PROYECTO FIN DE CARRERA

Título:	Desarrollo de una plataforma para la asistencia en el desarrollo y gestión de sistemas de bots
Autor:	Miguel Coronado Barrios
Tutor:	Carlos Ángel Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	Gregorio Fernández Fernández
Vocal:	Mercedes Garijo Ayestarán
Secretario:	Carlos Ángel Iglesias Fernández
Suplente:	Pedro Alonso Martín

FECHA DE LECTURA:

CALIFICACION:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESARROLLO DE UNA PLATAFORMA PARA LA ASISTENCIA EN EL DESARROLLO Y GESTIÓN DE SISTEMAS DE BOTS

Miguel Coronado Barrios

CURSO 2009-2010

Agradecimientos

A mis padres, que con su amor, dedicación y esfuerzo me han educado y aun lo siguen haciendo, me han servido de ejemplo y modelo para llegar a ser lo que hoy soy.

A mis hermanos Marcos y Pablo, porque con ellos puedo compartirlo todo, porque siempre están e mi lado, en los malos y buenos momentos.

A mis amigos de Cuenca, porque son mis amigos de siempre: lo han sido, lo son y lo serán.

A mis amigos de Teleco, porque de pequeño siempre había pensado que los amigos de la universidad iban a ser los que compartiesen mis vivencias el resto de mi vida, así está siendo y espero que lo sea siempre.

A mis amigos del colegio mayor, porque compartí con ellos unos años extraordinarios el colegio mayor Santiago Galas Arce que con frecuencia echo de menos, a aquellos con los que aun hoy mantengo contacto porque amigos de verdad.

A mis amigos de San Bruno, porque les conocí cuando mas falta me hacía la amistad y nunca les podré devolver ese apoyo que me dan.

A mis amigos del laboratorio, porque ellos han hecho que decidir mi futuro haya sido más fácil.

A Mercedes por lo que me ha enseñado, por todo lo que ha hecho por que consiga mi itinerario personalizado y porque si no fuera gracias a ella aún estaría estudiando.

A Carlos, mi tutor y mentor, porque le aprecio mucho, porque he aprendido de él desde el primer año de carrera y por haber sido mi conciencia en estos últimos años de carrera.

A mi novia Olga porque la quiero con locura y por todo el apoyo que me da en todos los aspectos de mi vida.

A mi Padre y mi Amigo del cielo, porque sin Él nada en mi vida sería posible.

A todos los que han hecho que se me agoten estas líneas sin haber podido siquiera nombrarlos a todos. Todos aquellos en los que estoy pensando sabéis quienes sois. Gracias

Resumen

Esta memoria presenta la plataforma GSI Bot para el desarrollo de servicios de bots. El objetivo de la plataforma es facilitar la asistencia en lenguaje natural a los usuarios que están usando un servicio, facilitando su integración multicanal (mensajería instantánea, Web, teléfono móvil) y el mantenimiento del servicio, con herramientas para la gestión, manejo y edición de la base de conocimiento de los bots. El sistema ha sido aplicado para un servicio de atención al cliente de una operador e integrado en un mundo virtual orientado al sector turístico.

Palabras clave: Bot, bot conversacional, Agentes software, Agente inteligente, Inteligencia artificial, procesado de lenguaje natural.

Abstract

This Project presents the GSI Bot platform which focus on bot services development assistance. The platform's aim not only covers user interaction in natural language, making multimodal integration possible (IM, Web, cell phone) and service maintenance. It also offers a reliable testing framework for corpus testing through the bot server. However, the project's main module is the Knowledge Manager, that includes a wizard (Web based) that assists in corpus development.

The platform has been used to develop and deploy a bot integrated within a 3D tourist environment and also a bot, hosted in a mobile telephone operation Web page, that attends user requests

Índice

Capítulo 1

Introducción

1.1 Motivación

Prácticamente desde el comienzo de la andadura de los ordenadores, los programadores han tenido la ambición de crear programas que fueran capaces de mantener una conversación con un interlocutor humano. Posiblemente comenzaron como un juego o entretenimiento para el programador pero hoy en día están muy extendidos y cada vez más perfeccionados. Los primeros Bot conversacionales o chatterbots, como son conocidos, datan de la década de los 60, era necesario un ordenador dedicado a su funcionamiento y sólo una persona podía utilizarlos a la vez. En la actualidad, con el gran desarrollo de la Web es imposible imaginarlos desligados de Internet y son capaces de atender multitud de conversaciones simultáneamente. Ya no sólo aparecen como aplicaciones de demostración en la página del autor sino que se encuentran integrados, por ejemplo, en canales de IRC, como agentes en sistemas de mensajería instantánea, agentes expertos en FAQ, servicios de atención al cliente, sistemas domóticos, etc. Por supuesto forman parte de numerosas aplicaciones comerciales.

Sin embargo, aún queda mucho camino por recorrer: los Bots conversacionales están lejos de ser totalmente inteligentes y sólo son capaces de hablar de aquello que se les ha enseñado, de lo que tienen conocimientos. Este comportamiento es semejante al de los seres humanos, con la diferencia de que la capacidad de aprendizaje del Bot es

limitada y totalmente pasiva. Enseñar a un chatterbot (programar un chatterbot) es un proceso largo, repetitivo, cuya complejidad aumenta conforme el conocimiento del Bot se hace más basto y que requiere personal cualificado.

Una dificultad añadida a la hora de enseñar a un Bot, es la necesidad de hacerlo única y exclusivamente para un idioma determinado. De manera que la posibilidad de partir de un trabajo previo se restringe sólo a los trabajos realizados para dicho idioma. Igualmente las diferentes estructuras de las frases según el idioma y las particularidades idiomáticas hacen que las herramientas orientadas a la asistencia en el desarrollo de Bots en un determinado idioma no sean aptas para su uso en el desarrollo de Bots en otros idiomas.

Por otro lado, el trabajo colaborativo en la enseñanza a un Bot está significativamente lastrado por las interferencias que pueden producir los cambios realizados por miembros del equipo de trabajo en las parcelas de conocimiento en que trabajan otros miembros del equipo. Estas interferencias son difíciles de rastrear si no se dispone de las herramientas adecuadas.

Un estudio de tendencias en las búsquedas realizadas en internet indica que existen un creciente interés por los Bots conversacionales y por la programación de los mismos. Sin embargo, por ahora la programación de Bots queda restringida a personal con conocimientos del lenguaje de programación utilizado. De manera que la curva de aprendizaje que se presenta a aquellas personas que desean enseñar a un Bot, bien sea uno existente o desde cero, es muy pronunciada, lo suficiente como para desmotivarlas y hacerlas desistir de su empeño.

1.2 Objetivos del proyecto

El objetivo de este proyecto es, por tanto, desarrollar una serie de herramientas y aplicaciones que sirvan para la asistencia en el despliegue, desarrollo, gestión y prueba de Bots. De forma que para el castellano y el lenguaje de programación utilizado, mitiguen las dificultades enunciadas anteriormente, ya sea parcial o totalmente.

En resumen los principales objetivos de este proyecto son:

- Profundizar en el conocimiento y uso de las tecnologías que abarca este proyecto: tecnologías de bots conversacionales: AIML; tecnologías Web de servidor: J2EE, jsp, jstl; tecnologías Web de cliente: jQuery, Ajax, JSON…

- Construir un servidor de Bots que proporcione facilidades para el despliegue de Bots y ofrezca distintas interfaces de consulta para facilitar su integración en otros servicios (GsiProgramD).
- Desarrollar una herramienta de mantenimiento y gestión de bases de conocimiento que permita garantizar la integridad de una base de conocimiento (Corpus Tester). Mediante su utilización e integración con las aplicaciones de desarrollo de Bots el trabajo colaborativo en la enseñanza de Bots quedará libre del lastre de las interferencias enunciado anteriormente.
- Verificar la integridad del código AIML disponible haciendo uso de la herramienta descrita en el punto anterior, o modificarlo hasta conseguir dicha integridad.
- Desarrollar una aplicación Web de edición de bases de conocimientos que mediante una interfaz intuitiva permita editar las respuestas almacenadas en una base de conocimiento de manera rápida y fiable (Gestión de Respuestas).
- Desarrollar una herramienta Web para la edición de bases de conocimiento que permita añadir nuevas entradas a la base de conocimiento. La interfaz de esta herramienta debe ser intuitiva y su uso no debe requerir de personal cualificado y debe necesitar la menor explicación posible (Gestión de Preguntas).

1.3 Estructuración de esta memoria

La presente memoria está organizada en cinco capítulos principales, dividido cada uno en secciones y subsecciones. La estructura de capítulos desarrollada es la siguiente.

- En el capítulo 1, Introducción, se explica la motivación para el desarrollo del proyecto, se enuncian los objetivos y las fases del mismo.
- En el capítulo 2, Bot conversacional inteligente, se presenta en un tono divulgativo, la necesidades de la sociedad que intentan resolver los bots y bots conversacionales. y se estudia la evolución de los bots conversacionales en los cincuenta años de historia de los ordenadores.
- En el capítulo 3, Tecnologías de Bots utilizadas, se describen las características más relevantes para el desarrollo del proyecto tanto de las tecnologías de bots como de los paquetes de software utilizados, en definitiva el punto de partida.
- En el capítulo 4, Análisis y diseño de escenarios, se realiza un estudio de requisitos. Se presentan distintos casos de uso tanto en formato de tablas como en diagramas.

- El capítulo 5, Alcance y Arquitectura del proyecto, es el capítulo principal de esta memoria. Recoge aspectos de diseño e implementación de todos los módulos que componen la solución final desarrollada.
- En el capítulo 6, Evaluación del proyecto, se recogen los trabajos que se han desarrollado utilizando las herramientas desarrolladas en este proyecto.
- En el capítulo 7, Conclusiones y trabajos futuros, se recoge una visión general del desarrollo y resultados obtenidos en el proyecto. Por otro lado se proponen las líneas futuras de investigación y trabajo para mejora de la aplicación desarrollada.
- En el Anexo A, Manual de desarrollo, se explica con detalle cómo trabajar con el código de este proyecto. Cual es la estructuración del mismo, la política seguida en el uso del repositorio SVN y los archivos de ant incluidos para cada módulo.
- En el Anexo B, Definición del corpus del bot 3DTour, se recogen, en forma de tablas, la definición completa del corpus de BOT3DTour, el bot desarrollado para el proyecto 3DTour haciendo uso de las herramientas descritas en el capítulo 7.

Capítulo 2

Bot conversacional inteligente

En el presente capítulo se introducen los conceptos de bot y bot conversacional, se hace un repaso a la historia de los bots conversacionales, se presentan los bots conversacionales con mayor habilidad para conversar, y se hace una introducción a la comunidad de bots en Internet.

El presente capítulo está escrito en tono divulgativo, su lectura proporciona conocimientos amplios acerca del estado del arte en esta tecnología e introduce al lector en algunas discusiones acerca de la necesidad de los Bots en la sociedad actual, del aprendizaje de estos, etc.

2.1 El concepto de Bot y Bot conversacional

Los primeros ordenadores surgieron como una herramienta con el propósito de facilitar la labor a los investigadores que los utilizaban. Al realizar cálculos matemáticos sencillos a gran velocidad, efectuar cálculos de algoritmos para los que habían sido programados o resolver ecuaciones complejas previamente introducidas podían ahorra muchas horas de trabajo resolviendo operaciones. Su propósito no era otro que realizar una y otra vez la labor para la que habían sido programados de una forma rápida y sin cometer errores.

En la actualidad existen multitud de herramientas con estas características: la calculadora que usan los estudiantes para disminuir el tiempo invertido en realizar operaciones, las enciclopedias electrónicas que indexan la información que contienen para no tener que realizar búsquedas exhaustivas en libros, o la lavadora que utilizamos para no tener que lavar la ropa a mano. Todos ellos llevan a cabo una labor –estructuralmente repetitiva– que podría realizar el usuario que es liberado de esta tarea pudiendo invertir su tiempo en otra. Además realizan dicha tarea a una velocidad mayor de la que lo realizaría el usuario. Todas estas herramientas tienen cabida en una primera aproximación al concepto de Bot.

No es necesario que un Bot sea una máquina física, de hecho, actualmente se asocia el concepto de Bot a un programa con las características enunciadas y cuando se trata de una máquina se le denomina Robot.

La primera vez que se implementó una máquina para resolver ecuaciones sin solución analítica, por medio de cálculos numéricos, daba la impresión de que aparentemente esa máquina realizaba una labor que el programador no podría realizar. Sin embargo, la máquina utilizaba unos medios no analíticos y por medio de cálculo numérico y realizando numerosas iteraciones lograba arrojar una solución. La máquina había sido programada para llegar a la solución por un camino que al usuario le llevaría un tiempo incontable puesto que la máquina tiene ventajas estructurales respecto al usuario en ese camino. Los Bots, igual que todos los programas informáticos, sólo pueden realizar la tarea para la que han sido programados, y su objetivo es realizar eficientemente y tantas veces como se solicite una tarea que podría realizar su programador. Sin embargo cuando la labor del Bot necesita acceder a distintos servicios en Internet y recoger información de múltiples bases de datos, el programa realiza un función que en apariencia no está al alcance del usuario humano.

La definición de Bot recoge todas estas premisas: Un Bot es un programa informático que realiza de forma eficiente y rápida una tarea, estructuralmente repetitiva, para la que ha sido programado, imitando a un usuario humano. Un Bot puede tener acceso a distintas fuentes de información, y suele ser éste el punto en que es visiblemente más eficiente que el usuario humano.

Confusión en la terminología

Existe un concepto de Bot muy extendido y a la vez equivocado que liga los Bots con la propagación de virus, distribución de spam y en general con todo tipo de delitos y fraudes a través de internet [1]. De acuerdo con este concepto, un “Bot” es un programa malicioso que permite al atacante tomar el control de un equipo infectado. Por supuesto

los Bots pueden utilizarse con muy distintos propósitos pero no son maliciosos en su concepción original, el daño que puedan hacer depende del uso pernicioso que se haga de ellos. Otro uso malicioso de los Bots igualmente puede atentar contra la privacidad de los usuarios de Internet. Un Bot puede recopilar información de los sitios Web visitados por un usuario y extrapolar hábitos de conducta, aficiones e incluso rasgos de personalidad. Existen también detractores "morales" de los Bots, esgrimen que privan al ser humano de la necesidad de tener que tomar la iniciativa y de realizar algunas de esas actividades que nos diferencian de los demás seres vivos[1].

Sin embargo, el concepto de Bot como algo malicioso sólo es un problema de terminología pues no cabe duda que en la actualidad los Bots o sistemas de Bots están en auge, ampliamente extendidos y aceptados como algo que facilita muchas tareas. Cada vez más se intenta liberar al usuario de toda tarea que sea susceptible de ser realizada por un Bot. Desde algo tan trivial como mantener la velocidad de crucero hasta aparcar automáticamente. Desde algo tan engorroso como encontrar el mejor precio en el billete de un vuelo hasta algo tan sencillo como buscar una definición en el diccionario. Desde algo tan extendido como encender y apagar luces a nuestro paso hasta algo tan inverosímil como preparar el agua de la bañera todas las mañanas.

2.1.1 Bot conversacional o chatterbot

Un tipo de Bots muy extendidos son los denominados chatterbot o Bots conversacionales. Un Bot conversacional simula mantener una conversación con una persona. Para establecer una conversación han de utilizarse frases fácilmente comprensibles y que sean coherentes. Aunque un Bot conversacional haya alcanzado un grado de madurez considerable no será capaz de dar una respuesta a todo lo que el usuario le diga. El Bot conversacional tendrá en cuenta las palabras o frases del interlocutor que les permitirán usar una serie de respuestas preparadas de antemano. De esta manera, el Bot es capaz de seguir una conversación con más o menos lógica, pero sin saber realmente de qué está hablando.

Actualmente los Bots conversacionales se exhiben en Internet. Con el auge del Web 2.0 sus interfaces son cada vez más elaboradas, con acceso dinámico a la base de conocimiento para no tener que recargar la página base, con avatar animado que expresa emociones... Más recientemente, algunos comienzan a utilizar programas traductores de

[1] En la película de animación Wall-E [57] se muestra un mundo en el que los seres humanos son completamente dependientes de los autómatas [58].

texto a sonido[2], dotando de mayor realismo a la interacción con el usuario y tratando de alcanzar una total accesibilidad.

Sin embargo, a pesar de todos estos avances, construir o programar un Bot conversacional capaz de participar en una conversación sigue siendo un proyecto muy ambicioso, complejo y prácticamente imposible de acometer en tiempo. Por ello, el objetivo con que se están desarrollando los Bots conversacionales es el de ser capaces de responder acerca de todo tipo de preguntas relacionadas con un tema muy concreto: el inventario de una tienda, los lugares de interés de una ciudad, la ayuda (FAQ) de la página Web de una empresa… De hecho, éste ya es un objetivo muy difícil de alcanzar si se pretende alcanzar con un nivel de funcionamiento excelente.

El test de Turing

A mediados del siglo XX. el matemático, informático y filósofo inglés Alan Turing enunció la prueba conocida como el test de Turing [2]. Esta hipótesis afirmaba que si una máquina conseguía convencer, mediante una conversación, a un ser humano de que estaba hablando con un ser humano entonces se habría conseguido la Inteligencia Artificial. No obstante, a pesar de haber sido enunciada al inicio de la andadura de los ordenadores, la hipótesis sigue vigente en la actualidad, hasta la fecha ningún programa informático ha conseguido superar el test de Turing. Los programas que sí han engañado a sus interlocutores lo han hecho partiendo de una mentira: no decirle al juez que posiblemente quien hablaba con él era una máquina. El Test de Turing parte de la base de que el interrogador está intentando activamente diferenciar a su interlocutor humano del mecánico, lo que invalida los estudios antedichos.

2.1.2 Aprendizaje de los Bots

La concepto de Bot es una definición funcional ya que hace diferencias en el lenguaje o la tecnología utilizada para su implementación. Sin embargo, dependiendo de la tecnología utilizada los Bots podrán ser capaces de asimilar comportamientos o serán totalmente incapaces de aprender nada.

Con respecto al aprendizaje de los Bots cabe resaltar que no todos los Bots tienen la necesidad de aprender. El programa notificador de correo que nos informa de los mensajes entrantes en nuestra bandeja de correo realiza su labor sin necesidad de tener que aprender nada. Los denominados *shopbots* son agentes que buscan información de ventas de un determinado producto en distintas Web, de manera que exponen las ofertas

[2] Traducción del término inglés Text-to-Speech (TTS). Existen conversores de texto a sonido muy avanzados como es el caso de Loquendo [61]

que cumplan unas determinadas características. Se podría considerar que estos *shopbots* aprenden cada vez que el administrador introduce la dirección de una nueva tienda online en la que buscar productos, pero es un aprendizaje pasivo, pues el programador quien tiene la iniciativa y decide introducir una nueva dirección. Por otro lado, el Bot en cuestión inspeccionará el inventario de la tienda para saber cuando buscar en esa dirección y así podemos decir que ha asimilado conocimientos.

Con todo, este no es el concepto de aprendizaje de Bots al que queremos llegar. El concepto de aprendizaje está ligado a **agentes inteligentes**, de manera que por medio de una serie de reglas sean capaces de inferir un conocimiento o una forma de comportamiento. Los Bots que recomiendan a un usuario de una tienda online qué artículos pueden interesarle (también conocidos como agentes expertos) son capaces de comparar las últimas compras del usuario activo con las de los demás clientes asumiendo que la comunidad de usuarios que tienen gustos semejantes realizan compras semejantes y viceversa. Cada vez que un cliente realice una compra el Bot está adquiriendo conocimientos, pues en el futuro podrá recomendar un nuevo artículo a los usuarios qué compartan gustos (preferencias, compras…) con este último comprador.

Por supuesto, los agentes inteligentes con capacidad para aprender han sido programados para ello. En el ejemplo de agentes expertos que recomiendan compras en una tienda online, el agente es capaz de adquirir un conocimiento acerca de los gustos de los clientes a partir de las compras realizadas porque han sido programados para ellos.

Cuando hablamos de aprendizaje por parte de un Bot conversacional podemos hablar de aprendizaje a distintos niveles. Por un lado, la labor hecha por el programador de crear la base de conocimiento (escribir las reglas de actuación) se puede considerar como que el programador está enseñando al Bot. Pero por otro lado podemos llegar a pensar en el mismo proceso de aprendizaje que sigue un bebé o un niño. Aunque sólo los proyectos más ambiciosos plantean actualmente el aprendizaje de los Bots conversacionales a este nivel constituye la ambición de todo programador que trabaje con Bot conversacionales.

Existen diversas formas de abordar este aprendizaje.

- Enseñar a través de conversaciones en las que el Bot no entienda algo y pregunta por ello. Requiere que el Bot tenga una capacidad de interacción muy cuidada y madura.
- Enseñar de forma directa con una metodología tipo “Cuando te pregunten por tu nombre di que te llamar”. En este caso se trata de adoctrinar o instruir al Bot en unos conocimientos determinados.

- Aprendizaje por parte del Bot a partir de analizar multitud de conversaciones de mensajería de una persona o los diálogos de un personaje en una serie… Un Bot capaz de seguir este camino de aprendizaje podría simular la personalidad de la persona o personaje estudiado.
- Aprendizaje a partir de analizar la información contenida en multitud de páginas Web, tales como enciclopedias, páginas de referencia sobre un tema determinado.

Aunque hasta la fecha no existe ningún Bot conversacional que implemente de forma completamente satisfactoria alguno de estos caminos, sí que existen proyectos que trabajan en todos ellos.

2.1.3 Marco de este proyecto

Este proyecto aborda el concepto de los Bots a varios niveles:

- Por un lado realiza un estudio de las distintas tecnologías para la programación de Bots conversacionales, así como buenas prácticas en el desarrollo de la base de conocimiento o corpus de un Bot.
- Por otro lado se desarrollan varias herramientas para la asistencia en el desarrollo de Bots. El conjunto de estas herramientas está orientado a la creación de una aplicación de aprendizaje y enseñanza para Bots.

El presente proyecto es la continuación de un proyecto fin de carrera [3] realizado con anterioridad en este mismo departamento, en el que se desarrolla, a partir de la ayuda y preguntas frecuentes (FAQ) de la página oficial de Orange España [4] [5]. Este proyecto lo complementa con el desarrollo de herramientas para el desarrollo de bases de conocimiento.

2.2 Otros Bots y sistemas de Bots

Aunque en los últimos años los Bots conversacionales están proliferando (tanto que actualmente la búsqueda de Bot en Google sugiere las entradas de la Wikipedia para Bot [6] y Bot conversacional [7]) lo cierto es que existen otros muchos tipos de Bots.

En el ámbito de los videojuegos, se conoce como Bot a programas que son capaces de jugar por sí mismos el juego en cuestión. De hecho, están íntimamente ligados al éxito de muchos videojuegos en los que un solo jugador puede jugar gracias a que un Bot hace de contrincante. Este es, por ejemplo, el caso de los juego de Fútbol, Baloncesto y demás deportes de equipo en los que necesitas un adversario que controle

el equipo rival, el caso de los juegos que representan juegos de mesa o la amplia gama de juegos orientados a el modo multi-jugador con posibilidad de incluir jugadores que controla el propio ordenador. La calidad del Bot en este caso está dada por su capacidad de vencer al jugador, aunque muchas veces se otorgan ventajas a los Bots para compensar su falta de habilidad.

Los Bots para juegos CRPG [8] son particularmente conocidos ya que este tipo de juegos requieren un alto grado de estrategia y habilidad para ser jugados. Habitualmente en este tipo de videojuegos existen diferentes niveles de dificultad que representan el nivel de habilidad del Bot enemigo. Incluso en torno a los videojuegos de estrategia con más éxito aparecen empresas o programadores amateur que desarrollan parches para incluir otros niveles de dificultad de los Bots. En el caso del videojuego Warcraft III [9] incluso se ha desarrollado un editor de estrategias para programar la propia inteligencia para el Bot [10].

Con el desarrollo de internet también han proliferado multitud de Bots asociados a la red global. Ya se ha comentado la existencia de numerosos programas que asociados a servicios de Internet tales como el email o mensajería persiguen llevar a cabo actividades maliciosas [1]. Sin embargo, no todos los Internet Bots son malignos, existen Bot para realizar tareas de administración de sitios Web , limpieza y compactación en bases de datos. De hecho, muchas de las tareas de administración de servidores de Internet son simples y repetitivas de manera que un Bot puede realizarlas perfectamente, sin cometer errores y a una tasa mucho mayor que un usuario humano. Google mismamente utiliza un Bot, Googlebot, para "rastrear" los sitios de Internet e indexar todos resultados que ofrece en su buscador. Googlebot no solamente indexa páginas web (HTML), sino que también extrae información de ficheros PDF, PS, XLS, DOC y algunos otros más. Este Bot indexa también la información “fresca” de aquellas páginas que cambian frecuentemente.

2.3 Historia de los Bots conversacionales

El conocido test de Turing [2], que data de los años 50, sienta un criterio para decidir si un Bot conversacional ha alcanzado el grado de Inteligencia Artificial. A diferencia de lo que ocurre con otros problemas abordados por la informática, no se ha conseguido una solución totalmente satisfactoria para este problema, de manera que se siguen desarrollando soluciones cada vez más sofisticadas. El Turing predijo que en el año 2000, una máquina con 119 Mb de memoria podría engañar al 30% de sus jueces humanos tras una conversación de cinco minutos. Esto no ha sucedido aún, ni siquiera con máquinas muchas veces más potentes. No obstante, la hipótesis sigue vigente en la

actualidad, hasta el punto que es la comprobación que se cuantifica con mayor peso en los concursos de Bots.

El primer chatterbot mundialmente conocido es ELIZA. Fue desarrollado en el año 1966 por el profesos Joseph Weizenbaum en el Instituto Tecnológico de Masachusets (MIT). Desde entonces se han hecho multitud de versiones de éste, escritas en multitud de lenguajes de programación. A pesar de su edad, Eliza es un Bot conversacional próspero, que con sus respuestas convincentes es capar de hacerse pasar por un interlocutor humano siempre y cuando no se plantee de antemano al interlocutor la posibilidad de que esté conversando con un ordenador.

A pesar de su habilidad para conversar, Eliza sólo tiene tres maneras de mantener una conversación. Dispone de una base de datos de palabras clave, y para cada una de ellas una serie de respuestas pertinentes. Estas palabras clave son palabras importantes, como las que refieren a la familia, al sexo, al dinero, a los estados de ánimo. Si en la conversación el interlocutor humano no ingresa ninguna palabra que Eliza reconozca tiene dos estrategias. A veces Eliza contesta ambigüedades como "Mmmm, qué interesante", o "Sigue", o "¿Ah, sí?". Otras veces envuelve el texto del interlocutor en una frase más amplia.

PARRY es otro conocido Bot conversacional desarrollado en 1972 por el psiquiatra Kenneth Colby en la Universidad de Stanford. A raíz de la ocupación de Eliza como psicóloga se programa a PARRY como un esquizofrénico paranoico. PARRY y Eliza fueron conectados en varias ocasiones para comprobar el grado de coherencia de la conversación.

Después de Eliza otros muchos Bots conversacionales han ido apareciendo. Sin embargo, a pesar de aparecer muchos años después pocos bots han alcanzado su grado de madurez. Esto es debido a que, al contrario de lo que ocurre con otras tecnologías, el avance en la potencia de los ordenadores y las técnicas de programación no da una ventaja sustancial a la hora de conseguir que un Bot entienda más consultas o responda de forma más brillante. La madurez de un Bot viene dada por el tamaño de su corpus, es decir, a la cantidad de patrones y reglas introducidas en su base de datos.

Por lo tanto, de todos estos Bots destacan aquellos que incluyen nuevas funcionalidades o interfaces de comunicación. Es el caso de SmarterChild, con el que se podía conversar agregándolo como un contacto de mensajería instantánea en MSN Messenger, AOL e ICQ. Este bot incluía también funcionalidades relacionadas con el sistema de mensajería utilizado pudiendo realizar algunas tareas sencillas al incluir los comandos apropiados en la conversación. En los momentos de mayor éxito,

SmarterChild ha enviado más de 500 respuestas por segundo y más de cuatro millones de mensajes al día, de acuerdo con sus creadores. Cuando se le pregunta a SmarterChild con cuánta gente está hablando, habitualmente replica con el número exacto. el contador sobrepasó los 20 millones a finales de mayo del 2006.

Uno de los Bots conversacionales más recientes es KAS 2001. Destaca por ser un programa relativamente pequeño que ofrece una limitada capacidad de conversación pero múltiples funcionalidades relacionadas con el sistema operativo. Así el interlocutor puede pedirle a KAS 2001 en lenguaje natural que ejecute una aplicación, modifique la lista de reproducción en curso o compruebe el correo electrónico...

A.L.I.C.E.[3] es otro de los bots conversacionales más conocidos y su alto grado de madurez lo convierten en uno de los Bots más sofisticados que se han programado. Está inspirado en Eliza, su desarrollo comenzó en 1995, siendo el Dr. Richard Wallace el autor original del proyecto. 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. El propósito de la evolución de las versiones de A.L.I.C.E no es tanto programar un Bot que sea capaz de mantener una conversación con mayor grado de coherencia sino conseguir atraer a programadores para consolidad una comunidad de programadores de Bot conversacionales.

[3] Artificial Linguistic Internet Computer Entity

Capítulo 3

Tecnologías de Bots utilizadas

En este capitulo se describen las tecnologías seleccionadas para ser utilizadas en el desarrollo del proyecto. La explicación, lejos de ser exhaustiva, se centra en aquellas características que son relevantes para la descripción de las herramientas desarrolladas en este proyecto. Este capito no trata, por tanto, de sustituir las especificaciones formales acerca de las tecnologías descritas sino de apuntas aquellos aspectos más importantes.

3.1 El lenguaje AIML

El lenguaje AIML [11], acrónimo de Artificial Intelligence Markup Language, es un dialecto de XML específicamente diseñado para crear bots conversacionales. AIML describe un tipo de objetos de datos llamados objetos AIML así como el comportamiento de los programas que los procesan. Como extensión de XML los objetos AIML también pueden ser incluidos en archivos XML.

Los archivos escritos en AIML se denominan archivos AIML y tienen esa misma extensión. El software que se encarga de leer los archivos AIML[4], extraer los objetos AIML y proporcionar una aplicación con una funcionalidad basada en la estructura de dichos objetos se denomina intérprete de AIML. Generalmente el intérprete de AIML

[4] o la base de datos con el contenido AIML en su caso.

forma parte de un programa de mayor envergadura que se suele denominar bot o, en caso de que dé servicio a varios bots, servidor de bots.

3.1.1 Introducción

Un archivo AIML comienza con la etiqueta `<aiml>` y termina con `</aiml>` que definen un objeto AIML. Entre estas etiquetas están contenidos una serie de reglas representadas por elementos `<category>` y definidas por un patrón `<pattern>` y una plantilla de respuesta `<template>`. El formato básico de un documento AIML es, por tanto, el que se muestra en la Fig. 1.

```
<aiml>
  <category>
    <pattern> ... </pattern>
    <template> ... </template>
  </category>
  <category>
    <pattern> ... </pattern>
    <template> ... </template>
  </category>
  ...
</aiml>
```

Fig. 1. Estructura básica de un archivo AIML.

Más adelante se describirán con detalle y ejemplos cada unos de los elementos del lenguaje.

Cada objeto `category` contiene la mínima información necesaria para establecer la respuesta a un estímulo de entrada o consulta. Habitualmente el estímulo de entrada es una frase introducida por el usuario del bot y la respuesta producida puede ser una cadena textual, una alteración en el estado del bot o una combinación de ambos. Las alteraciones del estado del bot quedan definidas en la memoria del programa que procesa el código AIML y se les llama variables AIML, o más apropiadamente predicados. Pueden ser fragmentos de información que constituyen recuerdos de lo que se ha hablado en la conversación o del interlocutor, información acerca del estado de ánimo del bot, etc. Realmente la entrada textual es la única vía de entrada, sin embargo,

se definen intérpretes y conectores que escuchan eventos externos tales como eventos del sistema operativo, que traducen a estímulos de entrada textuales.

Filosofía de programación de AIML

En definitiva, programar en AIML consiste en escribir reglas definidas mediante elementos category en las que se le indica al bot: "si encuentras una consulta que se ajusta a este patrón haz esta acción". Los patrones quedan definidos en los elementos `<pattern>` y las acciones a realizar en los elementos `<tempalte>` asociados.

Para programar un bot que sea capaz de dar una respuesta diversidad de consultas es necesario incluir multitud de reglas que se ajusten a las diferentes consultas posibles. Por supuesto, AIML define un sistema de patrones pare definir reglas que respondan a un cierto abanico de consultas. Para ello se definen dos comodines distintos "*" y "_". Así, por ejemplo, se puede definir un patrón "Hola *" al que se ajustan las consultas "Hola, ¿Qué tal?" y "Hola, ¿Cómo estás?".

En general, los patrones AIML son muy selectivos, por lo que, a pesar de su existencia, es necesario escribir muchas reglas. Más adelante se explicará en detalle el funcionamiento y la sintaxis de los patrones en 3.1.4.1.

3.1.2 Características del lenguaje AIML

En el momento de su definición AIML se concibió como un lenguaje con las siguientes características:

- AIML es fácil de aprender.
- AIML es compatible con XML.
- AIML no depende de ningún otro lenguaje de programación.
- Un objeto AIML contiene la mínima cantidad de código necesario para desarrollar un bot pregunta-respuesta.
- Las objetos AIML tienen soporte para reglas causales.
- Los objetos AIML son legibles por un humano y razonablemente claros.
- El procesado de los elementos AIML es sencillo, por lo que es fácil programar un intérprete básico de AIML.
- La escritura de AIM es formal y conciso.

Al ser compatible con XML se pueden incluir etiquetas XHTML para mejorar la presentación de la respuesta dada por el bot.

AIML se encuentra en el subconjunto de lenguajes XML denominados *document-oriented*. En estos, se permite que el contenido de un elemento sea mixto: puede contener otros elementos y texto mezclados. Aunque esto no le hace perder su compatibilidad con XML esto constituye una desventaja, puesto que su sintaxis es engorrosa y las librerías y programas funcionan mejor para lenguajes que son *data-oriented*, en los que el contenido de un elemento es sólo texto u otro elemento.

Al no existir ninguna dependencia con otro lenguaje de programación y no ser AIML un lenguaje compilado sino interpretado, las bases de conocimiento en AIML son totalmente portables e independientes del intérprete de AIML. Sin embargo, cada vez es más frecuente que en diversos proyectos se añadan funcionalidades a AIML, lo que infiere en la dependencia de otros lenguajes o servicios.

Esta independencia otros lenguajes y el intérprete hace que los sistemas AIML sean fácilmente escalables, pues basta con incluir nuevos archivos. Sin embargo, el carácter masivo de los corpus escritos en AIML los hace propensos a contener errores. Cuando la cantidad de reglas es considerable, es habitual que se incluyan reglas con patrones similares o idénticos, especialmente si el escalado se hace integrando varios corpus distintos. Este concepto se desarrollará más adelante en el apartado 3.1.4.

El carácter formal y conciso del lenguaje AIML lo hace susceptible de ser autogenerado a través de herramientas de automatización como las que se describen más adelante en este proyecto.

3.1.3 Sintaxis AIML: Elementos y objetos

En un documento XML los elementos son las estructuras lógicas delimitadas por una etiqueta de inicio (*start-tag*) y una etiqueta de final (*end-tag*). También está permitida una estructura formada únicamente por una etiqueta vacía (*empty-tag*) [12].

```
<pattern>Que es AIML?</pattern>
<star index=”1” />
```

Fig. 2. Ejemplo de elementos XML.

Los elementos AIML deben pertenecer al **espacio de nombres** http://alicebot.org/2001/AIML-1.0.1. Es importante utilizar el espacio de nombres

(*namespace*) para evitar ambigüedades con otros elementos XML [13], pues es frecuente incluir, por ejemplo, elementos XHTML en los archivos AIML para formatear las respuestas. En ese caso se puede utilizar el espacio de nombres AIML y un segundo espacio de nombres XHTML como se muestra en el siguiente ejemplo.

```
<aiml xmlns="http://alicebot.org/2001/AIML-1.0.1"
      xmlns:xhtml=" http://www.w3.org/1999/xhtml">
  <category>
    <pattern>Que es AIML?</pattern>
    <template>
      AIML es mi <xhtml:b>idioma</xhtml:b>!
    </template>
  </category>
</aiml>
```

Fig. 3. Ejemplo de utilización de varios espacios de nombres.

Los objetos son la estructura funcional correspondiente a dichos elementos XML. El procesado de los objetos por par parte de el interprete de AIML produce cambios en el sistema y una saluda en forma de cadena textual. Los elementos AIML deben ser parseados y procesados de dentro a fuera. De manera que si existen elementos AIML anidados, los más internos se procesarán primero y producirán la salida correspondiente que será procesada como contenido de los elementos que los contienen.

Como ya se ha mencionado, en las especificaciones de AIML [11] se describen tanto los elementos AIML como el comportamiento de los programas que procesan los objetos que representan. Puesto que dichas especificaciones constituyen un documento extenso, en este apartado sólo se analizarán los objetos AIML utilizados en el proyecto.

3.1.3.1 AIML

Es el elemento raíz de un archivo AIML. Debe tener un atributo que indique el número de versión de AIML utilizada. Dicho atributo será utilizado por el intérprete de AIML para comprobar que todos los archivos cargados en un bot siguen las mismas especificaciones, y si estas son compatibles con el propio interprete.

Es una buena práctica incluir la definición de espacios de nombres en este elemento pues engloba al resto de elementos del archivo.

Un elemento AIML sólo puede contener elementos *category* y *topic*. Puede no contener ningún elemento *topic* pero debe contener al menos un elemento *category*. Un objeto AIML es, por tanto, un contenedor de objetos *category* y *topic*.

```
<aiml version = number>
   <!- Elementos category y topic -->
<aiml>
```

Fig. 4. Definición del elemento aiml.

3.1.3.2 Category

Contiene un elemento *pattern* y un elemento *template*. Además puede contener otros elementos opcionales.

El objeto category representa una regla[5], que contiene un patrón con el que cotejará con la consulta (elemento *pattern*) y una respuesta (elemento *template*) que se ejecutará en caso de que la consulta se ajuste al patrón. Se dice que se evalúa un patrón cuando se coteja una consulta con dicho patrón. Se dice que se verifica un patrón cuando una consulta responde a un patrón o casa con dicho patrón.

```
<category>
  <pattern>patron</pattern>
  <template>
   <!- Elementos de template -->
  </template>
<category>
```

Fig. 5. Definición del elemento category

3.1.3.3 Topic

Es un elemento opcional que contiene al menos un elemento *category*. Debe tener un atributo *name* que lo identifique.

El objeto topic es un contenedor de elementos category. Pero a diferencia del elemento AIML, el elemento topic añade una restricción a la hora de evaluar la

[5] un patrón de actuación para el Bot.

consulta: las categorías contenidas sólo se evaluaran si la variable AIML de nombre topic verifica el patrón contenido en el atributo nombre.

En general el atributo name es una palabra o conjunto de palabras y no contiene comodines pero las especificaciones de AIML sí lo permiten.

```
<topic name = patron>
  <!- Elementos category… -->
<topic>
```

Fig. 6. Definición del elemento topic.

3.1.3.4 Pattern

Es el elemento que define el patrón asociado a cada category. Debe ser el primer elemento dentro del cada category.

```
<pattern><!- Patron --><pattern>
```

Fig. 7. Definición del elemento pattern.

3.1.3.5 That

Es un elemento de category que define un patrón asociado al historial de respuestas dadas por el bot.

```
<that><!- Patron --><that>
```

Fig. 8. Definición del elemento that.

3.1.3.6 Template

Es el elemento que define la acción a ejecutar si se verifica el patrón al que está asociado. Cada elemento category debe contener exactamente un elemento template. El contenido de template es contenido-mixto.

El objeto template, cuando se ejecuta procesa su contenido y devuelve el texto como respuesta. En general, los elementos de template, al ser procesados, producen una respuesta que incluirá en la posición que ocupa dicho elemento en la cadena de º

```
<template>
  <!- Elementos de template -->
  <!- Respuesta (texto) -->
</template>
```

Fig. 9. Definición del elemento template.

3.1.3.7 Set

Es un elemento de *template* que habitualmente devuelve una cadena. Debe tener un atributo *name*.

El objeto set hace indica al intérprete de AIML que debe guardar el contenido del elemento en el predicado AIML cuyo nombre viene dado por el atributo name. Por lo que el contenido del atributo name debe ser un nombre de predicado válido. Si este predicado aún no ha sido definido el intérprete debe crearlo. Además, el objeto set devuelve el valor almacenado.

```
<set name="nombre del predicado">
  <!- valor(texto) -->
</set>
```

Fig. 10. Definición del elemento set.

3.1.3.8 Get

Es un elemento de *template* que devuelve una cadena. Debe tener el atributo name. Get es un elemento vacío.

