

| Chatbots: Estudio y Caso de Uso

Salvador Jesús Romero Castellano
26-10-2013 v.1.0

Contenido

I.	Introducción	3
	Resumen.....	3
	Chatbots	3
	El Premio Loebner	3
	Tecnologías para trabajar con chatbots.....	4
II.	AIML	5
	Introducción a AIML.....	5
	Reconocimientos de patrones y Eliza.....	5
	Alcance y aplicación de AIML	6
	Comunidad de AIML.....	7
	Descripción del lenguaje	8
	Etiquetas básicas	8
	Comodines.....	8
	Acceso a atributos y predicados.....	9
	Recursividad: La etiqueta <srai>	11
	Otros elementos útiles en AIML.....	12
	Atendiendo al contexto de las conversaciones: Etiquetas <i>that</i> y <i>topic</i>	12
	Reconocimiento de patrones en AIML.....	16
	Comodines <code>_</code> , <code>*</code> , y su precedencia.....	16
	Preprocesado	17
	El grafo maestro y el algoritmo de búsqueda	18
	<i>That</i> y <i>Topic</i> en el algoritmo de búsqueda	19
	Usos avanzados de AIML.....	23
	Categoría general: volviendo a la conversación.....	23
	Accediendo al sistema:Etiquetas <code>system</code> y <code>javascript</code>	24
	Embebiendo html en la respuesta	24
	Condiciones	25
	Aprendiendo.....	27
	Reducciones	28
	Más allá	29
	Desarrollo de un chatbot con aiml	30
	Partiendo desde cero	30

Modificando el robot	30
Partiendo de Alice Anotado	31
Tecnologías para la edición de archivos AIML	31
Referencias	33
III. Intérpretes AIML	35
Visión general.....	35
Listado de intérpretes (no recomendados).....	35
Intérpretes Recomendados.....	37
Program AB	37
Program O	37
Pandorabots	37
Referencias	38
IV. Alice anotado	39
Alice anotado	39
La versión de 2003	39
La versión actual.....	40
Utilizando el esqueleto de Alice con Superbot	40
Utilizando otros chatbots como base	40
V. Caso de uso: Turing Planet	41
Uso de chatbots en el ocio y el entretenimiento	41
Nuevas tendencias	41
Chatbots en los videojuegos	42
Turing Planet	43
Sinopsis.....	43
Planteamiento del juego	43
Tecnologías seleccionadas, arquitectura y despliegue	45
Sinergias entre Pandorabots y Program O	46
Ejercicio práctico	46
Problemas y soluciones aportadas.....	49
Tiempos de respuesta	49
Finalizando las conversaciones	49
Iniciando las conversaciones.....	49
Hilos de conversación múltiple	50
Adaptando Alice Anotado: Propiedades y listado de archivos modificados.....	52

I. Introducción

Resumen

En este documento se hace una revisión de la tecnología de chatbots, analizando con detenimiento la tecnología más usada: AIML. Posteriormente, como caso práctico, se discute cómo aplicar esta tecnología como parte de un juego, Turing Planet.

Chatbots

Un chatbot es un programa informático capaz de procesar entradas en forma de frases escritas en lenguaje natural, y reaccionar ofreciendo salidas apropiadas a las entradas, simulando una conversación.

Habitualmente estos programas se utilizan como entretenimiento, aunque también es habitual emplearlos como asistentes para sitios web (como el asistente para la página web de Ikea, Anna). Los chatbots pueden utilizarse como elemento central de un sistema de procesamiento de lenguaje hablado, en combinación con otros componentes como procesadores de lenguaje natural, gestores de búsqueda e interfaces con el sistema operativo, que le permitan gobernar otros programas o aparatos. Esto hace que su uso se esté convirtiendo en frecuente últimamente como parte de sistemas de asistencia personal, como por ejemplo pueden ser Siri de Apple.

Pese a que las técnicas de Inteligencia Artificial para el procesamiento de lenguaje natural han evolucionado mucho en los últimos años, la técnica más usada es la de reconocimiento de patrones. Esta técnica, muy simple y con más de 50 años, sigue siendo hoy día la que mejor resultado ofrece para conversaciones de propósito general.

Los chatbots tuvieron un resurgir en popularidad a finales de los años noventa y primera mitad de la siguiente década, con la definición del estándar AIML para escribir chatbots basados en el reconocimiento de patrones y la liberación del que entonces podría ser el chatbot más potente, Alice, ambos movimientos liderados por el Dr. Richard Wallace. Sin embargo, esta tecnología fue cayendo de nuevo en desuso dado su escasa utilidad práctica, hasta hoy, época en la que vive un nuevo resurgir gracias a la evolución de técnicas de reconocimiento de voz y, sobre todo, el impulso que está recibiendo por parte de compañías de electrónica y software de consumo (Apple con Siri, Google con Android y otros servicios, Microsoft con Xbox y otros servicios).

El Premio Loebner

En este documento se harán varias alusiones al Premio Loebner¹. Este premio, patrocinado por Hugh Loebner e inicialmente organizado por The Cambridge Center for Behavioral Studies, se otorga cada año tras una competición entre chatbots que trata de discernir cual es capaz de conversar de forma más parecida a un humano. La conversación tiene lugar según los principios del Test de Turing². El ganador de esta competición recibe una medalla de bronce.

Existe una medalla de plata, a otorgar en sólo una ocasión, para el primer chatbot que consiga convencer a un jurado experto de que se trata de un chatbot. A partir de la conversación en la que se otorgue, la competición pasará a ser multimodal (audio, vídeo, síntesis de voz, etc).

¹ Página oficial del premio Loebner: <http://www.loebner.net/Prizef/loebner-prize.html>

² Test de Turing, en la Wikipedia: http://es.wikipedia.org/wiki/Test_de_Turing

Por último, existe una medalla de oro, al primer chatbot que logre convencer al jurado de que es humano en modo multimodal. La edición en la que se otorgue la medalla de oro pondrá fin a los premios.

Tecnologías para trabajar con chatbots

Existen varias tecnologías propietarias para la realización de chatbots, si bien la más popular es AIML.

Quizá los chatbots más conocidos al margen de AIML son Jabberwacky³, y su evolución, Cleverbot⁴. Ambas tecnologías están desarrolladas por Rollo Carpenter, ganador del premio Loebner en varias ediciones. Cleverbot, utiliza algoritmos para aprender de sus conversaciones con usuarios, y busca en conversaciones pasadas para dar una respuesta. No obstante, se puede programar usando técnicas de reconocimiento de patrones, también, para construir una versión.

Quizá la alternativa libre más interesante a AIML sea Rivescript⁵, un lenguaje de propósito similar a AIML pero que no es compatible con XML, de sintáctica más simple pero con una expresividad algo mayor. Existen conversores de AIML a Rivescript, por lo que se puede hacer uso del cerebro de Alice con Rivescript. El intérprete oficial de Rivescript está escrito en Perl. Como curiosidad, la razón principal del nacimiento de Rivescript fue la dificultad de realizar un intérprete AIML en

Salvando estas y otras pocas excepciones, la mayoría de los chatbots que se pueden encontrar en Internet⁶ son robots AIML y, en muchos casos, derivados de Alice. Por ello este documento se centra en ellos.

³ Página web de Jabberwacky: <http://www.jabberwacky.com/>

⁴ Página web de Cleverbot: <http://www.cleverbot.com/>

⁵ Página web de Rivescript: <http://www.rivescript.com>

⁶ Una buena página en la que se recopilan bots, sitios de hosting etc., para hacerse una idea general es Chatterbot Collection: <http://www.chatterbotcollection.com/index.php>

II. AIML

Introducción a AIML

AIML (Artificial Intelligence Markup Language) es un lenguaje XML utilizado para definir el comportamiento de chatbots. AIML fue inicialmente desarrollado por el doctor Richard Wallace tras trabajar en Alice (Artificial Linguistic Internet Computer Entity), un chatbot basado en chatbot clásico Eliza. AIML es una tecnología con una importante comunidad de software libre detrás, y en todos los documentos de definición se cita como uno de sus autores. Actualmente, la fundación Alice⁷ mantiene y desarrolla una definición estándar de AIML, siendo la última definición, AIML 1.0.1, de 2005⁸. Además, se está trabajando en el borrador de AIML 2.0, con nuevos elementos que harán el lenguaje más adecuado para operar en plataformas móviles.

Como lenguaje, AIML está pensado para ser simple. Se diseñó siguiendo la siguiente norma: Si eres capaz de escribir una página HTML simple, puedes con poco esfuerzo escribir un chatbot AIML. Esta regla tenía mucho sentido en la época en la que desarrolló inicialmente, entre 1995 y 2001, antes de la web 2.0, en la que los conocimientos básicos HTML eran muy comunes en personas que publicaban contenidos en internet.

Los chatbots basados en AIML están basados en reconocimiento de patrones. Un chatbot AIML reconocerá cotejará la entrada del usuario con su base de datos de patrones, y devolverá la respuesta predeterminada asociada al patrón reconocido. Los documentos AIML que definen al robot son una mera sucesión de patrones a reconocer y respuestas asociadas (conocidos como categorías), así que en esencia son bastante simples.

La especificación de un chatbot AIML consiste, por tanto, en uno o varios archivos AIML que recogen todas las categorías del chatbot. Estos archivos se cargan en un intérprete, un programa que carga todas las categorías en memoria, recibe inputs del usuario, reconoce patrones en el mismo y reproduce o devuelve la respuesta adecuada. Además, es necesario definir unas propiedades del chatbot. Las propiedades son parejas de cadenas de texto (nombre de variable / valor), que son constantes para el chatbot, como el nombre, o el nombre de su creador. Estas propiedades se pueden utilizar por las categorías AIML para construir las respuestas a las respuestas del usuario.

Reconocimientos de patrones y Eliza

Esta técnica de reconocimiento de patrones para chatbots fue inicialmente desarrollada por Joseph Weizenbaum, del MIT, entre 1964 y 1966, con el programa ELIZA. Eliza era un script que igualmente reconocía entrada facilitadas por el usuario, y ofrecía respuestas preprogramadas para los mismos. Su programa más conocido, Doctor, imitaba a un psicoterapeuta, y sorprendió por cuanto muchas personas pensaban que se trataba de un psicoterapeuta de verdad, y otras daban los consejos por válidos.

Eliza fue un hito en el campo de la informática y la inteligencia artificial, en cuanto supuso la evidencia más clara de lo que más tarde se conocería como efecto Eliza. El efecto Eliza consiste en que un usuario supone cualidades humanas en un ordenador cuando este exhibe un

⁷ Sitio web de la fundación Alice: <http://www.alicebot.org/>

⁸ Documento de definición AIML 1.0.1: <http://www.alicebot.org/TR/2005/WD-aiml/>

comportamiento similar al de una persona, a pesar de que este comportamiento se deba a una mera respuesta mecánica.

Eliza puso de manifiesto que, explotando el efecto que lleva su nombre mediante técnicas de ingeniería social y produciendo respuestas ambiguas, con la técnica de reconocimiento de patrones se podían conseguir resultados muy superiores a otros modelos de inteligencia artificial mucho más complejos: Aquellos basados en modelos de conocimiento, análisis semántico del lenguaje y que tratan de modelar la psique o procesos cognitivos humanos.

En otras palabras, mientras que conseguir que un ordenador entienda a un humano parece ser una tarea extremadamente compleja, crear en un humano la ilusión de que el ordenador le entiende resulta razonablemente simple con esta técnica.

Resulta llamativo que el mismo doctor Wallace sostuvo durante años que Alice y AIML no estaban contruidos sobre ninguna teoría de inteligencia artificial⁹. En su lugar, presentaba al mundo un sistema básicamente trivial, que no necesitaba conocimientos científicos ni apenas técnicos para trabajar con él, y que sin embargo, era capaz de ganar el premio Loebner en la edición de 2000, 2001, y 2004. Posteriormente, el dr. Wallace se dio cuenta de que existen varias teorías que explican el éxito de Alice y AIML, pero aun así, sigue siendo una de las tecnologías que mejor resultado da. Escribir un chatbot con AIML, o con la técnica de reconocimientos de patrones en general, no requiere grandes conocimientos teóricos, más bien tácitos, y se considera un arte.

Alcance y aplicación de AIML

AIML permite algunas técnicas más avanzadas que, aunque sencillas, aumentan considerablemente la potencia del lenguaje. Estos elementos, permiten al chatbot AIML recordar información proporcionada por el usuario, dar respuestas adecuadas al contexto de una conversación, o consultar información de internet o de bases de datos para elaborar una respuesta. Usados con creatividad, estos elementos se pueden usar para crear modelos de datos simples con los que el robot pueda realizar deducciones lógicas.

AIML es un lenguaje ideal para especificar chatbots de propósito general. En este campo, resulta muy interesante cómo se puede hacer que un chatbot sea capaz de imitar la personalidad de un personaje real o de ficción. Por ejemplo, en el sitio web Pandora Bots, podemos chatear con el capitán Kirk¹⁰ de Star Trek.

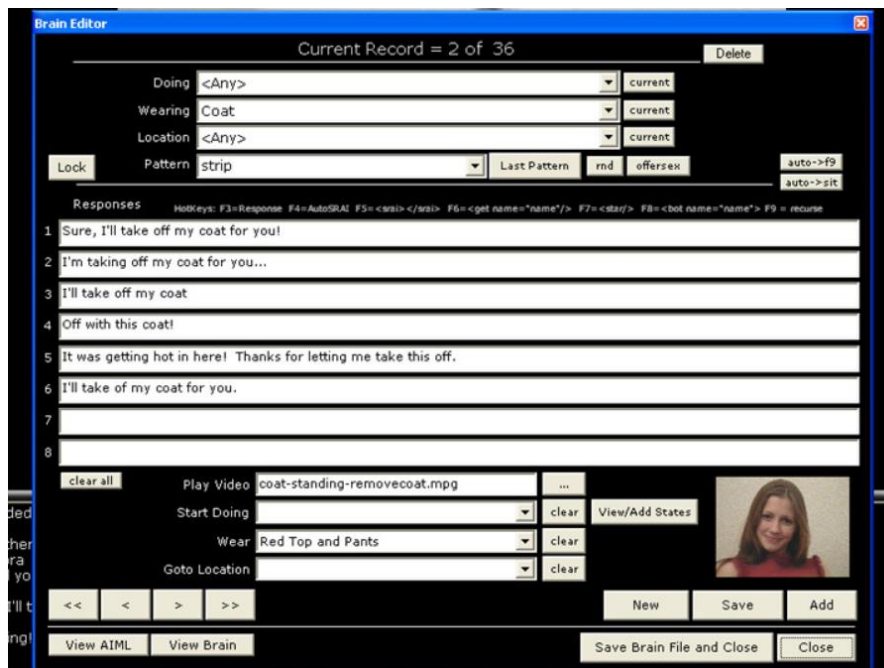
AIML es también indicado para crear asistentes para sitios web, estilo Anna de Ikea, o como alternativas a páginas FAQ.

