot de ayuda posee un completo registro tanto de incidencias de la aplicación como de conversaciones. Además estos registros son totalmente configurables y se almacenan de forma ordenada convirtiéndose en una herramienta muy potente a la hora de analizar las necesidad de modificaciones en la base de conocimiento o en la propia aplicación
- **Pruebas automatizadas:** Una vez terminado el proyecto se dispone de una batería de pruebas automatizadas, la cual se ha detallado a lo largo del tema "Fase de Pruebas", que no solo permite verificar el cumplimiento de los requisitos de la base de conocimiento de manera rápida y eficaz sino que además proporciona la capacidad de conocer a simple vista si cualquier cambio realizado sobre la aplicación ha afectado al funcionamiento de la misma.

9.2.2 Requisitos no Funcionales

- **Sistema multiusuario:** Como se ha detallado en el apartado de diseño del intérprete, el uso de *Program D* como núcleo de la aplicación proporciona la capacidad de atender a varios usuarios de manera simultánea. Como característica añadida no sólo se pueden atender a varios usuarios a la vez sino que cada uno de esos usuarios podría interactuar con un bot diferente si fuese necesario.
- **Escalabilidad:** La implementación original de *Program D* presentaba ciertos problemas de escalabilidad debido a la forma en la que se

gestionaban las propiedades de cada usuario mediante su almacenamiento en ficheros en el propio sistema. Este problema fue resuelto mediante una modificación que permite almacenar dicha información en variables de sesión y liberar recursos una vez dicha sesión ha expirado.

- **Extensibilidad:** A lo largo de la definición de la arquitectura de la aplicación y del diseño se ha hecho hincapié en la independencia de los diferentes módulos que la componen. Esta propiedad se traduce en la posibilidad de mejorar y ampliar la aplicación sin que dichas modificaciones afecten apenas al resto de la misma, incluso, en la mayor parte de los casos, sólo requerirá interactuar con cierta interfaz pudiendo dejar intacto el código de la aplicación.
- **Amigable e intuitivo:** La subjetividad de este requisito hace que sea difícil de valorar el grado de cumplimiento del mismo, sin embargo ha estado presente durante todo el proceso de planificación y diseño. Así como los aspectos de configuración de la aplicación se han mantenido todo lo completos que ha sido posible, la interfaz con el usuario se ha tratado de simplificar únicamente a los elementos necesarios. Esto se puede comprobar en ambas interfaces desarrolladas donde toda la operativa es transparente al usuario, siendo únicamente necesario escribir una respuesta para recibir la respuesta correspondiente.
- **Compatibilidad:** El uso de java como lenguaje de programación del núcleo de la aplicación así como la independencia de cualquier elemento externo como pueda ser una base de datos, dota a la aplicación de una compatibilidad casi total, pudiéndose ejecutar sobre cualquier plataforma que disponga de un servidor J2EE.
- **Seguridad:** Al ser una aplicación diseñada para ser utilizada por el público es necesario presentar una serie de mecanismos de protección contra ataques. Se han intentado minimizar los riesgos mediante una serie de salvaguardas como puede ser la limpieza de la entrada de usuario para prevenir los ataques mediante la introducción de código a través de la interfaz de la aplicación o la mejora de la escalabilidad para minimizar los efectos de un ataque de saturación de peticiones. Aun así es posible que la mejor herramienta contra las acciones dañinas de los usuarios sea el registro de incidencias, que puede permitir detectar ataques y actuar en consecuencia.

9.2.3 Conclusiones

Como se puede comprobar en los apartados previos, se han cumplido todos los requisitos tanto funcionales como no funcionales, superando en algunos puntos exigencias definidas inicialmente. Por lo tanto puede considerarse que se ha completado el desarrollo del proyecto de manera exitosa.

9.3 Dificultades y Limitaciones

El proceso de desarrollo de la aplicación no ha estado exento, por supuesto, de dificultades, algunas de las cuales se han podido solventar mientras que otras han sido tan inevitables que han debido ser sorteadas convenientemente.

Probablemente la mayor dificultad de este proyecto ha sido la gran cantidad de tecnologías que abarca, como puede comprobarse en el tema "Descripción de Tecnologías Usadas". Si bien es cierto que no se ha necesitado un nivel de conocimiento muy elevado en algunas de ellas, ha sido necesario mucho esfuerzo para lograr dominarlas al nivel exigido. Como dificultad añadida cabe destacar que las tecnologías que requerían de un mayor nivel de conocimiento eran aquellas menos extendidas, y por lo tanto con menor cantidad de documentación disponible.

Un problema derivado de este último punto es que, con algunas tecnologías, como es el caso de AIML, no sólo ha sido difícil encontrar información sino que al constituir el intérprete de AIML el núcleo de la aplicación ha sido igualmente complicado encontrar una aplicación que cumpliese las expectativas necesarias. Incluso una vez encontrada dicha aplicación, la necesidad de realizar ciertas modificaciones para adaptarse por completo a los requisitos del proyecto ha supuesto tener que analizar y comprender el funcionamiento de un proyecto de la complejidad de *Program D*.

9.4 Trabajos Futuros

Una vez concluido el proyecto, y a pesar de que éste se ha considerado concluido con éxito conforme a los requisitos iniciales, pueden considerarse una serie de posibles mejoras o ampliaciones que podrían resultar en una aplicación más completa o incluso modificaciones que permitan adaptar la aplicación a otros ámbitos. Se detallarán a continuación algunas de estas modificaciones:

9.4.1 Ampliación de la Base de Conocimiento

Uno de los aspectos más importantes de un bot conversacional es la base de conocimiento, ya que cuanto más amplia sea, más realistas serán las conversaciones. Esto lleva a que una de las mejoras más importantes que se pueda llevar a cabo sobre el bot de ayuda sea la ampliación de la base de conocimiento, tanto en la profundidad de los temas ya conocidos como en la introducción de nuevos temas de conversación.

Para conseguir llevar a cabo esta mejora resulta de gran utilidad el registro de conversaciones ya que permite analizar diversas características de las mismas, como los temas que se suelen tratar más a menudo, cuales son las entradas para las que el bot no ha tenido una respuesta apropiada o aquellos que no se han contemplado en la base de conocimiento.

Dadas las características de esta mejora se puede comprobar que es un proceso continuado y que difícilmente puede considerarse completo por mucho esfuerzo invertido en él, ya que siempre habrá temas por añadir o información adicional que dar. Aún así cualquier modificación destinada a mejorar la experiencia del usuario debe ser tenida en cuenta.

9.4.2 Interfaz de Modificación de las Bases de Conocimiento

La importancia del proceso de ampliación y mejora de de la base de conocimiento da lugar a una serie de trabajos relacionados para facilitar este proceso. El más importante de ellos es la creación de una interfaz amigable para su modificación.

En la fase de diseño del bot se ha podido ver como la base de conocimiento se encuentra en forma de archivos AIML que pueden no resultar de fácil modificación por alguien

que no esté familiarizado con el lenguaje AIML y la arquitectura de la base de conocimiento. Es por ello muy interesante la implementación de una interfaz que permita la modificación de la base de conocimiento de manera sencilla e intuitiva.

Al ampliar el rango de aquellas personas con capacidad de modificar la base de datos, no sólo se consigue una mayor flexibilidad en el mantenimiento de la aplicación si no que se permite repartir el esfuerzo necesario para llevar a cabo la tarea de "Ampliación de la Base de Conocimiento" definida en el punto anterior.

Esta modificación está siendo realizada por Miguel Coronado Barrios *[Coro09]* dentro del Grupo de Sistemas Inteligentes de la Escuela Técnica Superior de Ingenieros de Telecomunicación.

9.4.3 Humanización de la Interfaz de Usuario

Como se ha explicado en el apartado "Trabajo Desarrollado" se ha intentado mantener la interfaz lo más sencilla e intuitiva que ha sido posible. Sin embargo está claro que para un ser humano el modo de comunicación más intuitivo no es mediante la introducción de textos a través de un teclado sino a través de comunicación oral. Asimismo aunque la recepción de la respuesta por parte del usuario es menos forzada que la petición, se podría mejorar la experiencia del usuario mediante una respuesta de voz. Ambas modificaciones no sólo conseguirían dar lugar a una aplicación mucho más humana sino que permitirían su utilización por parte de usuarios con ciertas discapacidades.

Es por ello que una de las posibles mejoras con respecto a la versión que se ha desarrollado en este proyecto sería la inclusión de una interfaz con reconocimiento y síntesis de voz.

10 Manual de Instalación

En el presente capítulo se explicarán los pasos a seguir para la correcta instalación del bot de ayuda sobre un servidor J2EE y los parámetros de configuración que soporta la aplicación.