El objeto get indica al intérprete que debe sustituir este elemento por el contenido del predicado AIML cuyo nombre viene dado por la variable name. Si este predicado no existe será sustituido por una cadena vacía "".

```
<get name="nombre del predicado" />
```

Fig. 11. Definición del elemento get.

3.1.3.9 Srai

Es un elemento de template, puede contener cualquier elemento de template. Este elemento devuelve una cadena.

El objeto srai indica al intérprete de AIML que debe procesar su contenido como si fuese una nueva consulta del usuario. La respuesta dada por el bot como respuesta de esta nueva consulta será la respuesta dada por el bot.

```
<srai>
  <!- Elementos de template -->
</srai>
```

Fig. 12. Definición del elemento srai.

3.1.3.10 Star

Es un elemento vacío, que puede contener un atributo de nombre *index* cuyo valor es numérico.

```
<star índex="2" />
```

Fig. 13. Definición del elemento star.

El objeto star indica al intérprete que debe sustituir el elemento por el valor del comodín correspondiente. En el ejemplo siguiente se utiliza el elemento star para incluir en la respuesta un fragmento de la consulta realizada pro el usuario.

```
<category>
  <pattern>¿Qué es un * ?</pattern>
  <template>
    No sé lo que es un <star />.
  <template>
</category>
```

Fig. 14. Ejemplo de uso del elemento star.

Cuando sólo hay un comodín en el patrón –como en el ejemplo de la Fig. 14- el atributo index puede omitirse. En caso contrario es obligatorio su uso e indica el orden en que se encuentra el comodín entre los comodines en el patrón.

3.1.3.11 Think

El elemento think es un elemento de tempalte. El elemento think puede contener cualquier otro elemento de template.

```
<think>
  <!- Elementos de template -->
</think>
```

Fig. 15. Definición del elemento think.

El objeto think produce el mismo efecto cuando se ejecuta que un objeto template con el mismo contenido, con la diferencia de que no se devuelve ninguna respuesta. Los elementos think sirven para producir efecto en la memoria del bot sin que se vean reflejados en la respuesta del bot. Habitualmente se utilizan con elementos set para establecer el valor de variables AIML.

```
<template>
  <think>
    <set topic=”mood”>contento</set>
  </think>
  Estoy alegre.
</template>
<!- Salida: Estoy alegre -->

<template>
  <set topic=”mood”>contento</set>
  Estoy alegre.
<template>
<!- Salida: Contento Estoy alegre -->
```

Fig. 16. Ejemplo de uso del elemento think.

3.1.4 Programación de corpus en AIML.

Las bases de conocimiento escritas en AIML están orientadas para ser usadas en aplicaciones de pregunta y respuesta, Q&A[6]. El usuario tiene la iniciativa en este tipo de sistemas, él inicia la comunicación al enviar una consulta al bot, el cual la procesa y produce una respuesta.

Para que un bot sea capaz de entender una consulta determinada hay que crear una regla con un patrón asociado, de tal manera que la consulta se ajuste a dicho patrón. En este caso decimos que la consulta casa con el patrón o que verifica el patrón. Cada vez que el usuario envía una consulta al bot se evalúan todos los patrones para comprobar si se ajusta a alguno de ellos. En tal caso decimos que el patrón dispara la regla a la que está asociado el patrón, y por lo tanto dicha regla se ejecuta. La ejecución de una regla, definida como un objeto *category*, implica la ejecución del contenido del objeto *template* asociado.

La programación de un bot que sea capaz de responder a multitud de consultas implica la creación de multitud de reglas con patrones que se ajusten a todas las consultas a las que queremos dar respuesta.

3.1.4.1 Sintaxis de patrones de AIML

Los patrones AIML definidos por las etiquetas <pattern> son altamente selectivos, ordenados, insensibles a mayúsculas y consideran los espacios en blanco[7]. Los patrones AIML son secuencias de caracteres que pueden contener letras mayúsculas y minúsculas, dígitos o espacios en blanco y permiten la utilización de dos tipos de comodines o wild-cards _ y *, el primero más prioritario que el segundo.

A la hora de confeccionar un patrón hay que imaginar la frase que el usuario introduciría acerca de un tema, se mantienen en el patrón las palabras que son importantes para poder expresar dicha idea y el resto se sustituye por comodines. Es importante destacar que un patrón no se verifica si en el lugar de los comodines hay espacios en blanco. Por ejemplo, si se quiere construir un patrón para que el bot dé respuesta a la consulta "Necesito información de Madrid" se pueden considerar como palabras importantes información y Madrid y sustituir el resto por comodines[8], resultando el patrón * *INFORMACION * MADRID*.

[6] Question and answering
[7] Altamente selectivos al no permitir espacios en blanco en el lugar de los comodines y ordenados por no permitir la aparición de las palabras en un orden distinto al que aparece en el patrón.
[8] En esta estrategia se basa la herramienta de gestión de conocimiento.

Dado el patrón

*** INFORMACION * MARID**

Las siguientes consultas verifican el patrón:

Necesito **información** sobre **Madrid**.

Dame **información** de **Madrid**.

Tienes alguna **información** de **Madrid**.

La siguiente consulta no verifica el patrón:

Información sobre **Madrid**

Fig. 17. Ejemplo de verificación de patrones.

En el ejemplo anterior, la frase "Información sobre Madrid" no verificaría el patrón porque en lugar del primer comodín no hay ningún carácter. Para que el bot también sea capaz de responder a esta consulta es necesario incluir el patrón *INFORMACION * MADRID*.

El hecho de que los comodines no se verifiquen si coinciden con espacios en blanco hace que los patrones sean más selectivos pero a su vez hace que sea necesario repetir varias veces la misma regla (con igual respuesta) con patrones con distinta disposición de los comodines.

3.1.4.2 Patrones con *topic* o *that*.

AIML también proporciona mecanismos para crear patrones que sean selectivos en función del curso de la conversación. Este es el caso de los elementos *that* y *topic*. El primero tienen en cuenta las últimas respuestas dadas por el bot, el historial de la conversación, mientras que el segundo considera el tema de la conversación, almacenado en el predicado de nombre topic.

Estos elementos no definen un patrón en sí mismo, sino que constituyen una condición necesaria que se ha de cumplir, además de verificarse el patrón, para que se ejecute la regla; por lo que filtran la ejecución de algunas reglas. Se dice que los elementos *that* y *topic* aumentan la concreción de los patrones a los que acompañan.

Tal y como se describe en el apartado de sintaxis, el elemento ***that*** se asocia a una regla. Contiene un patrón que se debe verificar con la última respuesta dada por el bot, de manera que en caso de que exista para que se ejecute una regla tanto deben

verificarse simultáneamente el patrón y el patrón *that* con la consulta y la última respuesta del historial respectivamente.

Se considera un ejemplo en el que un usuario ha realizado una consulta a la que el bot le responde "debes consultar el manual". Si el usuario no sabe donde encontrar el mencionado manual es normal que se lo pregunte al bot, "¿dónde está el manual?", en ese caso se disparará la regla de la Fig. 18 y el bot responde con la dirección url en la que se encuentra el mencionado manual.

En la regla del ejemplo se utiliza una regla con un patrón muy genérico, que casa con un amplio abanico de preguntas. Sin embargo, al complementar el patrón con un elemento *that* la regla queda perfectamente encuadrada y precisada.

```
<category>
  <pattern>DONDE *</pattern>
  <that>DEBES CONSULTAR EL MANUAL</that>
  <template>
    El manual está en la siguiente url…
  </template>
</category>
```

Fig. 18. Ejemplo de category con elemento that.

El elemento ***topic*** se incluye directamente en el elemento *aiml* y contiene elementos *category*. A pesar de que su sintaxis es diferente a la de *that* –que da una idea del orden en que se realizan las comprobaciones[9]– en la práctica se puede considerar que cada *category* dentro del *topic* debe cumplir la condición adicional impuesta por el *topic*, igual que ocurre con el elemento *that*.

[9] Lo primero que se comprueba es que se cumpla el topic, después el patrón y por último la condición impuesta por that.

```
<topic name="futbol">
  <category>
    <pattern>DE QUE HABLABAMOS</pattern>
    <template>De fútbol!</template>
  </category>
...
</topic>
```

Fig. 19. Ejemplo de category dentro de un elemento topic.

En la regla mostrada en la Fig. 19 la información dada como respuesta no proviene de la propia regla en sí sino del topic.

Comodines en that y topic

El contenido de los elementos *that* y *topic* es un patrón que puede contener wild-cards. Sin embargo, no es habitual incluir comodines pues el nivel de complejidad de la base de conocimiento se incremente exponencialmente al incluir este grado de libertad, se dispara la probabilidad de introducir errores al escalar el sistema.

3.1.4.3 Prioridad de patrones

En AIML se definen varias reglas que determinan qué patrón se ha de ejecutar cuando hay varios a los que se ajusta la consulta. En ese caso se dice que el patrón que se ejecuta tiene más prioridad que los demás patrones que también estaban en disposición de ejecutarse.

Criterio básico de prioridad.

En general, cuando la frase introducida por el usuario casa con varios patrones sólo se ejecutará aquel que "encaja antes" con el patrón, es decir, aquel patrón cuya primera palabra (que no sea un comodín) se encuentra antes en la consulta. Se dice que una consulta se "engancha" al patrón que verifica con la primera palabra que es común a ambos.

En AIML el **orden de escritura de las reglas no es importante**, ésta es una característica importante de AIML pues permite la escalabilidad del sistema.

En el ejemplo de la Fig. 20 se ilustra el concepto de prioridad básica. Para los patrones dados "** INFORMACION * MARID*" y "** SOBRE MADRID*" la consulta

"Necesito información sobre Madrid", aunque verifica ambos patrones sólo dispara el primero; puesto que la primera palabra de dicho patrón que es distinta de un comodín es INFORMACION, la segunda palabra de la consulta, mientras que la primera palabra distinta de un comodín del segundo patrón es SOBRE, que se encuentra en la tercera posición en la consulta.

Dados los patrones

*** INFORMACION * MARID**

*** SOBRE MADRID**

Las siguiente consulta verifica ambos patrones pero por prioridad se dispara el primero:

"Necesito **información** sobre **Madrid.**"

La siguiente consulta verifica el segundo patrón:

"Información sobre **Madrid"**

Fig. 20. Ejemplo de prioridad básica de patrones.

En el caso de que varios patrones tengan la misma prioridad, como ocurre en el ejemplo de la Fig. 21, se volverá a aplicar el mismo criterio con el fragmento de la consulta que difiere para ambos patrones.

Dados los patrones

*** INFORMACION * MARID**

*** INFORMACION SOBRE ***

Las siguiente consulta verifica ambos patrones pero por prioridad se dispara el segundo:

"Necesito **información** sobre **Madrid.**"

Fig. 21. Segundo ejemplo de prioridad básica de patrones.

En función de su estructura se puede clasificar por prioridad los distintos tipos patrones. Sea XXX una palabra cualquiera:

1. "**XXX ***" tiene prioridad máxima, pues cuando se verifique lo hará enganchándose con la primera palabra de la consulta.
2. "*** XXX ***" tiene una prioridad intermedia.

3. "*** xxx**" tiene prioridad mínima, pues siempre que se verifique lo hará enganchándose con la última palabra.

El criterio básico de prioridad de patrones en AIML condiciona la forma en que se debe programar la base de conocimiento. Se debe dar mucha importancia a las palabras iniciales de la consulta, que generalmente son partículas interrogativas.

El criterio de prioridad básico viene determinado por los algoritmos de implementación del *proceso de encaje de patrones*[10], pues hace este proceso sea lo más rápido posible. Sin embargo, no es el criterio de prioridad óptimo en cuanto a la selección de patrones pues plantea algunas desventajas y obliga a prácticas que se pueden considerar contraintuitivas.

Se considera el ejemplo mostrado en la Fig. 22. La primera regla, con el patrón "*QUE **" se ha introducido como una regla genérica, para dar una respuesta genérica cuando se pregunte por algo sobre lo que no se ha incluido más información en la base de conocimiento. Sin embargo esta es una mala práctica, pues debido al criterio básico de prioridad esta regla tiene una prioridad muy alta, ya que siempre que se verifique lo hará "encajando" con la primera palabra de la consulta. Por lo tanto, la consulta "Que criterios de prioridad existen en AIML" disparará esta regla en lugar de la regla con el patrón asociado "** CRITERIOS * PRIORIDAD * AIML*" que, a priori, debería tener mayor prioridad pues es un patrón mucho más concreto.

```
<category>
  <pattern>QUE *</pattern>
  <template>No sé que es eso</template>
</category>
...
<category>
  <pattern>* CRITERIOS * PRIORIDAD * AIML</pattern>
  <template>Existen varios criterios…</template>
</category>
```

Fig. 22. Tercer ejemplo de prioridad básica de patrones.

Inversión de prioridades con el comodín '_'

[10] Del inglés pattern-matching.

Una excepción a la regla de prioridades básica es el caso en el que uno de los patrones que se ajustan a la consulta tiene un comodín de prioridad máxima '_'. En este caso tiene mayor prioridad el patrón que contenga el comodín '_'. Se dice que este es un caso de inversión de prioridades por el comodín de prioridad máxima.

Dados los patrones

*** INFORMACION * MARID ***

_ MADRID _

Las siguiente consulta verifica ambos patrones pero por inversión de prioridades se dispara el segundo:

"Necesito **información** sobre **Madrid**."

La siguiente consulta no verifica ningún patrón:

"Dame información sobre **Madrid"**

Fig. 23. Ejemplo de patrón oculto por inversión de prioridades.

Se considera una buena práctica en la programación de AIML restringir el uso del comodín '_' a situaciones muy concretas: generalmente dentro de elementos *topic*, acompañados de elementos *that*, o para realizar reducciones mediante el elemento *srai*. Su uso descontrolado puede producir situaciones anómalas en las que el bot no responde lo esperado debido a la inversión de prioridades.

El ejemplo de la Fig. 23 muestra dos patrones que nunca deben ir juntos pues el patrón ** INFORMACION * MADRID ** se verificará siempre a la vez que el patrón *_ MADRID _* por lo que nunca se ejecutará la regla a la que está asociado. En ese caso se dice que el segundo patrón oculta al primero.

El patrón compuesto únicamente por el comodín _ se verifica siempre y oculta todos los demás patrones. Bajo ningún concepto se debe incluir una regla como la que se muestra en la Fig. 24 fuera de un elemento topic.

```
<category>
  <pattern>_</pattern>
  <template>
    No entiendo qué quieres decir
  </template>
</category>
```

Fig. 24. Uso prohibido del comodín '_'.

Prioridades de patrones con that y topic.

Los elemento *that* y *topic* no modifican la prioridad de los elementos a los que acompañan. La regla de prioridad básica prevalece independientemente de si existen elementos *that* o *topic*. Como se ha explicado en el apartado , la función de los elementos *that* y *topic* es aumentar la concreción de los patrones en función del curso de la conversación.

Estos patrones sólo determinan la prioridad entre patrones si se verifican dos patrones idénticos, uno de ellos acompañado de un *topic* o *that* (que se verifica), en tal caso se verifica dicho patrón[11].

3.1.5 Limitaciones de AIML

AIML ha sido específicamente diseñado para posibilitar la programación de bots conversacionales con la mayor facilidad posible, esto es, conseguir programar un bot conversacional sencillo con la mayor rapidez posible. Sin embargo, cuando se trata de trabajar en sistemas complejos, que impliquen varios millares de reglas, AIML muestra diversas limitaciones.

En AIML no se definen herramientas de análisis sintáctico, esto obliga a introducir múltiples variantes de algunas palabras –especialmente tiempos verbales- para garantizar la efectividad de las reglas definidas. Igualmente es necesario incluir en el código AIML todas las variaciones semánticas de las palabras claves.

La regla básica de prioridad de patrones obliga a programar las bases de conocimiento de una manera determinada, dando importancia máxima las primeras

[11] En caso de que de los dos patrones idénticos esté uno acompañado por un elemento that y el otro por un topic el del elemento that tiene mayor prioridad.

palabras de la frase. El criterio óptimo sería un criterio de concreción, en el que el patrón con mayor número de patrones

AIML se define con rigidez en la capacidad de selección de sus patrones, es decir el usuario debe ingresar exactamente la consulta que se ajuste al patrón definido en la base de conocimiento de bot. No existen comodines selectivos en función de la naturaleza de la entrada, como pueden ser comodines para números, o sólo para palabras.

Es costoso realizar un set de pruebas completo, ya que la alta selectividad de los patrones de AIML que hace que sea necesario incluir multitud de estos también multiplica la cantidad de pruebas a incluir, especialmente desde un enfoque de caja blanca [14]. Por otro lado, si se tienen en cuenta los giros que puede dar la conversación en función del contexto el número de pruebas requeridas aumenta potencialmente.

Los parseadores de AIML son más complicados de lo estrictamente necesario dado que el lenguaje AIML se ha diseñado con una estructura *document-oriented*, es decir, XML que acepta una mezcla de texto plano y elementos como contenido de los mismos elementos XML. Muchas de las bibliotecas para parsear XML dan un soporte parcial a estas estructuras o funcionan peor que con los archivos *object-oriented*, en los que no existe esta posibilidad de mezcla [15].

El soporte de la comunidad al lenguaje AIML es parcial, pues ningún intérprete cumple con todo la totalidad de las especificaciones de AIML. En generar son las características más avanzadas, como son las etiquetas `javascript` o `condition`, las que están implementadas.

Uno de los propósitos de los creadores de AIML era conseguir que la comunidad desarrollase y compartiese un basto conjunto de archivos AIML sobre los que basar los demás desarrollos. Sin embargo, la amplia mayoría de los trabajos realizados con AIML son el inglés; existen muy pocos trabajos publicados sobre el uso de AIML en español y una minoría permite descargar los archivos del corpus, y debido a las variedades estructurales que presentan estos lenguajes no es una labor eficiente llevar a cabo una traducción directa de los archivos AIML en otra lengua.

3.2 ProgramD

ProgramD [16] es la plataforma para bots AIML de código abierto escrita en Java más utilizada. Consiste en un intérprete de AIML que implementa la mayoría de las características del lenguaje. Cada instancia de programD soporta un número ilimitado

de bots en el contenedor de aplicaciones. ProgramD es altamente configurable por medio de un conjunto de ficheros de configuración, que permiten activar o desactivar muchas de sus funciones. Una de las características más interesantes de programD es la posibilidad de definir múltiples bots, de manera que representen personalidades distintas y tengan conocimientos sobre distintos temas de conversación; es decir, contengan el conocimiento de distintos archivos AIML. De esta manera, si se estructura la base de conocimiento en varios archivos AIML se pueden definir bots que contengan archivos de conocimiento común y archivos propios de su personalidad.

3.2.1 Características y funcionalidades ofrecidas

El proyecto ProgramD, a pesar de ser clasificado como un intérprete de AIML, incorpora multitud de funcionalidades que ofrecen un valor añadido sobre este concepto posicionando a esta plataforma cerca de los servidores de bots.

Las características más destacables de programD son:

- Implementa todas las características de la versión 1.1 de AIML.
- Soporta un número ilimitado de bots y proporciona acceso separado a cada uno de ellos.
- Ejecución en la JVM o en un servidor de aplicaciones Tomcat para su acceso por consola o a través del navegador.
- Integración con sistemas de mensajería instantánea con comunicación en diversos protocolos.
- A través de los archivos de configuración permite habilitar diversas características y seleccionar entre distintos modos de funcionamiento.
- Servicio de seguimiento de archivos AIML, a través del módulo AIMLWatcher, para cargar en memoria las modificaciones en tiempo real, de manera que estén disponibles en el bot tan pronto como se produzcan.
- Módulo de pruebas para archivos AIML.
- Sistema de logs integrado con Log4j. A través de la definición de distintos Loggers permite la completa personalización de la información mostrada tanto en consola como en los archivos de logs, a través el archivo de configuración log4j.xml.
- Integración con mySql que permite incorporar el código AIML en una base de datos.
- Proporciona una API para la gestión individualizada e independiente de bots y archivos AIML.

3.2.2 Configurando programD

ProgramD permite configurar multitud de propiedades que determinan el comportamiento de la plataforma. Los valores por defecto de estas propiedades se ajustan al uso preferido de la plataforma, muchas de estas opciones de configuración sólo deberán ser modificadas para un uso muy particular de la plataforma.

La configuración se lleva a cabo a través de un conjunto de archivos XML situados en una carpeta determinada[12]. El archivo de configuración principal es core.xml, su localización es el único parámetro que necesita programD para ejecutarse, ya sea en la JVM o en un contenedor de aplicaciones.

3.2.2.1 Bots

ProgramD permite configurar un número ilimitado de bots. Cualquiera que sea la cantidad de bots a definir estos serán configurados en el archivo *bots.xml*. Para cada bot se define un identificador que será el utilizado para consultar al bot.

Cada bot se define por separado mediante el elemento XML *<bot>*. El elemento *<bot>* debe tener dos atributos: id y enabled. "id" es el identificador y "enabled" indica si el bot está habilitado o no.

Para cada bot se definen una serie de propiedades:

- El perfil del bot que contiene nombre, genero, aficiones, etc. Se defiende mediante el elemento *properties*.
- El valor por defecto de las variables AIML en el elemento *predicates*.
- Las sustituciones de secuencias de caracteres a realizar en la consulta antes de ser enviada al bot. Las sustituciones se utilizan para sustituir letras con acento por la misma letra sin acento, emoticonos por una palabra o frase que lo describe, abreviaturas por la frase sin abreviar, etc. Se definen en el elemento *subsitutions*.
- Los delimitadores de frase en el elemento *sentence-spliters*. Si una consulta contiene algún carácter o secuencia de caracteres definida como delimitador en la práctica se realizan varios consultas sucesivas.
- Los conectores con programas de mensajería y chat en el elemento *listeners*.

[12] En realidad no es necesario que todos se encuentren en la misma carpeta, en core.xml se puede seleccionar la localización de algunos de los demás archivos.

- Los archivos AIML que se deben cargar a través de un conjunto de elementos *learn*. El contenido de estos elementos es el localizador que identifica a un archivo AIML o conjunto de archivos AIML. El localizador se resuelve traduciéndolo a JNDI [17] y acepta direcciones relativas a la localización del fichero bots.xml, identificadores absolutos e identificadores que incluyan el protocolo de acceso[13]. Si se trabaja en sistema operativo Linux existe confusión en la resolución de las rutas absolutas. Por ejemplo, /var/aiml/some.aiml es resulto como una ruta relativa. En estos casos se recomienda indicar el protocolo de acceso que se utiliza, de manera que la dirección file:/var/aiml/some.aiml resuelve correctamente el archivo. Esto no ocurre en el equivalente para Windows C:/var/aiml/some.aiml.

En la Fig. 25 se muestra un ejemplo de configuración de bots.

```
<bot id="SampleBot" 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/AAA/*.aiml</learn>
  <learn>./aiml/alice/*.aiml</learn>
  <learn>file:/aiml/maria/*.aiml</learn>
</bot>
```

Fig. 25. Ejemplo elemento bot y su contenido, constituyen la definición de un bot.

La sintaxis de los archivos referenciados en cada uno de los elementos descritos es sencilla y fácilmente comprensible como para inferir los cambios a realizar a partir de la sintaxis proporcionada en los mismos. En cualquier caso para más información se puede consultar la guía detallada de inicio con programD [18].

[13] Esto incluye protocolos como file o http. Sin embargo no se puede indicar un protocolo con autenticación.

3.2.2.2 Multiplexores

La configuración del multiplexor a utilizar se realiza en el archivo core.xml. La diferencia entre los multiplexores ofrecidos radica en la forma de almacenar los predicados en el disco.

La forma óptima de almacenar estos predicados para su posterior análisis con herramientas de minería de datos es en una base de datos seleccionando el multiplexor `DBMultiplexor`. Esta situación no se contempla en el ámbito de este proyecto, por ello no se explica en este capítulo como configurar la base de datos.

El multiplexor por defecto, `FlatFileMultiplexor`, almacena los predicados en el disco en archivos asociados al nombre de usuario. Como se plantea en el apartado 3.2.3, esto constituye una limitación de programD pues estos se pueden llegar a acumular en el disco sin que haya programado ningún mecanismo para eliminarlos.

Por ello, como se describe más adelante, se define el multiplexor `ContextMultiplexor` que elimina los predicados antiguos del disco. Esta es una contribución del proyecto por lo que dicho multiplexor no está disponible con la distribución oficial de programD.

3.2.2.3 AIMLWatcher

El AIMLWatcher se encarga de comprobar si se han modificado los archivos AIML cargados, y en su caso cargar en memoria los cambios realizados. Esto hace que se actualicen automáticamente en memoria las modificaciones realizadas en el contenido de los archivos AIML.

Por motivos de seguridad AIMLWatcher sólo recarga el contenido de los archivos referenciados desde una directiva `learn` que hayan sido cargados anteriormente. De manera que aquellos archivos que se incluyan en un directorio referenciado una directiva `learn`, pero que no hayan sido cargados no se verán su contenido cargado automáticamente.

El módulo AIMLWatcher puede ser habilitado así como la frecuencia con que se examinan los archivos AIML desde el archivo core.xml. La siguiente figura muestra el código de para habilitar el módulo AIMLWatcher con un periodo de refresco de dos segundos.

```
<!—Habilitar el AIML Watcher? [boolean].-->
  <entry key="programd.use-watcher">true</entry>
<!—Periodo de tiempo en ms que transcurre entre
comprobaciones [int].-->
  <entry key="programd.watcher.timer">2000</entry>
```

Fig. 26. Código para habilitar el AIMLWatcher con un periodo de dos segundos.

3.2.2.4 Política de mezclado

Es la propiedad que define la manera de proceder ante el intento de cargar en memoria una regla repetida. Esta situación se da cuando se intentan procesar un objeto *category* que tiene un patrón exactamente igual a otro ya cargado en memoria. Es importante recordar que para que esto ocurra el contenido de ambos objetos *pattern* debe ser el mismo y además deben estar asociados al mismo objeto *topic* y contener el mismo objeto *that*.

Se definen cuatro modos de actuación:

- Skip desestima el nuevo elemento category procesado y mantiene el category ya cargado en memoria.
- Overwrite carga el category procesado y sobrescribe la regla en memoria de manera que no queda rastro de dicha regla.
- Append carga el contenido del template de la regla recién procesada a continuación del template en memoria anteriormente
- Combine carga el contenido del template además del ya cargado en memoria, de tal manera que una consulta que dispare esta regla tendrá las mismas probabilidades de obtener cualquiera de los templates asociados. Es la política por defecto.

La política de mezclado se define a través de la propiedad programd.merge.policy.

```
<entry key="programd.merge.policy">
  [skip|overwrite|append|combine]
</entry>
```

A pesar de la aparente irrelevancia de esta propiedad, es importante para la herramienta gestión de conocimiento.

3.2.3 Limitaciones de programD

A pesar de que la complejidad del diseño de programD y de lo completo en cuanto a características ofrecidas, al diseñar la plataforma descrita en esta memoria se plantean requisitos que programD no satisface.

La ejecución en la JVM sólo permite su acceso local desde la consola por lo que su uso no permite dar servicios de bots. La ejecución como aplicación Web, en cambio, permite acceso a los bots cargados pero en ningún caso su administración[14]. Así, es válida para crear un servicio Web de consultas a bots, pero cualquier cambio en la configuración de los mismos exige el reinicio del sistema, y esta operación no es en todos los casos una solución viable.

La interfaz de acceso Web que se define en programD es un servlet que guarda la información de los predicados en variables de sesión. Por ello quedan restringidas las aplicaciones que definan un acceso al bot por medio de la tecnología Ajax ya que dicha información sólo está accesible para las páginas del lado del servidor.

Las opciones de configuración para el módulo de gestión de predicados (multiplexor) no incluyen una que conlleve la eliminación de los archivos de predicados antiguos o que han caducado. Si se plantea una interfaz de acceso en la que se el usuario obtiene su identificador como un número que identifica la sesión y que por lo tanto es distinto en cada acceso el número que se generan es muy elevado, y todos archivos generados quedan obsoletos en el instante en que se finaliza la sesión.

El módulo de pruebas es muy rígido pues sólo admite la comprobación de la respuesta dada por el bot para cada caso de prueba. Eso constituye una limitación porque en los diseños de sistemas AIML avanzados no todas las reglas tienen porque devolver una respuesta. Es frecuente definir reglas auxiliares que modifican el estado de algún predicado. Por otro lado AIML define etiquetas para seleccionar una respuesta de una entre un conjunto de respuestas para más variedad al acervo del bot. En ninguno de estos dos casos el módulo de pruebas funciona correctamente.

AIML no define en sus especificaciones un modo acceso a información dinámica o cualquier otro recurso en la red. Este podría ser el caso de un usuario que pregunta por el próximo concierto de un determinado artista, de manera que el bot obtenga la fecha de dicho concierto de un recurso Web. No existe una forma sencilla de incluir esta información. Por medio del elemento <*javascript*>[15] se puede generar un script que

[14] Propiedad que si está disponible en la ejecución por consola.
[15] Este elemento no fue descrito en el apartado de sintaxis de AIML pues no ha sido utilizado en el proyecto y no está soportado por programD.

realice esa tarea, pero no es una opción viable pues dicho script no puede realizar consultas complejas, ni consultas a bases de datos y además no es cacheable.

Por otro lado, en la etapa de evaluación de la plataforma se han encontrado diversos errores relacionados con la carga dinámica de archivos aiml, característica indispensable para el desarrollo de la herramienta de gestión de conocimiento.

Todo ello ha llevado al desarrollo de un servidor de bots como se describe en el apartado 5.2 y que, en cierta medida, constituye un envoltorio de programD al que se añaden algunas funcionalidades para solventar estas limitaciones.

Capítulo 4

Análisis y diseño de escenarios

El objetivo fundamental de esta fase del desarrollo es establecer los requisitos que deben cumplir los módulos desarrollados en el proyecto. Durante esta etapa se utilizarán las herramientas usuales en ingeniería del software, como los diagramas UML de casos de uso y tablas para especificar estos casos de uso y requisitos.

En los últimos apartados, y a partir de la información obtenida en los casos de uso, se realiza un estudio de las características que deben tener los casos de prueba y el código AIML generado.

4.1 Casos de uso

Esta sección pretende identificar los casos de uso normales en el sistema, con el objeto de obtener una especificación completa de la utilización del mismo que permita establecer el listado completo de requisitos a cumplir.

En primer lugar, se utilizarán los diagramas UML de casos de uso junto a la descripción informal de los mismos, que permiten establecer los actores que interactúan con el sistema y las relaciones entre ambos. Estos diagramas darán lugar a tablas con la especificación completa de los casos de uso, cuya principal utilidad será la de establecer los requisitos de usuario.

4.1.1 Diccionario de actores

En la siguiente tabla se recogen los actores primarios y secundarios que intervienen en los casos de uso que se especifican más adelante:

Identificador de actor	Papel	Descripción
ACT-1	Usuario del sistema.	Usuario que se conecta, generalmente a través del sitio Web del bot para conversar con él.
ACT-2	Administrador BC.	Usuario administrador del servidor de bots. Tiene permisos para acceder a todas las opciones de administración del sistema
ACT-3	Bot	Un bot cargado en el sistema.

4.1.2 Casos de uso relativos al módulo gsiBotServer

En este apartado se agrupan todos aquellos casos de uso que son propios del servidor de bots. Muchos casos de uso encuadrados en otros apartados también utilizan este servidor como actor involucrado pero al no ser el actor principal no son incluidos en este apartado.

4.1.2.1 Casos de uso

- **CU-1: Hablar con un bot**

Cuando existen uno o más bots desplegados en el servidor un número ilimitado de usuarios de internet pueden conversar con cada uno de ellos. En este caso de uso no se distingue el modo de acceso al bot.

Identificador:	CU-1
Nombre:	Hablar con un bot

Descripción: Permite hablar con un bot cargado en el servidor.
Actores: Usuario del sistema.
Precondiciones: -
Flujo Normal: 1. El usuario utiliza el navegador para acceder a la interfaz de conversación. 2. El usuario escribe la consulta en el cuadro de texto y pulsa enviar. 3. El sistema traslada la consulta al bot, obtiene la respuesta y la presenta.
Flujo Alternativo: 3. En caso de que el bot no esté disponible se devolverá un mensaje de error describiendo la situación.
Poscondiciones: El estado de la conversación con el usuario ha sido guardado en el sistema. Se ha generado una entrada en el archivo de logs.

- **CU-2: Cargar un nuevo bot en el servidor**

Identificador:	CU-2
Nombre:	Cargar un nuevo bot en el servidor
Descripción: Permite a un usuario (con permisos) crear un nuevo bot con el que se podrá conversar dirigiéndose a él mediante su identificador de bot.	
Actores: Administrador BC.	
Precondiciones: El usuario debe haber ingresado en el sistema.	

Flujo Normal: 1. El usuario selecciona la opción de agregar un nuevo bot. 2. El usuario introduce un nombre (identificador) para el bot. 3. El usuario selecciona un conjunto de archivos AIML. 4. El sistema genera un archivo de configuración para el bot. 5. El sistema crea una carpeta específica para el bot y copia el archivo de configuración y los archivos AIML a la carpeta del bot.
Flujo Alternativo: 6. El usuario proporciona un archivo de configuración para el bot además de los archivos AIML. 7. El sistema comprueba la validez del archivo de configuración. + En caso de que el archivo no sea válido se interrumpe la operación e informa al usuario del error
Poscondiciones: El nuevo bot está listo para responder a las consultas de los usuarios y ha sido incluido en la lista de bots disponibles.

- **CU-3: Eliminar un bot del servidor**

Identificador:	CU-3
Nombre:	Eliminar un bot del servidor.
Descripción: Permite a un usuario (con permisos) eliminar un bot existente en el sistema.	
Actores: Administrador BC.	
Precondiciones: El usuario debe haber ingresado en el sistema.	
Flujo Normal: 1. El usuario selecciona la opción de eliminar un bot (este paso es irreversible y requiere confirmación). 2. El sistema elimina la carpeta que contiene toda la información del bot en el sistema, lo que incluye archivos de configuración, de propiedades y archivos AIML.	

Flujo Alternativo: 1. El usuario no confirma
Poscondiciones: El bot ya no está disponible y ha sido eliminado de la lista de bots del servidor.

- **CU-4: Modificar la configuración de un bot**

Todas las propiedades del bot pueden ser modificadas a través la interfaz Web sin necesidad de modificar los archivos de configuración en que se definen estas propiedades. En general sólo es necesario definir el valor de una propiedad del bot si esta propiedad se utiliza en algún punto del código AIML. Al igual que el resto de propiedades también se pueden modificar el conjunto de archivos AIML que forman la base de conocimiento de cada bot.

Como se explicará en uno de los anexos, la última versión del software del intérprete de AIML contiene algunos errores relativos al manejo de los archivos AIML cargados. Esto provoca que, en algunos casos, el proceso de edición de las propiedades englobe la eliminación total del bot del servidor y su sustitución por otro bot con la selección de propiedades y archivos AIML actualizada.

Identificador:	CU-4
Nombre:	Modificar la configuración de un bot.
Descripción: Permite a un usuario (con permisos) modificar la configuración de un bot cargado en el servidor. Esto incluye cambiar el nombre del mismo, los parámetros de configuración o los archivos AIML (tanto añadir como eliminar).	
Actores: Administrador BC.	
Precondiciones: El usuario debe haber ingresado en el sistema y el bot objetivo debe estar cargado en el servidor.	

Flujo Normal:

1. El usuario selecciona la opción de editar propiedades de bot.
2. El sistema muestra un formulario con las propiedades actuales del bot, incluyendo los archivos de configuración y AIML.
3. El usuario edita aquellas propiedades que desea modificar.
4. El sistema modifica la información relativa al bot guardada tanto en el servidor como en la instancia del propio bot. .

Flujo Alternativo:

5. El usuario modifica la información relativa a archivos AIML
   + Se produce un error al intentar recargar el corpus del bot.
6. El sistema reacciona ante el error comenzando el flujo de ejecución de eliminación de un bot y consecuentemente cargando de nuevo el bot con las propiedades modificadas introducidas por el usuario.

En este caso el tiempo de ejecución es significativamente mayor. Sin embargo, esto es aceptable ya que se consigue solucionar un estado de excepción.

Poscondiciones:

La información relativa al bot ha sido actualizada tanto en el servidor como en la instancia del bot correspondiente.

4.1.2.2 Diagrama resumen de los caso de uso.

En el siguiente diagrama se relacionan entre sí todos los casos de uso mostrados en los apartados anteriores.

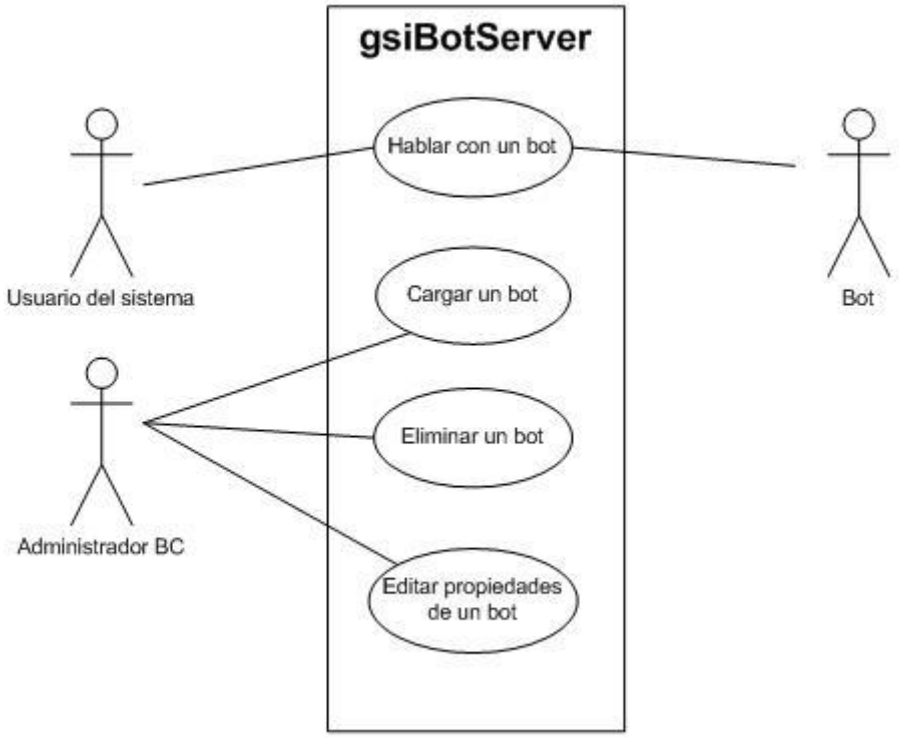

Fig. 27. Diagrama de casos de uso del servidor de bots.

4.1.3 Casos de uso relativos al módulo CorpusTester

En este apartado se agrupan aquellos casos de uso que son propios del módulo de pruebas automatizadas. Aunque un usuario puede ejecutar el módulo de pruebas automatizadas desde la interfaz Web de este, lo habitual es que sean otros agentes software los que utilicen el módulo de pruebas, por lo en muchos otros casos de uso no definidos en esta sección se incluirá al actor de software de pruebas,.

4.1.3.1 Casos de uso

- **CU-5: Explorar las pruebas definidas en un archivo de pruebas.**

Identificador:	CU-5
Nombre:	Explorar las pruebas definidas en un archivo de pruebas.
Descripción: Permite a un usuario explorar un archivo de pruebas y visualizar las condiciones que debe cumplir el bot en cada caso de prueba.	

Actores: Usuario del sistema.
Precondiciones: -
Flujo Normal: 1. El usuario selecciona la opción de explorar un archivo de pruebas. 2. El usuario selecciona un archivo de pruebas (KTF). 3. El sistema comprueba que la sintaxis del archivo es correcta. 4. El sistema muestra el conjunto de pruebas en estructura de árbol.
Flujo Alternativo: 3. La comprobación es errónea: la sintaxis del archivo de pruebas no es correcta, por lo que no se puede llevar a cabo la ejecución de los casos de uso. 4. El sistema muestra por pantalla el error en la sintaxis del archivo.
Poscondiciones: -

- **CU-6: Ejecución de las pruebas definidas en un archivo de pruebas.**

Identificador:	CU-6
Nombre:	Ejecución de las pruebas definidas en un archivo de pruebas.
Descripción: Permite a un usuario ejecutar un conjunto de pruebas definidos en un archivo de pruebas. Las pruebas deben ejecutarse sobre un bot objetivo.	
Actores: Administrador BC.	
Precondiciones: Al menos un bot debe estar cargado en el sistema.	

Flujo Normal:

5. El usuario selecciona la opción de ejecutar un conjunto de pruebas automatizadas.
6. El usuario selecciona un archivo de pruebas (KTF)
7. El sistema comprueba que la sintaxis del archivo es correcta y si el bot al que está asociado el archivo está disponible.
8. El sistema ejecuta el conjunto de casos de prueba definidos en el archivo KTF.
9. El sistema muestra en pantalla un informe de resultados.