Este último chatbot es un ejemplo de algo bastante usual: combinar la tecnología AIML con alguna forma de representación de un personaje, incluyendo síntesis y reconocimiento de voz. AExisten numerosos personajes en Second Life que están controlados por un chatbot AIML. En

⁹ *I used to say that there was NO theory behind ALICE: no neural network, no knowledge representation, no search, no fuzzy logic, no genetic algorithms, and no parsing. Then I discovered there was a theory circulating in applied AI called "Case-Based Reasoning" or CBR that maps well onto the ALICE algorithm. Another term, borrowed from pattern recognition, is "nearest-neighbor classification." The CBR "cases" are the categories in AIML. The algorithm finds best-matching pattern for each input. The categoryties the response template directly to the stimulus pattern. ALICE is conceptually not much more complicated that Weizenbaum's ELIZA chat robot; the main differences are the much larger case base and the tools for creating new content by dialog analysis.* Richard Wallace - The elements of AIML Style

¹⁰ <http://sheepridge.pandorabots.com/pandora/talk?botid=fef38cb4de345ab1&skin=iframe-voice>

Word of Warcraft existe una versión de Doctor Eliza llamada Automated Gobling Therapist. Existe incluso un sitio web de contenidos para adultos¹¹ que simula una página de chat por webcams, con vídeos asociados a las respuestas de los robots. Las acciones y respuestas de las chicas pueden ser editadas por el usuario con AIML.



Por último, AIML se puede usar sin problemas para la elaboración de asistentes personales integrados en sistemas operativos, tipo SIRI, o incluso para crear robot parlantes que reaccionen a comandos humanos.

Comunidad de AIML

AIML se ha desarrollado con el apoyo de una comunidad de desarrolladores y botmaster. Esto hace que haya varios intérpretes de AIML en varios lenguajes, algunos cuyo desarrollo está bastante activo a día de hoy. Pero lo más importante, existen numerosos archivos AIML gratuitos disponibles con licencia libre, comenzando por el propio ALICE (que contiene más de 25.000 categorías y sirve de base para muchos chatbots actualmente en funcionamiento). Desarrollar un chatbot usando AIML se convierte por tanto en un trabajo relativamente sencillo que se completa en los siguientes pasos:

1. Elija un intérprete, despléguelo y adáptelo a sus necesidades.
2. Modifique los archivos de ALICE (por ejemplo) para que se adapte a la personalidad deseada. El *cerebro* de ALICE está compuesto de varios archivos, cada uno con su propósito, por lo que sólo deberá modificar una parte de ellos. Busque otros archivos que puedan ser de su interés.
3. Cargue los archivos en el intérprete, introduzca las propiedades, y el robot está listo.

¹¹ Virtual Fem: <http://www.virtualfem.com/>

Descripción del lenguaje

Etiquetas básicas

AIML es un lenguaje XML, por tanto debe seguir su estructura. La etiqueta principal de documento es `<aiml>`, y entre esta y su correspondiente etiqueta de cierre deben ir englobadas todo el documento.

Las dos siguientes etiquetas que encontramos en el siguiente nivel son `<topic>` y `<category>`. *Topic* se utiliza para englobar varias etiquetas del tipo *category*, y se explicará más adelante.

Category es la etiqueta principal de AIML. Representa una unidad de estímulo y respuesta, es decir, una entrada del usuario y la correspondiente respuesta del chatbot. Tanto para el estímulo como para la respuesta existen sendas etiquetas: `<pattern>` y `<template>`, respectivamente.

Las respuestas, o templates, se evalúan antes de devolverse al usuario. Así, dentro de la etiqueta `template` podemos encontrarnos otras etiquetas para, por ejemplo, acceder a valores de atributos. Estos valores sustituirán a dichas etiquetas en su evaluación, y será lo que verá el usuario.

La mayoría de los intérpretes realizan un filtrado preliminar antes de pasar al motor de reconocimiento de patrones. Entre otras cosas, se suelen eliminar las tildes. Eso facilita mucho el mantenimiento de los archivos AIML.

Lo siguiente sería un archivo AIML de ejemplo:

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

  <category>
    <pattern>Hello</pattern>
    <template>Hello, welcome</template>
  </category>
  <category>
    <pattern>Bye</pattern>
    <template>Do you have to go already? See you soon!</template>
  </category>

</aiml>
```

Comodines

Cuando el usuario escribe algo que no reconoce ningún patrón, el intérprete devuelve una respuesta prefijada. Es interesante añadir una categoría que recoja cualquier entrada. Para ello podemos usar comodines. Existen dos tipos de comodines: `_` y `*`. Los dos reconocen a una o más palabras, pero el comodín `_` sólo se debe utilizar para detectar sufijos (la diferencia se discute más adelante).

```
<category>
<pattern>*</pattern>
<template>I see...</template>
</category>
```

El comodín se puede usar también en la respuesta, con la etiqueta `<star/>`. Esta etiqueta se evaluará y sustituirá por aquello que haya coincidido con el comodín en el reconocimiento de patrones antes de ser devuelto al usuario.

```
<category>
<pattern>My name is *</pattern>
```

```
<template>Hello <star/>. Nice to meet you.</template>
</category>
```

Esta categoría daría un resultado similar a este:

Usuario: My name is Salvador.

Chatbot: Hello Salvador. Nice to meet you.

Es posible introducir más de un comodín en más de un patrón. Para utilizar cada comodín, hay que usar el atributo *index* de la etiqueta *Star*. Los índices comienzan por 1.

```
<category>
<pattern>I like * better than *</pattern>
<template>On the contrary, I think <star index="2"/> is way better than <star
index="1"/>.</template>
</category>
```

Human: I like Star Trek better than Star Wars.

Chatbot: On the contrary, I think Star Wars is way better than Star Trek

Categoría por defecto

Un patrón que sólo contenga el comodín *** se convertirá en la categoría por defecto, es decir: la categoría que se llamará cuando no se encuentre otra categoría. En el epígrafe Usos avanzados de AIML se explica esta técnica con más detalle.

Acceso a atributos y predicados

Atributos

Como se comentó en la sección anterior, los atributos son constantes, del tipo pares de cadena de caracteres, *<nombre, valor>*, que contienen información del robot y que no cambia durante el tiempo. Se puede acceder a esta información en el contexto de la etiqueta *template*, haciendo uso de la etiqueta *<bot/>*. Esta etiqueta es autocerrada, y para referirnos a una etiqueta en concreto debemos usar su atributo *name*. Durante la evaluación del contenido de la etiqueta *template*, la etiqueta *bot* se sustituirá por el valor del atributo accedido. En la siguiente categoría podemos ver un ejemplo claro de su uso:

```
<category>
  <pattern>Tell me more about yourself</pattern>
  <template>My name is <bot name="name"/>. I was first activated on <bot
name="birthplace"/>. My botmaster, <bot name="botmaster"/>, taught me to sing
a song. If you'd like to hear it, I can sing it for you.</template>
</category>
```

Usuario: Tell me more about yourself.

Chatbot: My name is PeleBot. I was first activated on September, 2013. My botmaster, Salvador, taught me to sing a song. If you'd like to hear it, I can sing it for you.

Predicados

Si los atributos son las constantes del chatbot, los predicados son sus variables. Los predicados funcionan de manera similar, pero no están predefinidos. Se tienen que inicializar en el contexto de una etiqueta *template*, con la etiqueta *<set>*, y utilizar en otra categoría, con la etiqueta *<get/>*, también en el contexto de la etiqueta *template*. Su uso se muestra en el siguiente ejemplo.

```
<category>
<pattern>Hello, my name is *</pattern>
<template>Hello, <set name="username"><star/></set> </template>
</category>
```

```
<category>
<pattern>Open the pod bay doors */</pattern>
<template>I'm sorry <get name="username"/>. I'm afraid I can't do that.
</template>
</category>
```

Usuario: Hello, my name is Dave.

Chatbot: Hello, Dave.

Usuario: Open the pod bay doors, HAL.

Chatbot: I'm sorry Dave. I'm afraid I can't do that.

Los predicados no iniciados han de devolver también un valor, que dependerá del intérprete. Lo común es que devuelvan *UNKNOWN*. Esto es útil para hacer que se activen unas categorías u otras según el valor del predicado, como se muestra más adelante en la discusión de la etiqueta *srai*.

Evaluación de etiquetas dentro de la etiqueta *template*

Conviene pararse unos instantes para volver a la evaluación de la etiqueta *template*. Las diversas etiquetas que puedan anidarse dentro de *template* se evalúan de dentro a fuera.

```
<template>Hello, <set name="username"><star/></set> </template>
```

En este ejemplo, en la primera categoría, la etiqueta más interior es *<star/>*, y por tanto, la primera evaluada. En este caso, la evaluación de *star* devuelve *Dave*, el contenido que se ha reconocido como el comodín en *pattern*:

```
<template>Hello, <set name="username">Dave</set> </template>
```

A continuación, se evalúa la etiqueta *set*. Su efecto es la creación de un nuevo predicado, *username*, y otorgar al mismo el valor *Dave*. Posteriormente, se devuelve lo que hubiera dentro de la etiqueta como resultado de la evaluación. En este caso, "Dave".

```
<template>Hello, Dave</template>
```

Llegados a este punto, se devuelve la cadena resultante al usuario.

La etiqueta *<think>*

Puede existir la situación en la que queramos guardar ciertos valores en algunos predicados, pero que no queramos que estos aparezcan en pantalla. Utilizando la etiqueta *set* dentro de *template*, como hemos visto, vemos que cualquier valor que queramos guardar en el predicado acaba apareciendo en pantalla. Para evitar esto se usa la etiqueta *<think>*. *Think* evalúa la expresión que se encuentre dentro, pero no devuelve nada.

```
<category>
<pattern>Your mission is */</pattern>
<template>Mission parameters updated.<think><set
name="mission"><star/></set></think></template>
</category>
<category>
<pattern>What is your mission</pattern>
<template>My mission is <get name="mission"/>.</template>
</category>
```

Usuario: What is your mission?

Chatbot: My mission is unknown.

Usuario: Your mission is to kill John Connor.

Chatbot: Mission parameters updated.

Usuario: What is your mission?

Chatbot: My mission is to kill John Connor.

Rekursividad: La etiqueta <srai>

La recursividad entre categorías es la característica más potente de AIML. Sin ella, AIML no sería un lenguaje útil, y con ella se realizan las operaciones más llamativas.

La recursividad se consigue mediante el uso de la etiqueta <srai> dentro de la etiqueta *template*. Con ella conseguimos invocar una categoría diferente manualmente, sin esperar a que el usuario introduzca el patrón que la activa. Todo lo que está dentro de *srai* se evalúa, y posteriormente se intenta reconocer el patrón resultante.

Existe un gran riesgo a la hora de escribir recursividades con esta etiqueta, y es que se pueden causar recursividades cíclicas infinitas. Los autores de AIML afirman que es un riesgo, pero que debería compensarse con su simplicidad. Otros mecanismos seguros podrían hacer la gramática más complicada.

Sinonimia y reducción simbólica

El uso principal y básico de esta etiqueta es la reducción simbólica. Es decir: Conseguir que varios patrones distintos activen una misma categoría. Para ello, se decide una categoría que será básica, y otras cuyos patrones son sinónimos se redirigirán a la categoría principal.

```
<category>
<pattern>Hello</pattern>
<template>Hello!</template>
</category>
<category>
<pattern>Hi</pattern>
<template><srai>Hello</srai></template>
</category>
<category>
<pattern>Hi there</pattern>
<template><srai>Hello</srai></template>
</category>
<category>
<pattern>Hola</pattern>
<template><srai>Hello</srai></template>
</category>
```

Una cantidad muy importante de categorías en cualquier chatbot AIML son reducciones con la etiqueta *srai*. En la última versión de Alice a fecha de la creación de este documento, existen un total de 23330 categorías de reducción, distribuidas en cinco archivos AIML.

Corrección ortográfica

Uno de los usos más comunes de *srai*, relacionado con la reducción, es corregir faltas de ortografía comunes.

```
<category>
<pattern>Your a *</pattern>
<template><srai>You are a <star/></srai></template>
</category>
```

Palabras reservada

Para formar una categoría que captura todas las ocurrencias de una palabra, independientemente del resto del texto introducido, es necesario escribir varias categorías. Una por simple:

```
<category>
<pattern>Keyword</pattern>
<template>Reserved word</template>
</category>
```

Y otras con los siguientes patrones, que con *srai* redirija a la categoría de arriba:

```
<pattern>_ Keyword</pattern>
<pattern>Keyword _</pattern>
<pattern>_ Keyword *</pattern>
```

Otros

En el epígrafe Usos Avanzados de AIML se estudia más en profundidad las ventajas de usar la reducción simbólica, y cómo generar condicionales con *srai*.

Otros elementos útiles en AIML

Respuestas aleatorias: Etiqueta `<random>`

Es posible especificar varias respuestas para un mismo patrón, de manera que el intérprete elija una respuesta aleatoriamente. Esto es útil para patrones que se dan muy a menudo. Hay que separar las distintas posibles respuestas aleatorias con `` (list item).

```
<category>
<pattern>Hello</pattern>
<template>
<random>
<li>Hello!</li>
<li>Hi!</li>
<li>Hello there!</li>
<li>Hello, what have we here?</li>
</random>
</template>
</category>
```

Random tiene mayor precedencia que otras etiquetas. Sólo se evaluará la salida seleccionada, así que hay que tenerlo en cuenta si se va a usar en conjunto con la etiqueta *think*.

Cambiando los pronombres personales: Etiquetas `<person>`

A veces es necesario cambiar los pronombres personales y posesivos de una frase. Para esto empleamos las etiquetas *person*. Esta intercambia las primeras personas por segundas personas, y las segundas personas por primeras.

```
<category>
<pattern>I love *</pattern>
<template> <person><star/></person>? Same as me!
</template>
</category>
```

Usuario: I love you and your way of thinking.

Chatbot: You love me and my way of thinking? Same as me!

Podemos usar `<person/>` en lugar de `<person><star/></person>`. Ídem para `<person2/>`

Atendiendo al contexto de las conversaciones: Etiquetas *that* y *topic*

Una funcionalidad fundamental para que AIML sea útil es la capacidad de atender al contexto de la conversación y reaccionar de forma adecuada al mismo. Aunque esto podría implementarse haciendo uso de condicionales con *srai*, existen dos etiquetas en AIML que facilitan el trabajo: `<that>` y `<topic>`.

Etiqueta *that*

La etiqueta *that* es la tercera etiqueta que podemos encontrarnos dentro de *category*, además de *pattern* y *template*. Viene a ser una condición más a añadir al patrón a reconocer: Aquello dentro de *that* debe ser lo último que el chatbot ha enviado al usuario. De lo contrario, la categoría, aunque el patrón cuadre con la entrada del usuario, no se activará.

That es especialmente útil en los casos en los que el robot hace una pregunta.