10.1 Introducción

El bot de ayuda se suministra empaquetado en un archivo .war que incluye todos los componentes necesarios para su funcionamiento, de esta forma se consigue que sea sencilla su instalación sobre cualquier servidor J2EE, consistiendo únicamente en desplegar el archivo sobre el servidor deseado.

Dada la variedad de servidores J2EE que existen en la actualidad no es factible crear un manual de instalación para cada uno de ellos, sin embargo se darán los pasos a seguir para desplegar y arrancar la aplicación en un servidor *Tomcat 5.5*, con la intención de que sirvan de referencia de instalación en cualquier tipo de servidor.

10.2 Instalación y Configuración del servidor Web Tomcat 5.5

Como se ha comentado en la introducción, el bot de ayuda funciona desplegado sobre un servidor J2EE, por lo tanto el primer paso para poner en marcha la aplicación será instalar dicho servidor. En este apartado se explicará la instalación y configuración de un servidor Web *Tomcat 5.5*.

Hay que tener en cuenta que para el correcto funcionamiento del servidor Web *Tomcat 5.5* es necesario tener instalado previamente un entorno en tiempo de ejecución Java (JRE) en su versión 5.0 o superior.

10.2.1 Instalación

Los pasos a seguir para instalar y arrancar un servidor Web *Tomcat 5.5* son los siguientes:

1. **Obtención del archivo instalador del servidor:** Podemos obtener el archivo instalador del servidor Web *Tomcat 5.5* descargándolo de la página del propio servidor (http://tomcat.apache.org/) o del CD que se adjunta con el proyecto.

   Se dispone de tres versiones diferentes de instalador dependiendo de la plataforma y del tipo de instalación a realizar. Dos de estas versiones son genéricas para cualquier plataforma y se encuentran empaquetadas en archivos comprimidos (.zip y .tar.gz) mientras que la tercera es exclusiva de la plataforma Windows y viene en formato de archivo ejecutable (.exe).

2. **Instalación del servidor:** La elección del tipo de instalador (Archivo comprimido o ejecutable) dará lugar a dos procesos para la instalación del servidor.

   a. **Archivo comprimido:** En este tipo de instalación tendremos que descomprimir el contenido del archivo en el directorio deseado. Una vez descomprimido pasaremos a configurar las variables de entorno *JAVA_HOME* y *TOMCAT_HOME* mediante los siguientes comandos:

      - **Windows:**

        ```
        set TOMCAT_HOME = $TOMCAT_HOME
        set JAVA_HOME = $JAVA_HOME
        ```

      - **Linux:**

        ```
        export TOMCAT_HOME = $TOMCAT_HOME
        export JAVA_HOME = $JAVA_HOME
        ```

      En ambos casos $TOMCAT_HOME y $JAVA_HOME hacen referencia a los directorios raíz de instalación de tomcat y de java respectivamente, por ejemplo:

        ```
        set TOMCAT_HOME = C:\tomcat5.5\
        set JAVA_HOME = C:\JDK\
        ```

   b. **Archivo ejecutable:** Al lanzar el archivo en una plataforma Windows entraremos en un programa de instalación de tipo *wizard* que nos guiará a través de los pasos necesarios. En este tipo de instalación no es necesaria la configuración de variables ya que esta se realiza de manera automática.

3. **Arranque del servidor:** Al igual que en el caso de la instalación, el tipo de arranque viene determinado por el tipo de archivo instalador elegido:

   a. **Archivo comprimido:** Para arrancar y parar el servidor haremos uso de los *scripts* que se encuentran dentro del directorio "*bin*" dentro del directorio raíz de instalación. Así pues, definiendo de nuevo $TOMCAT_HOME como el directorio raíz de instalación de *Tomcat* tendríamos los siguientes comandos:

      - **Windows:**

        Arranque: `$TOMCAT_HOME\bin\startup`

        Parada: `$TOMCAT_HOME\bin\shutdown`

      - **Linux:**

        Arranque: `$TOMCAT_HOME/bin/startup.sh`

        Parada: `$TOMCAT_HOME/bin/shutdown.sh`

b. **Archivo ejecutable:** Durante la instalación del archivo ejecutable aparte de configurarse las variables de entorno de manera automática también se crean una serie de lanzadores en el menú de inicio de Windows. Así pues para arrancar nuestro servidor Web sólo tendremos que acceder al menú de inicio de Windows y una vez en el menú de *Tomcat* ejecutar "*Monitor Tomcat*". Esto hará aparecer un icono en la barra de tareas de Windows desde donde podremos realizar las diferentes acciones de arranque y parada del servidor.

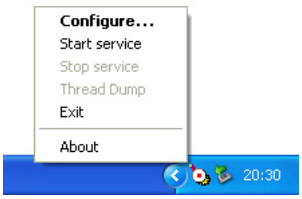

Ilustración 19: Opciones de la aplicación "Monitor Tomcat" sobre Windows

4. **Acceso a la Consola de Administración de Tomcat:** Para comprobar la correcta instalación y arranque del servidor Web accederemos a la consola de administración de *Tomcat* a través de un navegador Web. Para ello sólo tenemos que introducir la URL con la dirección de la máquina en la que está arrancado el servidor y el puerto de escucha. Para un acceso local y el puerto por defecto de *Tomcat* (8080) sería:

http://localhost:8080/

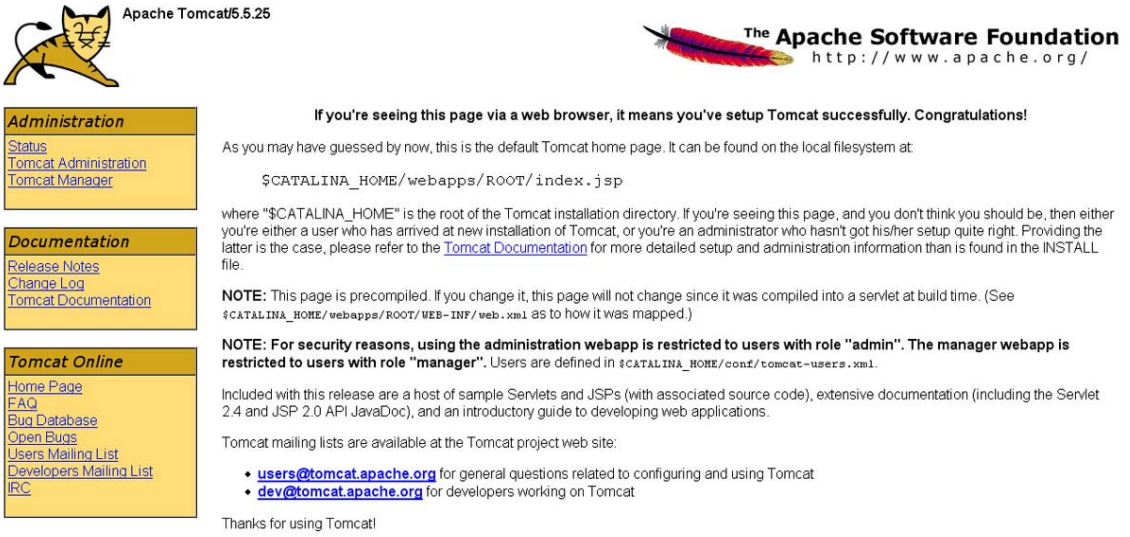

Ilustración 20: Consola de administración de Tomcat 5.5

10.2.2 Configuración

Por razones de seguridad no es posible que cualquier usuario pueda desplegar aplicaciones dentro del servidor, sino que tiene que tener los permisos necesarios para ello.

Salvo en el caso de la instalación mediante archivo ejecutable, en la que nos pide una contraseña de administrador durante los pasos de instalación, no se ha definido ningún usuario con permisos para poder desplegar aplicaciones, así que deberemos añadir uno. Para ello accederemos al directorio "*conf*" dentro del directorio raíz de instalación del servidor *Tomcat* y editaremos el archivo "tomcat-users.xml" que en un principio debería presentar un aspecto parecido al siguiente:

```
<tomcat-users>
      <user name="tomcat" password="tomcat" roles="tomcat" />
      <user name="role1" password="tomcat" roles="role1" />
      <user name="both" password="tomcat" roles="tomcat,role1" />
</tomcat-users>
```

Como puede observarse, por defecto no viene definido ningún usuario con permisos de gestor (*manager*) así que pasaremos a crear uno añadiendo una línea, de tal manera que el archivo quede:

```
<tomcat-users>
    <user name="tomcat" password="tomcat" roles="tomcat" />
    <user name="role1" password="tomcat" roles="role1" />
    <user name="both" password="tomcat" roles="tomcat,role1" />
    <user name="usuario" password="contraseña" roles="manager" />
</tomcat-users>
```

Donde "usuario" será el nombre del usuario que queremos que tenga permisos de gestión y "contraseña" será su contraseña asociada.