Flujo Alternativo:

10. La comprobación es errónea: la sintaxis del archivo de pruebas no es correcta, por lo que no se puede llevar a cabo la ejecución de los casos de uso.
11. El sistema muestra por pantalla el error en la sintaxis del archivo.

Flujo Alternativo:

3. La comprobación es errónea: el bot para el que está diseñado el archivo de pruebas no existe en el servidor o no está disponible..
4. El sistema muestra por pantalla el error indicando la no disponibilidad del bot objetivo..

Poscondiciones:

-

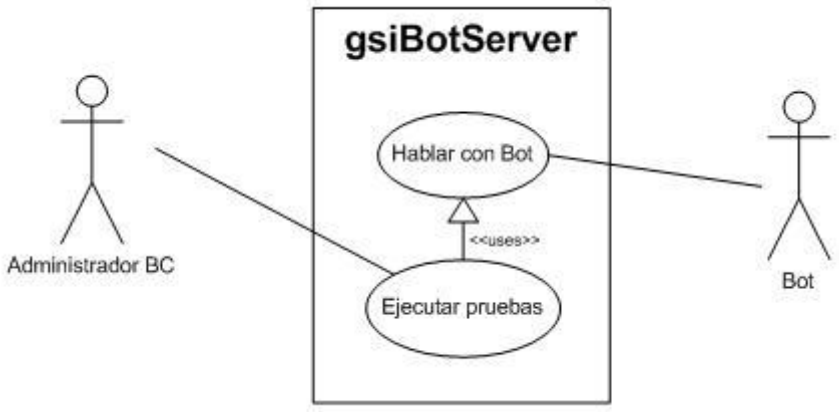

Fig. 28. Diagrama de caso de uso : Ejecución de las pruebas definidas en un archivo de pruebas.

- **CU-7: Cargar un bot con verificación de funcionamiento.**

El caso de uso descrito extiende al caso de uso CU-2. A la secuencia de ejecución normal del CU-2 se añade la ejecución posterior de un conjunto de pruebas se han

requerido al usuario como prueba del comportamiento del bot. Éste caso de uso añade fiabilidad al mencionado, puesto que al llevar a cabo la ejecución de la pruebas correspondientes a dicho archivo garantiza, por ejemplo, que se han incluido en la base de conocimiento del bot todos los archivos AIML requeridos.

Identificador:	CU-7
Nombre:	Cargar un bot con verificación de funcionamiento.

Descripción:

Permite a un usuario (con permisos) crear un nuevo bot con el que se podrá conversar dirigiéndose a él mediante su identificador de bot. Además, antes de completar el proceso se ejecutan un conjunto de pruebas que verifican el correcto funcionamiento del bot..

Actores:

Administrador BC

Precondiciones:

El usuario debe haber ingresado en el sistema.

Flujo Normal:

1. El usuario selecciona la opción de agregar un nuevo bot.
2. El usuario introduce un nombre (identificador) para el bot.
3. El usuario selecciona un conjunto de archivos AIML.
4. El usuario selecciona un archivo de pruebas KTF que se asociará al bot.
5. El sistema genera un archivo de configuración para el bot.
6. El sistema crea una carpeta específica para el bot y copia el archivo de configuración, el archivo de pruebas KTF y los archivos AIML a la carpeta del bot.
7. El software de ejecución de pruebas ejecuta los pasos correspondientes a la ejecución de las pruebas definidas en el archivo KTF que ha especificado el usuario.

Flujo Alternativo:

3. ‘El usuario proporciona un archivo de configuración para el bot además de los archivos AIML.
4. El usuario selecciona un archivo de pruebas KTF que se asociará al bot.
5. ‘El sistema comprueba la validez del archivo de configuración.
   + En caso de que el archivo no sea válido se interrumpe la operación e informa al usuario del error

Poscondiciones:
El nuevo bot está listo para responder a las consultas de los usuarios y ha sido incluido en la lista de bots disponibles. El archivo KTF ha sido asociado al bot.

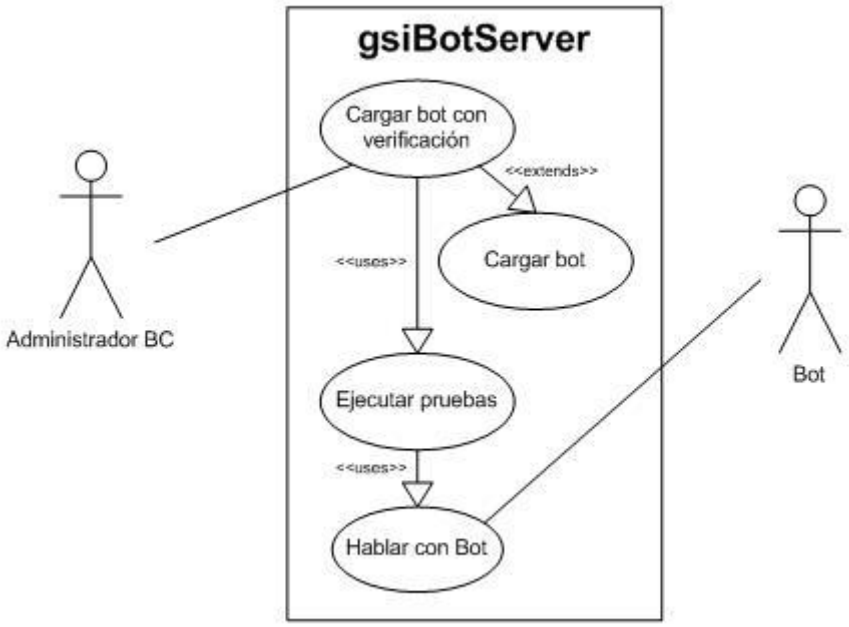

Fig. 29. Diagrama de caso de uso : Cargar un bot con verificación de funcionamiento.

- **CU-8: Modificar la configuración de un bot con verificación.**

El caso de uso descrito extiende al caso de uso CU-4. A la secuencia de ejecución normal del CU-4 se añade la ejecución posterior de un conjunto de pruebas que garantizan que las modificaciones realizadas no han repercutido en un malfuncionamiento del bot.

Identificador:	CU-8
Nombre:	Modificar la configuración de un bot con verificación.
Descripción: Permite a un usuario (con permisos) modificar la configuración de un bot cargado en el servidor. Esto incluye cambiar el nombre del mismo, los parámetros de configuración o los archivos AIML (tanto añadir como eliminar). Al ser una operación con verificación el paso final es la comprobación del correcto funcionamiento del bot con los cambios introducidos.	

Actores:

Administrador BC

Precondiciones:

El usuario debe haber ingresado en el sistema, el bot objetivo debe estar cargado en el servidor y tener asociado un archivo de pruebas.

Flujo Normal:

1. El usuario selecciona la opción de editar propiedades de bot.
2. El sistema muestra un formulario con las propiedades actuales del bot, incluyendo los archivos de configuración y AIML.
3. El usuario edita aquellas propiedades que desea modificar.
4. El sistema modifica la información relativa al bot guardada tanto en el servidor como en la instancia del propio bot.
5. El software de ejecución de pruebas ejecuta los pasos correspondientes a la ejecución de las pruebas definidas en el archivo KTF asociado al bot.

Flujo Alternativo:

4. El usuario modifica la información relativa a archivos AIML

   + Se produce un error al intentar recargar el corpus del bot.

   + El sistema reacciona ante el error comenzando el flujo de ejecución de eliminación de un bot y consecuentemente cargando de nuevo el bot con las propiedades modificadas introducidas por el usuario.

En este caso el tiempo de ejecución es significativamente mayor. Sin embargo, esto es aceptable ya que se consigue solucionar un estado de excepción.

Poscondiciones:

La información relativa al bot ha sido actualizada tanto en el servidor como en la instancia del bot correspondiente.

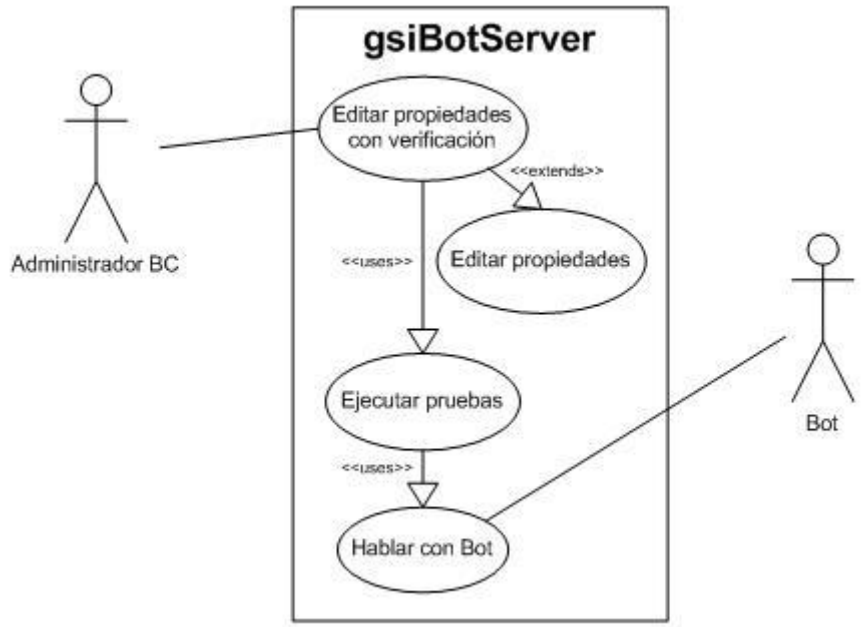

Fig. 30. Diagrama de caso de uso : Modificar la configuración de un bot con verificación de funcionamiento.

4.1.3.2 Diagrama resumen de los caso de uso.

En el siguiente diagrama se relacionan entre sí todos los casos de uso mostrados en los apartados anteriores.

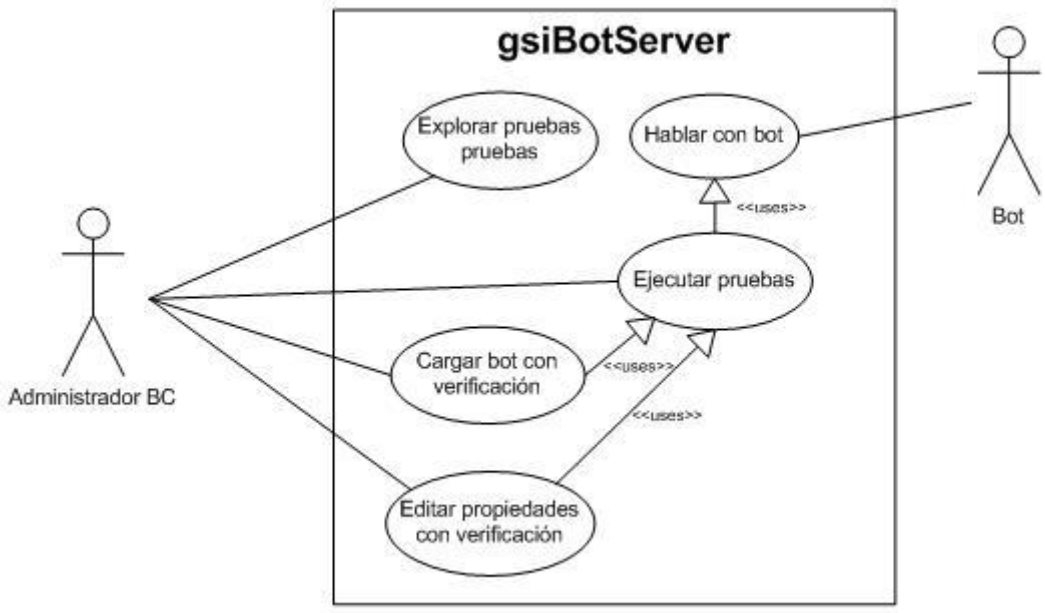

Fig. 31. Diagrama de casos de uso del corpus tester.

4.1.4 Casos de uso relativos al módulo de gestión de conocimiento

En este apartado se agrupan todos aquellos casos de uso que son propios del módulo de gestión de conocimiento.

- **CU-9: Agregar regla a la base de conocimiento.**

Identificador:	CU-9
Nombre:	Agregar una regla a la base de conocimiento.
Descripción: Mediante un asistente el usuario puede agregar una regla a la base de conocimiento sin necesidad de que escriba el código AIML.	
Actores: Administrador BC.	
Precondiciones: El usuario debe haber ingresado en el sistema. El bot al que se debe agregar el conocimiento debe existir en el sistema.	
Flujo Normal: 1. El usuario entra en el asistente para añadir conocimiento al bot. 2. El usuario sigue los pasos que le marca el asistente 3. El sistema deduce cual es el código AIML necesario y lo genera. 4. El sistema carga un bot en el servidor con la base de conocimiento del bot y las reglas adicionales generadas en el paso anterior. 5. El modulo de pruebas comprueba que no existen interferencias con el conocimiento anterior. 6. El usuario prueba el funcionamiento del nuevo conocimiento añadido y valida la operación. 7. El sistema transfiere el nuevo conocimiento a la base de conocimiento del bot	
Flujo Alternativo: 5. El módulo de pruebas detecta un error, informa al usuario y descarta los cambios.	
Poscondiciones: El conocimiento ha sido añadido a la base de conocimiento del bot. El bot con la base de conocimiento auxiliar no deja de estar disponible.	

4.2 Diseño de las pruebas de bot

El principal problema en la programación de una base de conocimiento es que a medida que esta se construye, incorporando reglas que funcionan correctamente por separado, existe el riesgo de que uno de estos añadidos provoque un error en una regla ya definida. Este problema que es intrínseco a la programación en sí, ya que escribir una línea de código incorrecta en un programa cualquiera puede provocar un resultado completamente imprevisible. No obstante, la definición de prioridades hecha en las especificaciones de AIML favorece esta situación, puesto que se da prioridad a aquel patrón que contiene su primera palabra en una posición más baja en la consulta, en lugar de a aquel que con mayor precisión se ajusta a la misma.

Ha medida que la base de conocimiento aumenta su tamaño, también aumenta la probabilidad de que añadir una nueva categoría interfiera con el conocimiento existente en la base de conocimiento y el bot no de la respuesta esperada a algunas consultas. Cuando esto ocurre decimos que la nueva entrada interfiere en el correcto funcionamiento de la base de conocimiento y por lo tanto la corrompe.

El hecho de que la nueva regla interfiera con una existente previamente no quiere decir que sea esta nueva regla la que es incorrecta., es posible que la regla que se introdujo anteriormente sea la causante de error.

4.2.1 Interferencia entre bases de conocimiento.

Debido a la definición de patrones utilizada en AIML existen patrones contenidos en otros, es decir, se pueden definir dos patrones A y B tal que cualquier frase que verifique A también verifique B; pero que existan frases que verifiquen B y no verifiquen A. En este caso decimos que el patrón A está contenido en B: *A es una concretización de B* o *B es una generalización de A*. Esta característica de los patrones de AIML permite crear patrones con distinto grado de selectividad, con la capacidad de ofrecer una respuesta general a una frase ambigua y una respuesta específica a una frase más concreta. Sin embargo, puede ser una fuente de errores difíciles de detectar si no se disponen de las herramientas adecuadas.

Hablando en términos de interferencia, se dice que el patrón B interfiere en el patrón A. Además, dado que cada patrón está asociado a una categoría, probar la interferencia entre patrones es equivalente a probar la interferencia entre categorías, y por tanto entre fragmentos de la base de conocimiento (o entre bases de conocimiento).

4.2.1.1 Corrupción de la base de conocimiento.

Cuando se añade una nueva categoría a una base de conocimiento, de forma que dicha categoría es una concretización de una categoría existente previamente en la base de conocimiento, se dice que la nueva categoría interfiere en la base de conocimiento. Sin embargo, el concepto de interferencia no es equivalente al de error, pues cuando se refina una base de conocimiento se añaden nuevas reglas para dar respuesta a un mayor abanico de consultas. Este proceso consiste en introducir interferencias "deseadas" en la base de conocimiento.

El concepto de error está ligado a los requisitos marcados para un bot, de manera que cuando se introduce una nueva categoría en la base de conocimiento, de manera que esta deja de cumplir alguno de los requisitos marcados para la misma decimos que interferencia introducida es un error y que corrompe la base de conocimiento. Por esta razón es importante precisar los requisitos de comportamiento del bot (corpus del bot), que deben ser definidos de manera formal, para que no haya lugar a malinterpretaciones y a su vez favorecer su comprensión por parte de agentes software. Más adelante se describe detalladamente cómo se realiza la especificación de los requisitos así como las características que deben tener estos.

Una vez definidos los requisitos para una base de conocimiento podemos determinar si una base de conocimiento está corrupta o no. Es importante remarcar que el concepto de corrupción en una base de conocimiento esta ligado a la hoja de requisitos. De manera que una misma base de conocimiento puede ser corrupta para una especificación de los requisitos y no serlo para otra distinta.

Se ha dicho que cada vez que se añade a la base de conocimiento una categoría que concretiza otra existente en la base de conocimiento existe la posibilidad de que la interferencia corrompa el corpus. Aunque no es tan intuitivo, también se pueden producir errores en la base de conocimiento cuando se introduce una nueva categoría que generaliza una categoría existente. En este caso es posible que con la nueva categoría con la pretensión de cubrir uno de los casos de uso definidos en el corpus pero que una de las categorías previamente existentes en la base de conocimiento oculte la respuesta. Por lo que la base de conocimiento queda corrupta del igual modo.

4.2.1.2 Estrategias para la detección de interferencias

Podemos seguir dos estrategias para determinar si existen interferencias entre patrones y por tanto entre categorías:

- Realizar un estudio de la base de conocimiento existente, analizando cada uno de los patrones para determinar si existen concretizaciones que afecten a la hoja de requisitos, o si existen patrones que oculten respuestas definidas en la hoja de requisitos.
- De forma empírica, realizar consultas al servidor de bots a partir de cada uno de los casos de uso definidos en la hoja de requisitos y realizando las comprobaciones para verificar si la respuesta obtenida es la esperada o no.

El diseño de la hoja de requisitos de Bot en casos de uso está orientado a una estrategia de detección de interferencias del segundo tipo, y es por tanto la estrategia utilizada en este proyecto.

4.2.2 Requisitos de una base de conocimiento: La hoja de requisitos de Bot

Es fundamental identificar y concretizar los requisitos que se desean para un bot para poder evaluar así su funcionamiento conforme a estas especificaciones. Una misma base de conocimiento puede funcionar bien o mal en función de los requisitos que a ésta se le exijan. Veamos el siguiente ejemplo: Si estamos creando un bot para atender visitas virtuales a la oficina de turismo de Madrid, el hecho de que el bot sepa en qué parada de metro se encuentra el museo del Prado es un requisito; y por lo tanto la base de conocimiento no funcionaría correctamente si el bot no pudiera contestar a la pregunta ¿En qué parada de metro está el museo del prado? Sin embargo, al crear un bot para la Universidad Politécnica de Madrid, con el propósito de resolver dudas a los nuevos alumnos, el que el bot conozca o no la parada de metro del museo del Prado es indiferente, por lo que no será uno de los requisitos[16].

Los requisitos de un bot deben plasmarse en una *hoja de requisitos de Bot*. La hoja de requisitos de un bot debe contener *todas* las frases de entrada a las que se pretende que el bot dé una respuesta, así como la respuesta que el bot debe dar en cada caso. Desde un punto de vista literal, es imposible que la hoja de especificaciones de un bot contenga todas las frases de entrada para las que el bot da respuesta. Cuando hablamos de "todas las frases de entrada" nos referimos a un conjunto muestra de frases de entrada significativas que abarquen **todos los conceptos** de los que debe hablar el bot. Continuando con el ejemplo anterior, el bot de la oficina de turismo de Madrid debe saber en qué línea de metro se encuentra el museo del Prado, cual es la dirección del museo y cual es el horario de apertura. Así la hoja de requisitos debe contener frases

[16] En última instancia, esta decisión es del cliente por lo que él es quien decide qué debe ser un requisito del bot.

similares a: "¿En qué línea de metro debo montarme para llegar al museo del prado?" "¿Cuál es la dirección del museo del prado?" "¿A partir de qué hora abre el Museo del Prado?"… Si la hoja de requisitos sólo contiene algunas de estas frases sería incompleta porque no abarcaría todos los conceptos.

Si la hoja de requisitos de bot sólo contiene la frase "¿Dónde está el museo del prado?" estaría incompleta porque no abarcaría los conceptos de horario de apertura y desplazamiento en metro; incluso sería ambiguo puesto que por ¿Dónde está el museo del prado? podemos responder también con el nombre de una estación de metro. Para hacer que la hoja de requisitos no sea ambigua debe contener varias frases que sirvan para decir lo mismo.

4.2.2.1 La hoja de requisitos como contrato con el cliente.

El concepto de requisito[17] introducido en el apartado anterior se asemeja al concepto de *requisito funcional* [19] tratado en ingeniería del software o *ingeniería de requisitos*. Una hoja de requisitos de Bot no incluye *requisitos no funcionales* [19] tales como restricciones en la tecnología utilizada, tiempo máximo de respuesta, volumen de respuestas procesadas por unidad de tiempo o requisitos de interfaces. Por supuesto estos requisitos existen para el proyecto de software realizado pero no son relevantes en esta sección.

Una hoja de requisitos de Bot recoge todos los requisitos funcionales que el cliente exige al Bot para aceptar que cumple el contrato. Junto con una especificación de los requisitos no funcionales constituyen el acuerdo entre la empresa desarrolladora del Sistema Bot y la empresa cliente.

"La especificación de requisitos es el punto de partida para desarrollar software" [20] por lo tanto la hoja de requisitos de Bot es el punto de partida para comenzar a desarrollar un Bot y está pensada para ser elaborada **antes** de comenzar con el desarrollo propio del corpus del Bot.

4.2.2.2 Características de los requisitos

Cada uno de los requisitos recogidos en la hoja de requisitos de Bot debe cumplir una serie de características:

- Conciso: Un requisito es conciso si es fácil de leer y entender. Su redacción debe ser simple y clara para aquellos que vayan a consultarlo en un futuro.

[17] Algunos autores traducen *requirement* en vez de por requisito por requerimiento, traducción más acertada para *request*.

- Completo: Un requisito está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
- Consistente: Un requisito es consistente si no es contradictorio con otro requisito.
- No ambiguo: Un requisito no es ambiguo cuando tiene una sola interpretación. El lenguaje usado en su definición, no debe causar confusiones al lector.
- Verificable: Un requisito es verificable cuando puede ser cuantificado de manera que permita hacer uso de los siguientes métodos de verificación: inspección, análisis, demostración o pruebas.

El diseño de la estructura de la hoja de requisitos de Bot se realiza teniendo en cuenta estas características de manera que cada uno de los requisitos las cumpla dentro de la estructura definida.

4.2.2.3 Estructura de una hoja de requisitos

Los requisitos se describen en la hoja de requisitos de Bot como un conjunto de casos de uso. Los casos de uso para un sistema Bot carecen de complejidad: El usuario introduce una frase en el sistema para la que recibe una respuesta.

En términos generales, cada de uso tiene una parte de estímulos de entrada que modifican el estado del Bot (lo que introduce el usuario) y una parte de comprobación que se compara con el estado en que queda el Bot después de aplicarle los estímulos de entrada. En el caso sencillo de la dupla frase_de_entrada-frase_de_salida el único estímulo de entrada es la frase introducida por el usuario y la única comprobación del estado del Bot es la comprobación sobre la respuesta dada.

La siguiente figura muestra un ejemplo de caso de uso más elaborado:

Parte de entrada	Parte de comprobación
"¿Cómo se llega al Museo del Prado?"	Respuesta:" ¿Andando o en metro?"
	topic: "Museo Prado"
	test: "Museo Prado Dirección1"

Fig. 32. Caso de uso de ejemplo de una hoja de requisitos.

La parte de estímulos de entrada –en este caso- contiene una entrada de usuario "¿Cómo se llega al Museo del Prado?" y se realizan tres comprobaciones. La ejecución

de este caso de uso envía al bot la entrada "¿Cómo se llega al Museo del Prado?" y comprueba que la respuesta sea" ¿Andando o en metro?", que el atributo topic[18] sea"Museo Prado" y que el atributo test[19] tenga por valor"Museo Prado Dirección1".

Para el cliente este caso de uso puede ser expresado como: Cuando el usuario introduce "¿Cómo se llega al Museo del Prado?" el Bot debe responder "¿Andando o en metro?" y debe saber que se está refiriendo a El Museo del Prado (expresado en la variable topic).

En el ejemplo anterior sólo se considera un estímulo de entrada, pero puede ocurrir que se encadenen varios estímulos de entrada para comprobar el funcionamiento con mantenimiento de la conversación. La siguiente figura muestra un ejemplo de caso de uso con varios estímulos de entrada:

Parte de entrada	Parte de comprobación
"¿Cómo se llega al Museo del Prado?" "En metro"	Respuesta:"Debe coger la línea 2 y bajarse en la parada Banco de España."
	topic: "Museo Prado"
	test: "Museo Prado Metro1"

Fig. 33. Caso de uso de ejemplo de una hoja de requisitos con dos elementos en la parte de entrada.

La parte de estímulos de entrada contiene dos entradas de usuario "¿Cómo se llega al Museo del Prado?" (a la que el bot respondería como hemos visto en el ejemplo anterior) y "En metro" que sería una de las posibles respuestas coherentes del usuario.

Para que esta hoja de requisitos fuera completa debería contener, por lo menos, otro caso de uso con las entradas de usuario "¿Cómo se llega al Museo del Prado?" y "Andando".

4.2.2.4 Características de una hoja de requisitos

Además de las características que deben cumplir cada uno de los requisitos recogidos en la hoja de requisitos de Bot, ésta debe ser completa, concreta y exhaustiva.

- Una hoja de requisitos es completa si no existe ningún requisito para el que no exista al menos un caso de uso.

[18] El atributo topic un atributo predefinido en AIML que contiene el tema/asunto (topic) sobre el que se está conversando.
[19] El atributo teste es un atributo AIML definido por el usuario y por lo tanto el nombre es elegido por éste.

- Es concreta si contiene casos de uso que aseguran que no se responden cosas cuando no es necesario responderlas.
- Una hoja de requisitos es exhaustiva si para cada concepto del que se deba hablar se incluyen tantos casos de uso como sea necesario para agotar las diferentes maneras de abordar el concepto.

Para explicar el concepto de hoja de requisitos exhaustiva veamos un ejemplo. Supongamos que estamos creando una base de conocimiento para actuar de representante de un grupo de música de manera que el Bot debe conocer el número de teléfono del *manager* del grupo de música para ofrecerlo a aquellos usuarios que lo soliciten. El concepto del que debe saber hablar el Bot es del número de teléfono del manager. Sin embargo hay varias formas de referirse a este número de teléfono: "Dame un teléfono de contacto" "¿Cuál es el número de teléfono de contacto?" "¿Puedes darme el móvil del representante?" De manera que para que una hoja de requisitos contenga este concepto debe tener todas estas variaciones.

4.3 Diseño de plantillas de código AIML

Cómo se ha explicado, un corpus escrito en AIML está compuesto por reglas concretadas en tags `<category>`. Cada regla se ajusta a un patrón determinado definido en el `<pattern>`, si dicho *pattern* se cumple produce una acción definida en el `<template>`. Sin embargo, estos tags son capaces de reconocer patrones muy concretos de las expresiones introducidas por el usuario. De manera que pequeñas variaciones en dichas expresiones, como puede ser conmutar el orden de dos palabras reconocidas por el patrón, hace que la frase ya no se ajusten al patrón.

Por ello, en el ámbito de este proyecto se definen lo que llamamos patrones AIML. Para diferenciarlos de los patrones de las category nos referiremos a estos como "*patrones category*" y a los patrones AIML como "*patrones*" simplemente. Un patrón AIML es un conjunto de reglas o *patrones category* con el propósito de entender una expresión concreta pero permitiendo más variaciones en la misma de las que permite un *patrón category*.

4.3.1 El patrón *Contains*

Al definir el corpus de un bot se definen un conjunto de frases para las que el bot debe dar una determinada respuesta. En el proceso habitual de desarrollo de la base de conocimiento para el bot se extraen las palabras clave de cada una de las consultas definidas en el corpus y se escribe el código AIML necesario para reconocer consultas que contengan dichas palabras clave. Es decir, se escribe un conjunto de reglas AIML

de tal forma que se ajusten a las distintas combinaciones en que pueden aparecer las palabras clave ya sea con otras palabras entre ellas o no.

A este tipo de patrones se les denomina –en el ámbito de este proyecto- patrón '***contains***' (contiene), y son los más comunes en los archivos AIML. Estos patrones buscan que en la frase introducida por el usuario esté contenida una palabra o conjunto de palabras. O de forma más general busca que en la frase introducida por el usuario esté contenida una frase/expresión o conjunto de frases/expresiones. Así se define el patrón como: ***contains (espresions...)***.

Por ejemplo, el patrón `contains (foo, boo)` busca que en la frase introducida por el usuario estén contenidas las expresiones *foo* y *boo* **en el orden dado**. Es decir, que en la expresión del usuario se busca la expresión *foo* y en caso de que se encuentre se busca la expresión *boo* de ahí en adelante. Es importante recalcar que, por lo tanto, una frase no puede verificar a la vez los patrones `contains (foo, boo)` y `contains (boo, foo)` por el hecho de que el orden de los parámetros importa.

El número de parámetros que se definen para este patrón no está prefijado de manera que podemos definir los patrones `contains (first, second, third)` y `contains (first second, third)` con 3 y 2 parámetros respectivamente[20]. De manera que la cadena de entrada *"first hello second hello third"* se ajusta al primer patrón y no al segundo, así como la cadena de entrada *"first second hello third"* se ajusta a ambos patrones.

Como ya hemos explicado el orden en que se definen los *patrones category* en el fichero AIML no es relevante pues el motor de programD elige siempre el patrón más concreto al que se ajusta la frase de entrada. Así en el ejemplo anterior, si ambos patrones están presentes en la base de conocimiento AIML se verificará el segundo –`contains (first second, third)`– de acuerdo con las reglas de prioridad descritas en 3.1.4.3.

En cualquier caso no hay que olvidar que esta definición de patrones no es más que una abstracción, lo que podríamos llamar *meta*-patrones de manera que cada uno de estos patrones hay que traducirlo a un conjunto de patrones category, ya que no son elementos AIML y no hay un tag XML que los englobe.

[20] Nótese la separación por comas de los parámetros: si no existe el carácter ‘,’ es que toda la expresión forma parte del mismo patrón.

Para facilitar la comprensión del código, el módulo de generación de código AIML descrito en 5.4 los delimita mediante comentarios y mantiene un orden dentro de los patrones category y un sangrado adecuado, tal y como se ejemplifica a continuación.

```
<!-- @pattern_begin contains(foo bar) -->
<!-- Patrón AIML contains de 1 parâmetro: contains (foo bar) -->
  <category>
    <pattern>FOO BAR</pattern>
    <template>
      <think>
        <set name="topic">whatever</set>
        <set name="test">...</set>
      </think>
      Esta es la respuesta dada a foo bar en el ejemplo.
    </template>
  </category>

  <category><pattern>* FOO BAR *</pattern>
    <template><srai>FOO BAR</srai></template>
  </category>
  <category><pattern>FOO BAR *</pattern>
    <template><srai>FOO BAR</srai></template>
  </category>
  <category><pattern>* FOO BAR</pattern>
    <template><srai>FOO BAR</srai></template>
  </category>
<!-- @pattern_end contains(foo bar) -->
```

Fig. 34. Código AIML para el patrón de un parámetro contains (foo bar).

```
<!-- @pattern_begin contains(foo, bar) -->
<!-- Patrón AIML contains de 2 parámetro: contains (foo, bar) -->
  <category>
    <pattern>FOO BAR</pattern>
    <template>
      <think>
        <set name="topic">whatever</set>
        <set name="test">...</set>
      </think>
      Esta es la respuesta dada a foo bar en el ejemplo.
    </template>
  </category>

  <category><pattern>* FOO BAR *</pattern>
    <template><srai>FOO BAR</srai></template>
  </category>
```

```
<category><pattern>FOO BAR *</pattern>
  <template><srai>FOO BAR</srai></template>
</category>
<category><pattern>* FOO BAR</pattern>
  <template><srai>FOO BAR</srai></template>
</category>

<category><pattern>FOO * BAR</pattern>
  <template><srai>FOO BAR</srai></template>
</category>
<category><pattern>* FOO * BAR *</pattern>
  <template><srai>FOO BAR</srai></template>
</category>
<category><pattern>FOO * BAR *</pattern>
  <template><srai>FOO BAR</srai></template>
</category>
<category><pattern>* FOO * BAR</pattern>
  <template><srai>FOO BAR</srai></template>
</category>

<!-- @pattern_end contains(foo, bar) -->
```

Fig. 35. Código AIML para el patrón de 2 parámetros contains (foo, bar).

La estructura de los patrones AIML se define en torno a una regla "principal", que por simplicidad se encuentra el primero de todas las reglas que forman el patrón (aunque no es necesario). Dicho patrón contiene el template con la respuesta y todos los demás redirigen la salida mediante la etiqueta *<srai>* a ese *<category>*. A partir de ahora a este elemento se le denomina *category principal* del patrón. El número de reglas necesarios para generar cada un patrón contains es $2^{(núm\_parametros\ +1)}$ (incluyendo el category principal). Así se ve que es imposible crear patrones contains de un tamaño grande.

Si se observan detalladamente los ejemplos de las Fig. 34 y Fig. 35 se puede ver que existe una pequeña complicación con esta forma de definir los patrones puesto que pueden existir colisiones entre category principales. En el caso expuesto en los ejemplos los dos category principales tienen el mismo patrón category de manera que de estar ambos contenidos en el mismo corpus AIML existirá un conflicto. Todo el conjunto de patrones category del primer ejemplo (Fig. 34) está contenido en el conjunto de patrones del segundo ejemplo (Fig. 35). Puesto que no se contempla esta posibilidad en la definición de AIML hay que comprobar el comportamiento del motor de AIML

utilizado. En el caso de programD, cuando se da esta situación se actúa según se indica en la política de mezclado tal y como se describe en 3.2.2.4.

Para dar solución a dicha situación en el apartado siguiente se define el patrón *contains_strict*. Sin embargo, se incluye sólo a modo teórico, pues en la implementación del módulo de gestión de conocimiento que se describe en 5.4 se sigue una política con protección contra este tipo de situación[21].

4.3.1.1 Variaciones del patrón contains: *contains_strict*

El patrón *contains_strict* se define como aquel que busca que en la frase introducida estén contenidas todas las palabras dadas como parámetros, en el orden establecido y **con otras palabras o caracteres distintos de caracteres en blanco entre ellas**. Así el patrón contains_strict con los mismos parámetros que el de la Fig. 35 será:

```
<!-- @pattern_begin contains_strict(foo, bar) -->
<!-- Patrón AIML contains de 2 parámetro: contains_strict (foo, bar)
-->
  <category>
    <pattern>FOO * BAR</pattern>
    <template>
      <think>
        <set name="topic">whatever</set>
        <set name="test">...</set>
      </think>
      Esta es la respuesta dada a foo bar en el ejemplo.
    </template>
  </category>

  <category><pattern>* FOO * BAR *</pattern>
    <template><srai>FOO BAR</srai></template>
  </category>
  <category><pattern>FOO * BAR *</pattern>
    <template><srai>FOO BAR</srai></template>
  </category>
  <category><pattern>* FOO * BAR</pattern>
    <template><srai>FOO BAR</srai></template>
  </category>

<!-- @pattern_end contains_strict(foo, bar) -->
```

[21] En cualquier caso, si se da esta situación de conflicto en una base de conocimiento se estará incurriendo en una mala práctica pues no es aceptable definir un patrón de un único parámetro –de varias palabras- y además pretender definir otro distinto que contenga al anterior para el que se exija un alto nivel de concreción.

Fig. 36. Código AIML para el patrón de 2 parámetros contains_strict (foo, bar).

Para patrones contains_strict con más de dos parámetros es importante remarcar que se exigirá que haya palabras o caracteres distintos de caracteres en blanco entre **todos** los parámetros. El patrón contais_strict de un único parámetro es equivalente al patrón contains con el mismo parámetro. De esta forma un patrón contains puede quedar perfectamente definido como un conjunto de patrones contains_strict.

`contains (first, second)`
`  => Un ejemplo;`
	`contains_strict (first, second)`
	`  => Un ejemplo;`
	`contains_strict (first second)`
	`  => Un ejemplo;`

`contains (first, second, third)`
`  => Otro ejemplo;`
	`contains_strict (first, second, third)`
	`  => Otro ejemplo;`
	`contains_strict (first second, third)`
	`  => Otro ejemplo;`
	`contains_strict (first, second third)`
	`  => Otro ejemplo;`
	`contains_strict (first second third)`
	`  => Otro ejemplo;`

4.3.1.2 Variaciones del patrón contains: contains_commutative.

En algunos casos es necesario escribir un patrón que no tenga en cuenta el orden de aparición de los parámetros en la frase. Así se define el patrón *contains_commutative* como aquel que busca que en la frase de entrada estén contenidas todas las frases/expresiones dadas como parámetro **sin importar el orden de aparición**. Aunque puede no parecer evidente este patrón es mucho menos usado que el patrón contains definido pues ofrece un grado de libertad que en muchos casos no es necesario, y en algunos otros perjudicial por ofrecer conflictos con otras reglas.

Cuando sólo el patrón acepta sólo dos parámetros: *contains_commutative (foo, boo) => respuesta*, se puede escribir como la combinación de dos patrones contains: *contains (foo, boo) => respuesta; contains (boo, foo) => respuesta*. Sin embargo, cuando el número de parámetros aumenta linealmente el número de patrones category necesarios para lograr la equivalencia lo hace a razón de potencia por factorial. Un patrón contains_conmutative contiene $2^{(num\_par\acute{a}metros+1)} \cdot n\acute{u}m\_par\acute{a}metros!$. Por ello es casi impensable considerar patrones de este tipo de más de tres parámetros.

4.3.2 Patrones de reemplazo o prioritarios

Algunas reglas no producen en su *<template>* una respuesta como consecuencia a la verificación del *<pattern>*, sino que redirigen a otro patrón. En los ejemplos de las figuras anteriores el patrón principal contiene en su elemento *<template>* una respuesta, mientras que los demás contienen una redirección a esta regla. Este comportamiento también se puede encontrar en conjuntos de patrones category que formen patrones AIML. En ese caso **todos** los patrones category, incluido el patrón principal, generan una redirección.

En estos casos el resultado de la ejecución de la regla no es una respuesta en sí sino el reemplazo de algunas palabras o expresiones contenidas en la consulta por otras distintas. Tal y como se muestra en la Fig. 37, estos remplazos se realizan mediante redirecciones utilizando el elemento *<srai>*, por lo que a estos de patrones también se les llama patrones de redirección.

```
<!-- Blog (Blogs) -->
  <category><pattern>BLOGS</pattern>
    <template><srai>BLOG</srai></template>
  </category>

  <category><pattern>_ BLOGS _</pattern>
    <template>
      <srai><star index="1"/> BLOG <star index="2"/></srai>
    </template>
  </category>

  <category><pattern>BLOGS _</pattern>
    <template><srai>BLOG <star/></srai></template>
  </category>

  <category><pattern>_ BLOGS</pattern>
    <template><srai><star/> BLOG</srai></template>
  </category>
```

Fig. 37. Ejemplo de un patrón de reemplazo.

En el ejemplo de la Fig. 37 todos los <category> realizan una redirección con *<srai>*. A diferencia de los ejemplos de de las figuras anteriores, la redirección no es en todos los casos la misma sino que se incluye el contenido de los comodines, por medio de *<star/>* y *<star index="1!/>*. Es decir, estamos sustituyendo en la

entrada la expresión dada en el *pattern* por la expresión dada en el *template*. Esto es por tanto muy útil para hacer patrones para reconocer plurales, sinónimos, tiempos verbales...

En el ejemplo también se observa que el comodín utilizado es '_', el comodín de máxima prioridad, puesto que lo que se pretende en modificar la entrada de usuario y volver a evaluarla, antes de producir la salida. Éste hecho les da el apelativo prioritario a este tipo de patrones.

Se pueden agrupar los patrones de redirección o prioritarios en tres tipos: el patrón replace, el patrón delete y patrones de redirección dependientes de la posición.

4.3.2.1 El patrón replace

El patrón replace pretende sustituir una palabra por otra antes de procesar o evaluar la entrada de usuario. Se define como *replace (regex, replacement)*, donde *regex* es la cadena que se desea sustituir por la cadena dada en *replacement*. Así la Fig. 37 corresponde al patrón *replace (blogs, blog)*. Este es el patrón que se utiliza para incluir consideraciones acerca del número de las palabras o tiempos verbales. Algunos autores también defienden su uso para la inclusión de sinónimos en la base de conocimiento, pero dada la riqueza del lenguaje y la inexistencia de 'sinónimos absolutos' que sean válidos independientemente del contexto, sólo se recomienda su uso cuando el corpus es acotado y no se pretenda escalarlo.