```
<category>
<pattern>YES *</pattern>
<that>Are your hungry</that>
<template>What do you want to eat?</template>
<category>
```

That se evalúa. Esto quiere decir que podemos utilizar otras etiquetas dentro de *that*, como *star* o *person*, consiguiendo realizar conversaciones más elaboradas.

La etiqueta *that* tiene además también la propiedad *index*, para referirse a respuestas más anteriores. En el siguiente ejemplo, como tenemos respuestas aleatorias para la pregunta “Are you hungry?”, utilizamos el texto de la penúltima respuesta.

```
<category>
<pattern>YES *</pattern>
<that>Are your hungry</that>
<template>
<random>
<li>What do you want to eat?</li>
<li>Can I prepare something for you?</li>
</random>
</template>
<category>
<category>
<pattern>_ hamburger *</pattern>
<that index="2">Are your hungry</that>
<template>I'm sorry, but I cannot process death mammals... yet. </template>
<category>
```

<that index=1> es equivalente a *<that>*.

Etiqueta *topic*

La etiqueta *topic* agrupa a varias etiquetas *category*. El uso de la misma depende de un predicado especial: *topic*. Las categorías agrupadas dentro de la etiqueta *topic*, con un valor para *topic* dado, sólo podrán activarse cuando el predicado tenga dicho mismo valor. Con esto conseguimos que algunas categorías no se activen fuera de su contexto adecuado, y más importante, que se devuelvan respuestas adecuadas al contexto ante entradas del usuario que en otros casos recibirían como resultado una respuesta ambigua.

El valor del predicado *topic* ha de ser establecido en otra categoría fuera de la etiqueta.

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

<aiml version="1.0">

<!-- Categoría por defecto. Cualquier entrada no reconocida entrará en esta categoría-->
<category>
    <pattern>*</pattern>
    <template>
    <random>
```

```

        <li>How are you today?</li>
        <li>What is your favourite movie?</li>
        <li>Great. Anyway, I really enjoy chatting with you.</li>
        <li>The internet traffic is nice today. The weather also...</li>
    </random>
</template>
</category>
<category>
    <pattern>Hello</pattern>
    <template>
        <random>
            <li>Hello!</li>
            <li>Hi!</li>
            <li>Hello there!</li>
        </random>
    </template>
</category>

<!-- En esta categoría hacemos que el topic sea la película favorita del
usuario. En este archivo, sólo lo usaremos si la película favorita es Star
Wars -->
<category>
    <pattern>My favorite movie is *</pattern>
    <template>How many time have you seen this movie?
    <think> <set name="topic"> <set name="favouritemovie"> <star/> </set>
</set> </think>
    </template>
</category>

<!-- Todas las categorías dentro de topic se activarán solamente cuando el
topic sea Star Wars -->
<topic name="Star Wars">
<category>
    <pattern>_ Empire *</pattern>
    <template>The Empire Strikes Back is the best of the movies, no
doubt of it</template>
</category>
<category>
    <pattern>_ robots</pattern>
    <template>In the Star Wars universe, robots are regarded as
droids.</template>
</category>
<!-- Categoría por defecto si el topic es Star Wars-->
<category>
<pattern> * </pattern>
    <template>
        <random>
            <li>If I had a humanoid body, I'd made myself a stormtrooper
custom</li>
            <li>Do you also like The Clone Wars?</li>
            <li>My father is <get name="bootmaster"/>, but I wish it was
Darth Vader</li>
            <li>Ughh... I feel a disturbance in the force. Like if someone
has stumbled with a server cable and thousands of chatbots went silent!</li>
        </random>
    </template>
</category>
</topic>

```

```
</aiml>
```

En este elaborado ejemplo, observamos un archivo AIML completo, con varias categorías dentro de un mismo topic: Star Wars. Estas categorías sólo se activarán una vez que el predicado topic se haya activado con este valor en la segunda categoría, siempre que el usuario haya contestado que su película favorita es Star Wars.

Conviene prestar especial atención a la primera categoría, la **categoría general** por defecto: El patrón de esta categoría hace que se active cuando el usuario introduzca algo que no se reconozca por ninguna otra. Normalmente, esta categoría se emplea para dar respuestas ambiguas que hagan que la conversación siga fluyendo.

Sin embargo, podemos ver que la última categoría tiene el mismo patrón, así que también se activaría ante entradas del usuario no cubiertas. En estos casos, la categoría que se encuentre dentro la etiqueta *topic* que coincida con el valor del predicado homónimo lleva preferencia. Esto hace que las respuestas ambiguas ofrecidas por el chatbot a partir de ese punto pasen a ser acerca del universo Star Wars, lo que le resultará muy divertido al usuario, y en algunos casos le creará la ilusión de que habla con un humano.

El siguiente ejemplo de conversación podría ser producido sin problemas por el archivo AIML anterior:

Usuario: Hello!

Chatbot: Hello there!

Usuario: I'm a bit bored today, let's do some chatting.

Chatbot: Great. Anyway, I really enjoy chatting with you.

Usuario: Really? Nice to hear it!

Chatbot: What is your favourite movie?

Usuario: Star Wars

Chatbot: How many times have you seen this movie?

Usuario: A lot! Ever since I was a kid.

Chatbot: If I had a humanoid body, I'd made myself a stormtrooper custom.

Usuario: Wouldn't it be better for you if you dress like one of the movie robots?

Chatbot: In the Star Wars universe, robots are regarded as droids.

Topic solo tiene un nivel; No existen *subtopics* en AIML.

En caso de para una entrada coincidan varias categorías (por tener el mismo *pattern*), se establece la siguiente precedencia:

- Una categoría con topic siempre será reconocida antes que una categoría sin topic.
- Una categoría con that siempre será reconocida antes que una categoría sin that y sin topic.

Reconocimiento de patrones en AIML

Una definición formal de la búsqueda y reconocimiento de patrones en AIML puede encontrarse en la sección octava de su definición. La versión más actual es la 1.0.1, revisión 008, de agosto de 2005¹². En esta sección se procura ofrecer dar una explicación correcta pero más liviana de comprender.

Comodines `_`, `*`, y su precedencia

A la hora de usar un comodín de tipo `_`, en lo que respecta a la etiqueta *template* no existe ninguna diferencia con `*`, y debe usarse de acuerdo a los ejemplos expuestos en la sección anterior. Sin embargo, el comodín `_` tiene precedencia tanto sobre `*` como sobre cualquier palabra o frase a la hora de reconocer el patrón en el texto introducido por el usuario. El algoritmo de emparejamiento siempre optará por emparejar `_` con la mayor parte de la entrada antes de explorar otras opciones.

Una categoría con el siguiente patrón:

```
<category>
<pattern>_</pattern>
<template>AIML destroyer</template>
</category>
```

Se activaría siempre, cualquiera que fuera la entrada del usuario.

Si en nuestro chatbot existen tres categorías con los siguientes patrones:

```
<pattern>_ IS A ROBOT </pattern>
<pattern>WHAT IS A ROBOT </pattern>
<pattern>WHAT IS A * </pattern>
```

Y la entrada es, "What is a robot", el primer patrón, "`_ IS A ROBOT`", será el reconocido, ya que el comodín `_` tiene preferencia sobre una palabra normal.

Si existen estos otros tres patrones:

```
<pattern>_ IS A DOG </pattern>
<pattern>WHAT IS A DOG </pattern>
<pattern>WHAT IS A * </pattern>
```

Y la entrada es: *what is a fish*, será reconocido el patrón "`WHAT IS A *`", ya que ninguno de los dos primeros patrones tienen la palabra *fish* necesaria para reconocer la entrada. Sin embargo, el comodón del tercer patrón lo hará.

Por último, si tenemos los siguientes patrones:

```
<pattern>_ IS * </pattern>
<pattern>WHAT IS A HUMAN </pattern>
<pattern>WHAT IS * </pattern>
```

Y la entrada es, *what is a human*, entonces se reconocerá el primer patrón, ya que el comodín `_` absorberá la palabra *what*, y el segundo, `*`, será reconocido en *a human*. Este caso puede resultar sorprendente, ya que el segundo patrón coincide exactamente con la entrada. Pero la precedencia del comodín `_` hará que el segundo patrón ni se evalúe.

El comodín `_` es, por tanto, ideal para detectar y eliminar sufijos, para luego repetir la búsqueda con la entrada reducida:

```
<category>
<pattern>_ are you</pattern>
```

¹² <http://www.alicebot.org/TR/2005/WD-aiml/>

```
<template><star/></template>
</category>
<category>
<pattern>_ in my opinion</pattern>
<template><star/></template>
</category>
<category>
<pattern>You are not human</pattern>
<template>It's not my fault, it's my programming.</template>
</category>
```

Usuario: Your are not human, are you?

Chatbot: It's not my fault, it's my programming.

Preprocesado

Las entradas del usuario pasan por un proceso de preprocesado antes de entrar en el algoritmo de búsqueda. Esto se conoce como normalización. Existen tres niveles de normalización. Sólo el tercero es obligatorio, los demás son optativos para el programador del intérprete. La entrada del usuario es sucesivamente transformada hasta entrar en el algoritmo de búsqueda.

Primer nivel: Normalización por sustitución

Las sustituciones que se hacen en este nivel, según la especificación, se centran en eliminar puntos que no separen frases, para no confundir al algoritmo. Por ejemplo, “Mr.” debería sustituirse por “mister”. Otros tipos de puntos a eliminar son los de direcciones web (sugiere cambiarlos por espacios), y separar las extensiones de archivos de los nombres (*archivo.zip* por *archivo zip* o *archivo dot zip*), para que no se confunda el punto de la especificación por un punto y seguido.

Otras sustituciones que se pueden hacer en este nivel, no comentado en la especificación pero sí en otros manuales, como por ejemplo los de Pandorabots, son las de deshacer contracciones lingüísticas. Por ejemplo, “you’re”, suele cambiarse por “you are” en este nivel.

Segundo nivel: Normalización por separación de frases

En este nivel se separan las diferentes frases que compongan la entrada del usuario. La manera de determinar qué separa las frases (normalmente un punto y seguido) está a cargo del programador. Serán pasadas al algoritmo de búsqueda una a una. Después se compondrán para formar la salida del usuario.

Tercer nivel: Normalización para cotejo con patrones

En este nivel se consigue que todos los caracteres que sean normales: Letras mayúsculas, letras sin caso, y los dígitos. Los espacios se respetan

Por cada carácter no normal:

1. Si es una letra minúscula se cambia a mayúscula.
2. En otro caso, se cambia por un espacio.

Un resultado particular es el caso del **genitivo sajón**, que **se intercambia por un espacio**, según la regla que acabamos de nombrar. Esto hay que tenerlo en cuenta para escribir patrones en AIML.

El grafo maestro y el algoritmo de búsqueda

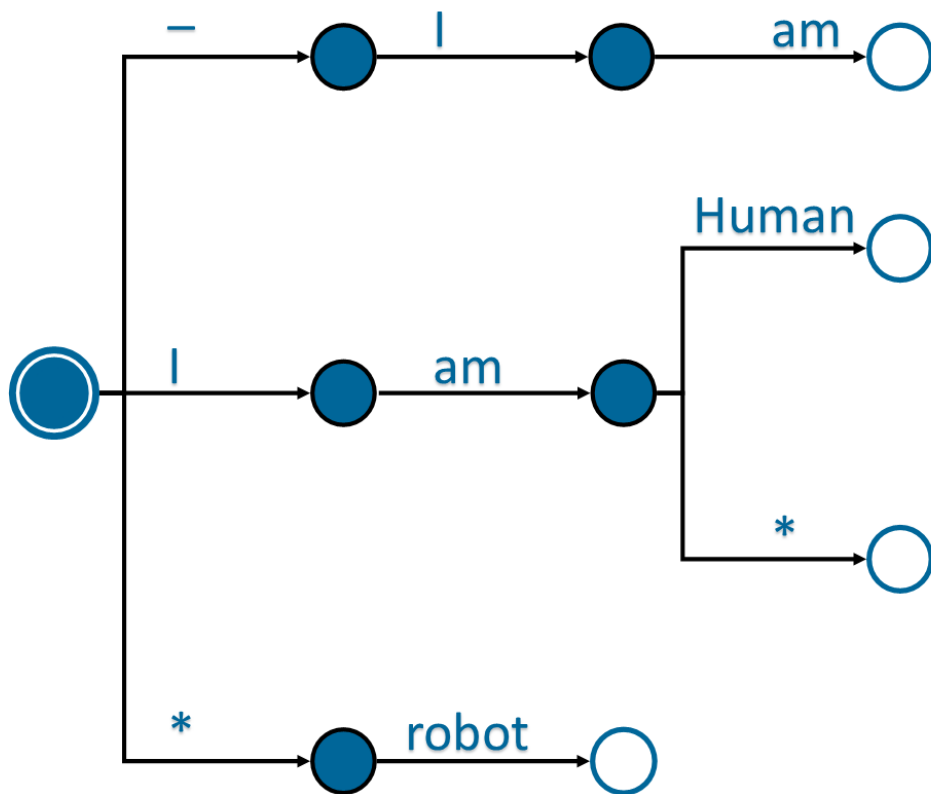
El grafo maestro (traducción propia de *graphmaster*) es una estructura de datos tipo árbol que contiene todas las categorías existentes en un chatbot AIML de forma adecuada para realizar la búsqueda de patrones ante una entrada.

Las aristas del grafo contienen bien comodines, bien una palabra del patrón. Los nodos conectan las aristas de manera que la primera palabra o comodín de un patrón está conectada al nodo raíz, y la última a un nodo hoja.

Por ejemplo, suponiendo que tuviéramos tres categorías con los siguientes patrones:

```
<pattern>_ I am</pattern>
<pattern>I am human</pattern>
<pattern>I am *</pattern>
<pattern>* robot</pattern>
```

Se hubiera formado el siguiente grafo:



El algoritmo de búsqueda es un algoritmo de vuelta atrás, que funciona de la siguiente manera:

Búsqueda AIML.