Problemas de disponibilidad de puertos

Es posible que los puertos que utiliza *Tomcat* por defecto para su funcionamiento estén ocupados previamente por otro proceso. Si se diese el caso *Tomcat* nos mostraría un mensaje de error informándonos de que no puede arrancarse el servidor.

Para solucionar este problema podemos terminar el proceso que está ocupando el puerto o bien modificar la configuración de *Tomcat* para que utilice otro puerto. Si optamos por la segunda opción deberemos editar el archivo "server.xml" que se encuentra dentro del directorio "*conf*" en el directorio raíz de instalación del servidor *Tomcat*, cambiando el puerto que está ocupado por otro que se encuentre libre, por ejemplo podríamos cambiar la línea:

```
<Server port="8005" shutdown="SHUTDOWN">
```

En la cual se puede ver que se está haciendo uso del puerto 8005 por la siguiente línea en la que se utilizará el puerto 8006 en su lugar:

```
<Server port="8006" shutdown="SHUTDOWN">
```

10.3 Instalación de la Aplicación sobre un servidor Tomcat

Para realizar la instalación de la aplicación sobre un servidor Web *Tomcat 5.5* y comprobar el correcto funcionamiento de la misma una vez instalado se han de seguir los siguientes pasos:

1. **Acceso al Gestor de Aplicaciones:** Una vez hemos accedido a la consola de Administración de *Tomcat* tal y como se ha explicado en el apartado de instalación del servidor *Tomcat 5.5*, seleccionamos la opción "*Tomcat Manager*" que aparece en el panel de administración del menú lateral.

Ilustración 21: Panel de administración de Tomcat 5.5

2. **Identificación en el Gestor de Aplicaciones:** Al seleccionar la opción "*Tomcat Manager*" nos aparecerá una ventana de identificación en la que tendremos que dar el nombre y *password* de un usuario con permisos de administración. Tras identificarnos llegaremos al Gestor de Aplicaciones.

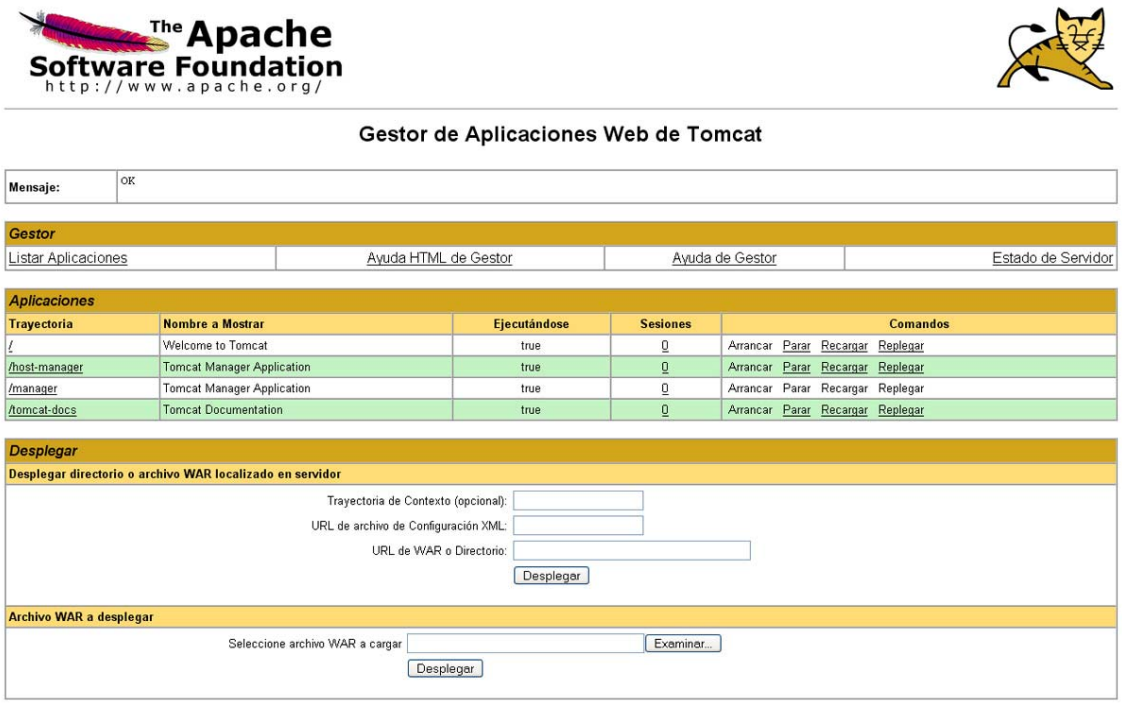

Ilustración 22: Gestor de aplicaciones de Tomcat 5.5

3. **Localización de Archivo de Aplicación:** En el Gestor de Aplicaciones tenemos dos opciones para desplegar una aplicación Web mediante un archivo WAR. En el primer caso el archivo está localizado en un servidor externo mientras que en el segundo se encuentra en la propia máquina. Eligiendo la segunda opción buscamos el archivo WAR con la aplicación en el sistema de directorios de la máquina pulsando en el botón "Examinar…".

Ilustración 23: Localización de archivo WAR en el panel de despliegue

4. **Despliegue de Aplicación:** Tras localizar el archivo a desplegar nos aparecerá la ruta del mismo en el cuadro de texto. Procedemos al despliegue pulsando en el botón "Desplegar".

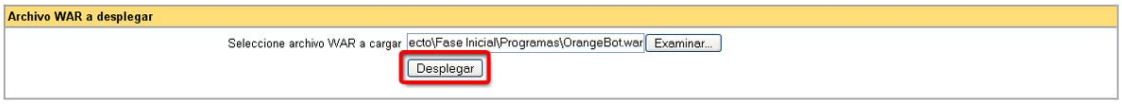

Ilustración 24: Despliegue de archivo WAR en el panel de despliegue

5. **Comprobación de Despliegue y Arranque de Aplicación:** Unos instantes después de pulsar el botón "Desplegar" volveremos a la pantalla del Gestor de Aplicaciones donde podremos comprobar si el despliegue se ha realizado de manera correcta. Para dicha comprobación disponemos de dos indicadores:

   a. **Mensaje de Despliegue:** En la parte superior de la consola en el apartado "Mensaje" debe aparecer "OK". Esto indica que el despliegue de la aplicación se ha realizado con éxito (Marcado en la imagen como 1).

   b. **Lista de Aplicaciones:** En el listado de aplicaciones desplegadas debe aparecer nuestra aplicación aparte de las que hubiese desplegadas previamente. Además se puede comprobar que además de haberse desplegado se ha iniciado correctamente la aplicación comprobando que en la columna "Ejecutándose" aparece el valor "*true*" y en la columna "Comandos" sólo se permiten las operaciones "Parar", "Recargar" y "Replegar" (Marcado en la imagen como 2).

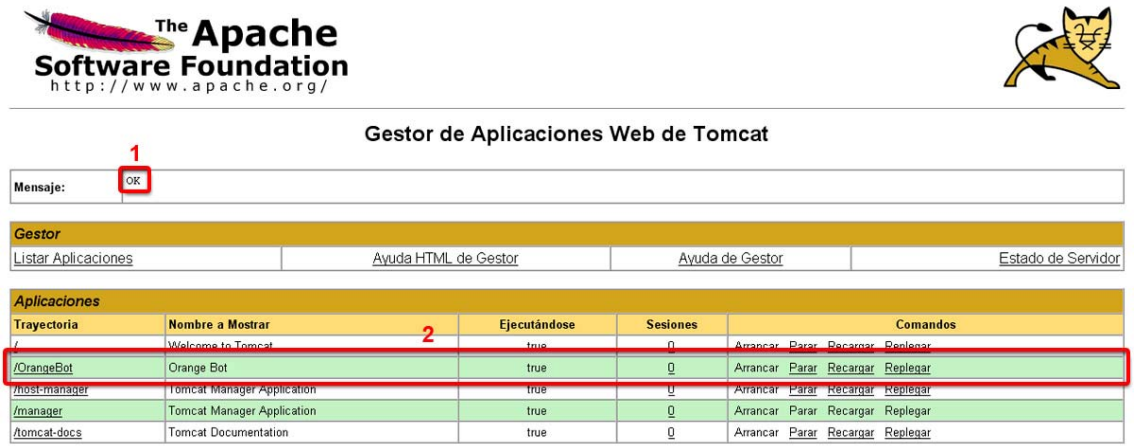

Ilustración 25: Comprobación de despliegue y arranque de la aplicación

6. **Comprobación del Funcionamiento de la Aplicación:** Una vez hemos desplegado y arrancado la aplicación podemos comprobar que la aplicación funciona de manera correcta. Para ello podemos bien pulsar en la trayectoria de nuestra aplicación en el listado de aplicaciones o bien desde un navegador introducir la dirección de la aplicación, que será la misma que utilizamos para entrar en la consola de administración de *Tomcat* añadiendo la trayectoria que nos aparece en la lista de de aplicaciones desplegadas. En este caso y suponiendo las mismas condiciones del punto 1 quedaría la siguiente URL para el caso de la interfaz Web:

   http://localhost:8080/OrangeBot

   Al acceder a la aplicación nos deberá aparecer una ventana tal y como se puede ver en la imagen.