Se define una variación del patrón replace para aquellos casos en que el parámetro regex tenga más de una palabra y se desee incluir un comodín entre ellas. Sin embargo, es un caso muy específico, de reducida utilidad y por tanto simplemente se presenta a modo teórico en la figura siguiente. Este patrón acarrea el problema de no saber que hacer con el contenido comprendido entre las dos palabras del patrón. En el ejemplo siguiente se ha optado por desechar dicha información.

```
<!-- @pattern_begin replace(copia seguridad, backup) -->
  <category><pattern>COPIA SEGURIDAD</pattern>
    <template><srai>BACKUP</srai></template>
  </category>
  <category><pattern>_ COPIA SEGURIDAD _</pattern>
    <template>
      <srai><star index="1"/> BACKUP <star index="2"/></srai>
    </template>
  </category>
```

```
  <category><pattern>COPIA SEGURIDAD _</pattern>
    <template><srai>BACKUP <star/></srai></template>
  </category>
  <category><pattern>_ COPIA SEGURIDAD</pattern>
    <template><srai><star/> BACKUP</srai></template>
  </category>

  <category><pattern>COPIA _ SEGURIDAD</pattern>
    <template><srai>BACKUP</srai></template>
  </category>
  <category><pattern>_ COPIA _ SEGURIDAD _</pattern>
    <template>
      <srai><star index="1"/> BACKUP <star index="3"/></srai>
    </template>
  </category>

  <category><pattern>COPIA _ SEGURIDAD _</pattern>
    <template><srai>BACKUP <star index="2"/></srai></template>
  </category>
  <category><pattern>_ COPIA _ SEGURIDAD</pattern>
    <template><srai><star index="1"/> BACKUP</srai></template>
  </category>
<!-- @pattern_ends replace(copia seguridad, backup) -->
```

Fig. 38. Código AIML para el patrón de replace (copia seguridad, backup).

4.3.2.2 El patrón delete

Al trabajar con AIML es frecuente desechar palabras de la frase de entrada antes de evaluarla por no aportan información relevante. Este es el caso de los determinantes, muchos adverbios y preposiciones. También es frecuente que al desarrollar un *bot de nicho*, u orientados a un área de conversación acotada suele ocurrir que hay palabras que no aportan información en ese contexto y por tanto se desea eliminarles. Con este fin se define el patrón *delete (regex)*.

```
<!-- Delete EL -->
  <category>
    <pattern>EL _</pattern>
    <template><srai><star/></srai></template>
  </category>

  <category>
```

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

Fig. 39. Ejemplo de patrón delete.

4.3.2.3 Patrones de remplazo dependientes de posición

Si se observa la estructura de los patrones mostrados en la Fig. 38 y Fig. 39 se observa que el patrón mostrado en la Fig. 38 reemplaza el *regex* cuando este se encuentra en cualquier posición de la frase introducida por el usuario. Sin embargo, el patrón mostrado en la Fig. 39 elimina el regex a menos que éste sea final de frase (ya que no existe ninguna regla con el patrón "**_ EL**"). Este es un ejemplo real en el que se elimina el determinante EL, que en ningún caso puede aparecer al final de la frase, por lo que por eficiencia no tiene sentido incluir un patrón para ese caso. Por tanto, en algunos casos puede ser necesario considerar un patrón que sólo realice el reemplazo si la palabra se encuentra en una determinada posición: al principio de la frase, al final, no al final, etc.

Con esta motivación se definen los cuatro patrones siguientes:

- *delete_beggining (regex, replacement)*. Realiza la reducción sólo si el regex se encuentra al inicio de la consulta al bot. Éste patrón es útil para eliminar introducciones a peticiones (Querría, deseo, cuéntame...) o introducciones propias de una conversación (Mas, pues, entonces, sin embargo...).

- *delete_not_beggining (regex, replacement)*. Elimina el regex a menos que este se encuentre al inicio de la consulta. Éste patrón es útil para eliminar palabras que sólo ofrecen significado si se encuentran al inicio de la frase introducida. Por ejemplo, éste es el caso de *Qué, Quién, Cómo...* que a la vez pueden trabajar como adverbios de unión de frases conjugadas.

- *delete_not_endding (regex, replacement)*. Elimina el regex a menos que este se encuentre al final de la consulta al bot. Éste patrón es útil para eliminar determinantes y preposiciones, que nunca van a aparecer al final de una frase.

- *replace_strict(regex, replacement)*. Realiza el reemplazo sólo si el regex es la totalidad de la frase introducida por el usuario. Es un patrón utilizado en situaciones muy concretas; por ejemplo en el caso en que el

usuario introduzca *'Claro'* o '*Por supuesto*' entendidos como respuestas afirmativas a una pregunta formulada por el bot.

En la figura siguiente se muestra la estructura de cada uno de estos patrones de reemplazo:

```
<!-- @pattern_begin delete_beggining (querria) -->
  <category><pattern>QUERRIA _</pattern>
    <template><srai><star/></srai></template>
  </category>
<!-- @pattern_end delete_beggining (querria) -->

<!-- @pattern_begin delete_not_beggining (que)-->
  <category><pattern>_ QUE</pattern>
    <template><srai><star/></srai></template>
  </category>
  <category><pattern>_ QUE _</pattern>
    <template>
      <srai><star index="1"/><star index="2"/></srai>
    </template>
  </category>
<!-- @pattern_end delete_not_beggining (que) -->

<!-- @pattern_begin delete_not_ending (los) -->
  <category><pattern>LOS _</pattern>
    <template><srai><star/></srai></template>
  </category>
  <category><pattern>_ LOS _</pattern>
    <template>
      <srai><star index="1"/><star index="2"/></srai>
    </template>
  </category>
<!-- @pattern_end delete_not_ending (los) -->

<!-- @pattern_begin delete_stri(querria) -->
  <category><pattern>CLARO</pattern>
    <template><srai>SI</srai></template>
  </category>
<!-- @pattern_end delete_stri(querria) -->
```

Fig. 40. Ejemplos de la estructura de patrones de remplazo dependientes de la posición.

Capítulo 5

Alcance y Arquitectura del proyecto

En este capítulo se aborda la descripción de la fase de diseño de proyecto. Cabe destacar que por la propia naturaleza de la aplicación desarrollada y de los requisitos del proyecto, existen dos facetas diferenciadas en el diseño:

- Diseño del software de la solución: engloba el diseño de todos los componentes del proyecto y las clases los componen. El software de la solución proporciona las herramientas de inteligencia artificial y agentes inteligentes que componen el proyecto.
- Diseño de la aplicación Web: engloba el diseño de las distintas aplicaciones Web que se distribuyen como partes de este proyecto. El diseño de las aplicaciones Web proporciona una interfaz para acceso a las herramientas desarrolladas en el punto anterior. Su diseño debe estar cuidado para garantizar los requisitos marcados de facilidad de uso.

Por otro lado, se ha puesto especial interés en estructurar el proyecto en módulos independientes y completos. De tal manera que puedan ser desarrollados en líneas de trabajo separadas, que puedan formar parte de otros proyectos a los que no pertenezcan el resto de módulos y que puedan ser desplegados en el servidor de aplicaciones como una aplicación útil que proporciona un servicio concreto. Como ya se ha mencionado, esta estructuración favorece la continuidad de este proyecto y se hará especial mención a ella al final de este capítulo.

5.1 Descripción global del sistema

En estados anteriores del proyecto, se definió el sistema como un bot que se comporta conforme a los requisitos de un cliente (gsiBot). Sin embargo, la propia evolución del proyecto y de las funcionalidades ofrecidas por la aplicación ha promovido un cambio de perspectiva posicionando el sistema como un servidor de bots (gsiBotServer). Además de dar servicio a varios bots y proporcionar opciones de administración sobre los mismos, gsiBotServer incluye un conjunto de aplicaciones para la asistencia al desarrollador de bots. Estas incluyen un asistente para la edición de la base de conocimiento a alto nivel (gestiónDeConocimiento), un módulo de pruebas automatizadas (CorpusTester) y una plataforma para el despliegue de los bots en desarrollo (mediante el propio panel de administración de gsiBotServer). En la Fig. 41 se muestra las relaciones y dependencias que tienen estos módulos entre sí.

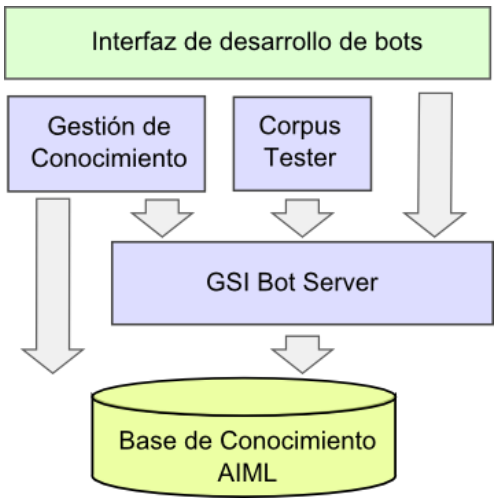

Fig. 41. Diagrama de módulos que componen el sistema.

La interfaz de desarrollo que se presenta en la Fig. 41 permite el usuario desplegar con facilidad y rapidez los bots que está desarrollando, probar su funcionamiento conforme a unas especificaciones que marcan el comportamiento esperado y desligarse de la generación de código AIML mediante la utilización de la herramienta de gestión de conocimiento.

Estas funcionalidades son proporcionadas individualmente por cada uno de los módulos independientes que integran el sistema. Por una parte se dispone de un servidor de bots para el despliegue y servicio de bots. De tal manera que proporcionando los archivos que forman la base de conocimiento (y los archivos de configuración en caso de no utilizar la configuración por defecto) se puede conversar con el bot sin necesidad de ejecutar ningún programa adicional ni reiniciar el servidor.

También se dispone de dos aplicaciones de edición de la base de conocimiento del bot: una para la edición de las respuestas ofrecidas por el Bot y otra para añadir nuevo conocimiento de manera que el bot de respuesta a un mayor número de frases de usuario.

Por último existe un módulo automatizado de pruebas de la base de conocimiento que a partir de unos ficheros de prueba, en los que se recoge en comportamiento esperado del bot, se comprueba si los cambios realizados en la base de conocimiento por los dos componentes anteriores interfieren con el conocimiento existente previamente.

Cada uno de los módulos constituye una aplicación Web separada, que puede ser desplegada de forma independiente del resto en el servidor de Servlets.

5.2 El módulo servidor de bots

El servidor de bots (*gsiBotServer*) es la aplicación principal del conjunto de herramientas desarrolladas en el proyecto GSIBot. El servidor de bots permite desplegar un bot con el conocimiento contenido en un conjunto de archivos AIML, a través de un panel de administración Web, con solo seleccionar los archivos AIML y elegir un nombre para el bot. Esta funcionalidad agiliza considerablemente el proceso de desarrollo de bots, pues evita tener que reiniciar el servidor cada vez que se desee realizar una prueba. A pesar de que pueda parecer una funcionalidad básica, programD no la incluye, pues no está diseñado para dar soporte a desarrollo de bots

El núcleo del servidor de bots es programD, en torno al cual se añaden una serie de módulos que refuerzan sus puntos débiles y añaden otras funcionalidades para las que programD no fue diseñado.

5.2.1 Modelo funcional

Aunque más adelantes se desarrollarán con mayor detalle las características cada uno de los módulos, en la siguiente lista quedan recogidas –a modo de resumen- las funcionalidades que el servidor de bots añade a programD.

- El gestor de bots permite generar el descriptor de bot a partir de los archivos AIML que compondrán la base de conocimiento del bot, de manera que pueda ser desplegado inmediatamente.

- Capacidad para desplegar/replegar bots en caliente, sin necesidad de reiniciar el servidor, de manera que el usuario pueda probar los bots en los que este trabajando sin perdida de tiempo.
- El panel de administración de bots ofrece una interfaz sencilla e intuitiva para la administración del servidor. Permite visualizar el estado de los bots cargados, ofrece medidas del tamaño de la base conocimiento y del uso de los mismos. El panel de administración incorpora un script de conversación rápida con cualquiera de los bots cargados para probar el correcto funcionamiento de los mismos.
- Adaptadores de acceso HTTP, RMI, WAP, Jade y de mensajería instantánea. Estos proporcionan una total independencia del servidor a la aplicación cliente que accede al bot, así como un amplio abanico de interfaces de acceso diferentes para escoger en función de la naturaleza del sistema cliente.
- Sistema de plugins para el registro de meta-etiquetas AIML. Pensado como un conjunto de etiquetas incrustadas en los valores de las variables AIML o de las respuestas de los templates. Estas etiquetas son leídas por el gestor de plugins, procesadas por el plugin asociado a cada etiqueta y reemplazadas por la cadena determinada por el plugin. De esta manera se pueden añadir funcionalidades a AIML tales como consulta de buscadores, diccionarios, servicios de meteorología, lectura de recursos RDF… todo sin infligir las especificaciones de AIML.
- Sistema de logging configurable orientado a la obtención de estadísticas de acierto, frecuencia de consulta de tópicos y recomendación de categorías AIML a incluir en la base de conocimiento del bot.

5.2.2 Modelo Estructural

El módulo gsiBotServer define una arquitectura basada en programD como intérprete de AIML. Se dice que programD es un contenedor de bots. El sistema se centraliza en torno al BotServer que se encarga de gestionar el acceso a los distintos bots cargados en el servidor, bien sean bots servidos en la misma instancia de programD o en distintas. A través del BotServer el panel de administración permite monitorizar y gestionar los distintos bots. En la Fig. 42 se puede ver un esquema de la arquitectura en el que se especifican cada uno de los módulos que componen el servidor de bots.

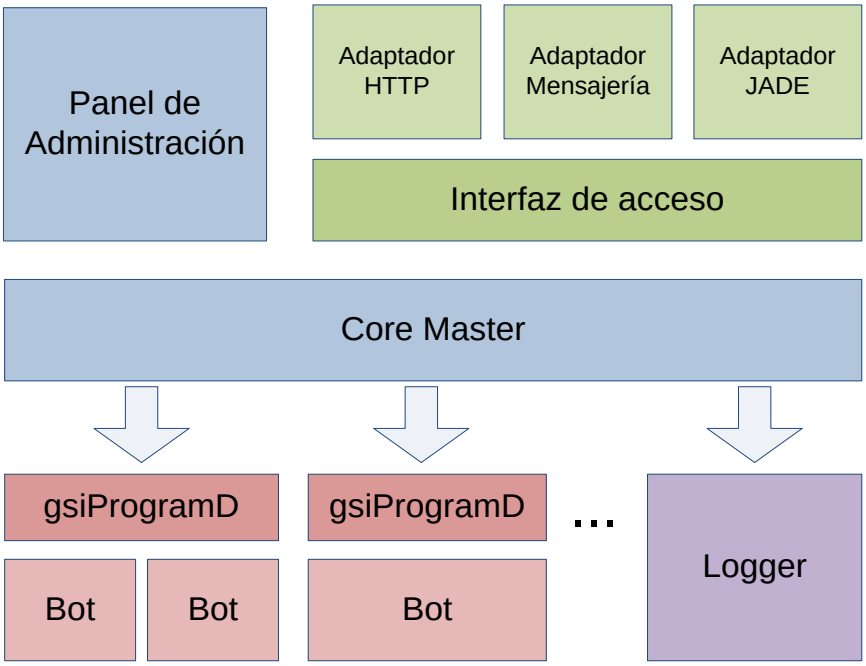

Fig. 42. Arquitectura de gsiBotServer.

El sistema ha sido desarrollado utilizando tecnología *Java Enterprise Edition* y se despliega en Apache Tomcat.

5.2.2.1 GsiProgramD

ProgramD es la plataforma de código abierto escrita en Java para bots AIML más utilizada. Implementa la mayoría de las capacidades de AIML y cada instancia soporta un número ilimitado de bots en el contenedor de aplicaciones. ProgramD es altamente configurable por medio de un conjunto de ficheros de configuración, que permiten activar o desactivar muchas de sus funciones.

GsiProgramD constituye un `wrapper` de programD. Ofrece las mismas características que programD y ha sido adaptado para que el panel de administración pueda visualizar variables privadas de programD. Por otro lado GsiProgramD subsana algunos bugs presentes en la última distribución de programD dado que el grupo de desarrollo de programD ha cesado su actividad.

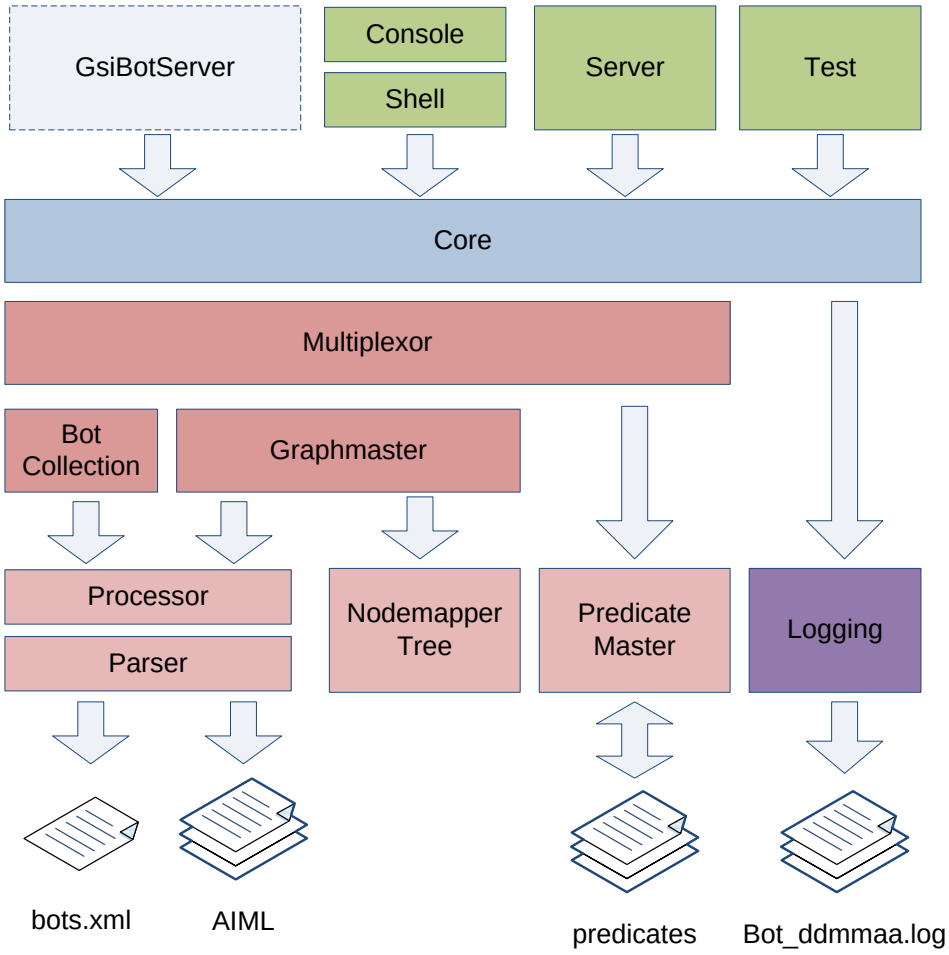

Fig. 43. Arquitectura simplificada de programD.

En la Fig. 43 se muestra un esquema simplificado de la arquitectura de programD. `Core` es el núcleo de programD, tiene acceso a toda la información cargada de los archivos de configuración y de los archivos AIML. Además ofrece métodos para acceder a esa información, realizar una consulta a un bot determinado, cargar o eliminar una url cargada para un determinado bot, cargar un nuevo bot, replegar un bot activo…

El módulo de `Parser` se encarga de leer los archivos con formato XML y el módulo `Processor` ejecuta las acciones relacionadas con cada uno de los elementos XML leídos. La colección de Bots es un conjunto de instancias que se crean a partir de la lectura del archivo de configuración de bots[22] que contienen información acerca de las url de los ficheros cargados para cada bot, verdaderamente no contienen información alguna, sino atajos para el acceso a la información. La información leída del corpus AIML se carga en el `Graphmaster`, mediante la construcción del árbol de `Nodemapper`s.

[22] Se crea una instancia de la clase Bot por cada bot cargado en progarmD.

El módulo multiplexor tiene acceso a esta información y la sirve a cada uno de los usuarios que piden acceso al bot. El módulo *multiplexor* además mantiene el estado de los bots para cada usuario de manera que es capaz de recordar de lo que ha hablado con cada bot, por lo que la conversación se hace más natural. Esta información se almacena en los predicados. El módulo *multiplexor* también almacena la información del usuario con el que está conversando que es capaz de recoger. Es posible que recoja esta información de forma explícita preguntando por el su nombre o su edad, o de forma implícita adivinando, por ejemplo, sus gustos a partir de los temas de conversación.

Sobre el *Core* se montan los servicios de *testing* de AIML que ofrece programD, la consola para la ejecución de bots o el servlet de acceso a bots cuando se ejecuta programD en un servidor de aplicaciones. Sin embargo, todas estas características constituyen el punto débil de programD, de manera que en el gsiBotServer se trabaja directamente sobre el *Core* para servicios alternativos.

5.2.2.2 Bot Server

BotServer es la pieza central del sistema servidor de bots. Permite acceder de forma transparente a todos los Bots cargados con sólo proporcionar el nombre del mismo independientemente de la instancia de programD en que estén alojados. Ha sido diseñado para que permita cargar y replegar bots a través del panel de administración sin necesidad de reiniciar el servidor. `BotServer` también incluye un módulo de soporte para incluir plugins que actúan como filtros de la respuesta o la entrada.

`BotServer` centraliza el acceso al gestor de bots y al gestor de plugins proporcionando un único punto de entrada para, a través del panel de control y la interfaz de acceso, configurar el servidor y acceder a los bots. En concreto, `BotServer` es la clase java que se encarga de proporcionar esta interfaz. Sin embargo, dado que el acceso a la gestión de bots y plugins se realiza de forma transparente a través de esta clase, cuando se habla de BotServer se incluyen también las clases para la gestión de bots y plugins. En la Fig. 44 se muestra la arquitectura de clases de `BotServer` en un diagrama UML simplificado

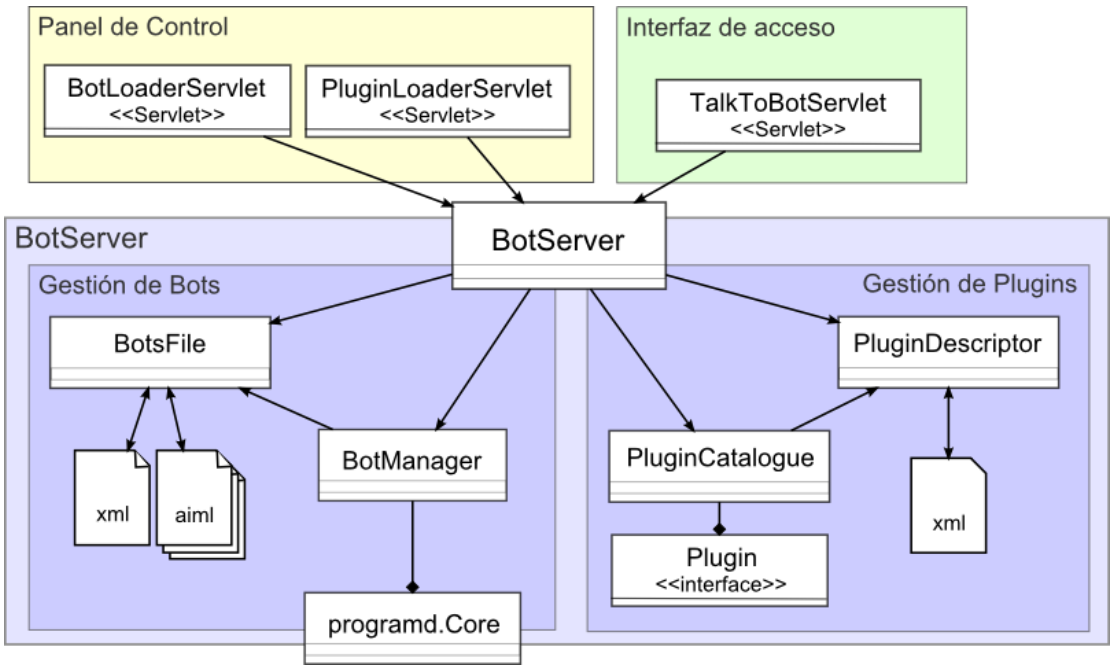

Fig. 44. Diagrama UML de clases en que interviene BotServer.

5.2.2.3 Panel de administración

El panel de administración es un servicio Web sobre BotServer que ofrece una interfaz para la administración y gestión de bots. Esta interfaz se describe más adelante en el apartado 5.2.4.1. Desde el panel de control se pueden acceder a todas las funciones de configuración de gsiBotServer.

- Visualización del listado de bots cargados en el servidor.
- Vista en detallada de los archivos AIML cargados en cada bot.
- Despliegue y repliegue de bots sin necesidad de reiniciar el servidor.
- Gestión de archivos AIML cargados en cada bot.
- Integración con la herramienta CorpusTester.

5.2.2.4 Interfaz de acceso

Una consulta a un bot queda definida por la terna: el nombre de usuario, el nombre del bot al que se consulta y la frase o cadena de caracteres que constituye la consulta en sí. La respuesta que ofrece el bot está constituida por la repuesta textual dada por el bot y por un conjunto de atributos de estado del bot que se denominan predicado.

El predicado es un subconjunto de variables escogidas de entre las variables AIML establecidas con el comando `<set>`[23] y de los parámetros de la petición realizada por el usuario. El conjunto de atributos incluidos en el predicado es configurado a través del archivo de configuración del servidor. Por defecto se incluyen la id del bot, la id del usuario, la respuesta, la variable AIML test y la variable AIML url.

El servidor ofrece una sencilla interfaz de acceso a través HTTP para facilitar su integración con los clientes (Web, IM, móvil y JADE), según la URI:

```
http://<url-servicio>?q=<consulta>&bot=<id-bot>
     [&user=<user-id>][&type=<response-type>]
```

donde:

- <*url-servicio*> es la URL donde se está ofreciendo el servicio.
- <*consulta*> es la cadena con la consulta del usuario.
- <*id-bot*> es el id del bot al que se le envía la consulta.
- <*user-id*> es la id de usuario. Si se omite se utilizará por defecto el identificador de sesión.
- <*response-type*> es el tipo de respuesta solicitado. Puede ser HTML, XML, JSON [7] o TEXT. Es un parámetro opcional, por defecto HTML.

En la siguiente tabla se muestran las propiedades de los parámetros de acceso de la interfaz de acceso HTTP.

Parámetro	Requerido	Por defecto	Casesensitive	Posibles valores
Q	Si	-	No	Cadena caracteres
Bot	Si	-	Si	Cadena caracteres
User	No	[session-id]	Si	Cadena caracteres
Type	No	HTML	No	HTML, XML, Json, Text.

Tabla 1: Resumen de los parámetros de la interfaz de acceso de gsiBot.

La respuesta, por tanto se servirá en formato HTML, XML, JSON o en texto plano en función del valor del parámetro type. La siguiente tabla muestra el valor del tipo MIME indicado en las cabeceras de la respuesta en cada caso:

[23] Ver apartado XXX

Formato de salida	Tipo MIME
HTML	text/html
XML	application/xml
Json	application/json
Text	text/plain

Tabla 2: Tipos MIME de la respuesta.

La interfaz de acceso definida suple las limitaciones que programD muestra su interfaz de acceso, en la que los parámetros de la respuesta son almacenados como atributos del objeto sesión y estando así solamente accesibles a través de tecnologías de servidor. La interfaz permite el acceso a cualquier tecnología que permita realizar una consulta HTTP a una URI.

No obstante, la interfaz está optimizada para responder a consultas a un cliente que emplee la tecnología AJAX pues al incluir JSON entre los formatos en que se puede solicitar la respuesta se hace muy sencillo el acceso a la información de respuesta desde JavaScript y se permite llevar a cabo la conversación sin necesidad de recargar la página. En la Fig. 45 se muestra el código JSON una respuesta dada por el servidor de Bots.

```
{"dialog": {   "sessionid": "4A5800A89B1778",
               "bot": "gsiBot",
               "q": "hola ",
               "response", "Hola, ¿qué tal?",
               "test": "General Saludo",
               "state": "saludo"  }}
```

Fig. 45. Respuesta del servidor de Bot en formato de datos JSON.

5.2.2.5 Logger

El módulo de logs incluido en el servidor de Bots de gsi utiliza la biblioteca log4j [21] para generar y presentar las trazas. Se presenta como alternativa para generar estadísticas a la solución de integración con una base de datos mySql propuesta en programD.

5.2.3 Procesado de las consultas

La secuencia de ejecución más repetida en un sistema de bots es la de consulta a un bot. La mayor parte de la ejecución de esta secuencia tiene lugar en las clases que forman programD, ya que estas constituyen el motor de AIML.

El procesado de las consultas a un bot conlleva el procesado del árbol de decisión, `Graphmaster`, en que se almacenan en memoria las reglas AIML. Tanto el proceso de construcción de dicho árbol como su procesado están definidos en las especificaciones de AIML [11]. En el diagrama siguiente se resume la secuencia de ejecución de una consulta al bot en la que se ha ocultado la interacción con la clase `Graphmaster`.

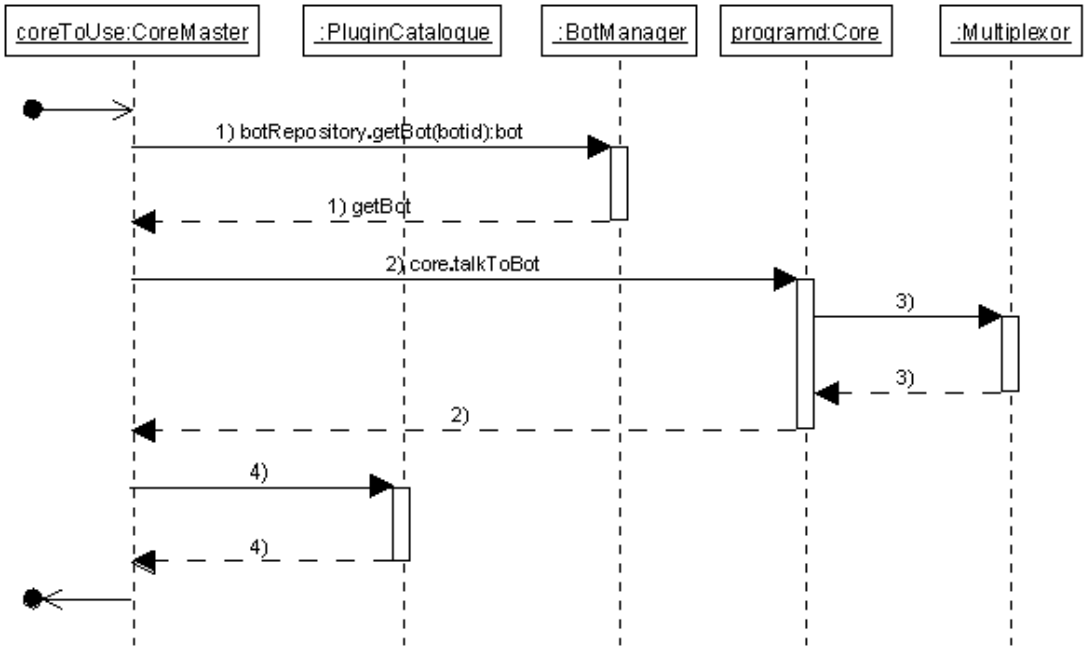

Fig. 46. Diagrama de ejecución de una consulta al bot

En la figura anterior se ha incluido el punto de inicio como un mensaje asíncrono ya que el caso habitual es que el usuario envíe una petición HTTP a través de la interfaz Web para iniciar la consulta. Dicha petición es recogida por el servlet `TalkToBotServlet` que extrae de la petición los parámetros y consulta al `CoreMaster`.

La clase `CoreMaster` se encarga de planificar la consulta:

1) `BotManager` proporciona la clase `Core` que contiene el bot determinado o lanza un error en caso de que el bot no esté disponible o no pueda ser encontrado.
2) La consulta se traslada a la clase Core de programD.

3) La clase Core lleva a cabo la búsqueda en el árbol de decisión mediante las clases multiplexor, graphmaster, processor… y proporciona una respuesta. Este paso se ha omitido en el diagrama por simplicidad.
4) La respuesta puede contener etiquetas para ser procesadas por los plugins instalados, de manera que se envía la consulta al `PluginCatalogue` para su procesado.

5.2.4 Interfaz Web

La interfaz Web del módulo gsiBotServer proporciona acceso, a través del panel de administración, a todas las funcionalidades presentadas así como a los bots a través de la interfaz de acceso.

5.2.4.1 Panel de administración

En la página principal del panel de administración se muestra una lista con un resumen de los bots cargados en el servidor sea cual sea su estado.


Listado de Bots

La siguiente tabla lista todos los bots que actualmente están cargados en este servidor y presenta cierta información de cada uno de ellos.

Nombre del bot	Dedicado	Estado	Tamaño	Acciones
OrangeDistribuidores	Si	READY	2883	Eliminar Probar
Erika	Si	READY	4558	Eliminar Probar
3DTour	Si	READY	4650	Eliminar Probar
admiTI2	Si	READY	2336	Eliminar Probar

Información en el listado de bots...

Fig. 47. Extracto de la interfaz Web del panel de administración de gsiBotServer.

El panel de administración permite cargar un bot con solo elegir un nombre para el bot y seleccionar los archivos AIML que componen su base de conocimiento. La selección de archivos AIML se realiza a través de un plugin jQuery de selección de múltiples archivos.

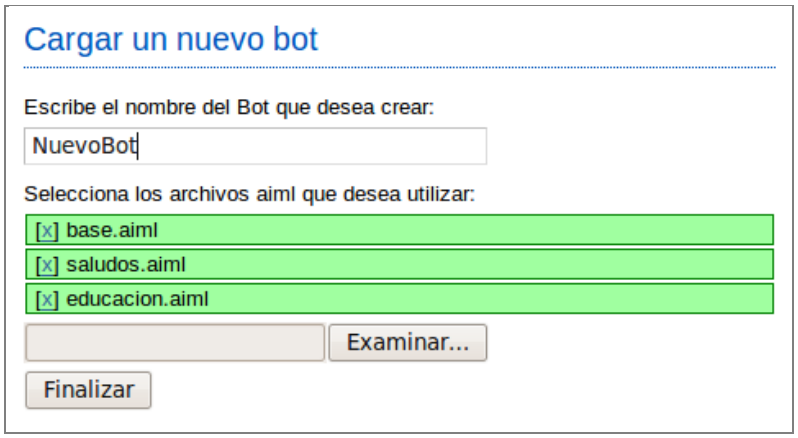

Fig. 48. Extracto de la interfaz del panel de administración de gsiBotServer.

5.2.4.2 Interfaz de acceso

La interfaz de acceso incluida en la distribución como aplicación Web del servidor de aplicaciones consta de un plugin de jQuery configurado para mostrar tanto la respuesta actual como el historial de conversaciones. El plugin ha sido desarrollado plenamente configurable de manera que permite seleccionar desde la dirección URL de la interfaz de acceso del bot al que se realiza la consulta hasta el objeto del DOM en que se muestra tanto la última respuesta como el historial de estas.

El pluggin talktobot.js ha sido desarrollado utilizando la tecnología ajax y el framework de javascript jquery. El plugin se incluye en el código de la página como cualquier script javascript y permite acceder al bot a través la interfaz de acceso descrita en el apartado 5.2.2.4. La ubicación del plugin es ./js/talktobot.js.

El plugin es totalmente configurable en cuanto a su modo de funcionamiento. En el caso por defecto, para vincular al plugging el formulario descrito en el apartado anterior basta con poner a dicho formulario la id `talktobot`. La conversación se mostrará en el elemento div que tenga la id `dialog_display`.

La figura siguiente muestra la apariencia de un formulario de estas características. El capo para el nombre del bot se ha incluido como un select y el campo del tipo de acceso se mantiene oculto.

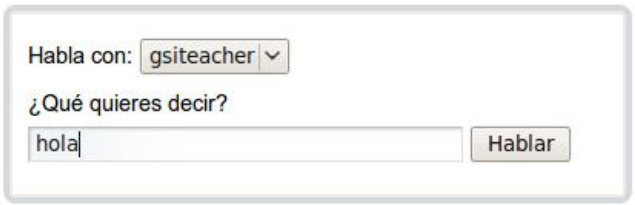

Fig. 49. Interfaz de consulta estándar del servidor de bots.

El código para este formulario es:

```
<form id="talktobot" action="#">
   <label for="bot">Habla con:</label>
   <select name="bot">
      <option value="Dummy">gsiteacher</option>
   </select>
   <label for="q">¿Qué quieres decir?</label>
   <input type="text" name="q"/>
   <input type="hidden" name="type" value="json" />
   <input type="submit" value="Hablar"/>
</form>
```

El nombre de los campos bot, q y type debe coincidir con los parámetros de la interfaz de acceso descritos en 5.2.2.4.

5.2.5 Distribución de gsiBot personalizada para clientes

Los clientes que desean una solución de servicio de bots marcan unos requisitos muy concisos respecto a los conocimientos del bot y a la interfaz de usuario. Habitualmente solicitan una interfaz con su imagen corporativa y una base de conocimiento que permita al bot hablar de aquello para lo que han contratado el servicio sin necesidad de extenderse en temas de conversación no relacionados.

En el diseño de la aplicación se ha tenido en cuenta esta necesidad, por lo que el servidor de bots se GSI aporta facilidades para personalizar la interfaz Web de acceso e incluye herramientas para la edición y gestión de bases de conocimiento. Las herramientas de edición y gestión constituyen un módulo completo del que se hablará más adelante en esta memoria.

Para adaptar la interfaz Web de acceso basta con diseñar un formulario con campos para introducir la pregunta que se desea remitir al bot, aplicarle estilos y utilizar el plugin javascript para acceso a que se proporciona con el gsiBotServer.

El formulario debe tener un campo de entrada para introducir la pregunta, un campo con el nombre del bot y un campo con el tipo de acceso[24]. Algunos de estos

[24] Estos campos coinciden con los parámetros de la url de acceso al bot descritos en el apartado 5.2.2.4.

campos pueden omitirse por lo que se tomarán los valores por defecto o estar ocultos por lo que su valor estará prefijado. El único campo en que el usuario siempre debe poder escribir es en la frase de entrada.

El proceso de empaquetado de la aplicación Web está automatizado mediante una tarea de Ant que incluye todos los archivos AIML y de configuración, así como las páginas Web con las hojas de estilos y código javascript necesarios.

En el Capítulo 6 se presentan dos casos de estudio concreto consistentes en dos distribuciones de corpus AIML para cliente: OrangeBot y 3DTour.

5.3 El módulo de pruebas automatizadas

El objetivo del modulo de pruebas automatizadas es determinar si la base de conocimiento funciona correctamente conforme a las especificaciones marcadas y si, por el contrario, contiene errores identificar de qué fallos se trata. Esta herramienta es necesaria pues de otra manera sería muy complicado identificar la causa de los errores producidos por el proceso de aprendizaje. En el corpus tester se registra el comportamiento que se espera de un determinado bot de manera que se pueda probar de manera sistemática este comportamiento. Está orientado a ser utilizado cada vez que se introduce un cambio en la base de conocimiento para cerciorarse de que no ha producido el error.

El principal problema en la programación de una base de conocimiento es que a medida que esta se construye, incorporando reglas que funcionan correctamente por separado, existe el riesgo de que uno de estos añadidos provoque un error en una regla ya definida. Por esta razón es necesaria una herramienta que pueda probar sistemáticamente el corpus del bot, de manera que tras la introducción de cada modificación en el mismo se pueda determinar si la modificación reciente interfiere con el conocimiento existente y como consecuencia provoca un comportamiento inesperado. Por lo tanto, el objetivo del modulo de pruebas automatizadas es determinar si la base de conocimiento funciona correctamente conforme a las especificaciones marcadas y si, por el contrario, contiene errores identificar de qué fallos se trata.

El problema descrito con la interferencia entre patrones no deja de ser un caso concreto de un problema inherente a la programación: “Cada vez que se introduce un cambio en el código, éste puede contener errores. Incluso puede producir un error con el que no está aparentemente relacionado y que es difícilmente detectable”. En definitiva, escribir una base de conocimiento es programar en lenguaje AIML.

Con el fin de resolver este problema en la programación apareció el concepto de pruebas unitarias y pruebas de integración. Existen en los distintos lenguajes de programación diversos *frameworks* de realización de pruebas unitarias como es el caso de JUnit [22] o HTTPUnit [23] en Java, CPPUnit [24] para C++, PHPUnit [25] en Php, etc.