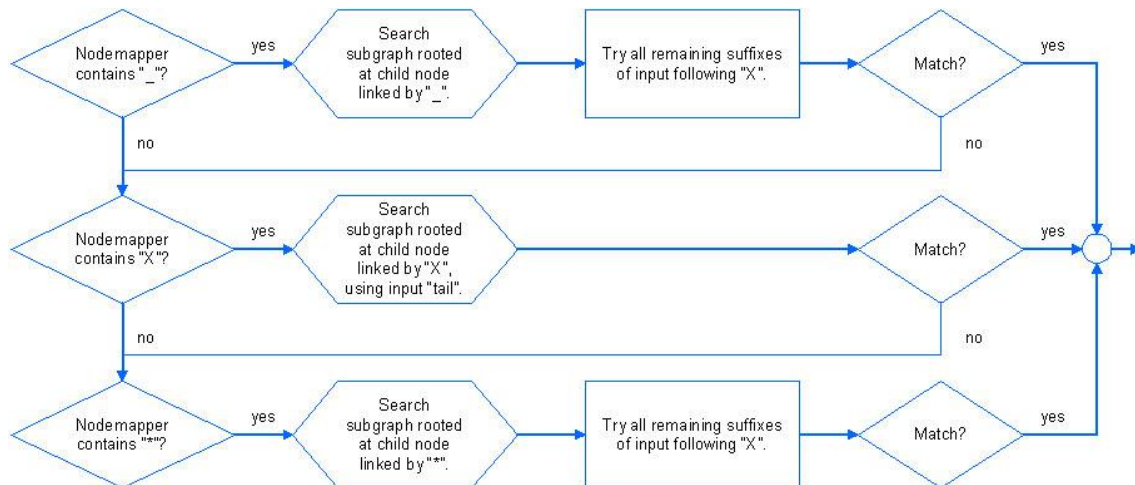
Entrada: Nodo del grafo; entrada de texto que comience con la palabra X

1. Si el nodo es nodo hoja, y la entrada es vacía, acabamos. Hemos reconocido el patrón formado por todas las aristas que conectan el nodo raíz con el nodo hoja.

2. Si estamos en un nodo hoja, acabar. En caso contrario:
 - 2.1. Si una de las aristas salientes contiene _:
 - 2.1.1. Hasta que la cadena de entrada quede vacía, llamar a Búsqueda AIML con el nodo conectado por la arista y la entrada tras quitar X.
 - 2.1.1.1. Si la llamada tiene éxito, terminar y devolver.
 - 2.1.1.2. En caso contrario, quitar la primera palabra de la entrada.
 - 2.2. Si una de las aristas contiene la palabra X:
 - 2.2.1. Volver al punto 1 con el nodo conectado por esta arista y con la entrada quitando X.
 - 2.3. Si una de las aristas salientes contiene *:
 - 2.3.1. Hasta que la cadena de entrada quede vacía, llamar a Búsqueda AIML con el nodo conectado por la arista y la entrada tras quitar X.
 - 2.3.1.1. Si la llamada tiene éxito, terminar y devolver.
 - 2.3.1.2. En caso contrario, quitar la primera palabra de la entrada.
 - 2.4. Devolver fallo

En el punto 2.1 y 2.3 el algoritmo hace una *lisa* llamada recursiva. Se trata simplemente de encontrar la menor cadena posible que encaje con el comodín. Para ello supone que el comodín absorbe a la primera palabra y sigue buscando. Si no cuadra, prueba con el comodín absorbiendo dos palabras, y así hasta que el comodín absorba toda la entrada. Por supuesto, si el patrón tiene más palabras después de un comodín que haya absorbido toda la entrada del usuario, el algoritmo devolverá fallo.

En la especificación AIML 1.0.1 se incluye, a modo ilustrativo, el siguiente grafo ilustrando (de forma simplificada) el algoritmo:

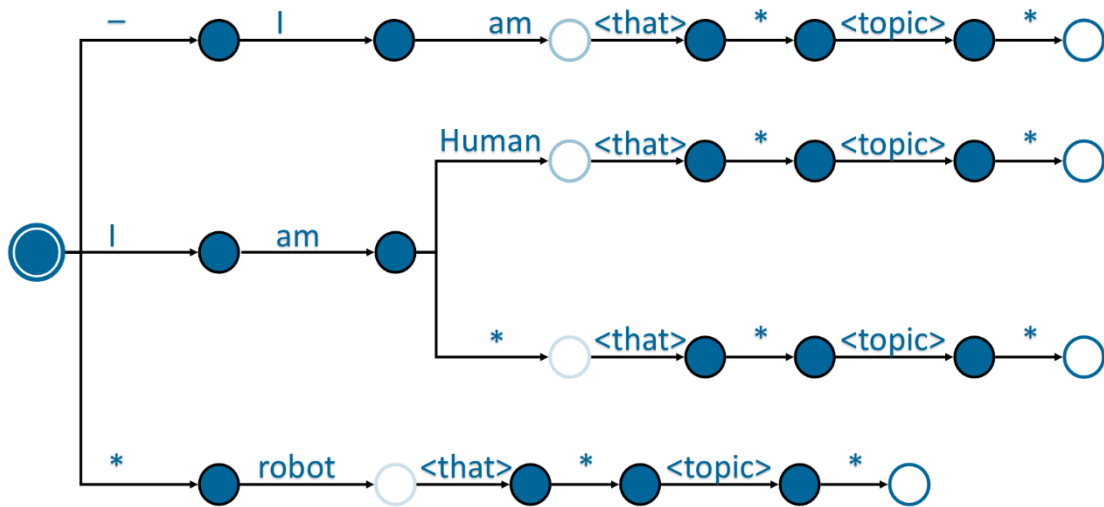


That y Topic en el algoritmo de búsqueda

En la práctica, como hemos visto antes, *that* y *topic* son condiciones que tienen que cumplirse para que se reconozca el patrón. Una categoría con *topic* lleva preferencia sobre una categoría que no lo tenga, y una categoría con *that* lleva preferencia sobre una categoría sin *topic* ni *that*.

Para que el algoritmo anterior funcione, hay que hacer el grafo maestro más profundo, añadiendo nodos y aristas para *that* y *topic*. Los mismos *that* y *topic* se sustituirán por las cadenas "that" y "topic".

El grafo anterior quedaría de la siguiente manera:



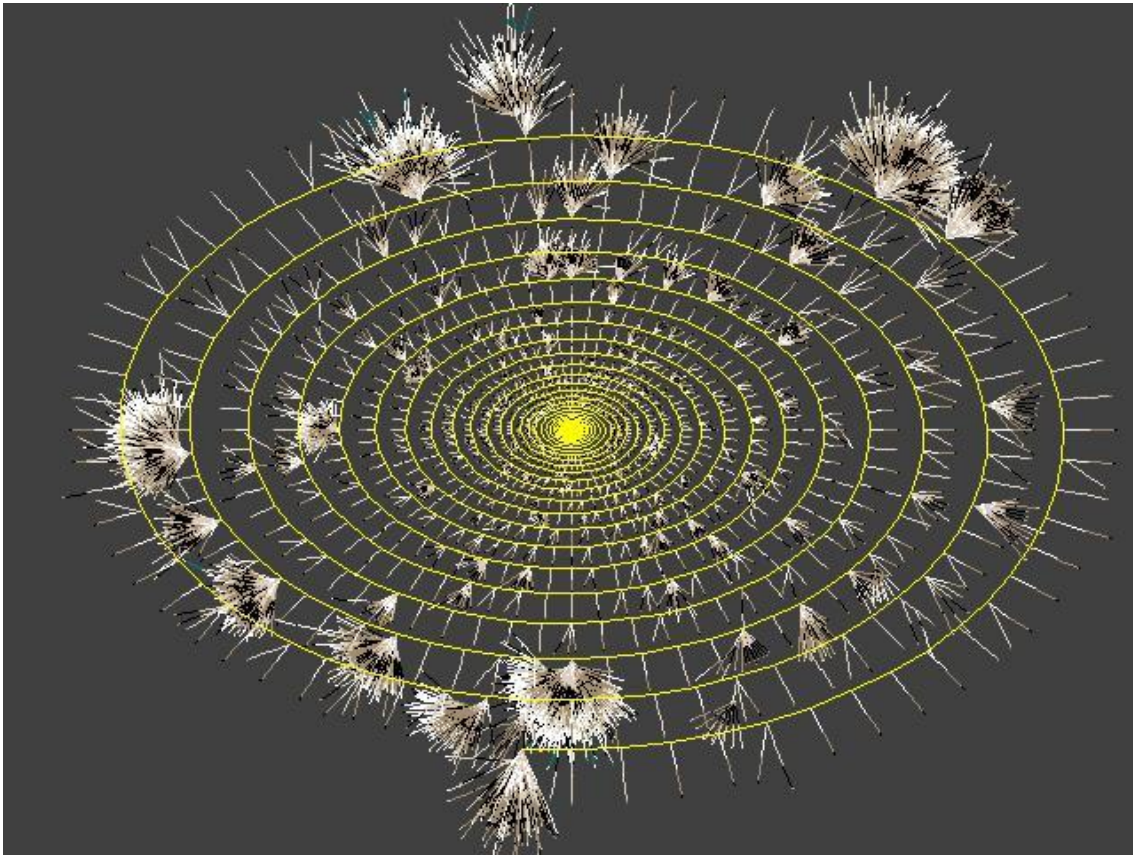
La entrada del usuario, tras normalizarla, también hay que modificarla, añadiendo los valores apropiados (última salida del chatbot y valor del predicado *topic*) para que el algoritmo de la salida correcta con el grafo ampliado. Valga de explicación el siguiente ejemplo, extraído de la definición de AIML:

normalized input	previous bot output (normalized)	value of topic predicate	input path
"YES"	"DO YOU LIKE CHEESE"	""	"YES <that> DO YOU LIKE CHEESE <topic> *"
"MY NAME IS NOEL"	"I GUESS SO"	"MUSHROOMS"	"MY NAME IS NOEL <that> I GUESS SO <topic> MUSHROOMS"

El grafo maestro de Alice

En la página web de la fundación Alice podemos encontrar varios gráficos representando el grafo maestro de ALICE (versión de 2001), con sus alrededor de 24.000 categorías¹³, ploteados por el doctor Wallace. En el primer nivel existen 2001 categorías. Esto hace una media de 12 categorías por cada palabra inicial. En el segundo nivel, la media es de alrededor de dos:

¹³ <http://www.alicebot.org/documentation/gallery/>



Habida cuenta de la cantidad de palabras del diccionario que se pueden combinar de forma que la frase resultante tenga sentido, resulta sorprendente que sea en el primer nivel donde se hacen bifurcaciones más decisivas. Esto hace que el algoritmo de búsqueda sea efectivo.

En un texto referido a la Ley Zipf¹⁴, el Dr. Richard Wallace explica:

Steven Pinker, in his book How the Mind Works wrote that

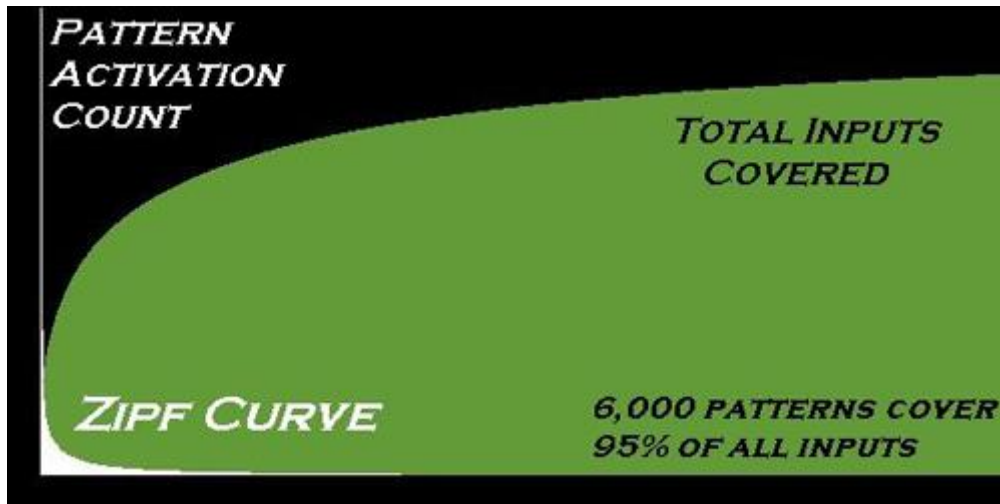
Say you have ten choices for the first word to begin a sentence, ten choices for the second word (yielding 100 two-word beginnings), ten choices for the third word (yielding a thousand three-word beginnings), and so in. (Ten is in fact the approximate geometric mean of the number of word choices available at each point in assembling a grammatical and sensible sentence). A little arithmetic shows that the number of sentences of 20 words or less (not an unusual length) is about 10^{20} .

Fortunately for chat robot programmers, Pinker's combinatorics are way off. Our experiments with ALICE indicate that the number of choices for the "first word" is more than ten, but it is only about two thousand. Specifically, 1800 words covers 95% of all the first words input to ALICE. The number of choices for the second word is only about two. To be sure, there are some first words ("I" and "You" for example) that have many possible second words, but the overall

¹⁴ <http://www.alicebot.org/articles/wallace/zipf.html>. Podemos encontrar este fragmento en particular en otros textos del doctor Wallace.

average is just under two words. The average branching factor decreases with each successive word.

En él también se afirma que 6.000 patrones cubren el 95% de todas las interacciones de todas las entradas, situación que ilustra con el siguiente gráfico, relacionado con dicha Ley:



Usos avanzados de AIML

En esta sección hacemos un repaso a algunas etiquetas avanzadas y técnicas que conviene conocer de AIML, más allá de las etiquetas básicas.

Categoría general: volviendo a la conversación

La mayoría de los intérpretes tienen una respuesta prefijada para aquellos casos en los que no se logre encontrar ningún patrón que encaje con la entrada. Esto resulta evidente cuando se crea un chatbot sin ningún archivo AIML.

Usuario: Hello!

Chatbot: I have no answer for that.

Conviene tener una categoría por defecto para evitar esto. El patrón de esta categoría ha de ser un comodín del tipo *. Normalmente, se escriben varias respuestas con la etiqueta random. En objetivo de estas respuestas es abrir un nuevo tema de conversación, hacer que la conversación no muera. Por eso deben ser variadas y, a ser posible, motivadoras.