Ilustración 26: Página de inicio de la aplicación desplegada

10.4 Configuración de la Aplicación

Aparte del modo de funcionamiento por defecto, la aplicación acepta la modificación de ciertos parámetros para variar la configuración inicial. En este apartado se explicará como modificar la configuración de la base de conocimiento y de los *logs*.

Hay que tener en cuenta que esta configuración requiere modificar archivos contenidos dentro del archivo .war, por lo tanto se requerirán las herramientas adecuadas para modificar dichos archivos sin cambiar la estructura del resto de la aplicación.

10.4.1 Configuración de la Base de Conocimiento

La base de conocimiento está estructurada en forma de temas contenidos en archivos AIML. Gracias a esta organización resulta sencillo añadir nuevos temas o quitar y modificar los temas existentes.

El archivo de configuración de la base de conocimiento es "bots.xml" que se encuentra dentro del directorio "*conf*" y que por defecto tiene el siguiente contenido:

```
<bot id="OrangeBot" enabled="true">
      <properties href="properties.xml"/>
      <predicates href="predicates.xml"/>
      <substitutions href="substitutions.xml"/>
      <sentence-splitters href="sentence-splitters.xml"/>
      <listeners href="listeners.xml"/>

      <learn>./aiml/comun/base.aiml</learn>
      <learn>./aiml/comun/reducciones.aiml</learn>
      <learn>./aiml/comun/verbos.aiml</learn>
      <learn>./aiml/comun/auxiliar.aiml</learn>
```

```
        <learn>./aiml/comun/auxiliarOrange.aiml</learn>
        <learn>./aiml/comun/saludos.aiml</learn>
        <learn>./aiml/comun/despedidas.aiml</learn>
        <learn>./aiml/comun/educacion.aiml</learn>
        <learn>./aiml/comun/conversacion.aiml</learn>

        <learn>./aiml/general.aiml</learn>
        <learn>./aiml/movil.aiml</learn>
        <learn>./aiml/fijo.aiml</learn>
        <learn>./aiml/internet.aiml</learn>
        <learn>./aiml/television.aiml</learn>
</bot>
```

Como puede observarse cada tema de conocimiento está representado por la ubicación del archivo AIML dentro de una etiqueta <*learn*> por lo que cualquier cambio que se quiera hacer en la base de conocimiento se realizará modificando dichas líneas.

Para quitar un tema de conocimiento eliminaremos la línea en la que se incluye dicho tema. Por ejemplo, si queremos eliminar el tema de telefonía móvil quitaremos la línea:

```
        <learn>./aiml/movil.aiml</learn>
```

Para añadir un nuevo tema de conocimiento tendremos que añadir la ruta del archivo al archivo de configuración, bien como ruta absoluta si queremos que el archivo se encuentre fuera de la aplicación, bien como ruta relativa si preferimos guardar el archivo junto con el resto de temas de la base de conocimiento (En el directorio “aiml” de la aplicación). Por ejemplo si queremos añadir el tema de conocimiento “cine” podríamos añadir cualquiera de las siguientes líneas dependiendo de si queremos dar una ruta absoluta o una relativa:

```
        Ruta absoluta:  <learn>/var/aiml/cine.aiml</learn>

        Ruta relativa:  <learn>./aiml/cine.aiml</learn>
```

En ambos casos el archivo “cine.aiml” será un archivo AIML que contenga los conocimientos deseados sobre el tema.

Hay que tener en cuenta que el añadir nuevos temas puede dar lugar a inconsistencias en la base de conocimiento, así que es recomendable que estos cambios

se realicen con mucho cuidado y se realicen las pruebas automatizadas oportunas una vez realizados para comprobar el correcto funcionamiento del bot.

10.4.2 Configuración de Logs

Al igual que en el caso de la configuración de la base de conocimiento existe un archivo XML para la configuración de *logs*, este archivo es "log4j.xml" y está localizado en la ruta "\WEB-INF\classes".

Este archivo ofrece la posibilidad de configurar tanto qué eventos almacenar en los *logs* como el formato de los mismos o incluso variables más específicas como el tamaño máximo que pueden ocupar.

Dado que los parámetros de configuración de los *logs* son los habituales de log4j no entraremos en profundidad y nos centraremos en como cambiar la ruta en la que se almacenan los *logs*.

Los *logs* del bot de ayuda se almacenan por defecto en la ruta "/var/log/orangebot/", si queremos modificar esa ruta tendremos que cambiarlo en el archivo de configuración. Por ejemplo, podemos ver como el *log* de actividades tiene la siguiente configuración por defecto:

```
<!--The main activity log file-->
<appender name="activitylog"
      class="org.apache.log4j.RollingFileAppender">
      <param name="File" value="/var/log/orangebot/activity.log"/>
      <param name="MaxFileSize" value="10MB"/>
      <param name="MaxBackupIndex" value="10"/>
      <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="[%d{ISO8601}] %p:
                  %m%n"/>
      </layout>
</appender>
```

Para cambiar la ruta en la que se almacenarán los *logs* deberemos cambiar la ruta que aparece dentro del parámetro de nombre "File", así podríamos escribir:

```
<param name="File" value="/logs/erika/activity.log"/>
```

Hay que tener en cuenta que tendremos que cambiar la ruta de todos los archivos de log que queramos modificar y que el directorio en el que queremos guardar los *logs* debe tener permisos de escritura para la aplicación.

11 Apéndices

11.1 Temas de Conversación

En este apartado se pretende definir un conjunto de temas de conversación que conformarán el conocimiento de un bot interactivo cuya función será ofrecer ayuda a los usuarios sobre los servicios ofrecidos por Orange.

Para ello se expone en cada una de las áreas de conocimiento del bot una serie de conversaciones tipo, siendo capaz el bot de responder a variaciones de dichas conversaciones.

Fuente de información:

- http://ayuda.orange.es/movil/
- http://ayuda.orange.es/internet/
- http://ayuda.orange.es/telefonia_fija/
- http://ayuda.orange.es/tv/

Todos los *topics* de los temas relacionados con Orange comienzan con una de las siguientes palabras:

- **Orange:** Temas relacionados con Orange en general.
- **Pregunta:** Preguntas de especificación de servicio en temas comunes.
- **Movil:** Temas relacionados con telefonía móvil.
- **Fijo:** Temas relacionados con telefonía fija.
- **Internet:** Temas relacionados con Internet.
- **Televisión:** Temas relacionados con televisión.

Las siguientes palabras del *topic* sirven para especificar el tema tratado dentro de cada una de esas categorías.