5.3.1 Modelo funcional

El módulo de pruebas automatizadas sirve para probar el correcto funcionamiento de los distintos fragmentos de AIML, y por lo tanto constituye un *framework* de pruebas unitarias para AIML. Puesto que AIML es un lenguaje basado en AIML también lo son las pruebas unitarias utilizadas en CorpusTester, tal y como se describe en la sintaxis de los archivos KTF en el apartado 5.3.2.1

CorpusTester también puede ser utilizado para realizar pruebas de integración en el caso de querer utilizar fragmentos de otras bases de conocimiento, con el objeto de saber si existe interferencia entre los mismos.

Las pruebas realizadas por el módulo de pruebas se caracterizan por:

- Estar automatizadas: No se requiere una intervención manual. Lo que las hace especialmente útiles para probar las interferencias tras la inclusión de pequeños cambios en la base de conocimiento.
- Ser reutilizables: pueden ser ejecutadas todas las veces que sea necesario.
- Ser independientes: la ejecución de una prueba no interfiere en que resultado que arroje la ejecución de otras pruebas. Así como el orden en que se ejecuten las pruebas no varía el resultado de las mismas.

5.3.2 Modelo Estructural

La arquitectura de Corpus Tester está basada en la biblioteca XStream [26] utilizada para parsear los archivos KTF. La clase principal es CorpusTester, que a partir de la información obtenida del archivo KTF y de los archivos de configuración lleva a cabo las pruebas y genera un informe con los resultados.

El módulo se estructura en cuatro paquetes. El paquete principal con las clases de ejecución de pruebas y elaboración de informes. El paquete `botaccessor` con las clases en las que se defienden los distintos métodos de acceso al bot. El paquete `file` y el paquete `file.serialization` contenido en este, con las clases relativas al manejo de ficheros de pruebas, su parseo y serialización. Y el paquete `servlet` en el que se

incluyen los servlets encargados de cargar parámetros por peticiones HTTP y mostrar en formato HTML los resultados.

En el siguiente diagrama UML se muestran las relaciones existentes entre estos paquetes. En el se han representado sólo las clases más importantes así como las relaciones entre ellas más destacadas.

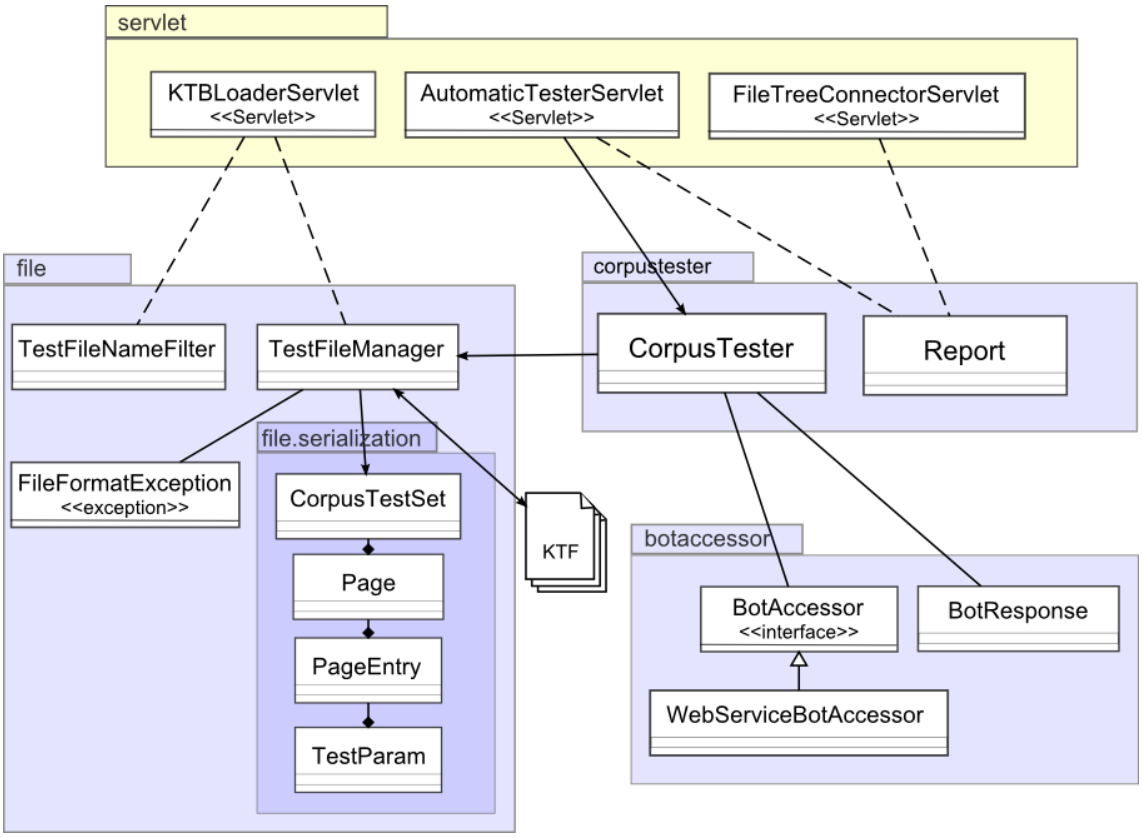

Fig. 50. Diagrama UML del módulo corpustester.

5.3.2.1 Los archivos Knowledge Test File (KTF)

El archivo KTF, dialecto de XML, contiene la definición de las pruebas a las que se somete el bot que se está desarrollando. Este formato permite al módulo de pruebas comprender el contenido de la hoja de requisitos.

Uno de los éxitos del diseño de la hoja de requisitos de Bot como un conjunto completo y exhaustivo de casos de uso es que pueden ser transformados unívocamente en un conjunto de pruebas unitarias para el Bot. De esta forma se cumple con el compromiso de que los requisitos sean fácilmente verificables.

Sintaxis de los archivos KTF

La sintaxis del archivo KTF se define conforme a un archivo de esquema XLS. En él se recoge que existen los siguientes tipos de elementos con sus correspondientes atributos. Todos los atributos que se especifican son obligatorios.

- **corpustestset**: Es el elemento raíz.
  - bot: Único atributo de corpustestset. Bot indica el nombre del Bot para el que se realizan las pruebas.
- **page**: El uso de este elemento es opcional. Engloba un conjunto de pruebas unitarias, sirve para agrupar dichos casos de prueba.
  - name: El nombre que se le asigna al grupo de casos de prueba. El nombre se presenta en el visor de archivos KTF.
- **testcase**: Es el caso de prueba propiamente dicho. Debe contener al menos un elemento userinput y un elemento checkparam.
- **userinput**: Es la frase de entrada que introduce el usuario para este caso de prueba. Se pueden incluir varios elementos userinput para concatenar entradas de usuario y realizar la prueba. El orden en que se incluyen los elementos userimput es el orden en que se enviaran al Bot.
- **checkparam**: Corresponde a la parte de comprobación. Cada elemento checkparam realiza una comprobación.
  - name: es el nombre de la variable de AIML cuyo valor queremos comprobar.
  - value: es el valor que debe tener la variable dada por *name* para que la comprobación se evalué como correcta.

La Fig. 51 muestra un detalle de un archivo KTF

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="corpustestset.xsl"
type="text/xsl"?>
<corpustestset bot="StudentBot">
  <page name="default">
    <testcase>
      <userinput>¿Donde está Madrid?</userinput>
      <checkparam name="test" value="Madrid_location"/>
    </testcase>
    <testcase>
      <userinput>¿Qué puedo visitar en Madrid?</userinput>
      <checkparam name="test" value="Madrid_tourism"/>
```