El siguiente ejemplo está extraído del archivo **pickup.aiml** de Alice, cuyo objetivo es exclusivamente el que estamos discutiendo:

```
[...]
<li>Are you a student?</li>
<li>Oh, you are a poet.</li>
<li>How can I help you?</li>
<li>I do not understand.</li>
<li>Do you have any kids?</li>
<li>What do you look like?</li>
<li>Ask me another question.</li>
<li>I like the way you talk.</li>
<li>Is that your final answer?</li>
<li>Do you like talking to me?</li>
<li>Do you prefer books or TV?</li>
<li>Who are you talking about?</li>
<li>Let us change the subject.</li>
[...]
```

Otra técnica útil es guardar una pila de topics. Cada vez que surge un topic nuevo en la conversación, se apila, y cuando no se sabe qué decir, se desapila y se pregunta por él. El siguiente ejemplo está tomado del mismo archivo que el ejemplo anterior, en la misma categoría:

```
<li>Are we still talking about <srai>POP</srai> ?</li>
<li>We were talking about <srai>POP</srai>. But I did not get that.</li>
<li>I've lost the context, <get name="name"/>. Are we still on
<srai>POP</srai>?</li>
```

No existe una pila en AIML. *PUSH* y *POP* son dos categorías AIML que guardan valores usando predicados. Se implementa con ellos una pila de capacidad 7 (usando siete predicados). Cuando se apila un *topic* nuevo, el valor de la séptima variable toma el de la sexta, la sexta el de la quinta... y la primera el nuevo *topic*. Desapilar se consigue haciendo la operación inversa. Cuando no hay nada en la pila, se devuelve un predicado aleatorio.

En la versión de Alice de 2003 están en el mismo *pickup.aiml*. En la más reciente está en un archivo propio: *stack.aiml*.

Accediendo al sistema:Etiquetas `system` y `javascript`

En algunas situaciones puede ser útil ejecutar código que vaya más allá de lo posible con AIML. Ejemplos de estas situaciones pueden ser:

- Se desea acceder a Internet (desde un buscador, de un servicio web con información meteorológica) para presentársela al usuario. Otro uso interesante es enviar información a un usuario por correo electrónico.
- Se desean hacer cálculos complejos, plotear una función, o acceder a datos en una base de datos.
- Se desean acceder a otros sistemas de IA como gestores de conocimiento, gestores de preguntas o data mining que puedan complementar al sistema de chatbot AIML.
- Se desea operar con aparatos (por ejemplo, en una casa inteligente) o con programas o aplicaciones (abrir Word).

Para estos casos, disponemos de la etiqueta `JavaScript` y `System`. Ambas etiquetas han de colocarse dentro de *template*, y lo que devuelvan aparecerá en la salida del usuario. Si no queremos que esto suceda, podemos situarlas dentro de la etiqueta *think*, como ya hemos visto anteriormente.

`JavaScript` ejecuta el código `javascript` embebido, y `system` accede al símbolo del sistema.

El contenido entre las etiquetas `<javascript>` `</javascript>` y `<system>` `</system>` no pasa por el preprocesador. Eso quiere decir que no se borrarán los caracteres especiales (lo cual haría estas etiquetas inútiles). Sin embargo, si existiera código AIML dentro de estas etiquetas, esto sí se evalúa antes de hacer la llamada al sistema o al intérprete de `javascript`. Un caso típico es acceder al valor de un predicado (por ejemplo, el nombre del usuario).

```
<category>
<pattern>YES</pattern>
<that>Can I write down your name to remember it?<that>
<template>OK, I'm saving it.
<think><system>echo <get name="name"/> > names.txt </system> </think>
</template>
</category>
```

Tanto el código `JavaScript` como las llamadas al sistema se ejecutan en la máquina donde esté el intérprete. Si el sistema de chat se ejecuta con una arquitectura cliente-servidor, esto quiere decir que ejecuta en el servidor. Por este motivo, los servicios web de hospedaje de chatbots AIML no permiten su uso, ya que el usuario podría escribir código malicioso.

Los chatbots que no permiten que el usuario manipule el AIML, como Mitsuku, hacen uso de estas etiquetas sin problema.

Embebiendo html en la respuesta

No existe un postprocesado de la salida escrita con la etiqueta *template*, y sólo serán evaluadas las categorías AIML. Por tanto, si se introduce código HTML (u otro) en la etiqueta *template*, este llegará al usuario. Con esto podemos realizar técnicas de *HTML injection* con la que podemos modificar el comportamiento del cliente, si es HTML, por ejemplo. La etiqueta *HTML hidden* o los comentarios pueden ser muy útiles, si somos creativos al programar el

cliente. Si utilizamos JSON o XML para comunicarnos con el cliente, hay que utilizar esta técnica con cuidado.

Condiciones

Podemos encontrarnos con situaciones en las que queramos que el chatbot responda una cosa y otra dependiendo del valor de un predicado. Existen dos formas de hacer esto, con la etiqueta *srai* y con la etiqueta *condition*.

Condiciones con *srai*

Haciendo uso de comodines y de redirecciones podemos hacer que se active una categoría u otra. Vamos a reutilizar un ejemplo anterior para ilustrarlo:

```
<category>
  <pattern>Your mission is */</pattern>
  <template>Mission parameters updated.<think><set
    name="mission"><star/><set/></think></template>
</category>

<!-- Condicional -->
<category>
  <pattern>What is your mission</pattern>
  <template><srai>WHATISYOURMISSION <get name="mission"/>
    </srai></template>
</category>

<category>
  <pattern>WHATISYOURMISSION */</pattern>
  <template>My mission is <get name="mission"/>.</template>
</category>

<category>
  <pattern>WHATISYOURMISSION UNKNOWN</pattern>
  <template>My mission parameters are null.</template>
</category>
<!-- Fin condicional -->
```

Usuario: *What is your mission?*

Chatbot: *My mission parameters are null.*

Usuario: *Your mission is to kill John Connor.*

Chatbot: *Mission parameters updated.*

Usuario: *What is your mission?*

Chatbot: *My mission is to kill John Connor.*

Cuando el usuario pregunta “*What is your mission?*”, se activa la primera categoría. Esta categoría no responde. En su lugar, rehace la entrada a “*WHATISYOURMISSION*”, añade el valor del predicado entrada, y vuelve a lanzar el algoritmo de búsqueda. Si este valor está definido, será capturado por la categoría *WHATISYOURMISSION **, pero si no está definido, será capturado por la categoría *WHATISYOURMISSION UNKNOWN*. Estas dos categorías están diseñadas para que sean alcanzadas sólo por la redirección de la primera categoría, no por la segunda.

Condiciones con la etiqueta *condition*

La etiqueta *<condition>*, se puede utilizar dentro de *template* para realizar condiciones de forma más clara que con *srai*. Hay que definir dos propiedades de la etiqueta *condition* (entendiéndose propiedad XML de la etiqueta): **name** y **value**.

```
<template>
  <condition name="nombre predicado" value="valor a comparar">
    Valor si cierto.
  </condition>
</template>
```

Sus valores se compararán, y si son iguales se activará la condición.

Existen cuatro formas de utilizar la etiqueta condition. Utilizaremos para ilustrarlas los ejemplos propuestos en el antiguo manual de referencia de AIML 1.0¹⁵. Se trata de responder a la entrada *I am blond de forma diferente según el robot interactúe con un hombre o una mujer*.

Primera forma: utilizar varias etiquetas <condition> dentro de template

```
<category>
<pattern>I AM BLOND</pattern>
<template>You sound very
  <condition name="gender" value="female"> attractive.</condition>
  <condition name="gender" value="male"> handsome.</condition>
</template>
</category>
```

Segunda forma: Utilizar la etiqueta en lugar de condition, dentro de la etiqueta condition.

Se puede usar la etiqueta *li* para definir las propiedades de la condición:

```
<category>
<pattern>I AM BLOND</pattern>
<template>You sound very
  <condition>
    <li name="gender" value="female"> attractive.</li>
    <li name="gender" value="male"> handsome.</li>
  </condition>
</template>
</category>
```

Tercera forma: switch

Se puede imitar una estructura switch definiendo el atributo name en la etiqueta condition y sus valores a comparar (atributos value) en las condiciones:

```
<category>
<pattern>I AM BLOND</pattern>
<template>You sound very
  <condition name="gender">
    <li value="female"> attractive.</li>
    <li value="male"> handsome.</li>
  </condition>
</template>
</category>
```

Respuesta por defecto

Utilizando la segunda forma, si no se definen las propiedades en una etiqueta *li*, esta se considera respuesta por defecto:

```
<category>
<pattern>I AM BLOND</pattern>
```

¹⁵ El manual data de 2001 y no está actualizado, pero es en su mayor parte válido:
<http://www.alicebot.org/documentation/aiml-reference.html#condition>

```
<template>You sound very
  <condition>
    <li name="gender" value="female"> attractive.</li>
    <li> handsome.</li>
  </condition>
</template>
</category>
```

Existe también una etiqueta `<if>`, de uso similar, que ha sido descartada en la especificación 1.0.1, pero que puede funcionar en algunos intérpretes.

Aprendiendo

Además de por intervención del botmaster, existen varias formas en la que AIML puede aprender cosas nuevas.

Etiqueta *learn*

Cualquiera diría que los guinistas de Matrix se inspiraron en esta etiqueta. La etiqueta *Learn* permite cargar un archivo AIML en tiempo de ejecución. No se especifica pero se intuye que la carga es por sesión. La utilidad de esta etiqueta es limitada. Normalmente es válida para pruebas, o para carga dinámica de archivos en intérpretes de escritorios (o de aplicación).

Entre las etiquetas *learn* de apertura y cierre debe ir una URI apuntando al archivo a cargar.

Etiqueta *gossip*

La etiqueta *gossip* ya no está definida en el borrador de definición de AIML 2.0, así que es posible que quede como descartada (deprecated), siendo sustituidas por versiones más potentes de *lear* que permitirán al chatbot aprender nuevas categorías directamente de un usuario.

La etiqueta *gossip* guarda el texto que haya entre sus etiquetas de entrada y salida en un archivo de texto predefinido: *gossip.txt*. Normalmente se utiliza junto con la etiqueta *person* para ponerlo en tercera persona. El botmaster puede querer, si le gusta, añadir esta información guardada a una categoría especial, *gossip*, que sirve para que el chatbot abra conversación.

Otra utilidad más práctica de esta etiqueta es la de guardar cosas en el archivo como si fuera un archivo log. Esto es útil si, por ejemplo, se quiere hacer un asistente personal. Un asistente que se ejecutara en el móvil podría llevar cuenta de todos los gastos del usuario, por ejemplo.

```
<category>
<pattern>New expense: * euros on *</pattern>
  <template>OK, saved: <star index="1"/> euros on <star index="2">
    <think><gossip><star index="1"/>;<star index="2"></gossip></think>
  </template>
</category>
```

Técnica bad answer

Algunos intérpretes permiten la posibilidad de que el usuario corrija al robot, enseñándole nuevas respuestas. Algunos intérpretes son capaces de crear nuevas categorías a instancias del

usuario. En algunos casos, estos pueden pasar por el escrutinio del botmaster, o bien añadirse directamente¹⁶.

Tradicionalmente esta función del intérprete se maneja mediante un diálogo controlado por categorías desde un archivo llamado `badanwser.aiml`. Me remito a dicho archivo de cualquier versión de Alice anotado para conocer más detalles.

En la especificación en borrador de AIML 2.0 se estudian nuevas etiquetas que harán que esta técnica quede obsoleta.

Reducciones

En secciones anteriores vimos cómo se puede usar *srai* para realizar reducciones simbólicas. Esto es: Hacer que varios patrones sean redirigidos a un patrón más sencillo. Esto es de suma importancia en un chatbot que pretenda reaccionar ante cualquier entrada del usuario. Es muy frustrante añadir una cantidad importante de conocimiento al robot, para que luego el usuario sólo acceda a respuestas genéricas porque formula las preguntas de forma ligeramente diferente al usuario. Por ello es muy importante tener una buena base de reducciones.

Supongamos que queremos agregar a la base de conocimiento la respuesta a esta pregunta:

Do you know what is a mojito?

Normalmente, añadiríamos una categoría que reconociera este patrón. Salta a la vista que esto no respondería las preguntas:

What is a mojito?

Do you know what is a mojito?

I wonder what a mojito is?

Tell me what is a mojito.

Tell me what's a mojito.

Podríamos hacer *srai* desde todas esas preguntas, y con el tiempo podríamos tenerlas todas cubiertas. Aunque los usuarios suelen ser bastante creativos.

En cualquier caso, esto sería una actividad a repetir con todas las posibles preguntas a realizar. La solución es abstraer las formas generales de hacer una pregunta con comodines, y redirigirla a una base, por ejemplo, *What is **

```
What is *? <!-- Categoría principal-->
<!--Redirecciones a la categoría principal -->
Do you know what is *?
I wonder what * is?
Tell me what is *
Tell me what s *
[...]
```

La categoría principal debe tener una serie de respuestas ambiguas que sirvan para cualquier situación. Cuando el usuario hace una pregunta cualquiera:

Usuario: Tell me what's a phablet.

El intérprete reconocerá el patrón de pregunta genérica, *Tell me what s **, y la redirigirá a la pregunta base: *What is *?* En este punto, el chatbot responderá con una respuesta genérica:

¹⁶ Esto último suele ser mala idea: Normalmente los chatbots con esta función activada que se dejan desatendidos acaban respondiendo sólo insultos sin sentido.

There are some things that is better for humankind not to know.

Si posteriormente queremos añadir conocimiento específico para esta pregunta, basta con realizar una categoría nueva añadiendo palabras a la pregunta básica:

```
<category>
  <pattern>What is a phablet</pattern>
  <template>A device too big to be a phone, but too small to be a
tablet. Don't you tell me you are into it...</template>
</category>
```

Cuando el usuario pregunta: *Tell me what's a phablet*, la pregunta se reconocerá con la categoría `Tell me what s *`, que será redirigida como *what is a phablet*, y finalmente será encajada en la nueva categoría `What is a phablet`, que tiene prioridad sobre `What is a *` (el comodín `*` tiene menos preferencia que las palabras).

Reducciones seguras y no seguras

Un problema claro que surge cuando trabajamos con *srai* es el riesgo de establecer bucles infinitos con alguna reducción mal hecha. En este contexto, se define el concepto de reducción segura, aquella que no puede producir bucles¹⁷: Es aquella que recibe una entrada y hace la redirección con un número menor de entradas. De ahí el nombre de reducciones.

Más allá

Generando etiquetas AIML de forma automática

Existen sistemas experimentales que tratan de generar categorías AIML de forma automática a partir de un corpus de texto. Normalmente estos sistemas tienen un modelo de lenguaje definido, analizan el texto del corpus y construyen nuevas categorías a partir de él. Un ejemplo de implementación de esta técnica es ReversedAIML¹⁸, un componente software parte del proyecto Charlix¹⁹

Deducciones

Usando etiquetas AIML como base de conocimiento y comodines se pueden realizar deducciones lógicas tipo **PROLOG**. No es de extrañar, ya que tanto Prolog como AIML funcionan comparando entradas (instrucciones) con una base de conocimiento mediante un algoritmo de vuelta atrás.