11.1.1 Temas Comunes a Varios Servicios

ID	Usuario	Topic
General		
T.1.1.1	o Orange o ¿Orange opera sólo en España? o ¿Orange es más barata que el resto de compañías?	Orange Indefinido
T.1.1.2	o ¿Qué es Orange? o Dime qué es Orange o Me podrías explicar qué es Orange	Orange Indefinido
Contratación y uso		
T.1.2.1	o Contratar o Contratar servicio o Quiero darme de alta	Pregunta Contratar
T.1.2.2	o Recargar o Recarga o ¿Cómo puedo recargar?	Pregunta Recargar
T.1.2.3	o Contrato o Ver Contrato o ¿Dónde puedo ver mi contrato?	Pregunta Contrato
T.1.2.4	o Factura o Facturación o Infórmame sobre la facturación	Pregunta Factura
T.1.2.5	o PIN o ¿Qué es el PIN? o Quiero información sobre el código PIN	Pregunta PIN

T.1.2.6	o Mensaje o Quiero mandar un mensaje o ¿Cómo mando un mensaje?	Pregunta Mensaje
Tarifas y Descuentos		
T.1.3.1	o Tarifa o Tarifas o Quiero información sobre tarifas	Pregunta Tarifas
T.1.3.2	o Bonos o Información acerca de bonos o ¿Para qué sirven los bonos?	Pregunta Bonos
Servicios		
T.1.4.1	o Buzón de voz o Buzón de teléfono o Servicio buzón de voz	Pregunta Servicio BuzonVoz
T.1.4.2	o Desvío de llamada o Servicio de desvío de llamadas o ¿Cómo se activa el desvío de llamadas?	Pregunta Servicio DesvioLlamadas
T.1.4.3	o Identificación de llamadas o Servicio de identificación de llamadas o Dame información sobre la identificación de llamadas	Pregunta Servicio IdentificacionLlamadas
T.1.4.4	o Ocultar numero o Ocultación de llamada saliente o ¿Cómo puedo ocultar mi número de teléfono al realizar una llamada?	Pregunta Servicio OcultarNumero

T.1.4.5	o Portabilidad o Conservar número o ¿Puedo conservar mi número al pasar a Orange?	Pregunta Servicio Portabilidad

11.1.2 Telefonía Móvil

ID	Usuario	Topic
General		
T.2.1.1	o Móvil o Móviles o Teléfono móvil o Telefonía móvil o Información sobre móvil o Información sobre telefonía móvil	Movil
T.2.1.2	o Recomienda tarifa móvil o ¿Qué tarifa de móviles mejor? o ¿Qué tarifa de móvil me recomiendas? o ¿Me podrías recomendar una tarifa para móvil?	Movil Tarifas
T.2.1.3	o Bonos móvil o Información acerca de bonos para móvil o ¿Para qué sirven los bonos de teléfono móvil?	Movil Bonos
T.2.1.4	o Contratar móvil o Contratar telefonía móvil o Quiero contratar el teléfono móvil con Orange	Movil Contratar

T.2.1.5	o Detalle de llamadas o Consulta detallada de llamadas o ¿Cómo veo el detalle de mis llamadas?	Movil DetalleLlamadas
T.2.1.6	o ¿Qué es el PIN del móvil? o ¿Qué es el código PIN del móvil? o Quiero información sobre el código PIN de mi móvil	Movil PIN
T.2.1.7	o ¿Qué es el PUK? o ¿Qué es el código PUK? o Quiero información sobre el código PUK	Movil PUK
Tarjeta		
T.2.2.1	o Tarjeta móvil o Móvil de tarjeta o Teléfono de tarjeta	Movil Tarjeta
T.2.2.2	o Recargar tarjeta móvil o ¿Cómo puedo recargar la tarjeta de mi móvil? o Quiero recargar mi tarjeta de móvil	Movil Tarjeta Recargar
T.2.2.3	o Consultar saldo o Ver saldo o ¿Cómo puedo ver mi saldo?	Movil Tarjeta VerSaldo
T.2.2.4	o Cambiar tarifa o Cambio de tarifa o ¿Cómo puedo cambiar de tarifa?	Movil Tarjeta CambiarTarifa
T.2.2.5	o ¿Cuánto dura mi tarjeta de móvil? o ¿Hasta cuándo es válida mi tarjeta de móvil? o ¿Cuándo caduca la tarjeta de mi móvil?	Movil Tarjeta Caducar

T.2.2.6	o Paso a contrato o Pasar a contrato o ¿Cómo puedo pasar a contrato?	Movil Tarjeta PasarAContrato
Contrato		
T.2.3.1	o Contrato móvil o Móvil de contrato o Teléfono de contrato	Movil Contrato
T.2.3.2	o Consulta factura móvil o Información sobre facturación en móvil o ¿Cómo puedo ver la factura de mi móvil?	Movil Contrato Facturacion
T.2.3.3	o Cambiar número o Cambiar número de teléfono o ¿Puedo cambiar mi número de teléfono?	Movil Contrato CambiarNumero
PostVenta		
T.2.4.1	o Teléfono móvil no funciona o Teléfono móvil roto o Mi teléfono móvil no funciona bien ¿Puedes ayudarme?	Movil Problema
T.2.4.2	o He perdido mi tarjeta de móvil o Me han robado la tarjeta del móvil o La tarjeta de mi móvil está estropeada o La tarjeta de mi móvil no funciona	Movil Problema Tarjeta
T.2.4.3	o Móvil de sustitución o Móvil durante reparación o ¿Que hago mientras están reparando mi móvil?	Movil Reparar

T.2.4.4	o ¿Cuándo estará arreglado mi teléfono? o ¿Cuánto tarda la reparación? o ¿Cómo sé cuando está arreglado mi teléfono?	Movil Reparar
T.2.4.5	o ¿Dónde hay un punto de servicio de postventa? o ¿Donde hay un servicio de asistencia? o ¿Cuál es el servicio de postventa más cercano?	Movil Postventa
Configuración de Móvil		
T.2.5.1	o Configurar móvil o Ayuda para configurar móvil o No sé configurar mi móvil ¿Puedes ayudarme?	Movil Configurar
T.2.5.2	o Configurar móvil con un SMS o Ayuda para configurar móvil mediante SMS o No sé configurar mi móvil mediante SMS ¿Puedes ayudarme?	Movil Configurar SMS
T.2.5.3	o Configurar móvil en Internet o Ayuda para configurar móvil a través de Internet o No sé configurar mi móvil con Internet ¿Puedes ayudarme?	Movil Configurar Internet
Llamadas Internacionales		
T.2.6.1	o Tarjeta Internacional o Tarjeta llamadas extranjero o ¿Qué es la tarjeta internacional?	Movil Internacional Tarjeta
T.2.6.2	o Comprar tarjeta internacional o Adquirir tarjeta internacional o ¿Dónde puedo comprar la tarjeta internacional?	Movil Internacional Tarjeta

T.2.6.3	o Llamar con tarjeta internacional o Llamar desde el extranjero con tarjeta internacional o ¿Cómo puedo llamar con la tarjeta internacional?	Movil Internacional Tarjeta
T.2.6.4	o Consulta saldo de tarjeta internacional o Consultar el saldo de una tarjeta internacional o ¿Cómo puedo consultar el saldo de mi tarjeta internacional?	Movil Internacional Tarjeta
T.2.6.5	o Recargar tarjeta internacional o Recarga de tarjeta internacional o ¿Cómo puedo recargar mi tarjeta internacional?	Movil Internacional Tarjeta
T.2.6.6	o ¿Cuánto dura mi tarjeta internacional? o ¿Hasta cuándo es válida mi tarjeta internacional? o ¿Cuándo caduca mi tarjeta internacional?	Movil Internacional Tarjeta
T.2.6.7	o Roaming o Roaming internacional o ¿Qué es roaming?	Movil Internacional Roaming
T.2.6.8	o Precio roaming o Precio llamadas desde extranjero o ¿Cuánto cuestan las llamadas desde el extranjero?	Movil Internacional Roaming
T.2.6.9	o SMS extranjero o Mandar SMS roaming o ¿Puedo mandar SMS desde el extranjero?	Movil Internacional Roaming
T.2.6.10	o Llamar a España desde el extranjero o Llamar a España roaming o ¿Cómo puedo llamar a España desde el extranjero?	Movil Internacional Roaming

T.2.6.11	o Llamar extranjero o Llamadas extranjero o ¿Puedo realizar llamadas desde otro país?	Movil Internacional Roaming
T.2.6.12	o Recargar en el extranjero o Recargar en otro país o ¿Puedo recargar mi tarjeta en el extranjero?	Movil Internacional Recargar
Mensajes		
T.2.7.1	o SMS o Mensaje Corto o Mensaje de Texto	Movil Mensaje Corto
T.2.7.2	o MMS o Mensajes con imágenes o ¿Cómo puedo mandar mensajes con imágenes?	Movil Mensaje Multimedia
T.2.7.3	o Mensajes de voz o Mandar mensaje de voz o ¿Cómo puedo mandar mensajes de voz?	Movil Mensaje Voz
Multimedia		
T.2.8.1	o Internet móvil o Internet en el móvil o ¿Cómo puedo acceder a internet desde mi móvil?	Movil Internet
T.2.8.2	o Correo en el móvil o Ver correo en el móvil o ¿Cómo puedo acceder a mi correo electrónico desde el móvil?	Movil Correo