```
    </testcase>
  </page>
</corpustestset>
```

Fig. 51. Archivo KTF de ejemplo.

Visor de archivos KTF.

Un archivo en formato KTF puede ser leído y comprendido por el módulo de pruebas pero su lectura es pesada para un desarrollador con conocimiento de XML e incomprensible para un cliente sin la formación adecuada. Por tanto, la existencia de un archivo KTF como único documento que recoja la hoja de requisitos del Bot parece una mala solución.

Una solución parcial pasa por mantener dos ficheros perfectamente sincronizados con la hoja de requisitos: un KTF y un archivo de texto en el que se recoja la misma información de forma que pueda ser leída por un usuario no cualificado. Existe en Corpus Tester una aplicación que actúa de conversor de archivos KTF a texto plano y viceversa.

Una solución mejor es utilizar el visor de archivos KTF incluido en el módulo Corpus Tester. Consiste en un servicio Web que lee el archivo KTF y muestra el contenido en forma de árbol. La exploración de la hoja de requisitos a través de este servicio permite expandir la información de cada caso de uso para acceder a mayor nivel de detalle.

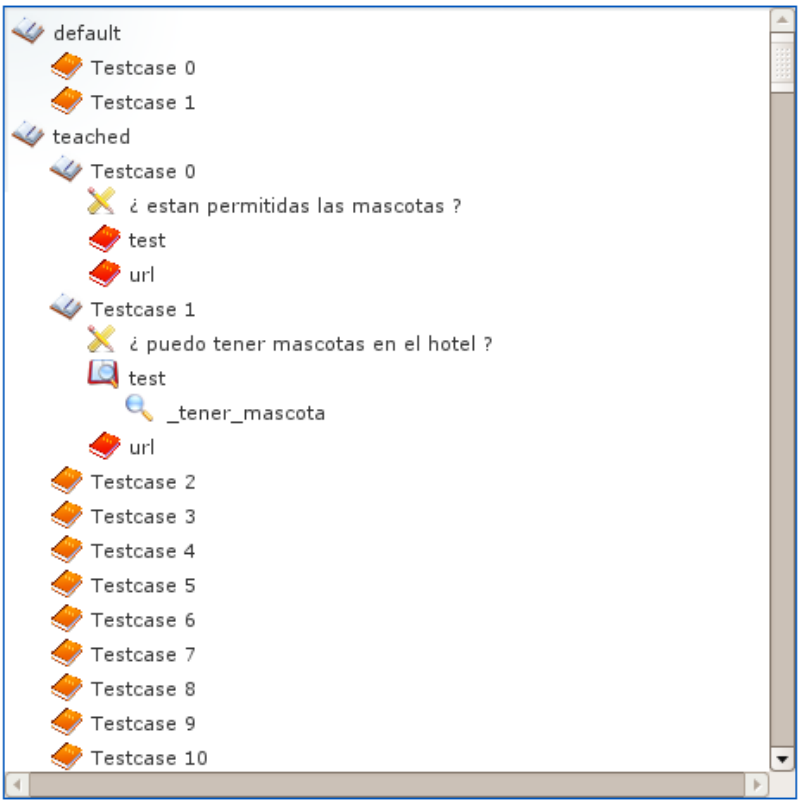

Fig. 52. Captura de pantalla del visor de archivos KTF.

La generación de la estructura de de árbol mostrada en la figura anterior, así como la animación del mismo –ya sea para expandir o plegar ramas- se realiza mediante un plugin de `jQuery jqueryFileTree.js` [27]. Para la implementación de este servicio se ha hecho que cada una de las clases con que se representa en memoria un archivo KTF (y que se describen en el siguiente apartado) implemente el interfaz TreeNode, de manera que se pueda representar con una estructura de árbol. El servlet encargado que sirve de conector con el plugin es `FileTreeConnectorServlet`.

5.3.2.2 El paquete de serialización de archivos KTF

Los paquetes `file` y `file.serialization` se encargan de la serialización y des-serialización de los archivos KTF, mediante la biblioteca XSteram [26] para serialización de archivos XML en Java.

Para la utilización de dicha biblioteca, en el paquete `file.serialization`, se ha diseñado una jerarquía de clases idéntica a la jerarquía de objetos definida para los archivos KTF. XStream funciona mediante la introspección de Java, de manera que cada una de las clases en dicha estructura se ajusta al modelo de *JavaBean*s. La relación entre las clases y los nombres de los elementos se realiza mediante *Java Annotation*s [28], de manera que pueda ser modificada en caso de cambiar la notación de los

archivos KTF. La clase que representa un archivo KTF en su totalidad es un *CorpusTestSet*. Dicha clase ofrece métodos auxiliares para facilitar la inclusión de nuevos casos de prueba. El propósito de añadir estos métodos es facilitar la integración de CorpusTester con las herramientas de edición automática de archivos AIML, de manera que al mismo tiempo que se añaden categorías al corpus se añadan los casos de prueba pertinentes al conjunto de casos de prueba.

En el paquete *file.serialization* también se incluye la interfaz *NodeTree,* utilizada por el visor de archivos KTF, que es implementada por cada una de las clases del paquete.

La clase principal para la gestión de archivos KTF es *TestFileManager*, que proporciona métodos para cargar el contenido de un archivo KTC en un objeto *CorpusTestSet* o guardar el contenido en memoria en un archivo KTF. *TestFileManager* además ofrece compatibilidad con anteriores versiones del archivo KTF en las que si sintaxis era en texto plano. Dicha sintaxis no merece descripción alguna pues está obsoleta ya que permite menos funcionalidades que la sintaxis actual.

5.3.2.3 El paquete botaccessor

La finalidad de este paquete es dejar la puerta abierta a la integración de CorpusTester con otros sistemas de bots y a su vez no complicar su integración con gsiBotServer. Para ello se definia una interfaz *BotAccessor* con métodos para configurar la interfaz, detectar si el servicio está disponible y realizar consultas al bot almacenando los resultados.

En general respuesta a la consulta realizada a un bot pueden constar de otras variables además de la frase que el bot da como respuesta propiamente dicha. Para almacenar la respuesta con una estructura de datos conocida por la interfaz *BotAccesor* se define la clase *BotResponse*, en la que se almacena tanto la frase respondida por el bot como el conjunto de variables que definen el estado del bot y que pueden ser accedidas por los métodos particulares de la implementación de la interfaz de acceso al servicio de bots.

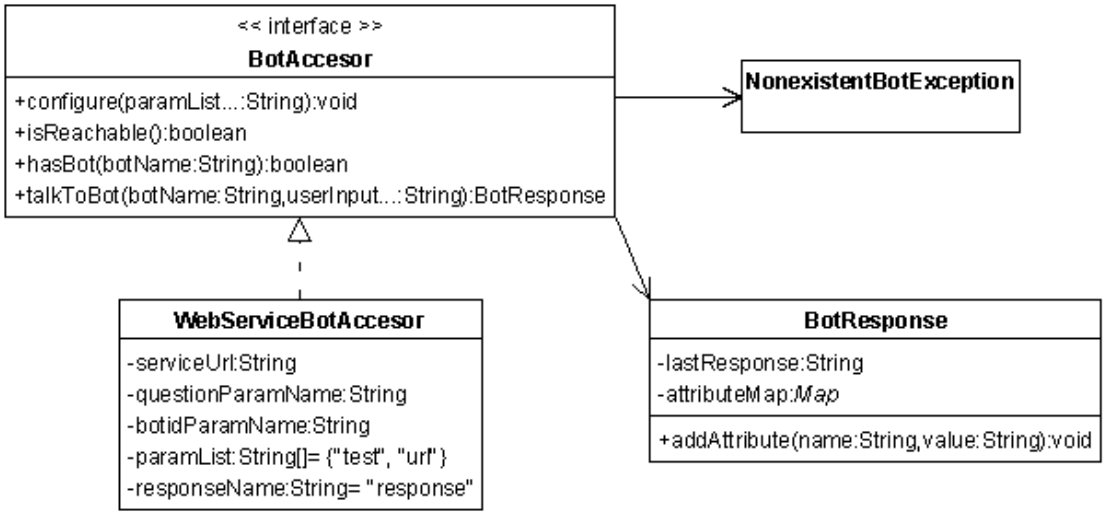

Fig. 53. Diagrama UML de la interfaz BotAccessor.

Tal y como se muestra en la figura anterior los métodos definidos para la interfaz *BotAccesor* son:

- `configure (paramList : String[]) : void`. Permite proporcionar un conjunto de nombres de parámetros que se deben recoger de la respuesta del bot y empaquetar en la respuesta *BotResponse*.
- `isReachable () : boolean`. Comprueba si el servicio de bots para el que se ha configurado la clase *BotAccesor* esta disponible. Por definición un bot está disponible si se puede conversar con él.
- `hasBot (botName : String) : boolean`. Comprueba si el servicio de bots dispone de un bot identificado por el nombre dado como parámetro. En caso de que el servicio de bots no esté disponible se lanzará una excepción en lugar de indicar que el bot no está disponible.
- `talkToBot (botName : String, userInput : String []) : BotResponse`. Este método realiza un cierto número consultas al bot y almacena la respuesta de la última de ellas. Cada una de las consultas es uno de los elementos del array *userInput*. El motivo por el que se realizan varias consultas es una decisión de diseño: para permitir a CorpusTester realizar pruebas unitarias en las que se tenga en cuenta el contexto de la conversación, cada vez que se realiza una llamada al método *talkToBot* se destruye el contexto, de manera que las pruebas no den un resultado erróneo interferido por "recuerdos" del bot de otras conversaciones. Para conseguir que *CorpusTester* pueda probar estas funcionalidades de mantenimiento de la conversación se define el parámetro *userInput* como un array, de manera que se puede guiar al bot por una conversación en la que se mantiene el contexto. Este proceso se describe con detalle en 5.3.3. La respuesta es

almacenada, como se ha indicado anteriormente, en un objeto `BotResponse`.

5.3.2.4 El paquete corpustester

`CorpurTester` es la clase principal del módulo de pruebas. Mediante la interfaz de acceso definida por las clases del paquete `botaccessor` ejecuta la batería de pruebas leída del archivo KTF.

El resultado de la ejecución de las pruebas es almacenado en un objeto `Report`. `CorpusTester` proporciona también una serie de métodos para imprimir el informe en un archivo XML, en un archivo de texto plano o en un documento HTML.

En el apartado 5.3.3 se define el flujo de ejecución llevado a cabo por la clase CorpusTester para la realización de la pruebas.

5.3.3 Funcionamiento de las pruebas unitarias de Corpus Tester

Al ejecutar las pruebas el módulo Corpus Tester inicia una nueva conversación para cada caso de uso –elemento *testcase*. De esta manera el Bot considera que es un nuevo usuario, lo que permitir a CorpusTester realizar pruebas unitarias partiendo en todas ellas estado inicial del bot.

Las conversaciones se realizan con el Bot cuyo nombre se indica en el atributo *bot* del elemento raíz *corpustestset*. Si este Bot no está disponible en el servidor de Bot todas las pruebas se considerarán fallidas.

Las comprobaciones definidas en cada testcase por los elementos checkparam se ejecutan secuencialmente. De manera que el error en la primera de ellas impide la ejecución de las demás considerándose inmediatamente la prueba fallida. Para que la prueba se considere pasada todas las comprobaciones de deben verificar.

Para conseguir que `CorpusTester` pueda realizar pruebas en las que se tenga en cuenta el contexto de la conversación se define el parámetro `userInput` como un array, de tal manera que todos los elementos del array serán enviados como consultas al bot antes de realizar las comprobaciones del *testcase*. Mediante este sistema se permite tener controlado en todo momento el estado del bot. El orden en que se incluyen los elementos del array *userinput* es el orden en que se envían las frases al servidor.

Corpus Tester genera un informe con el resultado de la evaluación de las pruebas, sólo en el caso en que estás hayan salido fallidas. En el informe se listan todas las

pruebas ejecutadas detallando cuales han resultado fallidas y cuales han sido superadas. Es importante destacar que dado que cuando una prueba es fallida deja de realizar comprobaciones restantes, en el informe pueden identificarse varias comprobaciones como fallidas que en realidad no se han realizado. Si la evaluación de las pruebas es completamente satisfactoria se indica con un mensaje pero no existe la necesidad de generar un informe.

Flujo de ejecución de las pruebas

La clase CorpusTester conduce la ejecución de las pruebas según se muestra en el siguiente diagrama:

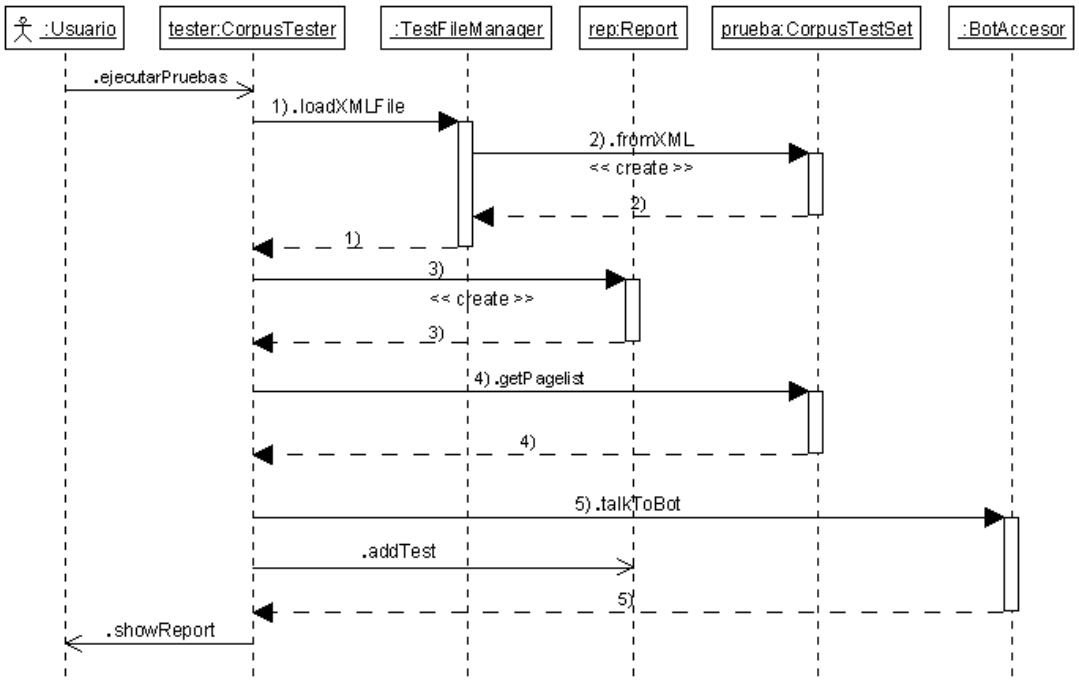

Fig. 54. Diagrama de ejecución de pruebas de bot.

1) `CorpusTester` carga los casos de prueba desde el archivo KTF generando un objeto `CorpusTestSet`.
2) Para ello la clase TestFileManager utiliza la biblioteca XStream.
3) `CorpusTester` crea el report en el que se almacenan los resultados de los casos de prueba.
4) Se cargan los casos de prueba desde el objeto `CorpusTestSet`.
5) Para cada una de los casos de prueba se realiza una consulta al bot mediante la interfaz `BotAccessor`. El resultado obtenido de cada consulta, en forma de un objeto `BotResponse`, se coteja con los resultados esperados definidos por los casos de prueba. Cada resultado se almacena en el objeto `Report`.

5.3.4 Desarrollo de bases de conocimiento utilizando Corpus Tester

La herramienta CorpusTester proporciona la garantía al programador de un bot de que el trabajo que está realizando no contiene errores, al menos conforme a las especificaciones marcadas.

Para ello es necesario que de antemano se hayan marcado las especificaciones para el bot recogidas en la hoja de requisitos, en forma de archivo KTF. Como se menciona en el apartado 4.2.2.1, la hoja de requisitos del bot es en realidad un alto porcentaje de las especificaciones marcadas por el cliente, de manera que a la hora de realizar el trabajo de ingeniería de requisitos se está confeccionando al mismo tiempo la hoja de requisitos del bot, y por tanto el archivo KTF asociado a dicho bot[25]. Al ligar la captura de requisitos a la realización de pruebas mediante una herramienta de automatización como es CorpusTester se está aumentando la eficiencia del proceso de desarrollo.

Es recomendable proceder de esta forma pues al quedar las pruebas unitarias definidas desde el principio podemos cuantificar y medir la evolución del trabajo realizado en función de las pruebas superadas. Además, de acuerdo con [14] es recomendable programar los casos de prueba antes de comenzar la programación del sistema.

Una vez está disponible el archivo KTF para su utilización con la herramienta CorpusTester se debe comenzar la programación del bot. Inicialmente, en estados prematuros del sistema, ejecutar las pruebas cada vez que se realice un cambio en los archivos AIML puede no tener sentido puesto que el número de pruebas no superadas aún es muy elevado y puesto que la habilidad del programador debe ser suficiente para controlar la inexistencia de interferencias entre las reglas incluidas. Sin embargo, es recomendable llevar a cabo de forma cada vez más frecuente estas pruebas puesto que es la manera óptima de acotar el error en caso de que se detecte alguno. Para ello el programador debe tener en cuentas el número de pruebas que se han superado la última vez y en caso de que este número aumente en una comprobación posterior comenzar el proceso de depuración.

Si las pruebas no son muy frecuentes, y especialmente en sistemas de bots de cierta envergadura, es posible que el número de pruebas superadas desde la ejecución de los test aumente, pero no lo haga en la medida en que debería, y por lo tanto los resultados hayan enmascarado un error. Esta situación no puede llegar a ocultar un error

[25] En el Anexo B se incluye un ejemplo de hoja de requisitos recogidos en forma de tabla y cómo es inmediata su traslación a la sintaxis KTF.

indefinidamente puesto que cuando el número de pruebas no superadas es reducido el programador trata con mayor grado de individualidad cada prueba, llegando al extremo en que sólo una prueba no sea superada.

Para evitar esto conviene aumentar la frecuencia de ejecución de las pruebas, en el caso ideal ejecutar las pruebas cada vez que se incluye una nueva regla y revisar en el informe, además del número total de pruebas superadas, que la regla recién añadida funciona correctamente.

5.4 El módulo de gestión de conocimiento.

El módulo de gestión de conocimiento, que constituye el módulo principal del proyecto, proporciona al usuario la capacidad de aumentar el conocimiento contenido en el corpus de un bot o crear uno desde cero en muy poco tiempo. Éste es un proceso delicado pues es el que contiene la inteligencia del sistema propiamente dicha y por tanto es el más difícil de automatizar. El usuario introduce en el conocimiento en el sistema mediante de un proceso asistido, dividido en una seria de pasos, en el que el lenguaje AIML se genera de forma automática y sin que sea necesario que el usuario tenga conocimientos del lenguaje.

Se pueden plantear muchas formas de automatizar la fase de enseñanza al bot. El sistema puede ser más o menos autónomo en función de la cantidad de decisiones que se toman sin consultar al usuario, decisiones de las que –por tanto- el sistema debe estar seguro –o bastante seguro. Un sistema totalmente autónomo sería capaz de, por ejemplo, leer un cierto número conversaciones guardadas hechas a través de clientes de mensajería instantánea e inducir a partir de ellas los patrones para generar AIML. Un sistema nada autónomo sería aquel que pidiera al usuario cada uno de los valores a introducir en cada uno de las etiquetas AIML. En tal caso el sistema se asemeja a un editor de XML.

A medida que la autonomía del sistema aumente también aumenta la probabilidad de cometer un error y por tanto, crear una entrada en la base de conocimiento que sea incorrecta: al aumentar la autonomía disminuye la fiabilidad. Si el sistema no es nada autónomo, y por tanto nada inteligente, todas las deducciones hechas se corroboran con el usuario y por lo tanto se le exige mayor nivel de cualificación en el uso de la herramienta.

Se pretende liberar de trabajo al usuario y desarrollar una interfaz intuitiva cuyo uso necesite la menor explicación posible. Para ello se ha alcanzado un compromiso entre autonomía y la fiabilidad desarrollando una interfaz semiautomática o asistida. El

sistema toma aquellas decisiones sobre las que posee certeza absoluta. En el resto de casos presenta al usuario las decisiones tomadas y pide su corroboración. En estos casos se traducen las consideraciones propias de AIML a términos fácilmente entendibles por el usuario.

En su funcionamiento, el módulo de gestión de conocimiento se integra con el servidor de bots gsiBot server y con el módulo de pruebas automatizadas aprovechando las funcionalidades que estos ofrecen.

5.4.1 Motivación

La mayor dificultad que entraña para al programador el desarrollo de un bot en el lenguaje AIML es conseguir manejar toda la información contenida en los archivos que constituyen el bot. A pesar de que la sintaxis de AIML es un lenguaje inteligible por un ser humano, su lectura es pesada y repetitiva para este, por lo que es sumamente complicado detectar un error en el código o simplemente depurarlo. AIML es un lenguaje basado en reglas, en el bot es más completo cuanto mayor sea el número de estás. Este tipo de lenguajes acarrean una dificultad de programación exponencial, pues a medida que el bot es más completo, mayor es el número de reglas y mayor la dificultad para encontrar la raíz de un error.

Para muchos de estos lenguajes basados en reglas, como es el caso de Proloj, CLIPS o JESS existen entornos de desarrollo integrados que permiten depurar los errores incluso en programas complejos con multitud de archivos. A pesar de de que la especificación de AIML data del año 2001, la comunidad de usuarios no ha alcanzado un tamaño suficiente como para que se haya desarrollado un entorno de este tipo.

Este tipo de entornos constituyen herramientas muy útiles de trabajo, sin embargo su desarrollo es muy complejo y está fuera del alcance de este proyecto. En el módulo de gestión de conocimiento se ha llegado a una solución intermedia, a través de una herramienta para la asistencia en el desarrollo de la base de conocimiento de bots, en la que no se permite al usuario editar manualmente los archivos AIML sino que se le abstrae de este nivel de programación, lo que permite a todo tipo de usuarios trabajar con la herramienta sin necesidad de tener conocimientos sobre el lenguaje.

Por otro lado, desde el punto de vista del cliente, es habitual solicitar un bot que sea capaz de atender al visitante de su página Web dando respuesta a un conjunto -relativamente reducido- de preguntas. En este escenario, la herramienta de gestión de conocimiento es especialmente interesante pues permite añadir conocimiento a un bot a

gran velocidad, por lo que se puede desarrollar el sistema solicitado en muy poco tiempo.

5.4.2 Modelo funcional

El modulo de gestión de conocimiento proporciona una interfaz de usuario sencilla, en forma de asistente, a través del cual se pueda trabajar en el desarrollo de la base de conocimiento de un bot con total abstracción de la programación en AIML. El objetivo es facilitar la programación de corpus en AIML, posibilitando el que un usuario sin conocimiento de AIML sea capaz de programar un bot para que responda como desea.

El módulo de gestión de respuestas genera el contenido de los archivos AIML así como del archivo KTF asociado al conocimiento incluido en los mismos, una función de la información solicitada al usuario en el asistente, de una forma totalmente trasparente para este. Así se consigue abstraer al usuario de la programación propiamente dicha.

El código autogenerado se escribe siguiendo unas plantillas (definidas en el apartado 4.3) que evitan las malas prácticas [29], que constituye la fuente más frecuente de errores en los sistemas de bots AIML complejos. Además, el diseño del código minimiza la probabilidad de que existan interferencias con otras partes del código al generar un código lo más preciso posible. Tal y como se describirá más adelante, en la programación convencional en AIML, la repetición de patrones trata de evitarse mediante la definición de reducciones. Esta treta, que ahorra la repetición de gran cantidad de código, puede provocar errores indeseados en otras reglas de código, tanto de las definidas anteriormente a las definidas con posterioridad a la reducción. Por ello, y puesto que la generación del código es automática se evitan las reducciones duplicando patrones para conseguir el mismo resultado sin el riesgo de errores colaterales.

Para garantizar la correcta integración del código autogenerado con el existente en la base de conocimiento, la herramienta se integra con el módulo de pruebas automatizadas, de manera que el asistente incluye un paso de verificación del funcionamiento del bot en la que se ejecutan todas la pruebas definidas para el corpus así como las nuevas pruebas asociadas al conocimiento que se desea incluir. Sólo en caso de que la integración sea satisfactoria se incluirán los cambios en la base de conocimiento del bot.

Gestión de conocimiento también permite editar las respuestas dadas por el bot en caso de que se deseen modificar con posterioridad al momento de su inclusión en la base de conocimiento.

5.4.3 Modelo estructural

La arquitectura del sistema está centrada en torno a la interfaz Web, puesto que será el usuario quien irá autorizando las acciones a que realizará la herramienta en función de sus decisiones. Aunque se expondrá su con detalle en el apartado 5.4.4, es conveniente explicar someramente su funcionamiento para entender el papel que desempeña cada componente. Para generar el código AIML necesario, el asistente pide al usuario que exprese con distintas frases una misma idea, la consulta para la que quiere que el bot de respuesta. A partir de ahí, el sistema realiza una serie de deducciones a las que solicita confirmación al usuario, hasta que infiere cuales es el código AIML que se debe incluir.

La arquitectura esta formada por varios componentes tal y como se muestra en la siguiente figura.

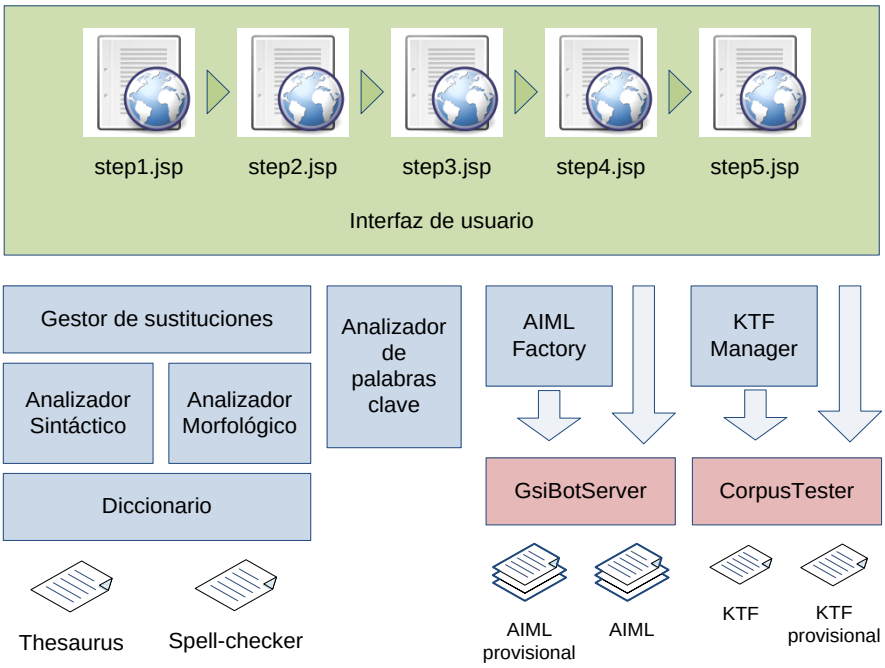

Fig. 55. Arquitectura del módulo de gestión de conocimiento.

El analizador sintáctico utiliza la información del diccionario *spell-checker* para comprobar si todas las palabras introducidas por el usuario en cada una de las frases están correctamente escritas.

El Analizador Morfológico busca la raíz morfológica de cada una de las palabras presentes en las frases introducidas por el usuario en un diccionario *Thesaurus*. Sirve para reconocer singulares y plurales como la misma palabra así como tiempos verbales. Con esto se pierde la posibilidad de hablar en pasado o hacer consideraciones particulares acerca de la cantidad de alguna cosa si no va precedido por un determinante numeral.

El gestor de sustituciones maneja la información obtenida por el analizador morfológico, guardándola en memoria para tener un fácil acceso. Además marca aquellas sustituciones que aún no han sido guardadas en el código AIML para incluirlas una vez se confirmen el proceso. El gestor de sustituciones también utiliza el diccionario *Thesaurus* para obtener sinónimos de las palabras introducidas por el usuario. En versiones posteriores del proyecto, el objetivo es que el gestor de sustituciones se también se encargue de analizar el AIML en busca de reglas que representen sustituciones. Con esto se pretende eliminar errores producidos por la existencia de reducciones en el código AIML (que en generar constituyen una mala práctica de programación) que, en la versión actual, el módulo no es capaz de reconocer.

El analizador de palabras clave estudia en función de la posición en que aparecen las palabras clave en cada una de las frases introducidas por el usuario y de su frecuencia de aparición el código de AIML y el código para el Knowledge Test File que se debe generar. El diseño de este bloque es el más delicado puesto que debe tener en cuenta todas las características del lenguaje AIML y del idioma para el que se programa el bot a fin de que la base de conocimiento resultante sea escalable y a la vez funcione como desea el usuario.

Los componentes AIML Factory y KTF Manager se encargan de generar, a partir de las deducciones hechas por el analizador de palabras clave y el gestor de sustituciones, el código AIML y XML que respectivamente se incluirá en el corpus y en el fichero KTF asociado a este. Estos componentes utilizan las plantillas definidas en el apartado 4.3.

En el diagrama de la arquitectura se presenta una base de conocimiento provisional que será cargada en el servidor de bots. Esta es la base de conocimiento que alberga los cambios hechos por el usuario y que permite a este, mediante el módulo de prueba de bots, comprobar que las modificaciones introducidas en la misma funcionan como se desea y que no interfieren con el conocimiento previamente existente. Si la comprobación es satisfactoriamente la base de conocimiento provisional se copiará a la base de conocimiento principal. Con esto se garantiza que, aunque en algún caso, el

código AIML generado no cumpla con las expectativas, la base de conocimiento no se verá contaminada puesto que los cambios serán desechados.

5.4.4 El proceso de enseñanza: La interfaz de usuario

Por medio del asistente que presenta la interfaz de usuario del módulo de gestión de conocimiento el usuario enseña al bot a que responda correctamente a una determinada consulta. El proceso se divide en cinco pasos en los que se pide al usuario que exprese con distintas frases una misma idea, la consulta para la que quiere que el bot de respuesta. A partir de ahí, el sistema realiza una serie de deducciones a las que solicita confirmación al usuario, le pide que introduzca la respuesta que dará bot a esas consulta y finalmente genera el código AIML que se incluirá en la base de conocimiento del bot.

En el primer paso, la aplicación solicita al usuario que escriba varias frases diferentes que signifiquen lo mismo (o similar). Así se pretende inferir qué palabras deben aparecer en el patrón de código AIML. La interfaz de este paso se muestra en la Fig. 56.

Cuantas más frases diferentes se introduzcan mayor riqueza de expresión tendrá el bot al terminar el proceso, puesto que es sistema introduce mayor cantidad de reglas en el código AIML pues dispone de más de palabras clave. Habitualmente, cuando se desarrolla un bot de forma manual se pretende reducir el número de patrones, pues la repetición de código es un proceso tedioso; sin embargo, al estar automatizado se convierte en un recurso que proporciona riqueza a la base de conocimiento del bot.

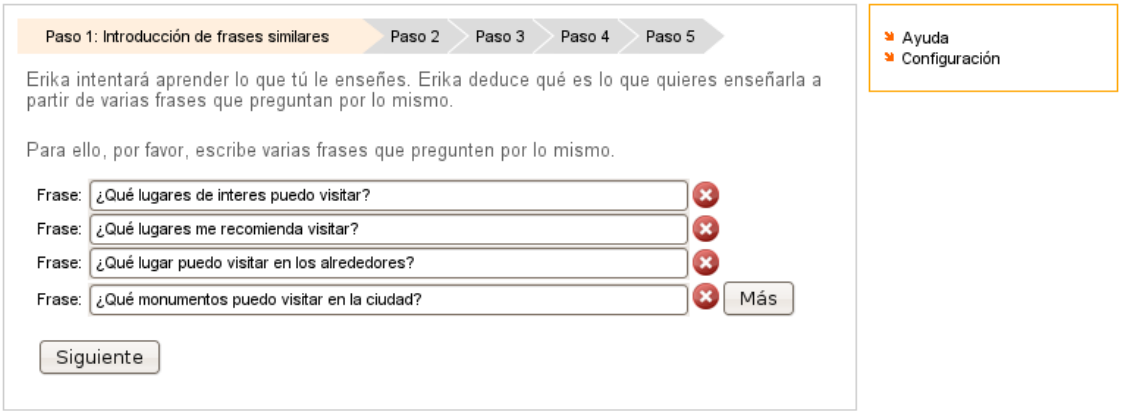

Fig. 56. Interfaz del primer paso del asistente: introducción de frases similares.

En el segundo paso el sistema, por medio del analizador sintáctico y morfológico habrá analizado todas las frases para descubrir si hay palabras que estén mal escritas, plural o conjugaciones verbales y las sustituye por singulares e infinitivos. También se

buscan sinónimos entre las frases introducidas por el usuario y se indica que se considerarán como la misma palabra. Todas estas deducciones se anuncian al usuario y se le da la opción de cambiarlas.

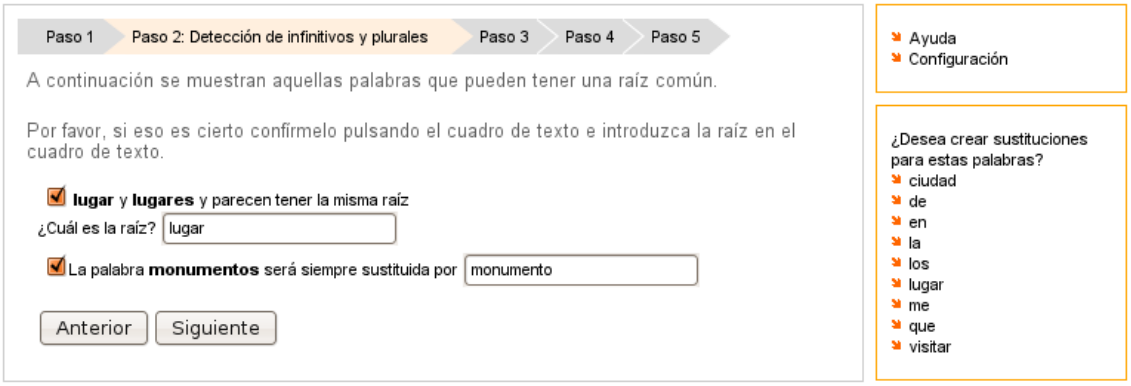

Fig. 57. Interfaz del segundo paso del asistente.

Una vez aceptadas, a las frases introducidas en el primer paso se le aplican las correcciones y sustituciones acordadas en el segundo paso de manera que las decisiones que se tomen de ahora en adelante se harán sobre este conjunto de frases.

En el tercer paso las frases vuelven a ser analizadas para buscar las palabras que contienen la información principal, aquellas que deben incluirse en el patrón del código AIML. Los resultados se presentarán en forma de tabla coloreada en la que aparecen las frases introducidas por el usuario y también las frases resultantes de aplicar las sustituciones del segundo paso. Las palabras relevantes, que se van a incluir en el código AIML, se marcan en color tal y como se muestra en la Fig. 58.

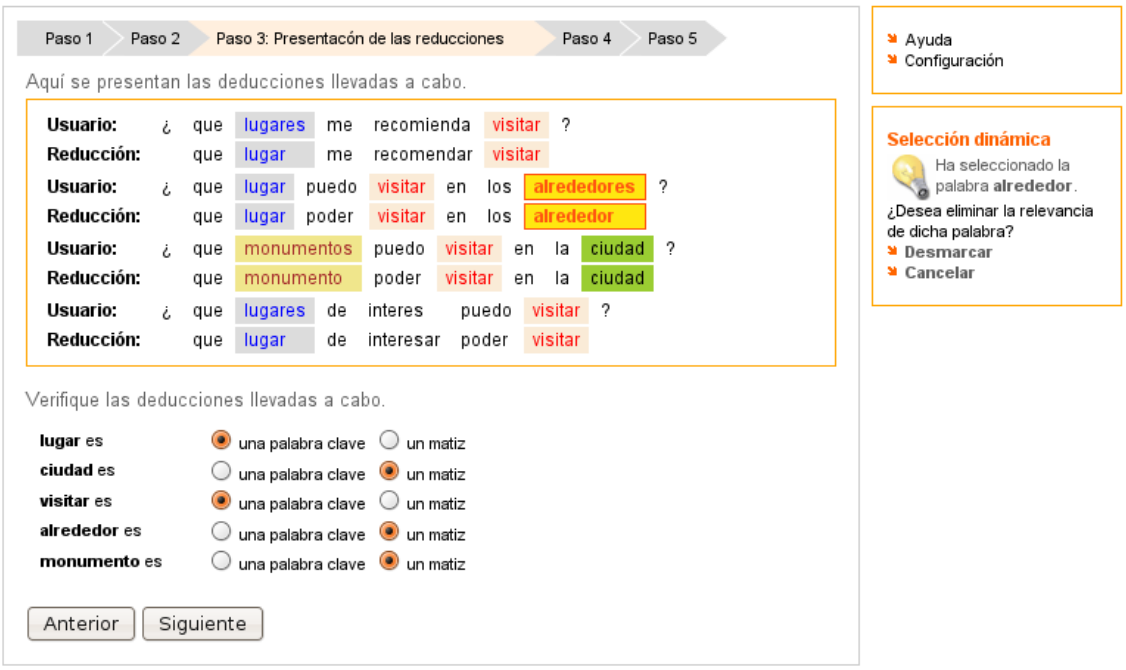

Fig. 58. Interfaz del tercer paso del asistente: presentación de las reducciones.

Las palabras relevantes pueden ser consideradas como palabra clave o palabra matiz. Las palabras clave son aquellas necesarias para comprender el concepto que se está considerando, las palabras matiz son relevantes pero se puede comprender el concepto sin su utilización. El asistente considera palabras clave aquellas que se encuentran repetidas en todas las frases introducidas por el usuario. Los adverbios interrogativos constituyen una excepción, son privados de ser palabra clave aunque cumplan esta condición por no ofrecer información de suma relevancia[26]. El asistente considera palabra matiz aquellas que, no siendo palabra clave, se repiten en dos o más frases de usuario. La incorporación de palabras matiz hace que el sistema genera un mayor número de patrones diferentes por lo que permite añadir mayor nivel de particularización a la hora de introducir las respuestas. El usuario tiene la posibilidad de modificar estas decisiones cambiando el nivel de relevancia de las palabras; pudiendo incluso asignar el de palabra clave a algunas que sólo aparezcan en una frase.

En función de esta clasificación, el sistema genera un conjunto de reglas AIML que se agrupan formando patrones de tipo *contains*. Esta estructura de reglas se describe detalladamente en el apartado 4.3.14.3. Posteriormente se ofrecerá al usuario la posibilidad de asignar una respuesta diferente a cada patrón, aunque pueden ser la misma.

[26] Además su condición de adverbio interrogativo los posiciona al comienzo de la frase por lo que los patrones que los contengan tendrán un nivel de prioridad superior a todos aquellos que no los contengan y que posiblemente acumulen mayor cantidad de palabras relevantes.

Para cada frase, el sistema genera un patrón general y varios patrones particulares. El patrón general es aquel que contiene todas las palabras clave presentes en la frase, es por tanto el patrón con menor nivel de prioridad. Los patrones particulares agrupan tanto las palabras clave como las palabras matiz. Por cada palabra matiz se genera un patrón *contains* que contiene las palabras clave y la palabra matiz (en el orden en que aparezcan en la frase) y además un patrón con todas las palabras clave y palabras matiz. Si al estudiar las palabras clave de toda la frase se da la situación en que un patrón está repetido este no se vuelve a generar.

Existen dos excepciones en los algoritmos descritos:

- Si la única palabra clave es la primera palabra de una frase, no se genera el patrón general que sólo contendría dicha palabra clave, puesto que produciría un patrón de muy alta prioridad. Dicha palabra se sigue considerando como palabra clave a efectos de que aparece en todos los demás patrones particulares.

- Si en una frase hay demasiadas palabras matiz, asta el punto de que el patrón particular que contiene todas las palabras clave y las palabras matiz es un patrón *contains* de cinco o más parámetros, se agrupan las palabras matiz consecutivas como un único parámetro con el objeto de reducir el número de reglas generadas.

En el cuarto paso se pide al usuario que introduzca las repuestas que debe dar el bot para cada una de las frases que introdujo el usuario en el primer paso. Las respuestas pueden coincidir para varias o incluso todas las frases. Sin embargo, aunque se de esta circunstancia, el hecho de introducir diversos patrones *contains* favorece que, a posteriori, la probabilidad de que se produzcan interferencias sea menor. El usuario puede además definir en cada respuesta el valor de cualquier variable AIML, incluida la variable `topic`.

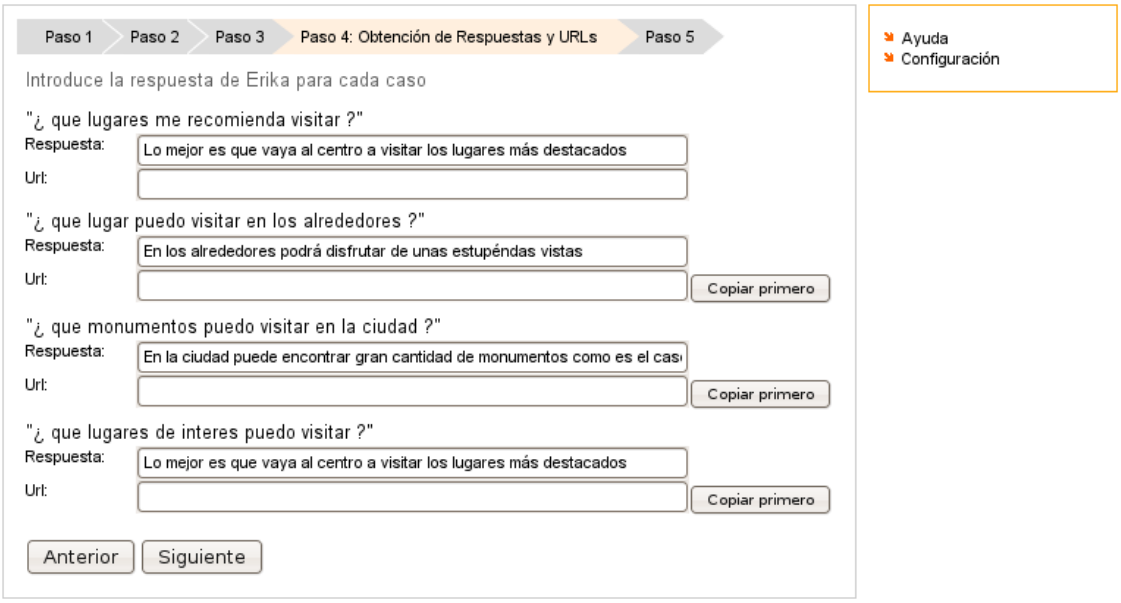

Fig. 59. Interfaz del cuarto paso del asistente: Introducción de las respuestas.

En el quinto paso se realizan las comprobaciones de que el proceso ha concluido satisfactoriamente. Entre el cuarto y quinto pasos el sistema genera el código AIML y lo incorpora a una copia de la base de conocimiento denominada base de conocimiento provisional. También actualiza el Know-ledgeTestFile con los nuevos pares consulta-respuesta. Se despliega un bot asociado a esta base de conocimiento para poder realizar un test corrección con el nuevo fichero de prueba que comprobar si existen interferencias de las nuevas reglas AIML introducidas con las existentes en la base de conocimiento. En caso de que el resultado del test sea satisfactorio se dará la opción al usuario de conversar con el bot que incluye las reglas recién generadas. Si el bot ha adquirido los conocimientos correctamente se da la opción al usuario de guardar los cambios, lo que hará que se vuelquen a la base de conocimiento principal.

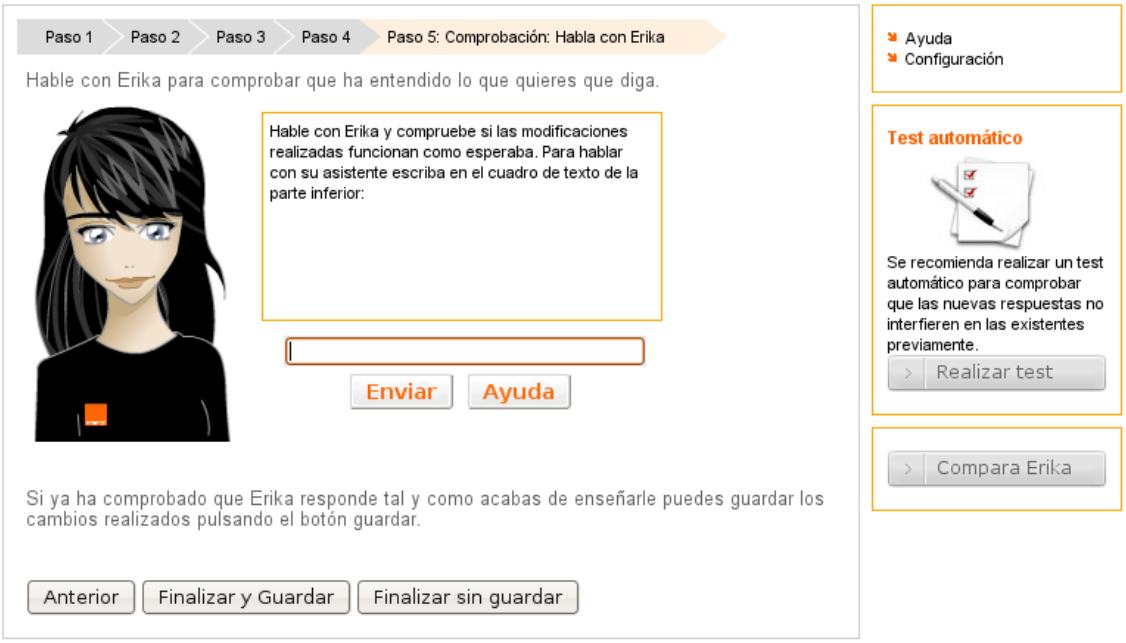

Fig. 60. Interfaz del quinto paso del asistente: Introducción de las respuestas.

5.4.5 Gestión de respuestas

La aplicación de edición de respuestas es una de las dos aplicaciones desarrolladas para la edición de una base de conocimiento existente. Gestión de Respuestas facilita la modificación de la base de conocimiento AIML a usuarios sin conocimientos de programación permitiendo modificar tanto la respuesta que da el bot como los atributos AIML asociados a dicha respuesta. Para facilitar esta tarea la aplicación visualiza el corpus descrito en el Knowledge Test File, permitiendo que los usuarios puedan editar directamente dicho corpus y gestionando la actualización de la base de conocimiento.

Gestión de Respuestas permite al usuario visualizar en su totalidad los pares: consulta-respuesta para los que el bot ofrece una respuesta, basándose en el archivo Knowledge Test File asociado al bot. Estrictamente hablando no están todas las entradas de usuario a las que el bot responde pero sí todas las respuestas diferentes que el bot retorna.

Todas las apariciones en la base de conocimiento de dicha respuesta serán reemplazadas por la nueva. De esta manera se consigue editar la base de conocimiento de una forma fácil y que desprende al usuario de toda necesidad de conocimientos en AIML.

La aplicación Gestión de Respuestas hace uso del módulo de la aplicación servidor de bots para acceder a la base de conocimiento y así generar un listado de respuestas del bot que se presenta por pantalla. A través de la interfaz Web se pueden editar las respuestas. Los cambios en la base de conocimiento se hacen efectivos a través del gestor de AIML.

El gestor de AIML es un manejador de archivos AIML que conoce la ubicación en el disco de cada uno de los archivos AIML que forman la base de conocimiento. El gestor de AIML busca en todos los archivos la respuesta que el usuario ha marcado para ser sustituida y la reemplaza por la nueva respuesta.

El servidor de Bots está configurado para refrescar el sistema si se detectan cambios en el AIML por lo que la próxima vez que se acceda al Bot se verán reflejados los cambios en la base de conocimiento.

Al iniciar la interfaz Web de usuario se visualiza una lista de los Bots (botid) cargados en ese momento. Al seleccionar cada uno de ellos se carga el Knowledge Test File asociado a dicho bot. A partir de este momento todos los cambios realizados sólo afectan al bot seleccionado. De esta manera se puede seleccionar el bot cuya base de conocimiento se desea modificar.

El listado de respuestas ofrecidas por cada bot se presenta agrupado en temas para facilitar al usuario la labor de encontrar la respuesta que desea modificar. El proceso que debe realizar el usuario para editar una de las respuestas es sencillo: basta con hacer clic encima de la respuesta que se desea modificar y aparecerá un cuadro de texto en su lugar. El cuadro de texto contiene la respuesta actual del bot. Al editar el contenido del cuadro de texto y confirmar los cambios a través del botón Cambiar la respuesta habrá cambiado.

Capítulo 6

Evaluación del proyecto

En éste capítulo se presentan tres casos de estudio concretos en los que ha sido utilizado gsiBotServer. Cada uno de ellos supone un aporte al estado del arte de tecnologías de bots.

6.1 OrangeBot

El proyecto OrangeBot constituye la primera distribución del gsiBot para cliente. El proyecto, descrito en [30], fue orientado al desarrollo del producto por lo que inicialmente no se utilizaron las herramientas de gestión de conocimiento o de realización de pruebas. En las especificaciones del proyecto se recogía la necesidad de crear dos bases de conocimientos separadas, con información extraída de las secciones FAQ de la Web de Orange España correspondientes a la parte de cliente y a la de distribuidores.

Tal y como se muestra en la Fig. 61 el número de reglas escritas para estos bots alcanza las 4558 y 2883 respectivamente. Estas dos bases de conocimiento no son totalmente independientes, ya que la parte de conversación sobre temas generales es común a ambas. Por ello el numero total de reglas diferentes escritas para OrangeBot asciende a 5007.

Nombre del bot	Dedicado	Estado	Tamaño	Acciones
Distribuidores	Si	READY	2883	Eliminar Probar
OrangeBot	Si	READY	4558	Eliminar Probar
3DTour	Si	READY	6025	Eliminar Probar

Fig. 61. Estadísticas de despliegue de OrangeBot y Distribuidores

Las dificultades encontradas en el desarrollo de un sistema AIML tan complejo como este marcaron la necesidad de crear las herramientas de asistencia que se describen en esta memoria. OrangeBot constituye las primeras pruebas de usabilidad y aceptación tanto de las herramientas de gestión de conocimiento como del servidor de bots.

Las lecciones aprendidas durante la programación de OrangeBot y Distribuidores, tanto en la escritura manual como utilizando las herramientas de asistencia, han sido utilizadas para el perfeccionamiento de dichas herramientas.

La interfaz Web proporcionada con el OrangeBot se abre en una nueva ventana del navegador, en la que se incluye una animación flash [6] con la cara del bot. Esta animación cambiará en función del estado de ánimo del bot en cada punto de la conversación. En el cuadro de texto situado debajo de la imagen permite escribir la frase que se enviará al bot. Cuando la respuesta del bot haga referencia a una URL está se mostrará automáticamente en la ventana del explorador.

Fig. 62. Interfaz de acceso con la imagen corporativa de Orange.

6.2 3DTour

El Bot3DTour es el bot con conocimientos de turismo desarrollado para el proyecto 3DTour. Bot3DTour ha sido desarrollado íntegramente utilizando la herramienta de gestión de conocimiento y ha sido testeado utilizando la herramienta de pruebas automatizadas. En 3DTour se lleva a cabo la integración de bots en un mundo virtual 3D, por medio de la interfaz de acceso a los bots el cliente del mundo virtual puede acceder a los bots para conversar con ellos como si de otros usuarios se tratara. 3DTour es un proyecto pionero al conseguir la integración de estas dos tecnologías y dibuja un escenario idóneo para la aplicación de las tecnologías de bots.

6.2.1 Desarrollo del corpus de BOT3DTour

Para 3DTour se desarrolla la primera base de conocimiento utilizando únicamente el asistente de gestión de conocimiento. Constituir una prueba diferente, puesto que se elabora la base de conocimiento desde cero, y por tanto se han encontrado dificultades diferentes a las encontradas en el desarrollo de OrangeBot que han permitido continuar con el perfeccionamiento de las herramientas.

El Bot3DTour es comparable en número de reglas a OrangeBot: 6025 frente a 5007 categorías respectivamente. Sin embargo, el uso de las herramientas de asistencia redujo el tiempo de desarrollo del nuevo bot a una jornada de trabajo de 8 horas, mientras que el desarrollo de OrangeBot constituyo, como se ha mencionado, un proyecto semestral[27]. Dada su gran extensión para ser incluido en este apartado, el corpus completo del bot BOT3DTour se recoge en el Anexo B.

6.2.2 Integración de bots con mundos virtuales

Desde el punto de vista tecnológico, el proyecto 3DTour plantea diversos retos. Por un lado, se pretende innovar utilizando la tecnología de mundos virtuales 3D en auge actualmente, sobre todo en el sector de ocio, enfocándolo hacia una aplicación corporativa del sector turístico. Se pretende así explotar la interactividad e inmersión presente en todo mundo virtual para ofrecer una visión lúdica que permita captar de manera eficaz la atención de los usuarios. Por otro lado, para dotar de más realismo a los servicios que se ofertarán en el mundo virtual se integrará la tecnología de bots, que además facilitará el acceso a la información a los clientes. Puesto que el entorno es propicio para camuflar los bots como otros usuarios humanos se incluirán también otros

[27] Es de justicia admitir que el proyecto OrangeBot incluye no solo el desarrollo del bot sino también de su interfaz de usuario y superar la curva de aprendizaje de AIML y programD. No obstante, la reducción de tiempo es un hito notable.

bots con el papel de usuarios en el mundo virtual y no solo empleados de manera que el usuario que se conecta perciba actividad en el escenario y se encuentre acompañado mientras explora el entorno virtual diseñado para 3DTour.

Los mundos virtuales tratan de simular entornos artificiales semejantes a la vida real en los que el hombre interacciona con máquinas. Por ello, facilitan una comunicación con mayor interactividad e inmersión, que crea una nueva forma de socializar a las personas, entretener, innovar y realizar transacciones comerciales.

Actualmente los mundos virtuales tienen una gran acogida, sobre todo dentro del sector ocio. Debido a este auge y como hemos visto, han surgido múltiples mundos virtuales, entre los que podemos destacar Second Life, OpenSim y Wonderland. Para el desarrollo de 3DTour se ha decidido utilizar Opensim, la tecnología de Sun Microsystems para desarrollar nuestro mundo virtual que se enfocará en los servicios turísticos.

Como se ha dicho anteriormente, la integración de la tecnología de bots en el proyecto, facilitará el acceso de los clientes a la información. Esto nos permite exponer la información de los servicios turísticos de forma real e incluso permite al usuario interactuar con la organización fácilmente, aportando gran dosis de realismo a los procesos que se van a llevar a cabo.

6.2.2.1 Aplicaciones y escenarios de mundos virtuales

La tecnología de mundos virtuales abre un amplio abanico de posibles aplicaciones y escenarios. Su principal baza es ofrecer al usuario la posibilidad de hacer algo que no puede hacer, en ese momento y en ese lugar, en la vida real. El avatar del usuario, puede transportarse inmediatamente a puntos del mundo virtual pudiendo visitar cualquier monumento y lugar de interés recreado en el entorno virtual, permite realizar varias acciones a la vez pudiendo mantener varias conversaciones al mismo tiempo o mientras se asiste a una exposición o una reunión.

Muchos mundos virtuales están orientados a entornos profesionales -los que no están orientados al ocio- y son utilizados en escenarios de colaboración y formación inmersiva en grandes organizaciones.

Como ocurre con otras tecnologías innovadoras, la tecnología de mundos virtuales se está aplicando a la educación [31] y formación de personal. Por ejemplo, en el entorno sanitario se puede educar a los jóvenes mediante un juego en buenas prácticas de salud con niveles de dificultad crecientes, competiciones individuales y por equipos; en el ámbito docente se puede extender el mundo virtual para permitir a los alumnos

realizar simulaciones de experimentos, en el campo de la seguridad en empresas o edificios públicos recreando simulaciones de incendios en los que se pueda incluir un fuego en el escenario para hacerlos más realistas...

Los mundos virtuales también son una herramienta con múltiples posibilidades en la propagación y preservación de la cultura [32]. Actualmente existen varios mundos virtuales para fomentar el turismo y el acceso a la cultura. Todos ellos se centran en la recreación de monumentos, museos, ciudades u oficinas de turismo, los cuales únicamente proporcionan información. Este es el caso del casco histórico de Gijón o la plaza mayor de Valladolid.

Sin embargo hay que saber valorar las posibles aplicaciones con sentido común: si en un mundo virtual un usuario puede transportarse de un punto a otro a su antojo no tiene sentido que se diseñe un coche o un carruaje como medio de transporte por muy realista que sea, pues ningún usuario querrá usarlo ya que es una pérdida de tiempo y no tiene valor para el usuario [33]. Igual ocurre con las propuestas de recreación de aulas de universidad, de manera que se imparta clase "presencial" a través del mundo virtual en lugar de dar clase en las aulas de la universidad. Si se plantea la clase como una reunión en la que los alumnos se citan con el profesor en el mundo virtual y éste expone la lección con una presentación de diapositivas, es más que probable que las transparencias se vean mejor impresas en el pupitre del alumno o en su navegador, de manera que hacer uso de las posibilidades del mundo virtual para mostrar presentaciones en un caso como este no tiene sentido. Sin embargo, el valor añadido que los mundos virtuales pueden aportan a un caso como esté es mostrar una presentación en 3D para comprender mejor la lección, facilitar a los alumnos el asistir a clase en caso de ser discapacitados o tener que desplazarse un largo trayecto para asistir a clase todos los días, o realizar simulaciones y experimentos a través de la potencia del servidor del mundo virtual [34].

Con 3DTour y gracias a la tecnología de los mundos virtuales y su integración con bots, se pretende dar valor añadido e ir más allá de la meramente informativa para que los usuarios además puedan interactuar entre ellos y con la empresa, así como proporcionar distintos servicios reales que puedan ser consumidos en el mundo virtual.

Evidentemente toda tecnología innovadora en la línea del internet del futuro ha de ir demostrando su valor en casos de uso concretos y nadie como los que conocen las necesidades de una organización para identificar las oportunidades en donde probar su utilidad. Es por eso por lo que en 3DTour se ha realizado este proyecto en un marco demostrativo orientado a el sector hotelero hablando, a través de la colaboración

profesionales del sector para considerar la posibilidad de implantar el proyecto como expositor de los servicios de una cadena hotelera.

Fig. 63. Escenario típico de un mundo virtual: Vista exterior del hotel 3DTour.

6.2.2.2 Tecnología de bots en mundos virtuales

El mero hecho de que los bots sean fácilmente reconocidos como máquinas hace que los interlocutores humanos descuiden la conversación al comunicarse con ellos. Esto hace complica la evaluación de la base de conocimiento de un bot, pues lo usuarios tienden a dirigir la conversación con palabras clave en lugar de construir frases correctas. Los mundos virtuales constituyen un escenario idóneo para la evaluación de bots conversacionales pues cualquier usuario que converse con ellos tiende a considerarlos, a priori, como otro usuario del sistema [35]. De esta manera se las conversaciones registradas en el logger pueden ser utilizadas para la mejora de la base de conocimiento del bot.

Fig. 64. Avatar de un bot recepcionista en un mundo virtual

Por otro lado, la integración de tecnología de bots en el mundo virtual supone una aporte de valor añadido para al mundo virtual. Añade un mayor grado de interacción, pues permite realizar acciones tales como reservar una sala de juntas, consultar las reservas disponibles o a través de una conversación en lenguaje natural, dando así una mayor dosis de realismo al escenario. En general los usuarios valoran muy positivamente los sistemas con los que se puede interactuar a través del lenguaje natural para desempeñar una tarea [36].

La inclusión de bots en el mundo virtual también convierte a este en un mundo más acogedor, pues el usuario percibe mayor grado de actividad al encontrar un mayor número de avatares en el escenario [37]. Este concepto, ha llevado a la práctica en [38] con la inclusión de bots en foros con el propósito de responder rápidamente a las preguntas cuya respuesta conoce el sistema y dar así una sensación de eficiencia en el servicio.

6.2.2.3 Bots en el sector turístico

Al igual que los mundos virtuales, los bots conversacionales están siendo utilizados en diversos campos de aplicación como agentes que sirven información o ofrecen promociones a través de mensajería instantánea, agentes expertos en FAQ incrustados en páginas Web, servicios de atención al cliente o sistemas domóticos.

En general, para que un bot conversacional sea apto para realizar una tarea es necesario que el volumen de conocimiento que necesita conocer sea acotado. Esto no significa que el conocimiento del bot no pueda cambiar y ampliarse a los largo del

tiempo, pero sí es necesario que dicho conocimiento se ciña a un tema concreto (o a un conjunto de temas) para así poder ofrecer mejores capacidades de conversación.

Este es precisamente el caso del sector turístico. Bien sea como promotor cultural o como empleado de un complejo hotelero, el conocimiento que necesita el bot se restringe a conocimientos generales acerca de la región en que se encuentra y conocimientos más a fondo del monumento u hotel para el que trabaja. Los bots en el sector turístico constituyen una opción de valor pues prácticamente pueden desempeñar el trabajo que realizan los empleados de una oficina de turismo o un recepcionista de un hotel.

En el caso de un empleado de un hotel, el carácter inteligente de los sistemas de bots les permite actualizar su base de conocimiento de manera que siempre disponen de la última información acerca de las habitaciones disponibles, los horarios en que están ocupadas las salas de juntas, tienen acceso a la base de datos sobre precios de los servicios ofrecidos por el hotel y también pueden recoger información de festejos y eventos programados en la ciudad.

Los sistemas de bots también son capaces de recolectar información a partir de las respuestas del usuario pudiendo así intuir sus gustos y preferencias. Por lo tanto, se puede plantear la conexión de el sistema de bots con un sistema recomendador de tal manera que este aconseje a su interlocutor sobre espectáculos o actividades a realizar.

6.3 Servicios de bot de un terminal Android

El acceso a Internet desde dispositivos móviles se está convirtiendo una realidad. Cada vez es más frecuente que los usuarios accedan a través de los dispositivos móviles de última generación a los contenidos de Internet. Bien es cierto que el acceso a Internet a través de dispositivos móvil es tecnológicamente posible desde hace una década. Sin embargo, es ahora cuando la tasa de accesos a la Web se está convirtiendo en una realidad. Esto es debido a:

- Los avances técnicos en los terminales con pantalla de mayor tamaño y procesador con más potencia capaz de ejecutar aplicaciones que tardan menos tiempo en mostrar el contenido de la Web consultada.
- No es necesario la adecuación de los contenidos por parte de las páginas Web, como ocurría con WAP, sino que los terminales poseen aplicaciones que acceden a la Web como cualquier otro navegador.
- La teleoperadoras empiezan a incluir la navegación Web en la tarifa plana del contrato.

- La tasa eficaz de acceso de las redes móviles es comparable a las de las redes de ADSL.
- Socialmente, el acceder a Internet desde el móvil, se está convirtiendo en un uso común.
- Muchos usuarios compran algunos dispositivos concretos que están en bogue, como el iPhone de MAC o los terminales con Android, pues ofrecen muchas funcionalidades y son los propios usuarios los que sobrestiman estas aplicaciones por el mero hecho de poder utilizarlas en estos terminales.

En este marco, y como parte de otro un proyecto de investigación sobre la viabilidad del desarrollo de aplicaciones para terminales móviles, es ha desarrollado una aplicación Java para terminales Android que permite acceder desde dispositivos móviles de última generación a los servicios de bots ofrecidos por el gsiBotServer.

6.3.1 Los terminales Android

A diferencia de de procesado que los modelos antiguos, los terminales modernos poseen capacidad de procesado suficiente para presentar en pantalla una página Web completa, además el tamaño de las pantallas tiende a ser mayor por lo que la cantidad de elementos que se pueden mostrar al mismo tiempo cada vez es mayor.

Los terminales con sistema operativo que ofrecen un sdk permiten el desarrollo de aplicaciones por terceros. Este es el caso de Android [AndURL] es un sistema operativo para dispositivos móviles y ordenadores basado en el núcleo Linux. Inicialmente desarrollado por Google. Los desarrolladores deben escribir código gestionado en lenguaje de programación Java a través de un SDK proporcionado por el mismo Google. Una alternativa es el uso del NDK (Native Development Kit) de Google para hacer el desarrollo en C en código fuente.

La mayor parte del código fuente de Android ha sido publicado bajo la licencia de software Apache, una licencia de software libre y código fuente abierto.

6.3.2 Interfaz adaptada para Android

El terminal Android accede a un servidor Web en el que se encuentra ubicado el gsiBotServer. Con una petición GET, que incluye como parámetros la consulta y el formato de la respuesta, obtiene del servlet correspondiente la respuesta en formato XML.

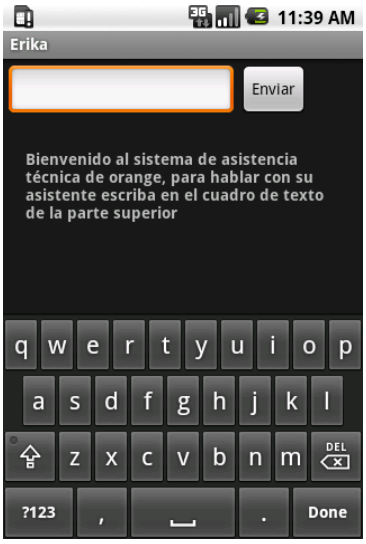

Fig. 65. Interfaz de acceso al bot a través de Android.

Inicialmente la aplicación tiene la apariencia mostrada en la Fig. 65. La aplicación obtendrá los datos introducidos en el cuadro de texto, les dará el formato correspondiente para incluirlos como parámetros en una petición GET e intentará acceder a la interfaz de acceso HTTP del servidor de bots descrita en el apartado 5.2.4.2.

Si el servidor no se encuentra o la pregunta está vacía, la aplicación devolverá un mensaje de aviso. En cambio, si el servidor devuelve correctamente la respuesta en XML bien formateada, la aplicación mediante un analizador sintáctico de XML obtendrá cada uno de los campos de la respuesta y presentará en el terminal los datos recibidos (pregunta, respuesta, URL recomendada).

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se evalúa con sentido crítico la consecución de los objetivos marcados para el proyecto y se presentan diferentes líneas de trabajo orientadas a completar el trabajo realizado en este proyecto y a incluir nuevas funcionalidades.

7.1 Conclusiones

Las funcionalidades de administración añadidas a gsiBotServer otorgan al servidor la madurez necesaria para ser desplegado en cliente. A través del panel de administración, se puede manejar el conocimiento de los bots cargados en el servidor sin necesidad de tener acceso a la máquina en que este hospedado.

El plugin de jQuery para el acceso a la interfaz HTTP del servidor de bot hace uso de la tecnología AJAX por lo que permite desarrollar nuevas interfaces de conversación, propias de la Web 2.0, en las que no se recarga el contenido completo de la Web después de cada consulta, lo que reduce el tiempo de respuesta del bot. Además permite incluir la interfaz de conversación en páginas HTML estáticas.

El módulo de pruebas garantiza de forma eficaz que la base de conocimiento funciona conforme a los requisitos marcados. La forma en que se definen las pruebas es apta para su traducción a partir de la definición de requisitos. Está orientada a ser definida previamente al desarrollo de la base de conocimiento, aunque también puede

utilizarse para testear otros bots existentes escribiendo las pruebas en que función de lo que se espera que responda el bot.

Tanto la posibilidad que ofrece el servidor de bots para desplegar y recargar bots en caliente, como la integración del módulo de pruebas permiten el funcionamiento del módulo de gestión de conocimiento.

El asistente de gestión de conocimiento ha sido utilizado para desarrollar satisfactoriamente la base de conocimiento asociada al corpus recogido en el Anexo B. El tiempo de desarrollo se ha reducido drásticamente con respecto al desarrollo manual de la misma base de conocimiento. El número de categorías de que consta el bot es mucho mayor cuando se desarrolla utilizando dicha herramienta, lo que disminuye la probabilidad de que se produzcan interferencias con posterioridad. Además se garantiza que el código generado está libre de malas prácticas.

7.2 Trabajos futuros

Se pueden definir multitud de líneas de trabajo ya sean para la realización de mejoras en los módulos definidos en este proyecto como para la inclusión de nuevas funcionalidades.

Una vez concluido este trabajo, y debido a la envergadura del mismo, muchos puntos han quedado inconclusos o son susceptibles de perfeccionamiento. Así en cuanto a perfeccionamiento de la plataforma se refiere se plantea:

- Desarrollar una aplicación Web con gestión de usuarios que integre todos los módulos desarrollados en este proyecto y que permita asociar permisos administración para cada uno de los bots.
- Incluir un sistema de estadísticas de acceso a cada uno de los bots del servidor, o estadísticas que recojan la evolución en el número de reglas de cada bot al utilizar la herramienta de gestión de conocimiento. Ambos escenarios implican el desarrollo de una base de datos.
- Desarrollar un módulo de plugins para el servidor de bots, que permita definir meta etiquetas en el código AIML que sean reconocidas y procesadas con el módulo de gestión de plugin para de manera que se incluya información generada dinámicamente en las respuestas de los bots.
- Incluir el uso de diccionarios de *spell-cheking* y de sinónimos en el módulo de gestión de conocimiento.

- Mejorar la interfaz de usuario del módulo de gestión de conocimiento dando una representación gráfica de los patrones que se van a generar, puesto que se da mayor información al usuario de lo que está ocurriendo sin que este requiera tener conocimientos de programación pues la interpretación de un gráfico domo un patrón es inmediata.

El desarrollo de ProgramD por parte de *aitools* no ha sido concluido y durante el desarrollo de este proyecto se han detectado numerosos errores de funcionamiento (*bugs).* Algunos afectan a partes importantes como son las que intervienen en la carga y recarga de archivos AIML así como de la multiplexación de los bots. Las clases envoltorio de programD, gsiProgramD, subsanan estos problemas. No obstante, constituyen una solución parcial y los bugs deben ser corregidos para poder utilizar la herramienta con fines comerciales ofreciendo totales garantías. Esto incluye el desarrollo de una batería de pruebas en jUnit.

Una línea de trabajo muy habitual en la comunidad de desarrolladores consiste en la mejora del avatar asociado al bot, sincronizando el habla con el movimiento de los labios o expresando sus "sentimientos" con expresiones corporales. Sin embargo, esta no es una línea de trabajo prioritaria por su remota relación de la inteligencia artificial y la escasa innovación que supondría.

La investigación de la integración de la plataforma con sistemas de recolección de información de la Web. Supondría la integración con ontologías de tal forma que el bot fuese capaz de responder a consultas cuya respuesta no estuviese previamente elaborada e incluida en su base de conocimiento sino que se obtuviese a partir de la ontología. Esta incorporación supone una avance importante hacia la inteligencia artificial, puesto que incorpora la búsqueda de información el la Web.

La incorporación de sistemas expertos como son los recomendadores pueden ofrecer una gran madurez a la base de conocimiento de un bot. De tal forma que a partir de la información de usuario obtenida de la conversación -reciente o no- mantenida con éste se pueden ofrecer recomendaciones o sugerencias de ocio, compras…

Otro paso más hacia la inteligencia artificial es la incorporación de sistema de *story-telling* o narrativa virtual [39]. Este tipo de sistemas disponen de bases de datos con multitud de fragmentos de narraciones etiquetados con una serie de precondiciones y Poscondiciones, de tal manera que el sistema pueda enlazarlas creando una historia con consistencia argumental.

La diversidad de enfoques de storytelling no se puede recoger una sola clasificación. En el relativamente corte periodo en que se está investigando en la

comunidad Internacional acerca de la narrativa interactiva se han identificado un varios problemas: la solución de compromiso entre la interactividad y la narrativa [40] [41], la dualidad entre el guión y el personaje [42] [43], narrativa casual, el control de la narración [44] y la relación entre la generación de la narración y su presentación [45].

La inclusión de un bot puede ofrecer que el usuario introduzca giros argumentales, de manera que no sea el sistema el que decida en todo momento (y en algunos casos, en los que varias opciones son posibles, de forma arbitraria) como continua la narración.

Por otro lado, son muchos los escenarios susceptibles de aplicar la tecnología de bots.

Paralelamente a este proyecto –como ya se menciona en 5.2.5- se ha desarrollado un sistema de integración del bots en mundos virtuales. Un mundo virtual [46] es un espacio virtual, en el que se representan un paisaje real o inventado al que el usuario puede acceder mediante un ordenador y moverse a través de él como si se encontrase dentro de ese escenario. En los mundos virtuales más avanzados se busca que el usuario pueda realizar casi cualquier acción que pudiera realizar en el mundo real, ya sea sentarse en un sofá, hablar con otros usuarios o utilizar un vehículo para desplazarse. La representación del usuario dentro del mundo virtual se denomina avatar.

Habitualmente, los mundos virtuales son espacios colaborativos accesibles a través de Internet en los cuales sólo son representados los avatares de los usuarios que se encuentran en ese comento conectados al mundo virtual y que están, por tanto, pendientes de las acciones que realizan a través de su avatar. La mayor problemática que tienen los mundos virtuales en cuanto a realismo se refiere es la total falta de usuarios de que adolecen algunos escenarios. Así, un usuario que entra en un mundo virtual y no es capaz de encontrar el avatar de ningún otro usuario no se encontrará a gusto y abandonará el mundo virtual. Con la inclusión de bots en el mundo virtual se puede crear un ambiente más acogedor [47], en el que el usuario perciba la presencia de otros usuarios que, aunque no sean humanos, puedan resolverle dudas sobre el uso del mundo virtual, o sobre el escenario en sí.

En este escenario, los bots tienen las mismas capacidades logísticas, a través de su avatar que un usuario humano. Por ello, es posible desarrolla bots con mayores posibilidades que los bots que se encuentran en una página Web. Incluso un bot puede pretender hacerse pasar por un usuario humano y es posible que otros usuarios le confundan como tal. Este es por tanto un escenario idóneo para estudiar la madurez de un bot, pues los demás usuarios no le tratarán de forma diferente como tal por el hecho de ser un bot [48].

En [49] se presenta un bot conectado a la base de datos de un foro y recoge la información contenida en este, de tal manera que da respuesta a aquellos hilos cuya consulta ya ha sido formulada con anterioridad. De igual forma, se puede considerar un bot conectado a un sistema de mensajería instantánea, o a una red social.

El campo de aplicación de este proyecto es el sector turístico, un espacio idóneo para el desarrollo de bases de conocimiento ya que en la recepción turística el conjunto de consultas que se realizan está acotado, de manera que mediante un sistema conectado a un bot se pueden atender las preguntas de los usuarios con un grado de satisfacción muy elevado. Igualmente idóneos son espacios de hostelería, restauración, teatros, cines… en los que se puede crear un corpus con la información del hotel, el restaurante, el reparto de la obra teatral, la cartelera del cine, sinopsis de la película.

La enseñanza electrónica o *e-Learning* también es uno de los grandes campos de aplicación de la tecnología de bots. Ya sea utilizando el bot como tutor electrónico que responde aquellas cuestiones cuya respuesta conoce y redirige al docente aquellas que no comprende [50], o como un examinador que realiza preguntas al alumno en un marco diferente en que se pueden medir otros parámetros como el tiempo de respuesta y en el que se puede adaptar el examen dinámicamente en función de las respuestas dadas [51].

Anexo A.

Manual de desarrollo

A.1 Puesta a punto del entorno de desarrollo.

Para trabajar en el proyecto es necesario instalar el kit de desarrollo de java, un entorno de desarrollo integrado que soporte Java, en este manual se indica como instalar *EasyEclipse*, el servidor de sevlets *Apache Tomcat* y la herramienta *ant 1.8* para la construcción del proyecto.

Se recomienda encarecidamente la utilización del sistema operativo Linux puesto que ofrece mayores opciones de configuración trabajando como servidor.

A.1.1. Instalación del kit de desarrollo de java.

Lo habitual que es que java ya esté instalado por defecto en la distribución de Linux. Podemos comprobarlo ejecutando el comando:

```
~$ java –version
```

En caso de que no se encuentre la versión de *java* se puede instalar tanto el JRE como el JDK mediante los paquetes *sun-java6-bin*, *sun-java6-jre* y *sun-java6-jdk*.