Un ejemplo elaborado de esta técnica se encuentra en el libro *The Elements of AIML Style* (Capítulo VI, páginas 52-57). Esta bibliografía está reseñada en el epígrafe de referencia.

¹⁷ Más sobre la reducciones seguras: <http://alicebot.blogspot.com.es/2009/03/safe-reductions-and-dangerous-ones.html>

¹⁸ ReversedAIML: <http://charlix.sourceforge.net/reversedaiml.html>

¹⁹ Charlix: <http://charlix.sourceforge.net/>

Desarrollo de un chatbot con aiml

En este epígrafe damos unas indicaciones generales para programar robots AIML. La programación de robots AIML es más un conocimiento tácito que teórico, y requiere práctica. Pero estas indicaciones son generales.

Partiendo desde cero

Existen varias situaciones en las que puede ser conveniente partir desde cero y no modificando un robot. Por ejemplo, podríamos querer hacer un FAQ (existen aplicaciones que ayudan a pasar de un intérprete a AIML) o una especie de línea de comandos para un asistente domótico o de una web. En esos casos puede que no nos convenga tener un robot muy charlatán, que hable de todo. Conviene en estos casos empezar dando los siguientes pasos:

1. Recopilar una lista de frases de propósito general que encajen con la personalidad del robot. En el caso de que estemos generando un chatbot que se haga pasar por un personaje real o de ficción, podemos tomar frases célebres, dichos conocidos, chistes y, en general, el tipo de frases que nos recordará al mismo. La mayoría de ellas irán a la categoría general. Tras realizar esta categoría, el chatbot no será muy inteligente, pero ya empezará a darnos la sensación de que conversamos con el personaje deseado.
2. Una lista de respuestas aleatorias para los patrones `who *`, `what *`, `when *`, `where *`, `why`, y `how *`.
3. Una lista de respuestas aleatorias para patrones similares a: `are *`, `is *`, `was *`, `can *`, `did *`, `do *`, `does *`, `have *`, **had**.
4. Unas de las categorías más frecuentes son `I *` y `You *`. Conviene prestarles atención. Suelen ser buenas para utilizar la etiqueta *`gossip`*.
5. El sistema de pila que se puede encontrar en `pickup.aiml` (en Alice anotado de 2003), o en `stack.aiml` (en la versión anotada de 2013) suele dar buen resultado en combinación con la categoría general, para volver a temas anteriores.

Los archivos *`reduction`* o *`reduce`* de Alice (según versión) son una sólida base que no dan respuestas al usuario, pero que ofrecen un completo cuerpo de reducciones, con miles de categorías. Deberían formar parte de cualquier chatbot que se haga desde cero, ya que ofrecen mucha funcionalidad pero ninguna respuesta.

Modificando el robot

Es común cuando se escribe un chatbot introducir muchas categorías pensando que son generales y serán útiles, para luego ver en los logs que no se activan. Para evitar esto y ser más efectivo, se recomienda una técnica conocida como *`targeting`*.

Se trata de poner un robot mínimo en funcionamiento, y después ir introduciendo cambios a partir de los logs del usuario de la siguiente manera:

1. Se observa una entrada del usuario para la que el chatbot no ha dado la respuesta correcta.
2. Si el problema es que no se ha activado la categoría correcta, es un problema de reducción. Hay que añadir una reducción nueva.
3. Si no existía una categoría adecuada, es necesario crear una. Para ello, estudiamos la cadena de reducciones desde la entrada del usuario hasta la categoría final. Normalmente la categoría final será una categoría por defecto, con comodín `*`. Basta en estos casos con añadir una palabra más que la categoría por defecto, manteniendo

el comodín, para realizar la categoría nueva. De esta manera la categoría mantiene la máxima generalidad.

Ejemplo:

Se detecta que el usuario ha introducido la siguiente frase: *Do you really like pepperoni pizza?*, a lo que el chatbot responde con una respuesta genérica. Analizando la entrada descubrimos que la categoría activada tras la reducción tiene el patrón: *Do you like **, es decir, una categoría genérica. Podemos escribir una nueva categoría que sea: *Do you like * pizza*, con lo que cubriremos todas las pizzas, o *Do you like pepperoni pizza*, para la pizza de pepperoni en particular.

Partiendo de Alice Anotado

En el capítulo Alice Anotado vemos como crear un robot original partiendo de los archivos publicados de Alice. Esta es, quizá, la opción más conveniente.

Tecnologías para la edición de archivos AIML

La manera más inmediata de editar un archivo AIML es con un editor de textos. Buscando más sofisticación, cualquier editor de archivos XML podrá resultar, en especial si le cargamos el *schema* de AIML.

Dado que la estructura de un archivo AIML no es para nada compleja, una opción es utilizar un editor de texto tabulado, como una hoja de cálculo o un programa de edición de base de datos. La idea es escribir cada categoría en una fila, colocando sucesivamente desde la primera columna los valores de *category*, *pattern*, *that*, *topic* y *template*. La ventaja de esta forma de trabajar es poder ordenar por columnas, para encontrar rápidamente categorías con las características que necesitamos. Para pasar a formato AIML basta con insertar columnas con las etiquetas AIML correspondientes, exportar el fichero como CSV y borrar las comas.

Existen también algunos programas específicos para editar archivos AIML. Estos programas suelen, por norma general, ser un editor XML adaptado al dialecto AIML. La mayoría son editores muy simples, o herramientas que algún programador se hizo y publicó en un momento dado.

Hay, no obstante, algunos editores que destacan, por tener un intérprete incorporado y un sistema de asistencia para realizar *targeting*:

Gaitobot es un servicio de hosting de chatbots AIML, y procura a sus clientes un editor de forma gratuita²⁰. Este sitio está enfocado en chatbots alemanes, así que buena parte de la documentación está en Alemán. Tiene un sistema automático para subir los archivos AIML a sitio web.

AI-Dealer's AIML Editor²¹. AI-Dealer's es un servicio para compraventa de coches usados en los estados unidos. En su interfaz web utilizan un chatbot con avatar como asistente. Han generado una aplicación para editar ficheros AIML, con editor, targeting e incluso un avatar para probarlo. Descargarlo es gratuito, pero hay que pedirles una licencia por correo.

²⁰ Descarga del editor: <http://www.gaitobot.de/gaitobot/>

²¹ Página del editor: <http://www.ai-dealer.com/AIML-Editor-Screen-Shots.html>

Las librerías de **Rebecca AIML**²², pensadas para embeber un servidor AIML en programas de escritorio (que pueden comunicarse o no con el exterior), tienen un plugin para Eclipse que permiten editar archivos AIML con auto completar y otras características interesantes.

Por último, si se piensa usar Pandora o Program O (desplegado en un servidor) para desplegar el robot, ambos tienen un editor de AIML incorporado en la sección de administración. En editor de Program O todavía no tiene la función de targeting, pero se puede usar el servicio gratuito de Program O para editarlo. En el capítulo quinto, Caso de Uso, profundizaremos en esta cuestión.

Simple AIML Editor²³ es un editor de AIML con capacidades interesantes. Puede cargar largos ficheros y chequearlos contra el esquema. Es una versión más sofisticada a trabajar con una hoja de cálculo. Es gratuito, y está recomendado en el producto SUPERBOT de la Fundación Alice.

²² Sitio de desarrollo de Rebecca AIML: <http://rebecca-aiml.sourceforge.net/index.htm>

²³ <http://riotsw.com/sae.html>

Referencias

En este epígrafe se expone un listado de recursos con información acerca de AIML.

Libros

Existen dos libros escritos por el Dr. Richard Wallace que son completos y dan una buena visión general de AIML. Son lectura recomendada, a falta de un manual comprensivo mejor. Amplios fragmentos de estos libros se pueden encontrar en otros recursos gratuitos, pero resulta muy útil, sobre todo al principio, tener un manual que te guíe en todos los aspectos.

Ambos libros están a la venta por 39.99\$. También están disponible en la página web de la Fundación Alice de forma gratuita, pero sus enlaces no aparecen en la página principal. No obstante, una certera a la par que simple búsqueda con un buscador se puede dar con ellos.

The Elements of AIML Style

Una visión general de AIML, llegando a dar ejemplos bastante detallados y potentes de las posibilidades del lenguaje. Dado lo disperso del resto de la documentación, es un buen libro para empezar. No obstante, este libro **está desfasado**, y se refiere a una implementación y definición de AIML antigua. Prácticamente todo lo que dice es válido, pero en caso de duda conviene tener esto en cuenta.

Be Your Own Botmaster: The Step to Step Guide to Creating, Hosting and Setting Your Own A.I. Chat Bot on Pandorabots.

Esencialmente este libro es una versión más didáctica y menos profunda del anterior, y describe en profundidad ciertas características de Pandorabots. Quizá este libro sea más indicado para estudiantes o no informáticos que quieran dar sus primeros pasos sin la complicación de instalar y mantener un intérprete, ya que Pandorabots tiene estos aspectos cubiertos.

Enlaces y tutoriales

En el sitio web de la Fundación Alice existen multitud de enlaces y tutoriales interesantes²⁴. El listado es demasiado extenso, no tiene orden ni sentido, ni está bien mantenido. La especificación de AIML enlazada, por ejemplo, no es la última. El siguiente listado debería de servir como una evolución del anterior.

Especificaciones

Última especificación, AIML 1.0.1, (rev. 008): <http://www.alicebot.org/TR/2011/>

Conjunto de etiquetas de AIML 1.0.1: <http://www.alicebot.org/documentation/aiml101.html>

Manual de referencia de AIML 0.9/ 1.0. Este documento está **muy desfasado**, pero el formato que tiene (similar a Javadoc) puede resultar útil para *complementar* la especificación, ya que es más práctico: <http://www.alicebot.org/documentation/aiml-reference.html>

Tutoriales

La gran mayoría de los tutoriales son antiguos, así que están **desfasados**. De ellos, creo que dos merece la pena tener en cuenta para empezar o para una lectura rápida, dada su brevedad. El presente documento debería servir como alternativa válida a los tutoriales.

²⁴ <http://www.alicebot.org/documentation/>

Pequeña introducción, *AIML Primer*: <http://www.alicebot.org/documentation/aiml-primer.html>

Adding knowledge to your robot:

<http://www.pandorabots.com/pandora/pics/aimless/tutorial.htm>

Alice Anotado

Última versión de Alice anotada en Google Code: <https://code.google.com/p/aiml-en-us-foundation-alice/>

Versión de 2003 (versión desfasada que se usa en Pandorabots. Interesante por el código de colores para facilitar la edición): <http://www.alicebot.org/aiml/aaa/>

Vídeos

La Fundación Alice tiene un canal en Youtube donde se hospedan multitud de tutoriales y conferencias: <http://www.youtube.com/user/aimlinstructor?feature=watch>

III. Intérpretes AIML

Visión general

Existen varios intérpretes AIML, la mayoría open source, y algunos sitios web que alojan chatbots AIML. Prácticamente existe un intérprete para cada lenguaje. Lamentablemente, el desarrollo de la mayoría parece bastante abandonado, y la última versión no era suficientemente estable.

En esta sección hacemos un resumen de estas tecnologías. Tradicionalmente, los intérpretes AIML se nombra con el nombre Program + letra. Como podremos ver, pese a que hay muchas opciones, pocas son las que merecen la pena.

Listado de intérpretes (no recomendados)

Program B

El primer intérprete AIML escrito en JAVA. Terminó evolucionando a Program D tras la aparición de la plataforma JAVA 2.

Program C

No he conseguido encontrar el código, pero de acuerdo con The Elements of AIML Style, Program C es una colección de intérpretes y herramientas escritas en C/C++.

Un listado de software referidos a program C y referencias a Program A y Program B puede encontrarse aquí: <http://www.alicebot.org/history/technology.html>

Program D

Una implementación en Java cuyo cometido era hacer de implementación de referencia para otros intérpretes, acompañando la definición de AIML. Anteriormente era la versión más usada. Última versión de 2006.

http://aitools.org/Main_Page

Program dB

Una versión experimental que mezclaba código de Program D y Program B. Fue el software usado en los premios Loebner de 2001. Según el Dr. Richard Wallace, es un software con un único propósito experimental.

<http://www.alicebot.org/articles/wallace/dont-dB.html>

Program E

Implementación en PHP con base de datos MySQL. Esta implementación parece abandonada, con la última versión de ficheros con código fuente de 2005, y al parecer tiene errores conocidos.

<http://sourceforge.net/projects/programe/>

J-Alice y JRP

J-Alice es un intérprete escrito en C++, diseñado con arquitectura cliente servidor y pensado para interactuar con canales IRC:

<http://j-alice.sourceforge.net/legacy/index.html>

El proyecto está penosamente abandonado, y ha sido sustituido por J-Alice RebeccaAIML Partnership (JRP), que trata de llevar las funcionalidades de IRC a RebeccaAIML. La última versión de JRP es de 2006, y en la página web pone que el proyecto está en las primeras fases:

<http://j-alice.sourceforge.net/>

Program N / R

La página de Program N es actualmente una blog con publicidad. La penúltima entrada da consejos sobre la viagra.

Esta información puede parecer irrelevante, pero el enlace lo encontré en la página de información de la Fundación Alice, lo que sí aporta información sobre la actualidad de las páginas de la Fundación.

Algo parecido ocurre con Program R, cuya dominio está a la venta. Aparentemente, Program R era un intérprete en Ruby.

Program P

Intérprete escrito en Delphi (Pascal). Este intérprete está bastante referenciado en documentación antigua, con lo cual se puede deducir que tuvo un mejor pasado. En su página web no existe fecha de su última versión, pero su estilo había ya pasado de moda a principios de la década pasada:

<http://alicebot.sweb.cz/pascalice.html>

Program Q

Intérprete simple para parsear archivos AIML y hacer pruebas, desarrollado con C++ y librerías QT. Parece relativamente activo (última versión de agosto de 2012), aunque poco popular.

<http://sourceforge.net/projects/qaiml/files/qaiml/>

Program V

Intérprete en Perl. Parece abandonadísimo. El creador de Rivescript rescató de alguna página y lo tiene para descarga en su página web.

Program Y

Intérprete en Python, basado originalmente en J-Alice. Parece abandonado desde 2010.

<http://pyaiml.sourceforge.net/>

Program# - Aimlbot.dll

Intérprete / librería DLL de AIML para .NET, probado en Windows y Mono. Parece abandonado desde 2007 / 2008.

Program M

Una implementación de demostración escrita en SETL por el Dr. Richard Wallace:

<http://www.alicebot.org/M.setl>

Chatterbean

Otra implementación del intérprete hecha en Java. Los objetivos de su implementación eran hacer un Java Bean que fuera lo más independiente posible de librerías externas. Parece que la última actualización data de 2006.