T.2.8.3	o Vídeo llamada o Videollamada o ¿Cómo puedo realizar una videollamada?	Movil Videollamada
T.2.8.4	o Televisión móvil o Ver la televisión en el móvil o ¿Cómo puedo ver la televisión en mi móvil?	Movil Television
T.2.8.5	o Juegos para móvil o Videojuegos para móvil o Dame información sobre juegos para móvil	Movil Juegos
T.2.8.6	o Música para móvil o Canciones para móvil o ¿Dónde puedo descargar canciones para mi móvil?	Movil Musica
T.2.8.7	o Tono móvil o Tonos para móvil o ¿Dónde puedo descargar tonos para mi móvil?	Movil Tonos
Servicios		
T.2.9.1	o Buzón de voz móvil o Activar buzón de voz de móvil o ¿Cómo puedo escuchar los mensajes de mi buzón de voz en el móvil?	Movil Servicio BuzonVoz
T.2.9.2	o Desvío móvil o Servicio de desvío de llamadas de teléfono móvil o ¿Cómo se activa el desvío de llamadas de móvil?	Movil Servicio DesvioLlamadas

T.2.9.3	o Identificación de llamadas móvil o Servicio de identificación de llamadas de móvil o Dame información sobre la identificación de llamadas para móvil	Movil Servicio IdentificacionLlamadas
T.2.9.4	o Llamada oculta móvil o Número oculto de móvil o ¿Cómo puedo ocultar el número de mi móvil en una llamada?	Movil Servicio OcultarNumero
T.2.9.5	o Portabilidad o Portar número o ¿Puedo conservar mi número al pasar a Orange?	Movil Servicio Portabilidad
T.2.9.6	o Orange World o ¿Qué es Orange World? o ¿Me puedes decir en qué consiste Orange World?	Movil Servicio OrangeWorld
T.2.9.7	o Tarjeta ECO o ¿Qué es la tarjeta ECO? o ¿Cuánto cuesta la tarjeta ECO?	Movil Servicio TarjetaECO
T.2.9.8	o Álbum de fotos o Álbum de fotos en el móvil o ¿Me puedes dar información sobre el servicio álbum de fotos?	Movil Servicio Album
T.2.9.9	o Pagar con el móvil o Dime como pagar con el móvil o ¿Cómo puedo pagar con el móvil?	Movil Servicio PagoConMovil
T.2.9.10	o Mobipay o ¿Qué es mobipay? o ¿Me podrías decir qué es mobipay?	Movil Servicio Mobipay

T.2.9.11	o Servicios de voz o Dime los servicios de voz o ¿Me puedes decir qué servicios de voz hay?	Movil Servicio Voz
T.2.9.12	o Afinidad o Servicio afinidad o ¿Me puedes dar información sobre el Servicio afinidad?	Movil Servicio Afinidad
T.2.9.13	o Ánimo o Estado de ánimo o ¿En qué consiste el servició mi ánimo?	Movil Servicio Animo

11.1.3 Telefonía Fija

ID	Usuario	Topic
General		
T.3	o Fijo o Teléfono fijo o Telefonía fija o Información sobre telefonía fija	Fijo
Contratación y Condiciones		
T.3.1.1	o Contratar fijo o Contratar telefonía fija o Quiero contratar el teléfono fijo con Orange	Fijo Contratar
T.3.1.2	o Alta teléfono fijo o Quiero darme de alta en telefonía fija o ¿Cómo doy de alta el teléfono fijo con Orange?	Fijo Contratar

T.3.1.3	o Factura fijo o Facturación fijo o Infórmame sobre la facturación en telefonía fija	Fijo Factura
T.3.1.4	o Contrato Fijo o Contrato de telefonía fija o Quiero ver el contrato de telefonía fija	Fijo Contrato
Tarifas y descuentos		
T.3.2.1	o Tarifas fijo o ¿Qué tarifas hay para fijo? o ¿Me puedes recomendar una tarifa para telefonía fija?	Fijo Tarifas
T.3.2.2	o Plan de tarifas o ¿Qué es un plan de tarifas? o ¿Qué plan de tarifas es el que más me conviene?	Fijo Tarifas Plan
T.3.2.3	o Plan hogar o ¿Qué es el plan hogar? o Quiero información sobre el plan hogar	Fijo Tarifas Plan
T.3.2.4	o Bonos fijo o Información acerca de bonos para fijo o ¿Para qué sirven los bonos de teléfono fijo?	Fijo Bonos
T.3.2.5	o Tarifa plana fijo o Información acerca de la tarifa plana para fijo o ¿En qué consiste la tarifa plana de telefonía fija?	Fijo TarifaPlana

Servicio indirecto y Marcación Directa		
T.3.3.1	o 1052 o ¿Qué es el prefijo 1052? o ¿Para qué sirve el prefijo 1052?	Fijo 1052
T.3.3.2	o Marcación directa o ¿Cómo funciona la marcación directa? o ¿Cuándo estará disponible la marcación directa?	Fijo MarcacionDirecta
T.3.3.3	o Qué es marcación directa o ¿Qué es la marcación directa? o ¿Me podrías explicar qué es la marcación directa?	Fijo MarcacionDirecta Descripcion
T.3.3.4	o Servicio indirecto o Información sobre servicio indirecto o ¿Me puedes dar información sobre el acceso indirecto?	Fijo ServicioIndirecto
T.3.3.5	o Qué es servicio indirecto o ¿Qué es el servicio indirecto? o ¿Me podrías explicar qué es el acceso indirecto?	Fijo ServicioIndirecto Descripcion
Servicios de Telefonía		
T.3.4.1	o Buzón de voz fijo o Servicio buzón de voz de fijo o ¿Para qué sirve el buzón de voz para fijo?	Fijo Servicio BuzonVoz
T.3.4.2	o Desvío fijo o Servicio de desvío de llamadas de teléfono fijo o ¿Cómo se activa el desvío de llamadas de fijo?	Fijo Servicio DesvioLlamadas

T.3.4.3	o Identificación de llamadas fijo o Servicio de identificación de llamadas de fijo o Dame información sobre la identificación de llamadas para fijo	Fijo Servicio IdentificacionLlamadas
T.3.4.4	o Ocultar número fijo o Ocultación de llamada saliente de fijo o ¿Cómo puedo ocultar mi número de teléfono al realizar una llamada con el teléfono fijo?	Fijo Servicio OcultarNumero
T.3.4.5	o Todo en 1 o Todo en uno de Orange o ¿Qué es todo en 1 de Orange?	Fijo Servicio TodoOrange
T.3.4.6	o Portabilidad fijo o Conservar número de fijo o ¿Puedo conservar mi número del teléfono fijo al pasar a Orange?	Fijo Servicio Portabilidad

11.1.4 Internet

ID	Usuario	Topic
General		
T.4.1.1	o Internet o Información Internet o Quiero que me des información acerca de Internet	Internet
T.4.1.2	o Configurar Internet o ¿Cómo se configura la línea de Internet? o Necesito configurar la conexión de Internet	Internet Configurar

T.4.1.3	o Instalar Internet o ¿Cómo se instala Internet? o Dime cómo instalar la conexión a Internet	Internet Instalar
T.4.1.4	o Problema Internet o No me funciona Internet o Mi conexión a Internet no funciona bien ¿Puedes ayudarme?	Internet Problema
T.4.1.5	o Tarifa plana Internet o Dame información sobre la tarifa plana de Internet o ¿Qué es la tarifa plana de Internet?	Internet Tarifa Plana
T.4.1.6	o Correo Internet o Correo electrónico o ¿Cómo accedo a mi cuenta de correo a través de Internet?	Internet Correo
Contratación y Condiciones		
T.4.2.1	o Contratar Internet o ¿Cómo contratar Internet? o Dime como puedo contratar Internet con Orange	Internet Contratar
T.4.2.2	o Alta Internet o ¿Cómo me doy de alta en Internet? o Dime como puedo darme de alta en Internet con Orange	Internet Contratar
T.4.2.3	o Contrato Internet o Contrato de Internet o Quiero ver el contrato de Internet	Internet Contrato

T.4.2.4	o Factura Internet o Facturación Internet o Quiero información sobre mi factura de Internet	Internet Factura
Internet ADSL		
T.4.3.1	o ADSL o Información ADSL o Quiero que me des información acerca de ADSL	Internet ADSL
T.4.3.2	o Contratar ADSL o ¿Cómo contratar ADSL? o Dime como puedo contratar ADSL con Orange	Internet ADSL Contratar
T.4.3.3	o Alta ADSL o ¿Cómo me doy de alta en ADSL? o Dime como puedo darme de alta en ADSL con Orange	Internet ADSL Contratar
T.4.3.4	o Configurar ADSL o ¿Cómo se configura la línea ADSL? o Necesito configurar la conexión de ADSL	Internet ADSL Configurar
T.4.3.5	o Instalar ADSL o ¿Cómo se instala ADSL? o Dime cómo instalar la conexión ADSL	Internet ADSL Instalar
T.4.3.6	o Router o Router ADSL o ¿Puedes darme información sobre mi router WiFi?	Internet ADSL Router