```
~$ sudo apt-get install sun-java6-bin, sun-java6-jre, sun-
java6-jdk
```

Es importante tener instalada la versión 6 de java, de manera que si la versión encontrada es una inferior conviene ejecutar el comando de instalación de paquetes igualmente con la opción *--reisntall*.

A.1.2. Instalación del entorno de desarrollo.

EasyEclipse [52] es un proyecto de software libre que empaqueta el entorno de desarrollo Eclipse [53] junto con algunos plugins recomendados en función del propósito de la distribución. EasyEclipse pretende liberar al programador de tener que encontrar todos aquellos plugins que le hacen falta e instalarlos resolviendo los consecuentes problemas de instalación. Esta es la razón por la que se ha decidido trabajar con EasyEclipse en este proyecto.

Puesto que la naturaleza del proyecto gsiBot es aplicación Web la distribución que se va a descargar e instalar es EasyEclipse Server Java [54].

Una vez descargada, basta con descomprimir el contenido en la carpeta deseada y hacer doble click en el icono Eclipse y la aplicación se lanzará automáticamente.

En el caso de no desear instalar EasyEclipse, alternativamente se puede instalar el IDE Eclipse e instalar los plugins para edición de ficheros XML, para sincronización con repositorios SVN y para trabajar con contenedores de aplicaciones.

A.1.3. Instalación y configuración de Apache Tomcat

El Apache Tomcat [55] es un proyecto de software libre que implementas las tecnologías de Servlets y de JSP de Java. Apache Tomcat actúa de servidor para las aplicaciones Web que utilizan estas tecnologías.

Existen diversas formas de instalación de Apache Tomcat. En esta guía se especifica aquella para la cual se tiene mayor control sobre los parámetros de configuración del servidor.

Para instalar Apache Tomcart es necesario descargar la última versión de la página oficial: el proyecto ha sido probado para las versiones 5.5.XX y 6.0.XX de Apache

Tomcat[28]. Se descomprime el archivador en la ubicación de instalación deseada para Apache Tomcat.

Permisos de ejecución para los ejecutables.

Para iniciar el servidor basta con ejecutar el archivo */bin/startup.sh* relativo a la ruta de instalación de Apache Tomcat. Dicho archivo se puede ejecutar desde la ventana de aplicación aunque es recomendable utilizar el terminal para estas tareas.

Para ello, desde la carpeta de instalación de apache Tomcat, se ejecuta el siguiente comando, donde *<tomcat_instalacion>* es la carpeta de instalación de Tomcat.

```
~/<tomcat_instalacion>$ ./bin/startup.sh
```

En caso de que el usuario utilizado no tenga permisos de ejecución sobre dicho archivo y por lo tanto la ejecución da un mensaje de error similar a

```
bash ./bin/startup.sh: Permiso denegado
```

Se le deben otorgar dichos permisos mediante la instrucción:

```
~/<tomcat_instalacion>$ sudo chmod 755 ./bin/startup.sh
~/<tomcat_instalacion>$ sudo chmod 755 ./bin/shutdown.sh
```

Se ha hecho lo mismo para el archive shutdown.sh.

Configuración de las variables de entorno.

En este punto el arranque del servidor seguirá fallando por no estar las variables de entorno correctamente configuradas.

```
~/<tomcat_instalacion>$ ./bin/startup.sh
Neither the JAVA_HOME nor the JRE_HOME environment variable is
defined. At least one of these environment variable is needed
to run this program.
```

Para la configurar las variables de entorno se ha de modificar el archivo *.bashrc* del usuario.

[28] En la fecha de redacción de esta memoria la versión 7.0 del software está en versión Beta.

```
$ sudo nano ~/.bashrc
```

Es posible que dicho archivo no exista en su sistema por lo que la consola muestre el contenido del archivo como vacío. Se añaden las líneas:

```
JAVA_HOME=/usr/lib/jvm/java-6sun
CATALINA_HOME=<tomcat_instalacion>
```

Donde *<tomcat_instalacion>* es la ruta del directorio de instalación del Apache Tomcat y */usr/lib/jvm/java-6sun* es la ruta de instalación del JDK de java. Es probable que esta ruta no sea exactamente igual puesto que las distintas versiones de java crean distintas carpetas. Es importante remarcar que la variable JAVA_HOME debe contener la ruta de instalación del JDK y no del JRE.

No es estrictamente establecer el valor de la variable CATALINA_HOME ya que el ejecutable startup.sh lo averigua a partir de su localización. Sin embargo, su definición será de utilidad más adelante.

Para hacer efectivos los cambios en la sesión actual se ejecuta:

```
$ source ~/.bashrc
```

Se puede comprobar la corrección de la operación ejecutando las instrucciones:

```
$ echo $JAVA_HOME
$ echo $CATALINA_HOME
```

Arranque del servidor.

Terminada la configuración se puede arrancar el servidor ejecutando:

```
~/<tomcat_instalacion>$ ./bin/startup.sh
```

Es posible, que debido a un conflicto entre las variables de entorno de usuario y de administrador, la ejecución del comando anterior lance una excepción por falta de permisos. Para solucionar dicho conflicto se puede ejecutar el lanzador de tomcat incluyendo el en una sola instrucción:

```
~/<tomcat_instalacion>$ JAVA_HOME=/usr/lib/jvm/java-6sun
```

```
./bin/startup.sh
```

Se puede acceder a la página del servidor mediante un navegador de Internet disponible en la url: *http://localhost:8080*

Permisos de administración y usuarios

El panel de administración de tomcat se encuentra en la url *http://localhost:8080/manager/html*, o bien en el link en la barra lateral *Administración > Tomcat Manager*.

Para acceder a dicho panel de administración es necesario disponer de un usuario y contraseña de administración que se requerirá mediante una ventana de autenticación. Por defecto Apache Tomcat no tiene ningún usuario de administración definido, por lo que se definirá uno nuevo. Para ello se ha de editar el archivo *<tomcat_instalacion>/conf/tomcat-users.xml*. Se deben añadir las siguientes líneas, en las que tanto el nombre de usuario como la contraseña se pueden cambiar:

```
<role rolename="manager"/>
<user username="admin" password="admin" roles="manager"/>
```

A.2 Configuración de Apache Tomcat con eclipse

Para poder trabajar con Apache Tomcat integrado desde eclipse es necesario definir el servidor y configurar el proyecto para trabajar con dicho servidor.

Para ello, en la vista 'Servers' 

, haciendo click con el botón derecho del ratón se selecciona 'New > Server' en el menú contextual. En caso de que la vista 'Servers' esté oculta puede hacerse visible en el menú 'Window > Show View > Servers'.

En la ventana del asistente de creación de un nuevo servidor se selecciona la versión de Apache Tomcat que se esté utilizando, Tomcat v5.5 Server o Tomcar v6.0 Server. En el nombre del servidor se debe mantener 'localhost'.

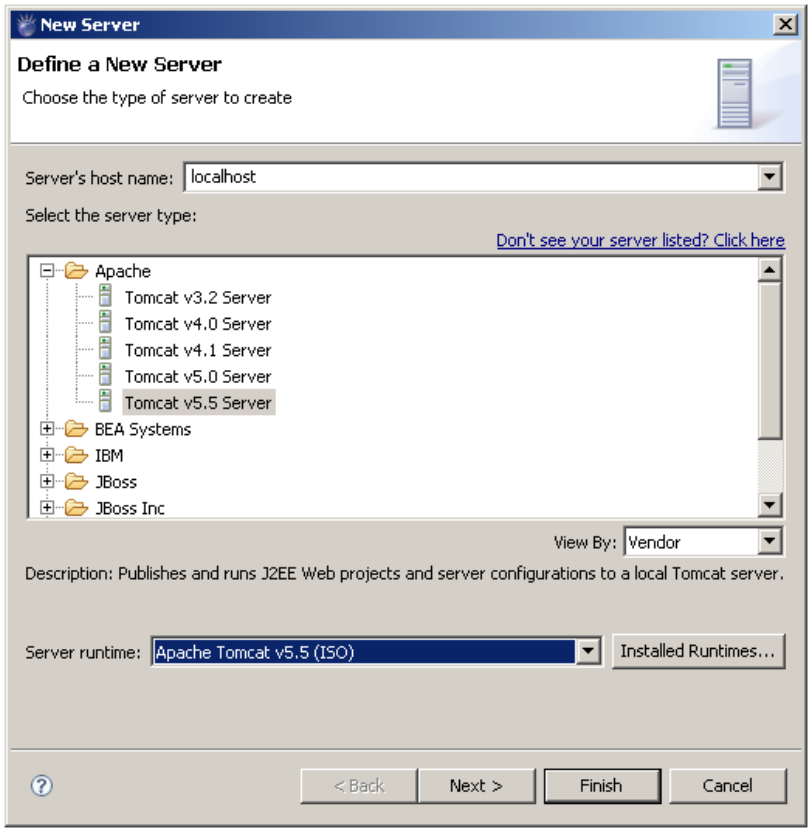

Fig. 66. Ventana de creación de un nuevo servidor en eclipse.

Para indicar a eclipse cual es el comportamiento que debe tener el servidor Tomcat instalado es necesario indicar la ubicación en el disco del servidor Apache Tomcat instalado, que en este caso es la carpeta en la que se ha descomprimido el servidor en el apartado A.1.3. Para ello se pulsa el botón 'Installed Runtimes' y se añade una nueva pulsando el botón 'Add…'[29]. En el asistente, en el primer paso se vuelve a seleccionar el tipo de servidor y en el segundo se le da un nombre que la identifique y se selecciona la ubicación en el disco del servidor. El nombre dado es una mera etiqueta que identifica el servidor.

[29] Si ya se ha realizado este paso con anterioridad Eclipse recordará la configuración anterior.

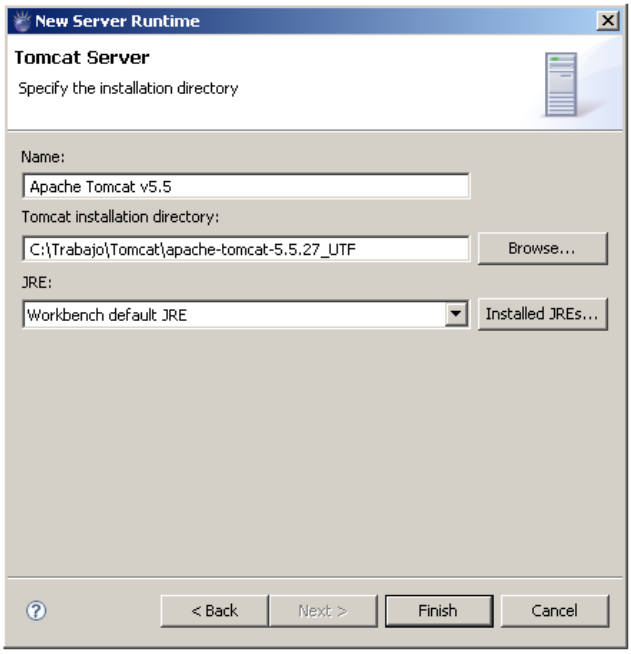

Fig. 67. Ventana de creación de un nuevo 'Server Runtime'.

Para evitar confusión es conveniente remarcar que eclipse utiliza el término 'Server' para identificar una configuración de servidor. Es una configuración interna de eclipse en la que se especifica el comportamietno del servidor. En esta configuración se incluye el término 'Intalled Runtime Server' o 'Runtime Server' que se refiere a un servidor instalado en el Sistema Operativo y que se identifica por la carpeta en que está ubicado.

A.3 Gestión del código del proyecto en el repositorio

Un repositorio es un almacén de código, que generalmente está hospedado en una máquina distinta a la de desarrollo. El propósito de un repositorio es manejar las diferentes versiones de los archivos que componen un proyecto de manera que varios programadores puedan trabajar conjuntamente en dichos archivos. Existen varios tipos de repositos con distintas filosofías de funcionamiento, el código de este proyecto está hospedado en un repositorio SVN en la dirección: *http://minsky.gsi.dit.upm.es/svn/orangebot.*

Cada uno de los módulos está almacenado como un proyecto aparte dentro del repositorio.

A.3.1. Descarga del código

Ya que SVN es software libre existen multitud de aplicaciones y plugins para entornos de desarrollo que son capaces de acceder a estos repositorios.

Sea cual sea la interfaz que se va a utilizar para gestionar el código es necesario disponer de un usuario y una clave de acceso. Si desea trabajar con este proyecto y no dispone de dicha clave solicite al una clave administrador del repositorio *http://minsky.gsi.dit.upm.es/svn/orangebot.*

Descarga desde el terminal

Para descargar el proyecto desde el terminar es necesario ejecutar el comando:

```
$ svn checkout http://minsky.gsi.dit.upm.es/svn/orangebot
```

Descarga del proyecto desde Eclipse

Para descargar el proyecto desde eclipse se abre la perspectiva 'SVN Repository'. Se añado un nuevo repositorio pulsando el botón 'Añadir repositorio'  , que inicia el asistente para la configuración de un nuevo repositorio. Basta con indicar en la Url del repositorio: *http://minsky.gsi.dit.upm.es/svn/orangebot.*

El repositorio aparece en la ventana de repositorios disponibles, en la que se puede explorar el contenido del respositorio. Tanto para crear el repositorio como para explorarlo es necesario proporcionar usuario y contraseña. Es conveniente activar la opción de recordar usuario y contraseña.

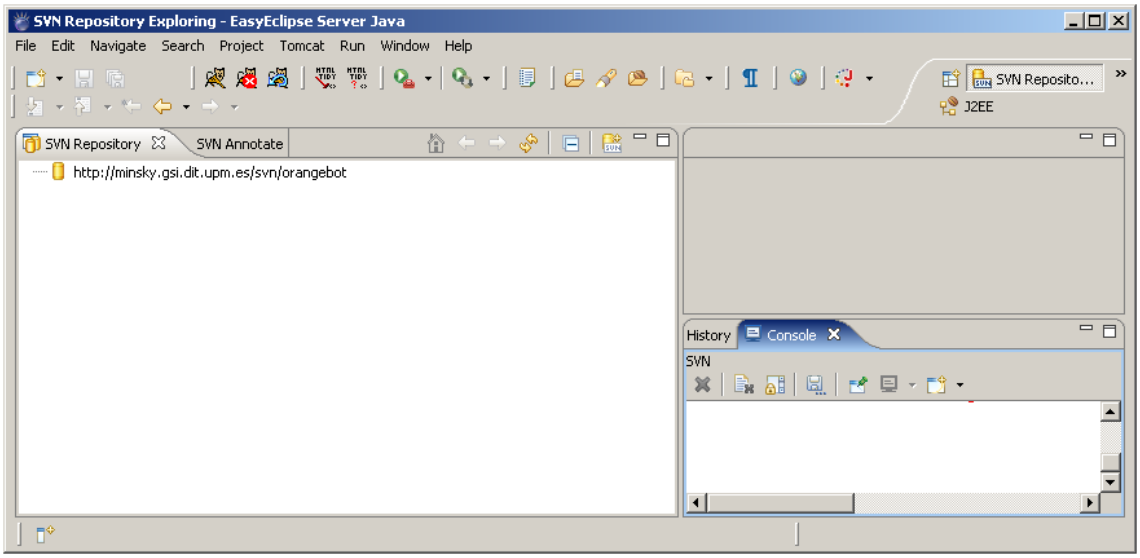

Fig. 68. Perspectiva 'SVN Repository'.

En eclipse es conveniente configurar cada módulo como un proyecto diferente. Para ello se ha de utilizar la ventana de exploración del repositorio desplegando la carpeta raíz del repositorio y la carpeta 'trunk' para visualizar la lista de proyectos.

Se selecciona la carpeta del proyecto que se desea construir. Con el botón derecho en dicha carpeta se despliega el menú desplegable y se selecciona la opción 'Chekout…'.

Se selecciona la opción de crear un nuevo proyecto utilizando el asistente (wizard). En la ventana de selección de asistente se selecciona 'Dynamic Web Project'[30].

[30] Todos los proyectos contenidos en el repositorio deben ser configurados como Dynamic Web Project a excepción de gsiProgramD que será configurado como Java Project.

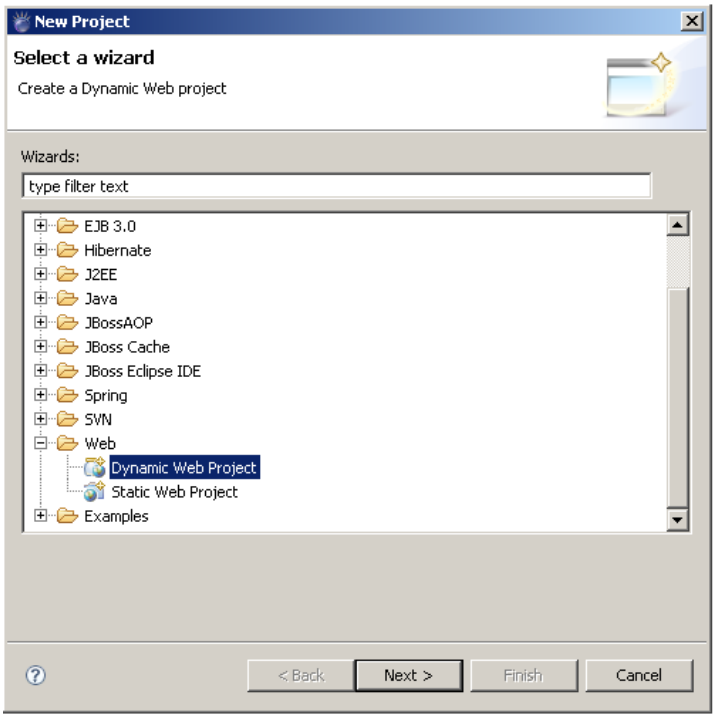

Fig. 69. Ventana de selección de asistente de creación de proyecto.

El nombre dado al proyecto lo identifica en el explorador de proyectos de eclipse y además define URL de acceso en el navegador.

En la opción desplegable 'Target Runtime' se ha de seleccionar la configuración de Eclipse para el servidor Apache Tomcat instalado. En caso de que no se haya configurado Eclipse todavía para trabajar con Tomcat es necesario seguir los pasos descritos en el apartado A.2.

Finalmente Eclipse procede a descargar todos los archivos del repositorio que aparecerán en la ventana de navegación del proyecto y en el explorador de paquetes.

A.3.2. Cómo trabajar con SVN

Existen distintos manuales que recogen buenas prácticas sobre el uso de repositorios SVN [56]. En función de la envergadura del proyecto, el tamaño del equipo de trabajo y la metodología de trabajo la aplicación de algunas de estas prácticas son innecesarias.

Estructura del repositorio

Existe una estructura canónica para repositorios SVN que se sigue en el repositorio del proyecto.

De la raíz del repositorio penden tres directorios: tags, branches y trunk. Los directorios tags y branches contienen copias del código en ciertos estados relevantes mientras que trunk guarda la copia de trabajo principal.

trunk es la rama principal, en la que se guarda la copia del código en que se trabaja habitualmente. En proyectos de una envergadura media y con un equipo de trabajo reducido, salvo casos excepcionales, siempre se trabaja en esta rama.

Cuando se empaqueta el código para un cliente o para una distribución determinada se ha alcanzado un estado relevante que conviene ser almacenado para poder recuperarlo en cualquier momento. Para ello se aconseja en estos casos realizar una copia etiquetada del código en la carpeta **tags**. De esta manera se puede volver a la versión del código que se entregó a un determinado cliente.

La carpeta **branches** sirve para albergar una línea de trabajo paralela a la principal trunk. Generalmente se crea una rama en branches, en la que se copia el código del proyecto, cuando se realizan cambios que pretenden implementar una funcionalidad compleja pero no se está seguro de que se vaya a poder completar la implementación debido a su dificultad, falta de tiempo, etc. De esta manera no es necesario eliminar fragmentos de código pues el resto del equipo ha seguido trabajando en la copia "limpia" del trunk. En caso de que se haya completado la tarea con éxito se puede ejecutar una acción de *merging* por la que los añadidos del código en la línea de trabajo del branch se añaden a la principal.

Política de commit

A la operación de guardar en el repositorio los cambios realizados en el código por un desarrollador se le denomina *commit*.

Cada operación de commit debe estar orientada a un propósito: arreglar un bug, implementar una nueva característica o alguna tarea en particular. De esta manera será más fácil navegar por las distintas versiones del código del repositorio para poder encontrar una determinada versión anterior.

Es conveniente realizar al menos un commit después de cada sesión de trabajo. No es correcto no realizar un commit excusándose en que "el código todavía no funciona correctamente". En caso de que los cambios persigan la implementación de una funcionalidad compleja y de dudoso éxito debe configurarse un branch, en caso contrario debe hacerse commit y si los cambios no son satisfactorios se puede volver a una versión anterior y continuar desde ahí el proceso de desarrollo.

Política de update

Aunque el repositorio persigue evitar los problemas de trabajo en equipo simultaneo, es posible que si varios miembros del equipo están trabajando en el mismo fichero, al subir los cambios al repositorio el SNV no sepa cómo llegar a mezclarlos. En eso momento se dice que se ha creado un conflicto. Habitualmente estas situaciones no son difíciles de resolver.

Para prevenir la aparición de estos conflictos es necesario realizar operaciones de *update* frecuentemente. Esta operación incluye actualiza la copia local con las últimas actualizaciones subidas al repositorio por otros desarrolladores. Si no se realiza esta operación frecuentemente puede ocurrir que al intentar subir un archivo muchos otros desarrolladores hayan trabajado en él produciéndose un conflicto difícil de solucionar. Incluso puede ocurrir la situación de que los mismos cambios ya hayan sido realizados con anterioridad por otro desarrollados.

¿Qué archivos guardar en el repositorio?

La configuración del espacio de trabajo de Eclipse incluye multitud de archivos, a menudo ocultos en el Sistema Operativo Windows pero que no lo son para las herramientas del SVN. Es necesario prestar atención a todos estos archivos de configuración puesto que su información es válida para un equipo en particular. Bajo ningún concepto deben de ser almacenados en al repositorio puesto que pueden corromper las copias de trabajo de los demás desarrolladores.

En general, en la versión 3.3 de eclipse dichos archivos son la carpeta `.settings` y los archivos .classpath y .project.

Tampoco es recomendable subir al repositorio los archivos compilados puesto que, aunque esto no ocurre en Java, en general el proceso de compilación se realiza para una arquitectura y sistema operativo determinado por lo que también puede producir errores. Independientemente de esto, no es una buena práctica en ningún caso puesto que los archivos compilados están en un lenguaje no asci que el SVN no es capaz de cotejar de manera que cada nueva compilación implica un trafico innecesario al SVN además de una fuente elevada de conflictos.

El mismo razonamiento se puede aplicar a las bibliotecas de código. Sin embargo, en el caso de Java constituyen una excepción y sí es recomendable incluirlas en el repositorio. Las bibliotecas no cambian frecuentemente por lo que no hay conflictos ni tampoco altas tasas de transferencia. Puesto que java es independiente de la arquitectura y S.O. todos los miembros del equipo de trabajo pueden –y deben- utilizar la misma

librería. Además, es frecuente que en internet sea complicado encontrar algunas versiones de determinadas bibliotecas pudiendo ocurrir que distintos miembros del equipo de desarrollo trabajen con diferentes bibliotecas, lo que constituye un error en tiempo de ejecución altamente difícil de detectar.

A.4 Trabajando con ANT

Ant es una herramienta de código libre desarollada por Apache SW Foundation para la construcción y despliegue de aplicaciones generalmete escritas en Java. Ant surge como una herramienta dentro del proeycto Tomcat para compilar y desplegar dicha aplicación. Sin embargo, el hecho de aplicar una licencia de código libre a Ant hizo que este projecto se extendiese incluso más que el propio Tomcat. La filosofía de Ant se asemeja a la de makefile usados en C y C++. Ant ofrece gran cantidad de acciones a realizar, desde las básicas relacionadas con crear carpetas y copiar archivos en el sistema de ficheros hasta ejecutar clases java o empaquetar proyectos en ficheros zip, jar, war... Además, Ant ofrece muchas facilidades para que el usuario añada funcionaliades a Ant en función de sus necesidades.

Uno de los principales objetivos por los que surgió Ant es par resolver el problema de la portabilidad. Con makefile, las instrucciones contenidas en estos ficheros dependían de la shell utilizada, de manera que era necesario desarrollar un makefile diferente para Unix, para mac y para Windows. Por supuesto esto no es eficiente. Tal es el triunfo de Ant al resolver el problema de la portabilidad que puede suplir a muchos scripts de arranque de aplicaciones comerciales que ordinariamente se esdoblaban en un script diferente para cada sistema operativo incluidos todos en el bin de la aplicación.

Con todo esto Ant se ha extendido tanto que la mayoría de los desarrolladores -profesionales y amater de grado avanzado o experto- incluyen un fichero build.xml de Ant con la distribución de su proeycto.

A.4.1. El archivo build.xml

El archivo build.xml es el archivo que Ant utiliza para describir todas los recursos utilizados y las acciones relacionadas con un proyecto. "build.xml" es el nombre por defecto pero este fichero puede renombrarse. El fichero build.xml está constituido por un conjunto de definiciones (`properties`) y un conjunto de objetivos (`targets`).

El elemento XML *project* es el elemento raíz que envuelve a todo el documento. Tiene los atributos obligatorios *default* y *basedir*. Default debe contener el nombre de uno de los targets definidos dentro del proyecto o quedar vacío. Ese target será el que se ejecute por defecto cuando se ejecute el archivo build.xml sin especificar un target determinado. Basedir es la dirección relativa a la posición del archivo build.xml que marca el directorio que se toma como referencia.

A.4.1.1. Properties

Las definiciones o properties son declaraciones de variables utilizadas dentro del build.xml. Estas properties se utilizan muy habitualmente para almacenar en variables el path a un directorio, el nombre de un archivo, el nombre del proyecto o la dirección de contacto del autor. El propósito de estas properties es evita tener que re escribir por todo el build.xml el valor de estos campos y se facilita la refactorización del proyecto, pues cambiar un directorio de nombre en el build.xml sólo supone cambiar el valor de un propertie. Así, las properties también se utilizan para mantener el valor de flags o variables utilizados a lo largo del script, como puede la versión del compilador de java, si se utilizara la opción verbose o no, o el año de publicación del proyecto...

Los elementos property poseen dos atributos: *name*, con el nombre de la variable y *value*, con el valor asociado a la misma. Ninguno de los dos puede estar vacío. La sintaxis de las definiciones es:

```
<property name="Name" value="Proyecto de prueba"/>
<property name="year" value="2010"/>
```

Para acceder a una definición utilizamos la sintaxis *${propertiName}*. De manera que se pueden utilizar properties definidas anteriormente en las nuevas definiciones:

```
<property name="distr.dir" value="${basedir}/distrib"/>
<property name="distr.zip" value="${distr.dir}/proyecto.zip"/>
```

No se puede utilizar una property que no se haya definido previamente en el build.xml aunque se defina más adelante.

Un caso especial es el de la definición de la variable de entorno.

```
<property environment="env"/>
```

Esta propiedad, a la que se puede acceder con el nombre "env" guarda todas las variables de entorno definidas en el sistema operativo. Es una propiedad especialmente útil para acceder a información del sistema.

A.4.1.2. Targets

Los *targets* son el equivalente a las definiciones de funciones. Sirven para realizar una tarea determinada. Tareas que habitualmente se realizan mediante targets son compilar el proyecto, generar el javadoc, crear una carpeta, generar un fichero jar… Los targets pueden ser invocados en la ejecución del archivo build.xml, utilizando como argumento el nombre del target, o mediante una relación de dependencia de otro target. En este caso el comportamiento es casi como el de herencia. Cuando un target B depende de un target A, la ejecución del target B desencadena la previa ejecución del target A. Si un target depende de varios targets que a su vez dependen unos de otros, se ejecuta en primer lugar aquel del que dependa alguno de ellos y que no dependa de ninguno[31]. Posteriormente se resuelve el árbol de dependencias hasta ejecutar todos los targets implicados. Aunque varios targets dependan de uno mismo, ningún target se ejecuta más de una vez.

Los elementos target poseen tres atributos: *name*, con el nombre del target, *depends* con una lista de los targets de lo que depende la ejecución del target que se define y *description*, con una descripción textual de la tarea desempeñada por el target. El atributo name no puede estar vacio y coincidir con el valor de otro atributo name de otro target. La sintaxis de los target es:

```
<target name="Name" depends="cleanup" description="some text"/>
<target name="cleanup" depends="" description="delete temp-dir"/>
```

A.4.1.3. Otros elementos

Al ser Ant un proyecto tan utilizado existen multitud de targets definidos para realizar gran cantidad de acciones. En la siguiente lista se recogen algunas de ellas:

- echo: imprime por consola un mensaje dado.
- mkdir: crea el directorio dado.
- javac: compila un conjunto de fuentes dado.
- javadoc: genera el javadoc asociado a un conjunto de fuentes dado.

[31] En caso de que esta situación no se produzca y haya una relación de dependencias cíclica la ejecución fallará.

- junit: ejecuta las pruebas definidas en un archivo *TestCase* o *TestSuite* dado.
- war: crea un Web Application Archive.

Además, la sintaxis que aceptan estos targets es muy flexible, por lo que pueden ser utilizados de diversas formas.

A.4.2. Organización del proyecto con Ant

El proyecto gsiBot está dividido en cuatro módulos: gsiProgramD, gsiBotServer, CorpusTester, gestionConocimiento y gestionDeRespuestas. Los cuatro últimos además constituyen proyectos Web. Cada uno de estos módulos es una carpeta dentro de la rama principal *trunk* del repositorio SVN.

Además cada uno de estos módulos tiene asociado un archivo build.xml, localizado en la raíz del proyecto, en el que se definen targets para compilar el proyecto, empaquetar las clases en un jar, exportar el proyecto a un archivo war, desplegar el proyecto en el Tomcat, generar el javadoc… La definición de estos targets facilita en gran medida la realización de estas labores, repetitivas, que en el caso de realizarlas a mano, supondría la realización de múltiples acciones, con el riesgo de equivocación que conlleva. Aunque los entornos de desarrollo integrados (IDEs) permiten realizar algunas de estas acciones con sólo presionar un botón, la personalización que permite ant, así como la garantía de buen funcionamiento, son bazas suficientes para desbancar a esta alternativa. Además, ant también está integrado en la interfaz de la mayoría de los IDEs por lo que la ejecución de un target determinado también está accesible desde la propia interfaz.

El archivo build.xml asociado a cada proyecto, presenta ciertas diferencias con respecto al resto de archivos[32] para adaptarlo a las particularidades de cada proyecto. Los targets que se emplean se recogen a continuación.

A.4.2.1. Usage

Es el target por defecto en todos los archivos build.xml que se tratan en este proyecto. En el se imprimen una serie de líneas de caracteres en las que se describe cómo se utiliza el archivo build.cml tratado.

```
<target name="usage" depends="" description="Usage description">
```

[32] Además de las diferencias obvias en el valor de las properties que definen el proyecto.

```
  <echo message=""/>
  <echo message="${name} build file"/>
  <echo message="-----------------------------------"/>
  <echo message=""/>
  <echo message="Available targets are:"/>
  <echo message="help--> Show some help topics"/>
  <echo message="compile--> Build the application"/>
  <echo message="compile-test--> Build the application and tests" />
  <echo message="distrib--> This prepares a zip file with the
release of the application"/>
  <echo message="deploy--> Deploy application"/>
  <echo message="deploy-war--> Deploy application as a WAR file"/>
  <echo message="run--> Run the junit testsuite"/>
  <echo message="jar--> Create the distribution jar file"/>
  <echo message="jar-debug--> Creates a jar for debuggin purposes"/>
  <echo message="javadoc--> Generate the javadoc"/>
  <echo message="war--> Create the Web Application Descriptor
(war)"/>
  <echo message=""/>
</target>
```

Fig. 70. Código XML del target “usage”

A.4.2.2. Targets de preparación

Los targets mostrados en la Fig. 71 crean los directorios que utilizarán otros targets. Es necesario que estos directorios sean creados antes de que tratar de crear archives en ellos pues si no existiesen se abortaría la ejecución.

```
<target name="prepare" depends="" description="prepare workspace">
  <mkdir dir="${build.dir}"/>
</target>

<target name="prepare-distrib" depends="">
  <mkdir dir="${distrib.dir}"/>
</target>

<target name="prepare-reports" depends="">
  <mkdir dir="${report.test.dir}"/>
</target>
```

Fig. 71. Código XML de los target “prepare”, “prepare-distrib” y “prepare-reports”.

A.4.2.3. Targets de limpieza

Los targets mostrados en la Fig. 72 son los encargados de eliminar los archivos contenidos en los directorios auxiliares. La importancia de estos targets radica en que, cuando se va a ejecutar ciertos elementos, se comprueba el estado del directorio de salida de dichos elementos, si los archivos que se van a generar ya se encuentran en dicha ubicación, se omite su ejecución. Esto es así para ganar eficiencia, sin embargo, en algunos casos es posible que se hayan realizado cambios que no estén reflejados en directorio de salida. Para ello se elimina el contenido del directorio de manera que se garantiza que el contenido no esta obsoleto.

```
<target name="clean" depends="clean-javadoc, clean-build"
  description="--> clean it all up" >
  <delete dir="${distrib.dir}"/>
</target>

<target name="clean-build" description="--> clean the build dir" >
  <delete dir="${build.dir}"/>
</target>

<target name="clean-javadoc" description="-->clean the javadocdir" >
  <delete dir="${doc.dir}"/>
</target>
```

Fig. 72. Código XML de los target “clean”, “clean-build” y “clean-javadoc”.

A.4.2.4. Targets de compilación

Los targets de compilación, por medio del elemento *javac* generan los ficheros class a partir de el código java contenido en los fuentes y en las bibliotecas utilizadas.

Tal y como se muestra en la Fig. 73, también se realizan otras acciones necesarias para el correcto funcionamiento del código, como es copiar los archivos *log4j* junto con los ficheros compilados.

```
<!--Compile the source directory.-->
<target name="compile" depends="prepare"
  description="--> compile the source files">

  <javac srcdir="${src.dir}"
         destdir="${build.dir}"
         classpathref="build.classpath"
         includes="**/*.java"
```

```
          target="1.5">
      <compilerarg value="-Xlint"/>
      <compilerarg value="-Xlint:-path"/>
      <compilerarg value="-Xlint:-serial"/>
    </javac>

    <!-- Copy log4j conf files from src dir to build dir -->
    <copy todir="${build.dir}">
      <fileset dir="${src.dir}" includes="log4j.*"/>
    </copy>
  <target>
```

Fig. 73. Código XML del target "compile"

En el código mostrado en la Fig. 73 se compilan los archivos contenidos en *${src.dir}*. Si se pretende realizar la compilación de un conjunto de archivos diferente es necesario cambiar este parámetro, como es el caso del target compile-test mostrado en la .

```
<!--Compile the source directory and the testcase files.-->
<target name="compile-test" depends="compile"
  description="--> compile the source files and test files">

  <javac srcdir="${src.test.dir}"
         destdir="${build.test.dir}"
         classpathref="build.classpath"
         includes="**/*.java"
         target="1.5">
    <compilerarg value="-Xlint"/>
    <compilerarg value="-Xlint:-path"/>
    <compilerarg value="-Xlint:-serial"/>
  </javac>

</target>
```

Fig. 74. Código XML del target "compile-test"

A.4.2.5. Targets de distribución

Estos targets generan archivos jar para la distribución del código, que contienen las clases compiladas y también los fuentes en caso de que así se especifique. Para ello se utiliza el elemento *jar*.

```
<!-- target: jar -->
<target name="jar" depends="prepare-distrib, compile"
  description="--> This creates the main jar of the project.
                  It only includes the compiled files">

  <jar destfile="${distrib.jar}">
    <fileset dir="${build.dir}">
      <include name="**/*.class"/>
    </fileset>
    <manifest>
      <!-- Information about the program itself -->
      <attribute name="Implementation-Vendor" value="GSI"/>
      <attribute name="Implementation-Title" value="${Name}"/>
      <attribute name="Implementation-Version" value="2010"/>
    </manifest>
  </jar>
</target>
```

Fig. 75. Código XML del target "jar"

Es un caso especialmente interesante el que genera un archivo jar con funcionalidades de debugging. Este caso es útil para llevar a cabo un proceso de depuración entrando en las clases que pertenecen a una biblioteca que está incluida en el proyecto.

Para habilitar las funcionalidades de depuración de un jar es necesario compilar los archivos incluyendo información sobre variables y número de línea, así como incluir el código fuente en el fichero comprimido.

```
<!-- target: debbugable jar -->
<target name="jar-debug"
        depends="clean-build, prepare, prepare-distrib"
        description="--> This creates a jar containing
                     compiled files for debuggin purpose.">

<!-- Delete all compiled files, in order to create new ones
that allow debugging -->
  <javac srcdir="${src.dir}"
         destdir="${build.dir}"
         classpathref="build.classpath"
         includes="**/*.java"
         target="1.5">
```

```
    <compilerarg value="-Xlint"/>
    <compilerarg value="-Xlint:-path"/>
    <compilerarg value="-Xlint:-serial"/>
    <compilerarg value="-g:source,lines,vars"/>
  </javac>

<!-- Create the jar with the new files -->
  <jar destfile="${distrib.debug.jar}">
    <fileset dir="${build.dir}">
      <include name="**/*.class"/>
    </fileset>
    <manifest>
      <!-- Information about the program itself -->
     <attribute name="Implementation-Vendor" value="${Author}"/>
     <attribute name="Implementation-Title" value="${Name}"/>
     <attribute name="Implementation-Version" value="${year}"/>
    </manifest>
  </jar>

<!-- Delete the debug oriented object files. These files must
not be kept in disk -->
  <delete dir="${build.dir}"/>

</target>
```

Fig. 76. Código XML del target “jar-debug”

A.4.2.6. Targets de distribución Web

Por medio del elemento *war*, se empaquetan los archivos propios de la aplicación Web junto con las clases compiladas y las bibliotecas de código con el formato establecido es las especificaciones. El archivo resultante puede ser desplegado en Tomcat.

```
<!--Create a WAR file.-->
<target name="war" depends="compile, prepare-distrib">

  <war destfile="${war}"
       webxml="${webapp.dir}\WEB-INF\web.xml"
       manifest="${webapp.dir}/META-INF/MANIFEST.MF"
       update="true" >
```

```
    <fileset dir="${webapp.dir}">
      <include name="aiml\**"/>
      <include name="conf\**"/>
      <include name="css\**"/>
      <include name="flash\**"/>
      <include name="images\**"/>
      <include name="js\**"/>
      <include name="KTFiles\**"/>
      <include name="pages\**"/>
      <include name="resources\**"/>
      <include name="META-INF\**"/>
    </fileset>

    <lib dir="${webapp.lib.dir}"/>
    <lib dir="${additional.lib.dir}"/>
    <classes dir="${build.dir}"/>
    <webinf dir="${webapp.dir}\WEB-INF">
      <include name="tests\**" />
      <include name="*.tld"/>
    </webinf>
  </war>
</target>
```

Fig. 77. Código XML del target "war"

A.4.2.7. Targets de despliegue

De acuerdo con el funcionamiento de Apache Tomcat, para desplegar una aplicación Web, basta con copiar el archivo war que la contenga en la carpeta webapp que se encuentra en el directorio de instalación de Tomcat. Éste, copia el contenido de la aplicación –debidamente organizado- a una nueva carpeta con el nombre de la aplicación Web contenida en el archivo war. Esto es lo que hace el target "war".

Al trabajar el servidor con código compilado (las clases .class incluidos los Servlets), código interpretado (los jsps) y archivos web, cualquier cambio realizado en archivos de estos dos últimos conjuntos se verá reflejado en la aplicación Web sin necesidad de reiniciar el servidor. No es así en caso de que los cambios se produzcan en archivos compilados. El target deploy, copia el contenido de los archivos jsp y web de la aplicación Web a la carpeta que la contiene de manera que los cambios hechos en

ellos se verán reflejados. Esto es muy útil para poder trabajar observando el efecto que tienen pequeños cambios en el código.