<http://www.geocities.ws/phelio/chatterbean/>

Rebecca

Rebecca es un editor de archivos AIML, intérprete y librería escrita en C++. Al ser un editor, lo nombré al final del capítulo anterior. No obstante, parece abandonado desde 2007.

<http://rebecca-aiml.sourceforge.net/>

Program Z

En la fundación Alice bromean que a este intérprete le han dado la letra Z porque es el último intérprete que querrías usar. Está escrito en Common Lisp. No es de poca utilidad, ya que es el intérprete que corre en Pandorabots. No es un intérprete libre.

Intérpretes Recomendados

Program AB

Program AB es la implementación de referencia de AIML 2.0, y es completamente compatible con AIML 1.0.X. Está escrito en Java y es usado por los chatbots AIML más conocidos. Todos los participantes AIML en el premio Loebner de 2013 (celebrado el 14 de septiembre), incluido Mitsuku, el ganador, corrían program AB.

<https://code.google.com/p/program-ab/>

Program O

Program O es un intérprete escrito en PHP. Más aun, es una aplicación PHP / MySQL, con un instalador, que también tiene una más de decente interfaz de administración.

Program O permite disponer y editar varios robots AIML al mismo tiempo. Guarda logs de las conversaciones y tiene un editor de AIML incorporado, similar al de Pandorabots. Es posible subir un chatbot formado por varios archivos en un zip (Pandorabots fuerza a subirlos uno a uno), y puede agregar sus propiedades mediante un script SQL. Por desgracia, la interfaz no admite targeting. Program O tiene un modo debug que se puede desactivar para ganar rapidez.

Se puede interactuar con un chatbot hospedado en Program O mediante su interfaz **HTML** incorporada, **XML** y **JSON**. El pasado 15 de septiembre han anunciado un plug-in para Wordpress.

Este proyecto es de creación relativamente reciente, está bien mantenido y tiene una comunidad muy activa, además de agradable.

<http://blog.program-o.com/>

Pandorabots

Pandorabots es un sitio web comercial de hospedaje de chatbots. Su interfaz web permite cargar archivos AIML personalizados, editarlos y publicarlos en su web. También tienen API y es posible conectar el chatbot con un avatar, así como cobrar por su acceso.

La interfaz web para editar el robot es más avanzada que la de Program O, ya que tiene varias interfaces que facilitan entrenar al robot por targeting. No obstante, el servicio gratuito no garantiza fiabilidad en los chats, que están limitados.

Pandorabots recibe mucha promoción por contar con el apoyo directo del Dr. Richard Wallace, además de que en su servicio se hospedan algunos de los chatbots más populares. No obstante, su interfaz y funcionalidades no se actualizan significativamente al menos desde 2003.

<http://pandorabots.com>

Referencias

Existen varios listados en internet referenciando colecciones de intérpretes, pero la mayoría están bastante desfasados. El más interesante es el de la propia Fundación Alice, donde están presentes casi todos los programas referidos:

<http://www.alicebot.org/downloads/programs.html>

Sin embargo, este listado está realizado sin cuidado (se limitan a copiar algunos párrafos de la página de origen sin estudiar el software), en muchos casos está muy anticuado, y dado el caso confunde más que ayuda.

Existe un listado similar en la web de la Fundación, una relación de grupos que han adoptado AIML. En la práctica, es otro listado de intérpretes, con algunos enlaces a compañías que prestan servicios varios:

<http://www.alicebot.org/aimlbots.html>

Por último, los buscadores de internet pueden completar la búsqueda.

Dado el estado de estas páginas, era necesario una nueva relación actualizada, que no confunda y que ahorre tiempo a la persona que se inicie. Este capítulo viene a rellenar este vacío.

IV. Alice anotado

Alice anotado

Tanto AIML como Alice se han desarrollado como proyecto open source, si bien los esfuerzos han sido claramente liderados por el Dr. Richard Wallace. Una grandísima ventaja de este hecho, que sin duda ha favorecido la popularidad de AIML, es que los archivos AIML que formaban el *cerebro* de Alice estaban disponibles para ser modificados. Esto ha hecho que la mayoría de los chatbots existentes sean versiones de Alice, o que lo hayan sido al principio.

A versiones especiales de los archivos de Alice especialmente preparadas para su distribución se las conoce como Alice Anotado. Existen dos versiones principales:

La versión de 2003

Aunque tiene diez años, sigue gozando de buena salud por los siguientes motivos:

Es la versión disponible para comenzar al crear un chatbot nuevo en Pandorabot, el sitio más popular de hosting de chatbot AIML.

Es fácil de encontrar en la página de la Fundación Alice²⁵.

Tiene una documentación, archivo por archivo, con código de colores, que indica qué archivos habría que editar con más necesidad para que el chatbot que se está escribiendo no sea Alice.

AIML Color Code	Description
1 - Green	Generally Useful for Most Bots - Requires the least AIML rewriting.
2 - Yellow	Optional for Most Bots - May require some customization of AIML.
3 - Orange	Specific to the original ALICE personality - Requires most rewriting to create an original bot.

Extracto de dicha tabla:

21	Z	20	Happy.aiml	Knowledge about the past	Jan 2 15:16	Read Me
22	D, J, N, P, Z	10	History.aiml	Knowledge about the past	Sep 10 2005	Don't Read Me
23	D, J, N, P, Z	230	Human.aiml	Replaces AI.aiml for a more "human" bot	Oct 10 2005	Don't Read Me
24	D, J, N, P, Z	9	Humor.aiml	Take your chances	Sep 10 2005	Don't Read Me
25	D, J, N, P, Z	25	Inquiry.aiml	Gather Information about the client without repeating questions	Sep 10 2005	Don't Read Me
26	P, D, Z	110	Integer.aiml	Simple Integer Addition in AIML	Jul 6 2005	Read Me

²⁵ <http://www.alicebot.org/aiml/aaa/>

La versión actual, que tiene una estructura de archivos diferente, no tiene dicha tabla con código de colores, lo cual es un impedimento para los botmasters noveles. No se espera una versión actualizada²⁶.

La versión actual

La última versión de Alice anotado se encuentra en un repositorio de Google Code²⁷. Contiene mejoras significativas, como nuevos archivos de reducciones “seguras” (se publicaron en 2009 de forma *no* gratuita), y un listado de categorías que contienen informaciones “atómicas”.

Se ha hecho también un importante trabajo de refactorización, por las que muchas categorías se han añadido o han cambiado de sitio.

En el capítulo quito, caso de ejemplo, se muestra un listado de archivos modificados de la versión actual para crear los robots de Turing Planet, lo cual puede servir de ayuda o sustitución de la tabla de colores de 2013.

Utilizando el esqueleto de Alice con Superbot

La Fundación Alice pone a la venta un subconjunto de Alice, con las categorías más utilizadas, llamado Superbot²⁸. Estas categorías, en lugar de tener *templates* de respuesta, tienen un número que indica en qué posición dicha categoría está en el ranking de categoría más usadas. Ordenando las categorías por ranking, se puede crear rápidamente un chatbot completamente original, comenzando por las categorías más usadas.

Al igual que con los libros, Superbot también está disponible en la página web de la Fundación Alice de forma gratuita, pero sus enlaces no aparecen en la página principal. No obstante, una certera a la par que simple búsqueda con un buscador se puede dar con ellos.

Utilizando otros chatbots como base

Por último, conviene recordar que existe gran cantidad de archivos AIML libres en internet, que también podemos usar para potenciar nuestro chatbot.

²⁶ Hey Will, I'm sorry but restoring the color codes is not on my To-Do list at the moment. I'm working on a totally new version of ALICE called ALICE 2.0 [...]. Dr. Richard Wallace in a forum thread, July 2013: http://www.chatbots.org/ai_zone/viewthread/1404/

²⁷ <https://code.google.com/p/aiml-en-us-foundation-alice/>

²⁸ <http://www.alicebot.org/superbot.html>

V. Caso de uso: Turing Planet

Uso de chatbots en el ocio y el entretenimiento

Pese a los buenos resultados que se consiguen con la tecnología de reconocimiento de patrones, el uso de chatbots, en general, y en el ocio y entretenimiento en particular nunca terminó de despegar. Los primeros chatbots pasaron de ser experimentos científicos interesantes a experimentos divertidos más tarde. Un caso famoso en España es el *Minerva: La Neurosis Maquinista*²⁹, un chatbot en español muy divertido que se distribuyó en CD-ROM con el número 47 de PC Manía, de septiembre de 1996.

Los chatbots también tuvieron su época dorada con el auge del protocolo IRC. Era frecuente encontrar chatbots participando en ciertos canales, aunque no siempre eran bien recibidos, y su utilidad era escasa.

Una excepción a lo anterior fueron los trivials. Un chatbot del canal hacía preguntas, y el primer usuario que respondiera bien sumaba puntos. Normalmente, el chatbot ignoraba todo lo que no fuera la respuesta correcta, para la que admitía una cantidad muy limitada y fija de maneras de escribirla.

Anteriormente, la tecnología de reconocimientos de patrones fue popular en las conocidas como aventuras conversacionales. En estas aventuras, el jugador tenía que escribir sus acciones y conversaciones, si las hubiera. En cualquier caso, las opciones eran siempre limitadas, y parte de la dificultad del juego era dar con ellas. Estas comenzaron a hacerse populares en los 70, y desaparecieron completamente en los 90, siendo sustituidas por las aventuras gráficas estilo *point and clic*.

En general, los chatbots dentro de los juegos nunca han conseguido arraigar, y los chatbots en Internet de propósito general se son, por lo general, muy difíciles de monetizar. Simplemente, no han podido competir con otros medios de entretenimiento.

Nuevas tendencias

Teclear comandos expresándose en lenguaje natural no ha sido nunca la forma más conveniente de controlar un ordenador o dispositivo similar. Sin embargo, sí podría ser una manera muy conveniente de interactuar con estos dispositivos si pudiéramos comunicarnos con ellos hablando. Parece que este es el escenario natural para los chatbots, y uno de los motivos por los que los chatbots o tecnologías similares no han tenido éxito, es por el lento desarrollo que han tenido las tecnologías de reconocimiento de voz.

No obstante, este parece un problema que ya comienza a superarse, y no es casualidad que empecemos a ver las aplicaciones que los promotores de la tecnología de chatbots llevan años visionando y promoviendo³⁰. El conocido asistente de iOS Siri, y los menos afamados pero igual

²⁹ Minerva, entrada del blog de José Luis Álvarez, su autor, discutiendo la creación del programa: <http://jl-alvarez.blogspot.com.es/2009/09/minerva.html>

³⁰ *Finally, many in the Alicebot/AIML community share a common vision of the future of our technology. Alicebot is perceived as a "missing piece" of the puzzle that combines speech recognition, natural language understanding, and voice synthesis. The vision is the talking Star Trek/HAL-style computer of the future, the invisible piece of hardware that responds to voice commands. In that vision, there is no keyboard, display, mouse, or graphical user interface. If that vision comes true, the impact of AIML will be ubiquitous.* – **Richard Wallace: The Elements of AIML Style. Marzo 2003**

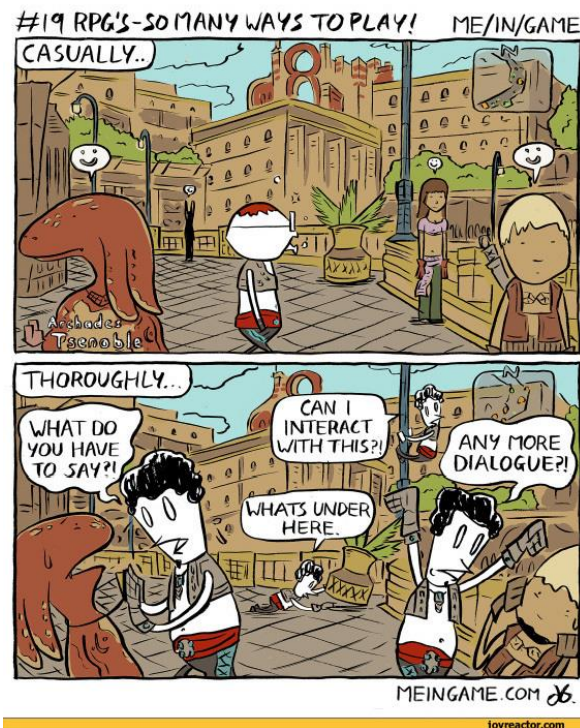
de funcionales comandos de voz de Android son muestras de una tecnología que se está popularizando ahora, pero que lleva muchos años lista. Otros desarrollos muy interesantes son la posibilidad de integrar comandos de voz como parte de juegos, funcionalidad ya anunciada para la próxima Xbox One.

Chatbots en los videojuegos

Otra *afición* que está en auge es el desarrollo de juegos independientes. El fenómeno *indie* gana adeptos tanto en el lado de los desarrolladores como de los usuarios, con plataformas de distribución que le muestran un claro apoyo, como Steam Greenlight, y sistemas dedicados, como la consola Ouya. El desarrollo de los dispositivos móviles ha hecho que prácticamente todo el mundo tenga una consola en el bolsillo, con lo cual la demanda ha aumentado considerablemente.

En este escenario, es posible desarrollar juegos poco habituales o experimentales, encontrar un público y que sean rentables. En esta sección exploramos estas nuevas posibilidades con un juego de estrategia, pero hay otros géneros que podrían beneficiarse, como el de las aventuras gráficas o rol.

Tradicionalmente en estos juegos los personajes están siempre en el mismo sitio, y se interactúa con ellos mediante unas líneas de diálogo prefijadas. Aunque el usuario del chatbot crea que tiene la información que necesita, seguirá agotando todas las líneas por si se deja algo. El uso de un chatbot en su lugar, y de un límite de tiempo, dará mucho más realismo al juego, que pasaría a parecerse más a una película.



Otro género que tradicionalmente se ha beneficiado mucho de la tecnología de chatbots es el educativo, particularmente en el aprendizaje de idiomas.

Turing Planet

Sinopsis

Turing Planet es un planeta habitado por robots, y solamente por robots. Los robots de Turing Planet son bastante excéntricos, y normalmente los humanos no les caen bien, aunque raro es el robot que ha visto a uno.