T.4.3.7	o Driver o Descargar drivers o ¿Dónde puedo descargar los drivers de mi router?	Internet ADSL Driver
T.4.3.8	o Manual o Descargar manuales o ¿Dónde puedo descargar los manuales de ADSL?	Internet ADSL Manual
Internet Básico		
T.4.4.1	o Internet básico o ¿Qué es Internet básico? o Quiero información sobre la conexión de Internet básica	Internet Basico
T.4.4.2	o Configurar Internet básico o ¿Cómo se configura Internet básico? o Necesito configurar la conexión de Internet básica	Internet Basico Configurar
T.4.4.3	o Instalar Internet básico o ¿Cómo se instala Internet básico? o Dime cómo instalar la conexión de Internet básica	Internet Basico Instalar
T.4.4.4	o Programa de conexión o Descargar programa de conexión o ¿Dónde puedo descargar el programa de conexión?	Internet Basico ProgramaConexion
Servicios para Todos		
T.4.5.1	o Barra Orange o ¿Cómo se usa la barra de orange? o ¿Cómo puedo instalar la barra de orange en mi navegador?	Internet Servicio BarraOrange

T.4.5.2	o Blogs o Servicio de blogs o ¿Qué son los blogs de Orange?	Internet Servicio Blogs
T.4.5.3	o Buscador o Servicio buscador o ¿Qué es el servicio de buscador?	Internet Servicio Buscador
T.4.5.4	o Directorio o Servicio directorio o ¿En qué consiste el servicio de directorio?	Internet Servicio Directorio
T.4.5.5	o Chat o Servicio chat o ¿Para qué sirve el chat?	Internet Servicio Chat
T.4.5.6	o Clasificados o Servicio clasificados o ¿Qué son los clasificados?	Internet Servicio Clasificados
T.4.5.7	o Foros o Servicio de foros o ¿Qué son los foros?	Internet Servicio Foros
T.4.5.8	o Internet servicio fotos o ¿Cómo funciona el servicio fotos de internet? o ¿En qué consiste el servicio fotos para internet?	Internet Servicio Fotos
T.4.5.9	o Desktop Search o Servicio Desktop Search o ¿Qué es Desktop Search?	Internet Servicio DesktopSearch
T.4.5.10	o Postales o Servicio de postales o ¿En qué consiste el servicio de postales?	Internet Servicio Postales

T.4.5.11	o Web personal o Página personal o Quiero información sobre el servicio de página personal	Internet Servicio WebPersonal
T.4.5.12	o SMS Internet o Mandar SMS por Internet o ¿Cómo puedo mandar un SMS a través de Internet?	Internet Servicio SMS
Servicios para Clientes de Orange		
T.4.6.1	o Acelerador o Servicio acelerador o Quiero información sobre el servicio acelerador	Internet Servicio Acelerador
T.4.6.2	o Antivirus o Servicio antivirus o ¿Me podrías dar información sobre el servicio antivirus?	Internet Servicio Antivirus
T.4.6.3	o Control parental o Servicio de control parental o ¿En qué consiste el servicio de control parental?	Internet Servicio ControlParental
T.4.6.4	o Backup o Servicio de backup o Necesito información sobre el servicio de backup	Internet Servicio Backup
T.4.6.5	o 4x4 o Servicio 4x4 o ¿Me podrías dar información sobre el servicio 4x4?	Internet Servicio 4x4

11.1.5 Televisión

ID	Usuario	Topic
General		
T.5	o TV o Tele o Televisión o Televisión Orange	Television
Orange TV		
T.5.1.1	o Orange TV o Orange Televisión o Información Orange TV	Television OrangeTV
T.5.1.2	o ¿Qué es Orange televisión? o ¿Qué es Orange TV? o ¿Puedes decirme qué es Orange TV?	Television OrangeTV Descripcion
Televisión Digital Terrestre (TDT)		
T.5.2	o TDT o Televisión Digital Terrestre o ¿Cómo puedo ver la TDT en mi descodificador?	Television TDT
Guía de Instalación		
T.5.3.1	o Contratar TV o Contratar televisión o Quiero contratar Orange TV	Television Contratar

T.5.3.2	o Alta TV o Quiero darme de alta en Orange TV o ¿Cómo me doy de alta en la televisión?	Television Contratar
T.5.3.3	o Contrato TV o Contrato televisión o Contrato Orange TV	Television Contrato
T.5.3.4	o Factura televisión o Facturación tv o Quiero información sobre mi factura de televisión	Televisión Factura
T.5.3.5	o Instalación TV o Instalar televisión o Guía de instalación de televisión o ¿Cómo puedo instalar Orange TV?	Television Instalar
T.5.3.6	o Configurar TV o ¿Cómo se configura la televisión? o No sé configurar la televisión	Television Configurar
T.5.3.7	o Usar TV o ¿Cómo se usa la televisión? o Quiero información de uso de la televisión de Orange	Television Usar
T.5.3.8	o Recargar TV o ¿Cómo recargo la televisión? o Dime como se recarga la cuenta de televisión	Television Recargar
T.5.3.9	o Descodificador o Necesito ayuda con el descodificador o Dame información sobre el descodificador, por favor	Television Descodificador

T.5.3.10	o Cómo instalar descodificador o ¿Cómo se instala el descodificador? o ¿Me podrías decir como instalo el descodificador?	Television Descodificador Instalar
T.5.3.11	o Euroconector o ¿Cómo conecto el euroconector? o ¿Qué es el euroconector?	Television Euroconector
T.5.3.12	o Indicadores luminosos descodificador o ¿Qué significa cada indicador luminoso del descodificador? o ¿Qué quiere decir una luz roja fija en el descodificador?	Television Descodificador Luces
Manual de Usuario		
T.5.4.1	o Canal o Canales o Información de canal	Television Canales
T.5.4.2	o Canal de TV o Canales de TV o ¿Qué canales de TV puedo ver?	Television Canales TV
T.5.4.3	o Mosaico o Visualización en mosaico o ¿Cómo activo el modo mosaico?	Television Mosaico
T.5.4.4	o Miniguía o Función de miniguía o ¿Para qué sirve la miniguía?	Television Miniguia
T.5.4.5	o Programas de televisión o Programación de televisión o ¿Dónde puedo ver la programación de los diferentes canales de televisión?	Television ProgramasTV

T.5.4.6	o Mando a distancia o Mando de la tele o ¿Cómo funciona el mando a distancia?	Television MandoDistancia
T.5.4.7	o Número de identificación de cliente o ¿Qué es el número de identificación de cliente? o ¿Dónde encuentro el número de identificación de cliente?	Television IdentificacionCliente
T.5.4.8	o Cuenta familiar o ¿Qué es la cuenta familiar? o ¿Para qué sirve la cuenta familiar?	Television CuentaFamiliar
T.5.4.9	o Cuenta parental o ¿Qué es la cuenta parental? o ¿Para qué sirve la cuenta parental?	Television CuentaParental
T.5.4.10	o Código adulto o ¿Para qué sirve el código adulto? o ¿Cómo puedo cambiar el código adulto?	Television CodigoAdulto
T.5.4.11	o Qué es código adulto o ¿Qué es el código adulto? o ¿Me puedes explicar qué es el código adulto?	Television CodigoAdulto Descripcion
T.5.4.12	o Código PIN TV o PIN de televisión o ¿Qué es el código PIN de televisión?	Television PIN
T.5.4.13	o Código confidencial de servicio o ¿Qué es el código confidencial de servicio? o ¿Dónde está el código confidencial de servicio?	Television CodigoConfidencial
T.5.4.14	o Control Parental o ¿Qué es el control parental? o ¿Para qué sirve el control parental?	Television ControlParental

T.5.4.15	o Distribución de saldo o ¿Qué es la distribución de saldo? o ¿En qué consiste la distribución de saldo?	Television DistribucionSaldo
Servicios de Televisión		
T.5.5.1	o Videoclub o Servicio de videoclub o ¿Cómo puedo acceder al servicio de videoclub?	Television Videoclub
T.5.5.2	o Menú videoclub o Información menú videoclub o ¿Cómo funciona el menú del videoclub?	Television Videoclub Menu
T.5.5.3	o Catálogo videoclub o Quiero ver el catálogo del videoclub o ¿Qué opciones tengo en el catálogo del videoclub?	Television Videoclub Catalogo
T.5.5.4	o Videos del videoclub o Películas del videoclub o ¿Qué películas hay en el servicio de videoclub?	Television Videoclub Catalogo
T.5.5.5	o Lista de películas o ¿Cómo puedo ver la lista de películas? o Dime la lista de películas	Television Videoclub Catalogo
T.5.5.6	o Catálogo de películas o ¿Cómo puedo ver el catálogo de videos? o Dame el catálogo de películas	Television Videoclub Catalogo
T.5.5.7	o Comprar un vídeo o ¿Cómo puedo comprar un vídeo? o ¿Qué películas puedo comprar?	Television Videoclub Comprar