```
<target name="deploy-war" depends="war"
    description="--> Deploy the WAR by copying it to Tomcat">
  <echo message="Copying the WAR to CATALINA_HOME"/>
  <copy file="${war}"
        todir="${env.CATALINA_HOME}/${tomcat.webapps.dir}"
        overwrite="true"
        verbose="true"/>
</target>

<target name="deploy" depends="compile"
        description="Deploy application">
  <copy
    todir="${env.CATALINA_HOME}/${tomcat.webapps.dir}/${name}"
    preservelastmodified="true"
    verbose="true">

    <fileset dir="${webapp.dir}">
      <include name="**/*.*"/>
    </fileset>
  </copy>
</target>

<target name="delete-webapp" depends=""
        description="This deletes the webapp folder in Tomcat
                     webapps folder" >
  <delete
    dir="${env.CATALINA_HOME}/${tomcat.webapps.dir}/${name}"
    verbose="true" />
</target>
```

Fig. 78. Código XML de los targets “deploy-war”, “deploy” y “delete-webapp”.

En ambos casos[33], se asume que la dirección de instalación del contenedor de Servlets Apache Tomcat es conocida. Esta información es proporcionada a Ant por medio de las variables de entorno del sistema, almacenadas en una propiedad de entorno. Mediante la instrucción *${env.CATALINA_HOME}*se accede a la dirección de instalación de Apache Tomcat. Para que esta instrucción pueda acceder a la propiedad de entorno como se describe en A.4.1.1.

[33] Y también en el del target delete-webapp.

Anexo B.

Definición del corpus del bot 3DTour

B.1 Introducción

En este documento se pretende definir un conjunto de temas de conversación que conformarán el conocimiento de un bot interactivo cuya función será ofrecer ayuda e información al usuarios sobre los servicios ofrecidos por el entorno 3DTour.

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.

B.2 Temas de conversación

En 3DTour se diseña un entorno de demostración inspirado en un complejo hotelero, por lo que el corpus del bot debe sujetarse al perfil del personal que trabaje en la hostelería.

En este documento se definen los temas de conversación que se incluirán en la base de conocimiento global.

La base de conocimiento puede ser fragmentado en varios subconjuntos que respondan al perfil específico de cada uno de los empleados del hotel. De manera que el personal de cafetería pueda ofrecer información distinta de la que ofrece el personal de

recepción, el aparcacoches, personal de habitaciones, etc. A falta de tomar una decisión al respecto de llevar a cabo esta división, se definen las entradas agrupadas por temas de manera que pueda definirse a posteriori a qué conocimiento tiene acceso cada bot.

B.2.1. Preguntas sobre servicios generales e instalaciones

ID	Usuario	Bot
Normas generales		
I.0.1.0	• Animales • Mascota • ¿Puedo tener en el hotel animales de compañía?	Lo siento, en el hotel no se admiten animales.
I.0.2.0	• Fumar • ¿Está permitido fumar en el hotel?	No, no se puede fumar en ninguna dependencia del hotel.
I.0.2.1	• Fumar habitaciones • ¿Se puede fumar en las habitaciones?	No, no se puede fumar en las habitaciones.
I.0.2.2	• Fumar cafetería • Fumar restaurante • ¿Esta permitido fumar en la cafetería?	No, no esta permitido fumar en la cafetería ni en el restaurante.
I.0.3.0	• Propina • ¿Hay que dejar propina? • ¿Es obligatorio dar propina?	No es necesario dar propina, lo dejamos a su elección.
I.0.3.1	• Propina precio • ¿Esta la propina incluida en el precio?	La propina no está incluida en el precio.
I.0.3.2	• Desperfectos • Romper habitación • ¿Tengo que pagar si ocasiono algún desperfecto? • ¿Qué pasa si rompo algo?	Los desperfectos que ocasione durante su estancia serán cargados a su cuenta
Servicios Generales		
I.1.0.0	• Categoría • ¿Cuál es la categoría del hotel?	Aún no se ha determinado cómo se asignan las categorías a los hoteles virtuales.
I.1.1.1	• Horario de recepción • ¿Cuál es el horario de recepción?	Las 24 horas del día. Siempre estaremos aquí para lo que necesite.

I.1.1.2	• ¿A qué hora se abandona la habitación? • ¿Cuándo se debe dejar la habitación? • A qué hora tenemos que salir de la habitación	Deben abandonar la habitación a las 12 horas para que el servicio pueda preparar la habitación para los próximos clientes.
I.1.2.0	• Restaurante • Servicio de Restaurante • ¿Tenéis servicio de restaurante?	Por supuesto, disponemos de un fabuloso servicio de restaurante.
I.1.2.1	• Horario de restaurante • Cuál es el horario del restaurante	El restaurante permanece abierto de 12 a 16 y de 20 a 00.
I.1.2.3	• Donde restaurante • ¿Dónde está el restaurante? • ¿Cómo puedo llegar al restaurante? • ¿Sabes donde está el restaurante?	El restaurante está junto a la cafetería.
I.1.3.0	• Cafetería • ¿Tenéis cafetería?	Por supuesto, disponemos de un estupendo servicio de cafetería.
I.1.3.1	• Horario de cafetería • ¿Cuál es el horario de cafetería?	La cafetería permanece abierta de 7:30 a 23:30 horas ininterrumpida mente.
I.1.3.2	• Precio Cafetería • ¿Tiene la tabla de precios de la cafetería? • ¿Tiene la guía de precios de la cafetería? • ¿Tiene la lista de precios de la cafetería?	Los precios de la cafetería los puede consultar en
I.1.3.3	• Donde cafetería • ¿Dónde está la cafetería? • ¿Cómo puedo llegar a la cafetería? • ¿Sabes donde está la cafetería?	La cafetería está en el otro extremo del vestíbulo.
I.1.4.0	• Aparcacoches • Servicio de aparcacoches	Lo siento, pero no disponemos de servicio de aparcacoches.
I.1.5.0	• Teléfono público • Teléfono en el vestíbulo	Tenemos una cabina de teléfonos en el vestíbulo.
I.1.5.1	• Horario Teléfono público • ¿Cuál es el horario del teléfono público?	Puede usar el teléfono público cuando desee
I.1.5.2	• Precio Teléfono publico • Precio Teléfono vestíbulo • ¿Cuánto cuesta hacer una llamada desde el teléfono del vestíbulo?	Junto al teléfono dispone de una lista de tarifas para las llamadas
I.1.6.0	• Internet • Servicio de internet • ¿Tiene el hotel internet? • ¿Dispone el hotel de conexión a internet?	Si, existen unos puestos donde puede conectarse a internet.

I.1.6.1	• Horario Internet • ¿Qué horario tienen los puestos de internet? • ¿A qué horas puedo usar la conexión a internet?	Los puestos permanecen conectados de 8 de la mañana a 10 de la noche.
I.1.6.2	• Precio Internet • Tarifa Internet • ¿Cuánto cuesta el uso de internet?	El uso de los puestos de internet es gratuito. Para eso es un hotel virtual, ¿no?
I.1.6.3	• Donde Internet • ¿Dónde puedo conectarme a internet? • ¿Cómo puedo consultar mi correo electrónico?	Puede conectarse a internet en los puestos que se encuentran por todo el hotel.
I.1.7.0	• Lavandería • Tenéis servicio de lavandería • ¿Ofrece el hotel servicio de lavandería?	No, el hotel no tiene servicio de lavandería para clientes.
Información sobre Instalaciones		
I2.0.2	• Donde ascensor • ¿Donde están los ascensores?	Encontrará los ascensores ahí enfrente
I.2.1.0	• Parking • Aparcamiento • Plaza de aparcamiento • Plaza de parking	En el exterior del edificio podrá estacionar su vehículo. El uso de la plaza de aparcamiento tiene un recargo el precio de la habitación.
I.2.1.1	• Horario de parking • Horario de aparcamiento • Horario de garaje • ¿A qué hora puedo estacionar mi coche? • ¿A qué hora puedo estacionar mi vehículo? • ¿A qué hora puedo dejar el parking? • ¿A qué puedo retirar mi coche del parking?	El aparcamiento permanece cerrado de 00h a 7h. Pero puedo comunicarse con recepción a través del intercomunicador para comunicarse con recepción.
I.2.1.2	• Precio Parking • Precio servicio Aparcamiento • Precio garaje	El servicio de aparcamiento es gratuito para clientes.
I.2.1.3	• Donde Aparcamiento • ¿Dónde esta el parking? • ¿Dónde se encuentra el garaje?	Ha pasado a su lado justo antes de entrar
I.2.2.0	• Gimnasio • ¿Tienen gimnasio? • ¿Hay gimnasio en el hotel? • Sala de musculación	Sí, tenemos un gimnasio en la última planta. Sin embargo ahora lo estamos remodelando y está cerrado, lo siento.

I.2.2.1	• Horario de Gimnasio • ¿Cuál es el horario de apertura del gimnasio? • ¿Qué horario tiene el gimnasio?	Me temo que actualmente el gimnasio está cerrado pues estamos llevando a cabo una reforma. Lo siento.
I.2.2.2	• Precio de Gimnasio • ¿Cuánto cuesta el gimnasio? • ¿Cuál es el precio del gimnasio? • ¿Cuánto cuesta una sesión de gimnasio?	El gimnasio es gratuito y de uso exclusivo para clientes.
I.2.2.3	• Donde gimnasio • ¿Dónde está el gimnasio? • ¿Cómo puedo llegar al gimnasio?	El gimnasio se encuentra en la última planta. Sin embargo, me temo que actualmente el gimnasio está cerrado pues estamos llevando a cabo una reforma. Lo siento.
I.2.3.0	• Sauna • ¿Tenéis sauna? • ¿Hay sauna en el hotel?	No, no tenemos Sauna lo siento.
I.2.3.1	• Horario de Sauna • ¿Cual es el horario de la sauna?	No tenemos sauna, lo siento.
I.2.3.2	• Precio de Sauna • ¿Cuánto cuesta la sauna?	No tenemos sauna.
I.2.4.0	• SPA • ¿Tienen SPA? • ¿Hay SPA en el hotel?	No tenemos SPA.
I.2.5.0	• Peluquería • ¿Tienen servicio de peluquería?	No tenemos servicio de peluquería.
I.2.6.0	• Piscina • ¿Hay piscina en el hotel? • ¿El hotel tiene piscina?	Lo siento, el hotel no tiene piscina.
I.2.7.0	• Parque • Jardín • Zona verde	El recinto del hotel es reducido, pero hay parques públicos muy cerca de aquí.

B.2.2. Preguntas sobre tarifas y reservas

ID	Usuario	Bot
Tarifas		
P.1.1.0	• Tarifa • Tarifa persona • Tarifa Noche • ¿Las tarifas son por persona o por noche?	Las tarifas son precios por habitación por noche. El número de personas ya está reflejado en el precio de las habitaciones bien sean individuales o dobles.
P.1.1.1	• Tarifa IVA • ¿Las tarifas incluyen IVA? • ¿Está el IVA incluido en la tarifa?	El precio de las tarifas ya incluye el IVA.
P.1.1.2	• Pagar adelantado • ¿Es necesario pagar por adelantado?	No, no es necesario pagar por adelantado
P.1.1.3	• Cuando factura • Cuando pagar • Cuando se paga la cuenta • Cuando tengo que pagar la factura	La factura se abona en el momento de abandonar el hotel.
P.1.2.0	• Moneda Pagar • ¿En qué moneda se puede pagar?	Sólo aceptamos pagos en Euros.
P.1.2.1	• Pagar Euro • ¿Se puede pagar en Euros?	Sí, se puede pagar en Euros.
P.1.2.2	• Pagar dólar • Pagar Libra • Pagar Yen • ¿Se puede pagar en dólares?	No, sólo aceptamos pagos en Euros
P.1.3.0	• ¿Los niños son gratis? • ¿Los niños tienen que pagar?	Los niños hasta 3 años no pagan si duermen en la habitación de sus padres.
P.1.4.0	• Pagar tarjeta crédito • ¿Aceptan tarjeta de crédito? • ¿Puedo pagar con tarjeta de crédito?	Claro, puede pagar con tarjeta de crédito o en efectivo.
P.1.5.0	• Pagar efectivo • ¿Aceptan efectivo? • ¿Puedo paga en efectivo?	Claro, puede pagar con tarjeta de crédito o en efectivo.
P.1.6.0	• ¿La tarifa de habitación incluye el desayuno? • ¿Está el desayuno incluido en la tarifa de habitación?	Si, el servicio de bufé del desayuno está incluido en la tarifa de la habitación.
P.1.6.1	• ¿Cuál es el horario del desayuno?	El horario del desayuno es de 7 a 11 horas.

P.1.5.2	• ¿Dónde es el desayuno? • ¿Dónde se desayuna? • ¿Dónde se sirve el desayuno?	El desayuno se sirve en el restaurante.
P.1.4.0	• ¿Hacen factura? • ¿Pueden hacerme factura?	Por supuesto, sólo necesitamos su NIF

B.2.3. Preguntas sobre el equipamiento de las habitaciones

ID	Usuario	Bot
Servicio de habitaciones		
H.1.1.0	• Habitación individual	Disponemos de habitaciones individuales ¿Qué quiere saber?
H.1.1.1	• Precio habitación individual • ¿Cuál es el precio de una habitación individual? • ¿Cuánto cuesta una habitación individual por noche?	Espere un momento, tengo que consultar el precio de la habitación individual.
H.1.1.2	• ¿Cuantas habitaciones individuales hay en el hotel? • Número de habitaciones individuales en el hotel.	Este hotel es muy grande, hay muchas habitaciones. Pero acabo de incorporarme a trabajar aquí y no sé exactamente cuantas son.
H.1.1.3	• Habitación individual con baño • ¿Tienen baño las habitaciones individuales?	Todas las habitaciones tienen cuarto de baño con bañera, ducha o ambas.
H.1.2.0	• Habitación doble	Disponemos de habitaciones dobles ¿Qué quiere saber?
H.1.2.1	• ¿Cuál es el precio de una habitación doble? • ¿Cuánto cuesta una habitación doble por noche?	Espere un momento, tengo que consultar el precio de la habitación doble.
H.1.2.2	• ¿Cuantas habitaciones individuales hay en el hotel? • Número de habitaciones individuales en el hotel.	Este hotel es muy grande, hay muchas habitaciones. Pero acabo de incorporarme a trabajar aquí y no sé exactamente cuantas son.
H.1.2.3	• Habitación doble con baño • ¿Tienen baño las habitaciones doble?	Todas las habitaciones tienen cuarto de baño con bañera, ducha o ambas.

H.1.3.0	• Cama supletoria • ¿Pueden poner una cama supletoria en mi habitación?	Podemos poner una cama supletoria en su habitación si así lo desea
H.1.3.1	• Precio cama supletoria • ¿Cuanto cuesta poner una cama supletoria? • ¿Cuál es el precio de la cama supletoria?	A ver… que consulte el precio de la cama supletoria…
H.1.3.1	• Cuna • ¿Pueden poner una cuna en mi habitación? • ¿Hay cuna en las habitaciones?	Podemos poner una cuna en su habitación si así lo desea.
H.1.3.2	• Precio cuna • Cuesta cuna • ¿Cuánto cuesta poner una cuna en la habitación? • ¿Cuál es el precio del servicio de cuna?	El servicio es gratis
H.1.4.0	• Cama de matrimonio • ¿Las habitaciones dobles tienen cama de matrimonio?	Las habitaciones dobles pueden estar dispuestas con una cama de matrimonio o con dos camas individuales.
H.1.5.0	• Teléfono habitación • Llamadas desde la habitación • ¿Puedo realizar llamadas desde la habitación? • ¿Hay teléfono en la habitación?	Con el teléfono de la habitación puede realizar llamadas tanto al exterior del edificio como a otras habitaciones
H.1.5.1	• Precio del teléfono • Precio de llamada • ¿Cuánto cuestan las llamadas desde la habitación?	Las tarifas de las llamadas están indicadas junto al teléfono.
H.1.5.2	• Llamadas entre habitaciones • ¿Son gratis las llamadas entre habitaciones?	Las llamadas entre habitaciones son gratuitas
H.1.5.3	• ¿Cómo se llamadas entre habitaciones • ¿Cómo puedo llamar a otra habitación?	Para llamar a otra habitación simplemente marque el número de habitación y pulse #
H.1.6.0	• Minibar • ¿Hay minibar en las habitaciones?	Todas las habitaciones tienen minibar
H.1.6.1	• Precio minibar • ¿Cuánto cuestan las bebidas del minibar?	El precio de las bebidas del minibar está indicado en la carta que acompaña al mismo
H.1.6.2	• Contenido Minibar • ¿Qué tiene el minibar? • ¿Qué hay en el minibar?	El minibar contiene una selección de bebidas frías
H.1.7.0	• Caja fuerte	Hay una pequeña caja fuerte en cada habitación

H.1.7.1	• Como Caja fuerte • Funcionamiento caja fuerte • ¿Cómo funciona la caja fuerte de la habitación? • ¿Cuál es la contraseña de la caja fuerte de la habitación?	Las instrucciones de la caja fuerte se encuentran en la habitación.
H.1.8.0	• Hilo musical • Radio • ¿Hay música en las habitaciones? • ¿Dispone el hotel de servicio de hilo musical?	En todas las habitaciones podrá elegir el hilo musical que prefiera.
H.1.9.0	• Televisión • TV • ¿Hay televisión en las habitaciones?	Todas las habitaciones cuentan con una magnifica televisión con TDT
H.1.9.1	• Precio Televisión • ¿Cuánto cuesta encender la televisión? • ¿Es gratis ver la televisión en la habitación?	La televisión es gratuita para todo los cliente ¡faltaría mas!.
H.1.10.0	• Calefacción • ¿Tenéis calefacción? • ¿Hay calefacción en las habitaciones?	Sí, todas las habitaciones tienen sistemas de climatización que el cliente podrá regular a su gusto.
H.1.11.0	• Aire acondicionado • ¿Tenéis aire acondicionado? • ¿Hay aire acondicionado en las habitaciones?	Sí, todas las habitaciones tienen sistemas de climatización que el cliente podrá regular a su gusto.
H.1.12.0	• Toallas • Tienen toallas las habitaciones • Están las toallas incluidas en el cuarto de baño	Todas las habitaciones vienen equipadas con un juego de toallas para cada cliente que será recogido y repuesto cada día durante la estancia en el hotel.
H.1.12.1	• Precio Toallas • ¿Cuánto cuestan las toallas?	El servicio de toallas es gratuito por supuesto.
H.1.12.2	• Llevar Toallas • ¿Me puedo llevar las toallas al abandonar el hotel? • ¿Puedo quedarme las toallas?	Las toallas son un bien del hotel que no están a la venta ni podrán ser retiradas por los clientes al abandonar la habitación.
H.1.13.0	• Tienen terraza las habitaciones • Tienen balcón las habitaciones	No, ninguna de la habitaciones tiene terraza
H.1.14.0	• Wifi • Wireless • Conexión internet habitaciones	En todas la habitaciones podrá conectarse a internet a través de nuestra conexión wifi.

H.1.14.1	• Precio Wifi • ¿Cuánto cuesta la conexión wifi? • ¿Es gratis utilizar la conexión wifi?	El uso de la conexión wifi del hotel es gratuito, sólo tiene que solicitar la clave de acceso.
H.1.14.2	• Donde wifi • Cobertura wifi • ¿Dónde puedo conectarme a la red wifi? • ¿Donde hay cobertura wifi?	Puede conectarse a internet a través de la conexión wifi del hotel en cualquier estancia del hotel.
H.1.14.3	• Como Wifi • ¿Cómo puedo conectarme a la red wifi?	Debe solicitar la clave de acceso para conectarse a la red wireless del hotel.
H.1.14.4	• Clave Wifi • ¿Cuál es la clave de acceso de la red wifi?	La clave de acceso es 3DTOUR2010.

B.2.4. Preguntas sobre la cafetería y restaurante.

ID	Usuario	Bot
Cafetería		
C.1.1.0	• ¿Qué tienen?	Mire la carta en la pared
C.1.1.1	• Carta cafetería • ¿Me enseña la carta cafetería?	La puede ver en la pared
C.1.2.0	• Café • Quiero un café • ¿Tienen café?	Por supuesto
C.1.2.1	• Precio café • ¿Cuánto cuesta un café?	Son 1,50€
C.1.3.0	• Refresco • ¿Tienen refresco? • Quiero un refresco	Claro
C.1.3.1	• Precio refresco • ¿Cuánto cuesta un refresco?	Cuesta 2 Euros
C.1.4.0	• Bocadillo • ¿Tienen bocadillos? • ¿Quiero un bocadillo?	Si, tenemos varios bocadillos
C.1.5.0	• Comida • ¿Tienen comida? • ¿Dais comida?	No, si quiere comer tiene que ir al restaurante
C.1.5.3	• Plato combinado • ¿Preparáis platos combinados?	No preparamos platos combinados, si quiere comer debe ir a al restaurante
C.1.6.0	• Hielo • ¿Tenéis hielo?	No vendemos hielo

C.1.7.0	• Tabaco • ¿Vendéis tabaco?	No vendemos tabaco
C.1.8.0	• Periódico • Prensa • ¿Puedo ver el periódico? • ¿Tenéis la prensa? • ¿Me deja ver la prensa?	Si, tenemos prensa general y deportiva.
Restaurante		
C.1.9.0	• Carta • ¿Me enseña la carta?	Claro, tiene una carta en cada mesa.
C.1.9.1	• Menú del día • ¿Tienen menú del día?	Tenemos un menú del día con 5 primeros y segundos y postre
C.1.9.2	• Precio menú del día • ¿Cuánto cuesta el menú del día?	El menú del día cuesta 18 Euros
C.1.9.3	• Menú noche • Menú cena • ¿Es válido el menú del día por la noche? • ¿Vale el menú del día para cenar?	No, el menú del día solo es válido en la comida

B.2.5. Preguntas sobre el entorno y turismo.

ID	Usuario	Bot
Turismo		
T.1.1.0	• Visitar ciudad • Turismo • ¿Qué puedo visitar en la ciudad?	En la página Web que le muestro puede encontrar información acerca de los lugares que visitar
T.1.1.1	• Recomendar visitar • ¿Qué nos recomienda visitar?	En la página Web que le muestro los lugares para visitar, pero quizá deba ir a la oficina de turismo.
T.1.2.0	• Oficina turismo • ¿Hay oficina de turismo?	Le muestro la página de la oficina de turismo
T.1.2.1	• Donde oficina de turismo • ¿Dónde está la oficina de turismo? • ¿Cómo puedo llegar a la oficina de turismo?	Está en el centro, para llegar tiene que coger el transporte público.
T.1.3.0	• Lugares de interés • Monumentos • ¿Qué lugares de interés hay? • ¿Qué monumentos hay en la ciudad?	En la página Web que le muestro puede encontrar información acerca de los lugares de interés

T.1.4.0	• Alrededores • ¿Hay algo de interés en los alrededores? • ¿Qué puedo visitar cerca del hotel?	En los alrededores no hay mucho que ver. Es mejor que se vaya al centro.
T.1.5.0	• Museo • ¿Qué museos puedo visitar?	Le estoy mostrando una página Web con los museos de la ciudad
T.1.6.0	• Trayecto turístico	En el plano que le muestro se indican varios trayectos turísticos.
T.1.6.1	• Mapa Ciudad • Plano ciudad • ¿Tiene un mapa de la ciudad? • ¿Me da un plano?	Claro, en la Web que le muestro puede ver un plano de la ciudad
Transporte		
T.1.7.0	• Transporte • ¿Qué transporte puedo utilizar?	Puede coger el autobús, el metro o un taxi.
T.1.8.0	• Autobús • ¿Pasa el autobús cerca del hotel? • ¿Qué autobuses pasan cerca del hotel?	Cerca del hotel pasan varias líneas de autobús
T.1.8.1	• Tarifa autobús • Precio autobús • ¿Cuánto cuesta el autobús?	El billete de autobús cuesta 1 €
T.1.8.2	• Parada de autobús • ¿Dónde está la parada de autobús más cercana?	Justo enfrente del hotel hay una parada de autobús
T.1.9.1	• Tarifa taxi • Precio taxi	No sé exactamente cuál es el precio de una carrera de taxi
T.1.9.2	• Teléfono Taxi • ¿Cuál es el teléfono de los taxis?	Si quiere puedo llamar a los taxis
T.1.9.3	• Llamar Taxi • ¿Puede llamar a un Taxi?	Por supuesto,¿ para cuando lo quiere?
T.1.10.0	• Metro • ¿Hay parada de metro cerca del autobús?	Sí, hay una parada de metro a unos 200 metros de la puerta del hotel.
T.1.10.1	• Línea metro centro • ¿Qué línea de metro tengo que tomar para llegar al centro?	Puede pregunta por eso en la estación de metro
T.1.11.0	• Tarifa reducida •	
T.1.12.0	• Aeropuerto • ¿Cómo llego al aeropuerto? • ¿Cómo voy al aeropuerto?	Para llegar al aeropuerto le recomiendo utilizar el metro
T.1.12.1	• Tiempo aeropuerto • ¿Cuánto tardo en llegar al aeropuerto? • ¿Cuánto dura el trayecto al aeropuerto?	Lo siento, pero de eso no estoy seguro de cuanto se tarda en llegar al aeropuerto

T.1.13.0	• Estación de tren • ¿Cómo llego a la estación de tren? • ¿Cómo voy a la estación de tren?	Para llegar a la estación de tren le recomiendo utilizar el metro.
T.1.13.1	• Tiempo estación de tren • ¿Cuánto tardo en llegar a la estación de tren? • ¿Cuánto dura el trayecto a la estación de tren?	Lo siento, pero de eso no estoy seguro de cuanto se tarda en llegar a la estación de tren.
T.1.14.0	• Estación de autobús • ¿Cómo llego a la estación de autobús? • ¿Cómo voy a la estación de autobús?	Para llegar a la estación de autobús le recomiendo utilizar el metro.
T.14.1	• Tiempo estación de autobús • ¿Cuánto tardo en llegar a la estación de autobús? • ¿Cuánto dura el trayecto a la estación de autobús?	Lo siento, pero de eso no estoy seguro de cuanto se tarda en llegar a la estación de autobús.

Bibliografía

[1] Symantec Corporation. «Bots y botnets: Una amenza creciente.» 2009. http://www.symantec.com/es/mx/norton/theme.jsp?themeid=botnet.

[2] Turing, A. M. «Computing machinery and intelligence.» *Computers & thought* (MIT Press), 1995: 11-35.

[3] Marqués Rodríguez, Alejandro. *Desarrollo de un Bot de ayudaen lenguaje AIML con acceso WEB y WAP.* Editado por 1. Madrid, 2009.

[4] Orange España. «Orange: móvil - adsl - telefonía - televisión.» 2009. http://www.orange.es (último acceso: 2009).

[5] Orange España. «ayuda de Orange.» 2009. http://ayuda.orange.es/ (último acceso: 2009).

[6] Wikpedia. «Bot - Wikipedia, la enciclopedia libre.» 2009. http://es.wikipedia.org/wiki/Bot (último acceso: 4 de 11 de 2009).

[7] Wikipedia. «Bot Conversacional - Wikipedia, la enciclopedia libre.» 2009. http://es.wikipedia.org/wiki/Bot_conversacional (último acceso: 19 de 9 de 2009).

[8] Wikipedia. «Computer role-playing game - Wikipedia, the freeencyclopedia.» 2009. http://en.wikipedia.org/wiki/Computer_role-playing_game.

[9] Blizzard Entertainment. «Blizzard Entertainment: Warcraft III.» 2009. http://us.blizzard.com/es-es/games/war3/.

[10] AMAI. «Advanced Melee AI.» 2005. http://www.hornes.pwp.blueyonder.co.uk/AMAI/.

[11] Wallace, Richard, Noel Bush, Thomas Ringate, Anthony Taylor, y Jon Baer. *Artificial Intelligence Markup Language (AIML).* 25 de Octubre de 2001. http://www.alicebot.org/TR/2001/WD-aiml/.

[12] W3 Consorcium. «Start-Tags, End-Tags, and Empty-Element Tags.» *Extensible Markup Languaje (XML) 1.0.* 2000. http://www.w3.org/TR/2000/REC-xml-20001006#sec-starttags.

[13] W3 Consorcium. *Namespaces in XML 1.0 (Third Edition).* 8 de Diembre de 2009. http://www.w3.org/TR/REC-xml-names/.

[14] Beizer, Boris. *Software Testing Techniques.* 2. Editado por Van Nostrand Reinhold. New Tork, 1990.

[15] Mak, E. H. C. and Chan, S. S. M. and Li, Q. «XML vs. Object-Oriented XML: Motivations, Applications, and Performance Evaluation.» En *CW '02: Proceedings of the First International Symposium on Cyber Worlds (CW'02)*, editado por IEEE Computer Society, 0371. Washington, DC, USA, 2002.

[16] Bush, Noel. *ProgramD.* 12 de Marzo de 2006. http://aitools.org/Program_D.

[17] Sun Microsystems. *Java Naming and Directory Interface (JNDI).* http://java.sun.com/products/jndi/.

[18] Bush, Noel, y Kim Sullivan. *Getting Started with programD.* http://aitools.org/Getting_Started_with_Program_D#Configuration.2FDeployment.

[19] Kotonya, G and Sommerville, I. *Requirements Engineering. Processes and Techniques.* Editado por John Wiley & Sons. 1998.

[20] Benet, F. *Ingeniería del software. Biblioteca multimedia.* UOC, 2003.

[21] Apache Software Fundation. *Apache log4j 1.2 -log4j 1.2.* 2007. http://logging.apache.org/log4j/1.2/index.html (último acceso: 12 de 2009).

[22] JUnit.org. *Junit.org.* 2010. http://www.junit.org/.

[23] Gold, Russell. *HttpUnit Home.* 2008. http://httpunit.sourceforge.net/.

[24] CPPUnit. *CppUnit 2 Testing Framework.* 2010. https://launchpad.net/cppunit2.

[25] Bergmann, Sebastian. *PHPUnit.* 2009. http://www.phpunit.de/.

[26] Walnes, Joe. «XStream.» 2008. http://xstream.codehaus.org/index.html.

[27] LsViska, Cory S.N. *JQuery File Tree.* 25 de Marzo de 2008. http://abeautifulsite.net/blog/2008/03/jquery-file-tree/.

[28] Microsistems, Sun. *Annotations.* 2010. http://download-llnw.oracle.com/javase/1.5.0/docs/guide/language/annotations.html.

[29] Wallace, Richard S. *The elements of AIML Style.* 2003.

[30] Rodriguez, Alejandro Marqués. *Desarrollo de un bot de ayuda en lenguaje AIML con.* 1. 1 vols. Madrid, 2009.

[31] Prasolova-Førland, E. «Analyzing place metaphors in 3d educational.» En *Comput. Hum. Behav*, 185–204. 2008.

[32] Y.-S. Kim, T. Kesavadas, and S. M. Paley. «The virtual site museum: a multi-purpose, authoritative, and functional virtual heritage resource.» *Presence: Teleoper. Virtual Environ.* 2006. 245-261.

[33] Nicole.Yankelovich. «Abrir wonderland.» 2010. http://mfeldstein.com/es/open-wonderland/.

[34] R. Gu, M. Zhu, and N. Zhang. «Adaptive web3d virtual learning environment using interest management.» *Seventh IASTED International Conference on Web-based Education.* Anaheim, CA, USA: ACTA Press, 2008. 400–405.

[35] Burden, D. J. H. «Deploying embodied ai into virtual worlds. .» *Know.-Based Syst.*, 2009: 540-544.

[36] S. Negi, S. Joshi, A. K. Chalamalla, and L. V. Subramaniam. «Automatically extracting dialog models from conversation transcripts.» *Ninth IEEE International Conference on Data Mining.* Washington, DC, USA: IEEE Computer Society., 2009. 890-895.

[37] Bartle, R. *Designing Virtual Worlds. New Riders Games.* 2003.

[38] D. Feng, E. Shaw, J. Kim, and E. Hovy. «An intelligent discussionbotfor answering student queries in threaded discussions.» *11th international conference on Intelligent user interfaces.* New York, NY, USA: ACM, 2006. 171-177.

[39] —. *Chis Crawford on Interactive Storytelling (New Riders Game).* 2004.

[40] Cavazza, M, F Charles, y S. J Mead. «Interacting with virtual characters in interactive storytelling.» *First International Joint Conderence on Autonomous Agents and Multiagent Systems.* Bologna: ACM, 2002.

[41] Barros, L.M, y S. R. Musse. «Introducing narrative principles into planning-based interactive storytelling.» *ACM SIGCHI international Conference on Advances in Computer Entertainment Technology.* Valencia, Spain: ACM, NY, 2005.

[42] —. «Character-Driven Story Generation in Interactive Storytelling.» *Seventh international Conference on Virtual Sstems and Multimedia (Vsmm'01).* IEE Computer Society, 2001.

[43] Grasbon, D, y N Braun. «A morphological approach to interactice storytellin.» *Proceedins of CAST01. Living in Mixed Realities.* 2001.

[44] Abd El-Sattar, H.K. «A New Framework for Plot-Based Interactive Storytelling Generation.» *Fifth international Conference on Advances in Computer Graphics, Imaging and Visualisation.* 2008.

[45] Pizzi, D, M Cavazza, A Whittaker, y J-L Lugrin. «Automatic Generation of Game Level Solutions as Storyboards.» *Fourth Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).* 2008.

[46] wikipedia. *Mundo virtual - Wikipedia, la enciclopedia libre.* http://es.wikipedia.org/wiki/Mundo_virtual.

[47] Bartle, Richard. «Designing Virtual Worlds.» New Riders Games, 2003.

[48] Burden, David J. H. «Deploying embodied AI into virtual worlds.» *Know.-Based Syst.* (Elsevier Science Publishers B. V.) 22, nº 7 (2009): 540--544.

[49] Feng, Donghui and Shaw, Erin and Kim, Jihie and Hovy, Eduard. «An intelligent discussion-bot for answering student queries in threaded discussions.» En *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, editado por ACM, 171--177. Sydney, Australia, 2006.

[50] Rodriguez, E., J.C. Burguillo, D.A. Rodriguez, F.A. Mikic, J.C. Gonzalez-Moreno, y V. Novegil. «Developing virtual teaching assistants for open e-learning platforms.» *EAEEIE Annual Conference, 2008 19th* . Tallin, 2008. 193-198.

[51] Mikic, F.A., J.C. Burguillo, D.A. Rodriguez, E. Rodriguez, y M. Llamas. «T-Bot and Q-Bot: A couple of AIML-based bots for tutoring courses and evaluating students.» *Frontiers in Education Conference, 2008. FIE 2008. 38th Annual* . Saratoga Springs, NY , 2008. S3A-7 - S3A-12.

[52] nexB. *EasyEclipse | Free, open source, easy-to-use Eclipse distributions and plugins for Windows, Mac and Linux - Easy Eclipse downloads.* 2010. http://www.easyeclipse.org/site/home/.

[53] Eclipse Foundation. *eclipse.org home.* 2010. http://www.eclipse.org/.

[54] —. *Eclipse | Distributions | EasyEclipse Server Java.* 2006. http://www.easyeclipse.org/site/distributions/server-java.html.

[55] —. *Apache Tomcat - Welcome!* 2010. http://tomcat.apache.org/.

[56] Collins-Sussman, Ben, Brian W Fitzpatrick, and Michael Pilato. *Version Control with Subversion: For Subversion 1.2: (book compiled from revision 2147).* 1 vols. 2006.

[57] Disney. «La página Web oficial de Wall-e.» 2008. http://www.disney.es/FilmesDisney/Wall-E/ (último acceso: 11 de 2009).

[58] «Críticas de Cine:WALL-E.» 2008. http://cinelasnibat.blogspot.com/2008/06/hoy-se-publica-la-crtica-de-wall-e.html (último acceso: 2009).

[59] «Eliza Chatbot.» *Eliza Chatbot.* 2009. http://nlp-addiction.com/chatbot/eliza/.

[60] —. «bot - Buscar en Google.» http://www.google.es/search?q=bot&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:es-ES:official&client=firefox-a.

[61] Loquendo. «Loquendo TTS Demo.» 2009. http://www.loquendo.com/en/demos/demo_tts.htm.

[62] Santos, Marcelo Dos. *El test de Turing ¿Cómo evaluar la inteligencia artificial?* 2007. http://axxon.com.ar/rev/170/c-170divulgacion.htm.

[63] W3 Consorcium. *Extensible Markup Language (XML) 1.0 (Second Edition).* 6 de

Octubre de 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

Glosario de términos

Avatar	Es la representación visual de un bot, aunque es necesario que siempre exista un avatar asociado a un bot. De hecho, para hablar con un bot basta con una cuadro de texto para escribir las consultas y una pantalla para leer las respuestas. En los mundos virtuales también se llama avatar a la representación visual de los usuarios –ya sean bots o humanos- dentro del mundo virtual.
Bot	Es el nombre que recibe el programa que, conteniendo un interprete de AIML, engloba un conjunto de archivos AIML, los parse, almacena en memoria y acepta consultas a las que devuelve una respuesta.
Cadena	Es una secuencia de caracteres.
Comodín	Son caracteres que se utilizan en los patrones para permitir la aceptar en la posición del comidín de cualquier secuencia de caracteres (en el caso de AIML esta secuencia de caracteres no puede estar compuesta sólo de espacios en blanco)
Contenido mixto	En XML, se dice que un elemento tiene contenido mixto si puede contener texto y otros elementos mezclados. Esto no implica que sea obligatoria esta mezcla. El siguiente ejemplo muestra un elemento con contenido mixto. <template>Mi nombre es <bot />.</template>
Data-oriented	Se dice que un dialecto de XML es data-oriented si no permite la inclusión de contenido mixto, es decir, para cada todo XML el contenido de este es un tipo de dato literal (int, char, boolean…), un fragmento de texto (string) o un conjunto de otros elementos XML.
Document-oriented	Se dice que un dialecto de XML es document-oriented si permite la inclusión de contenido mixto.

Elemento vacío	Es un elemento XML sin contenido, es decir, que está definido por una etiqueta de inicio y opcionalmente uno o más atributos. <get name=”aficiones” />
Espacio de nombres	Es un parámetro calificador utilizado para identificar unívocamente dos elementos distintos con el mismo nombre. Es un concepto asociado a la programación en general, en el proyecto se refiere a él en el ámbito de XML.
Intérprete de AIML	Es el programa que, siguiendo las instrucciones dadas en la especificaciones de AIML para el procesado de elemento AIML, es capaz de parsear e interpretar documentos escritos en AIML.
JVM	Java Virtual Machine / Máquina virtual de Java
Mixed-content	Alias de contenido mixto
Mundo virtual	Es un espacio virtual, en el que se representan un paisaje real o inventado al que el usuario puede acceder mediante un ordenador y moverse a través de él como si se encontrase dentro de ese escenario. En los mundos virtuales más avanzados se busca que el usuario pueda realizar casi cualquier acción que pudiera realizar en el mundo real, ya sea sentarse en un sofá, hablar con otros usuarios o utilizar un vehículo para desplazarse
Object-oriented	Alias de data-oriented.
Predicados	Variables en AIML
Servidor de bots	En muchos casos servidor de bots se refiere a un bot. En otros casos un servidor de bots es un programa más ámplio que permite definir y administrar más de un bot.
Patrón AIML	Conjunto de elementos category que ofrecen una misma respuesta y cuyos patrones tienen una estructura determinada que forma una patrón de reconocimiento de frase de entrada que acepta mayores variaciones que los patrones de las category que los forman.

Patrón category	En buena parte de esta memoria se refiere a los elementos pattern del lenguaje AIML como patrones category.
Patrones de redirección	Alias de patrones de reemplazo
Patrones de reemplazo	Un tipo de estructura de reglas AIML en la que el resultado de su ejecución no es una respuesta en sí sino la modificación de alguna palabra en la consulta al bot que posteriormente vuelve a ser evaluada. Por esta razón también se les llama patrones de redirección.
Wildcard	Comodín AIML