T.5.5.8	o Alquilar un vídeo o ¿Cómo puedo alquilar un vídeo? o ¿Qué películas puedo alquilar?	Televisión Videoclub Comprar
T.5.5.9	o Buscar vídeo o ¿Cómo puedo buscar un vídeo? o Quiero buscar una película	Televisión Videoclub Buscar
T.5.5.10	o Ver video o Ver películas o ¿Cómo se ven los vídeos que he comprado?	Televisión Videoclub Ver
T.5.5.11	o Grabar vídeo o Grabar películas o ¿Se pueden grabar las películas que compro?	Televisión Videoclub Grabar

11.1.6 Temas de Conversación no Relacionados con Orange

Los *topics* de todos los temas no relacionados con orange comienzan con la palabra "General". La segunda palabra sirve para caracterizar el tema a tratar y el resto sirven para concretar aún más dentro del tema tratado. Dado el creciente número de temas a tratar por el bot, se irán añadiendo nuevos términos a la segunda palabra del *topic* y posteriores según se vaya ampliando la base de conocimiento.

ID	Usuario	Topic
Saludos		
T.6.1	o Hola o Hi o Buenas o Buenos días o Buenas tardes o Buenas noches	General Saludo

Despedidas		
T.6.2	o Adiós o Ciao o Bye o Hasta otra o Hasta luego	General Despedida
Insultos y Mala Educación		
T.6.3.1	o Insultos	General Insulto
T.6.3.2	o Mala educación	General MalaEducacion
T.6.3.3	o Sexo	General Sexo
Interacción social		
T.6.4.1	o Piropos	General Piropo
T.6.4.2	o Ligue	General Ligue
Elementos conversacionales		
T.6.5.1	o Sí o No o Gracias o Vale o …	General Conversacion

Otros temas de conversación		
T.5.5.2	o Cine	General Cine
T.5.5.3	o Deporte	General Deporte
T.5.5.4	o Viajes	General Viajes
T.5.5.11	o …	General …

11.1.7 Temas de Conversación no Conocidos

Si el bot no entiende lo que quiere decir el usuario devolverá el *topic* "Fallo". Si en la siguiente interacción con el usuario sigue sin entender la entrada del usuario, devolverá "Fallo Repetido" y volverá a la situación inicial.

ID	Usuario	Topic
Insultos y Mala Educación		
T.7. 1	o Respuesta desconocida por el Bot	Fallo
T.7.2	o Respuesta desconocida habiendo dado el Topic "fallo" en la pregunta anterior	Fallo Repetido

Referencias

Bibliografía

[Aiml02] Aimless, D. (2002). *A Tutorial for adding knowledge to your robot*.

[Alle94] Allen, J. (1994). *Natural Language Understanding*. Addison Wesley ed.

[Bate95] Bates, M. (1995). *Models of natural language understanding*. Proceedings of the National Academy of Sciences of the United States of America, Vol. 92, No. 22, pp. 9977-9982.

[Bush01] Bush, N. (2001). *Artificial Intelligence Markup Language*. (http://alicebot.org/TR/2001/WD-aiml/).

[Bush02] Bush, N. (2002). *Getting Started With Program D*. (http://www.alicebot.org/resources/programd/readme.html).

[Bush96] Buschmann F., Meunier R., Rohnert H (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons ed.

[Carb92] Carbonell, J. (1992). *El procesamiento del lenguaje natural, tecnología en transición*. Congreso de la Lengua española, Sevilla.

[Coro09] Coronado, M. (2009). *Desarrollo de una aplicación Web de gestión y automatización de bases de conocimiento de programas bot con tecnología Java Enterprise Edition*

[Dale00] Dale, R., Moisl, H., Somers, H. (2000). *Handbook of Natural Language Processing*. CRC Press ed.

[Flan02] Flanagan, D., Ferguson, P. (2002). *JavaScript: The Definitive Guide*. O'Reilly & Associates ed.

[Gosl05] Gosling, J., Joy, B., Steele, G., Bracha, G. (2005). *The Java language specification*. Addison-Wesley ed.

[Herm00] Hermjakob, U., Hovy, E.H., Lin, C. (2000). *Knowledge-based question answering*.

[Hovy00] Hovy, E.H., Gerber, L., Hermjakob, U., Junk, M., Lin, C. (2000). *Question answering in webclopedia*.

[Jord92] Jordán, A. G. (1992). *Lenguas y tecnologías de la información*. Congreso de la Lengua española, Sevilla.

[Leon98] Leonard, A. (1998). *Bots: the origin of new species*. Penguin Books ed.

[Mate96] Mateescu, A., Salomaa, A. (1996). *Formal Languages: An Introduction and a Synopsis*. Springer Verlag ed.

[Maul94] Mauldin, M.L. (1994). *Chatterbots, Tinymuds, And The Turing Test: Entering The Loebner Prize Competition*.

[Ring02] Ringate, T. (2002). *AIML Primer*. (http://www.alicebot.org/documentation/aiml-primer.html).

[Roze97] Rozenberg, G., Salomaa, A. (1997). *Handbook of Formal Language Theory*. Springer Verlag ed.

[Sesh05] Seshagiri, A. (2005). *Basic AIML User Manual*.

[Wall03] Wallace, R.S. (2003). *The Elements of AIML Style*.

[Weiz65] Weizenbaum, J. (1965). *ELIZA: a computer program for the study of natural language communication between man and machine*. Communications of the ACM, Vol. 9, No. 1.

Enlaces

[AlicURL] Página del bot conversacional A.L.I.C.E.:
http://www.alicebot.org

[ApacURL] Página de Apache Foundation:
http://www.apache.org/

[Al4jURL] API de Log4j:
http://logging.apache.org/log4j/1.2/apidocs/index.html

[CajaURL] Página del asistente virtual de Caja Madrid:
http://asistente.cajamadrid.es/Response/Bea.jsp

[ChatURL] Página de Chatterbean:
http://www.geocities.com/phelio/chatterbean/

[CssURL] Especificación de CSS1:
http://www.w3.org/TR/REC-CSS1-961217.html

[ElizURL] Página del bot conversacional ELIZA:
http://www-ai.ijs.si/eliza/eliza.html

[HtmlURL] Especificación de HTML 4.01:
http://www.w3.org/TR/REC-html40/

[HUniURL] Página de HttpUnit:
http://httpunit.sourceforge.net/

[iGodURL] Página del bot conversacional i-God:
http://www.titane.ca/igod/

[IkeaURL] Anna, asistente virtual de IKEA:
http://193.108.42.79/ikea-es/cgi-bin/ikea-es.cgi

[iURL] i, Asistente virtual del Ministerio de Cultura:
http://www.mcu.es/MC/CIC/AsistenteVirtual.html

[JabbURL] Página del bot conversacional Jabberwaky:
http://www.jabberwacky.com/

[JavaURL] Página de Java:
http://www.sun.com/java/

[jMedURL] Página de jMedia:
http://www.contentwithstructure.com/extras/jmedia

[jQueURL] Página de jQuery.Corner:
http://www.methvin.com/jquery/jq-corner-demo.html

[JscrURL] Javascript reference:
http://docs.sun.com/source/816-6408-10/contents.htm

[JspURL] Página de JSP:
http://java.sun.com/products/jsp/

[JstlURL] Página de JSTL:
http://java.sun.com/products/jsp/jstl/

[JUniURL] Página de JUnit:
http://www.junit.org/

[Lg4jURL] Página de Log4j:
http://logging.apache.org/

[MitsURL] Página del bot conversacional Mitsuku:
http://www.mitsuku.com/

[ProDURL] Página de ProgramD:
http://aitools.org/

[TeleURL] Asesor virtual de Telefónica Online:
http://www.telefonicaonline.com/on/io/es/atencion/buscador/

[WmlURL] Página con especificaciones relacionadas con la tecnología WAP:
http://www.wapforum.org/

[XhtmURL] Especificación de XHTML 1.0:
http://www.w3.org/TR/xhtml1/

[XmlURL] Especificación de XML:
http://www.w3.org/TR/REC-xml/