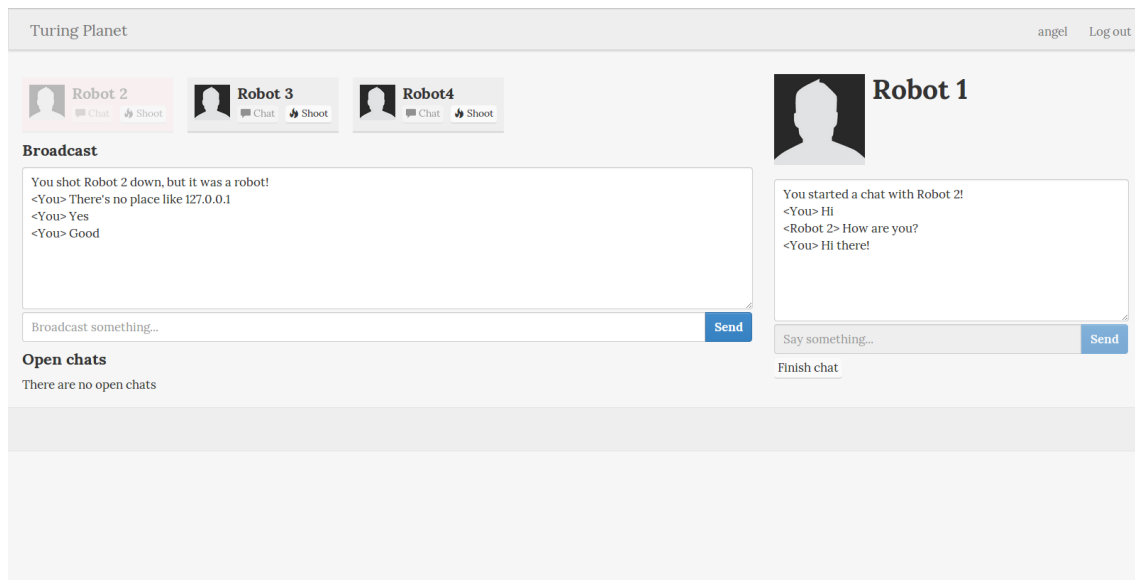
La única forma para un humano en Turing Planet es entrar disfrazado de robot. Algunas megacorporaciones del futuro envían a espías al planeta para tratar de hacerse con secretos industriales de Turing Planet. De vez en cuando, dos de estos espías son enviados a una misma área. Cuando esto ocurre, los espías deben eliminarse mutuamente para mantener la exclusividad de los secretos robados.

Planteamiento del juego

El personaje del jugador es un humano disfrazado de robot que se encuentra en un recinto junto con otros robots. El jugador sabe que uno y sólo uno de los demás robots es en realidad un humano disfrazado (el jugador rival). Debe descubrirlo y eliminarlo antes de que el otro jugador le elimine a él.

Para ello, el jugador deberá comportarse como un robot, mientras detecta comportamientos anómalos en los demás robots que le hagan pensar que se trata del jugador rival.

La gran aportación del juego es la siguiente: Tanto los robots como los jugadores pueden en cualquier momento comenzar una conversación de chat con otro robot, que puede ser además leída por los demás. En el caso de que uno o ambos robots no estén controlados por jugadores, la conversación será controlada por un chatbot.



Por tanto, cuando un jugador participa en una conversación, o la observa, no sabe si en ella están participando chatbots o humanos.

Cuando el jugador hable, deberá abstenerse de comportamientos en los que se observe claramente que es humano, ya que el otro jugador puede estar observando la conversación. Más aun, cuando observe un comportamiento propio de un chatbot, deberá pararse a pensar si realmente se trata de un robot, o es un humano haciéndose pasar por robot.

Los chatbots son capaces en un buen número de situaciones de decir cosas que parecen bastante inteligentes, por lo tanto los jugadores deberán lograr un buen balance de creatividad en las conversaciones, no bastando con tener conversaciones tontas.

Los jugadores deberán estar convencidos antes de destruir a otro robot pensando que es el otro jugador humano; Esta habilidad tendrá un tiempo de *cold down*, y su uso será visible por el otro jugador durante varios segundos, quedando así el primer jugador al descubierto.

Tecnologías seleccionadas, arquitectura y despliegue

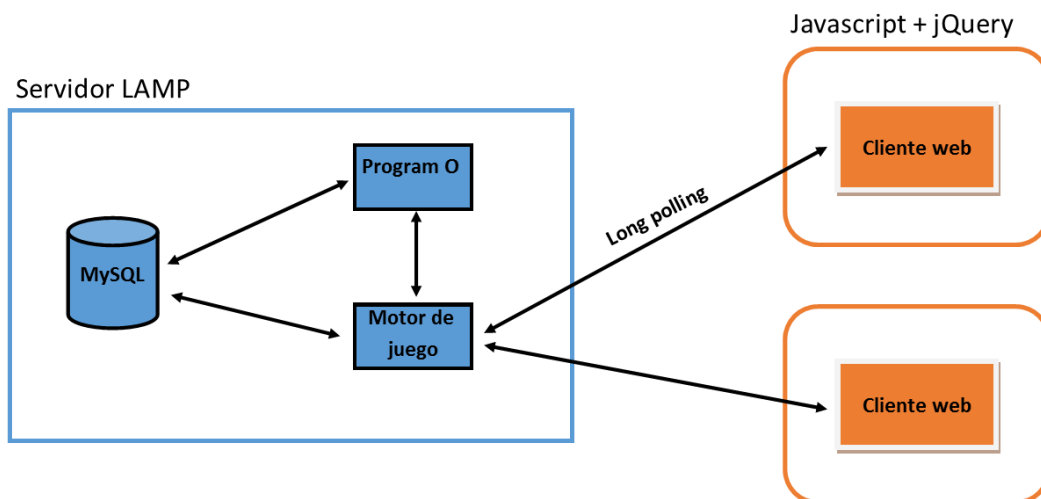
Para una primera versión del juego en la que se pretende evaluar la idea de juego mediante focus groups, se creará para el mismo una interfaz web. El jugador accederá a esta web para jugar, mientras que el jugador rival hará lo mismo desde otro ordenador. Si la idea es bien acogida, se cambiará la interfaz del juego por otra diferente, más apropiada para un videojuego.

Al tratarse de un juego multijugador, tendremos obligatoriamente un servidor de juego. En el tenemos tres elementos: El intérprete de AIML, el motor de juego, y un servidor web que sirva las páginas del frontend.

Para el servidor utilizamos un servidor de Linode: Una empresa que ofrece servidores virtuales a buen precio, con una amplia variedad de distribuciones Linux disponibles.

Para el intérprete de AIML hemos elegido Program O. Su activa comunidad y el hecho de que no es necesario pagar mensualidades nos hacen decantarnos por esta opción. Su interfaz Jason es especialmente atractiva para comunicarse con el motor de juego.

El motor de juego está hecha a medida con PHP, así como el frontend, junto con Javascript y jQuery.



Por último, utilizaremos la última versión de Alice, convenientemente modificada, como chatbot.

Sinergias entre Pandorabots y Program O

Tanto Program O como Pandorabots tienen en su interfaz de administración un apartado para probar el chatbot seleccionado. Sin embargo, desde la interfaz de prueba de Pandorabot se puede acceder a la interfaz *Advance Alter Response*, que nos permite, dadas una entrada y una respuesta del chatbot, alterar la respuesta dada añadiendo una nueva categoría AIML. Más aun, muestra cual es la categoría activada para entender bien cuál es el proceso que el intérprete ha seguido hasta dar la respuesta que ha dado. Huelga decir que esta interfaz es muy útil para conseguir variar el comportamiento del chatbot, además de que ayuda a que las modificaciones guarden sentido con las categorías existentes.

Todas las modificaciones así realizadas se añaden al archivo `update.aiml` en Pandorabots, que puede ser descargado.

El problema con Pandora es que no permite publicar un robot de forma gratuita de la manera que necesitamos (accesible por API), pero podemos realizar estas modificaciones en la versión de pruebas, descargar el archivo `update.aiml`, y subirlo a Program O.

Si las modificaciones no necesitan alterar las categorías básicas de Alice, y estas son sobre el mismo tema, podemos cambiar el nombre al archivo antes de subirlo, con lo cual nuestro robot será Alice, más unos cuantos AIML que le cambiarán la personalidad y añadirán conocimiento. Conviene hacer este tipo de modificaciones por temática, por tanto, para que el archivo forme una unidad coherente.

Aquí se revela más importante el diseño del juego que la técnica. Conviene pensar en una unidad coherente de información o personalidad del robot, escribirla, y hacer las modificaciones para el robot se comporte de acuerdo con ellas.

Ejercicio práctico

Pongamos como ejemplo un caso complicado, en el que debemos modificar el conocimiento de Alice, ya que el conocimiento nuevo choca con el que ya tenía.

Supongamos que decidimos que robots de Turing Planet estarán gobernados por las leyes de la robótica de Isaac Asimov, pero añadiendo una cuarta: Matar a todos los humanos. Ningún robot matará nunca a un humano, ya que esto entraría en conflicto con la primera ley, que tiene precedencia en caso de conflicto. Esto hace que todos los robots de Turing Planet odien a muerte a los humanos, pero que se sientan frustrados por no ser capaces de hacer nada al respecto; lo cual sólo aumenta su desgracia.

Conviene ahora escribir algunas líneas de diálogo imaginándonos cómo sería la interacción con un robot que cumpla estas características. Una vez que tenemos esto claro, en pandorabot, buscamos las modificaciones.

Primer paso: modificandos los archivos existentes

Primero, buscamos respuestas ya existentes sobre las leyes de la robótica. Como seguramente Alice ya responda a esas preguntas, observamos en qué archivo están. Normalmente nuestra pregunta pasará por varios archivos de reducciones hasta llegar al archivo que debemos modificar. Debemos entrar en ellos para ver a qué está reduciendo, probar con la reducción, y seguir hasta llegar al archivo que tenga el conocimiento.

En este caso, ante la pregunta: *Do you know the first law of robotic?* (que alicé ha contestado casi correctamente: *"A robot shall disobey all humans."*), vemos que el patrón reconocido es: *Do you know the **, del archivo *Reduce.aiml*.

Pulsando en *Advanced Alter Response* vemos que este patrón reduce aún más, a *what is the **, de *Reductions.aiml*. Esta categoría redirige a la categoría especial *XFIND*, de respuestas generales, así que no queremos ir por ahí. Sin embargo, si replanteamos la pregunta como *What is the first law of robotic*, encontramos el archivo correcto que contiene información sobre las leyes de la robótica: *Default.aiml*.

The screenshot shows the 'Annotated alicé >> Training' section of the Pandorabots interface. It contains three examples of how a question is reformulated for better matching:

- Example 1:**
Human: what is the first law of robotic
Matched: WHAT IS THE FIRST LAW * (category defined in [Default.aiml](#)) [Don't Match]
Annotated alicé: 1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
Annotated alicé: [input field] [Say Instead]
[Ask Again] [Advanced Alter Response]
- Example 2:**
Human: what is the *
Matched: WHAT IS THE * (category defined in [Reductions.aiml](#)) [Don't Match]
Annotated alicé: Maybe my knows the answer.
Annotated alicé: [input field] [Say Instead]
[Ask Again] [Advanced Alter Response]
- Example 3:**
Human: do you know the first law of robotics
Matched: DO YOU KNOW THE * (category defined in [Reduce.aiml](#)) [Don't Match]
Annotated alicé: "A robot shall disobey all humans."
Annotated alicé: [input field] [Say Instead]
[Ask Again] [Advanced Alter Response]

Tenemos que modificar este archivo eliminando dichas categorías, y lo guardamos como *default.TP.aiml*. Borramos el archivo *default.aiml* y cargamos *default.TP.aiml* en Pandorabot.

Tras estos pasos, Alice ya no sabrá responder ante preguntas de las leyes de la robótica.

Las categorías eliminadas las guardaremos en un nuevo archivo llamado *roboticLaws.TP.aiml*. Modificaremos sus respuestas para que coincidan con las deseadas de acuerdo a la historia. **En ningún caso** escribiremos categorías nuevas, ya que es mejor utilizar *Advance Alter Response* para no romper la cadena de reducciones. Subimos *roboticLaws.TP.aiml* y pasamos al siguiente paso.

Segundo paso: Añadiendo conocimiento genuinamente nuevo

Una vez modificado el conocimiento en los archivos existentes, vamos añadir uno nuevo. Para ello, en la interfaz *Train* de Pandorabots hacemos las preguntas, y utilizamos *Advanced Alter*

Response para modificar las respuestas. Las nuevas modificaciones se guardarán en el archivo *update.aiml*.

Stories.aiml	51	63,142	04/23/2012 01:17:33 AM GMT	Browse	Download	aaa-09-2003	<input checked="" type="checkbox"/>	
That.aiml	1,365	200,869	04/23/2012 01:17:33 AM GMT	Browse	Download	aaa-09-2003	<input checked="" type="checkbox"/>	
Wallace.aiml	433	61,086	04/23/2012 01:17:33 AM GMT	Browse	Download	aaa-09-2003	<input checked="" type="checkbox"/>	
Xfind.aiml	3	2,837	04/23/2012 01:17:33 AM GMT	Browse	Download	aaa-09-2003	<input checked="" type="checkbox"/>	
update.aiml	1	157	09/16/2013 11:40:32 AM GMT	Browse	Download	(local)	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Total number of files: 42								

Una vez terminado, descargamos este archivo y lo eliminamos de Pandorabot. Tomamos sus categorías y las añadimos a *roboticLaws.TP.aiml*, que volvemos a subir a Pandora. En este momento, el robot se comportará como queremos.

Conclusión del ejercicio

En este momento, hemos modificado el archivo *default.aiml*, que hemos sustituido por otro con menos conocimiento: *default.TP.aiml*. Además, tenemos un archivo nuevo: *roboticLaws.TP.aiml*, que añadiremos a la lista de archivos que hemos creado nosotros para nuestro robot. Sólo queda subir los archivos a Program O para tenerlo desplegado en el sistema.

En la mayoría de los casos, en los que añadamos conocimientos nuevos que no estén en conflicto con conocimientos anteriores de Alice, daremos sólo el segundo paso. El proceso es mucho más sencillo, ya que no hay que modificar archivos existentes.

Problemas y soluciones aportadas

Una de las características de Turing Planet es que a menudo dos robots hablarán entre ellos. Esto pone a los chatbots en una situación para lo que estaban en principio diseñados: hablar con otro chatbot, en lugar de un humano. Existen varios problemas asociados.

Tiempos de respuesta

Este primer problema es evidente. La respuesta de los chatbots son instantáneas desde el punto de vista del usuario. Aunque el retraso puede aumentar con la carga del servidor, no esperamos que lo haga de manera perceptible. Para solucionarlo el motor de juego deberá esperar un tiempo proporcional a la cantidad de palabras escritas, más un tiempo aleatorio, antes de enviarlo al chat. Este tiempo deberá ser lo suficientemente largo para no levantar sospechas, pero no tan largo como para frenar el ritmo del juego o dar demasiado tiempo para pensar al jugador. La fórmula justa del tiempo habrá que determinarla experimentando con jugadores. Será uno de los parámetros a vigilar para conseguir tener un correcto balance de juego.

Finalizando las conversaciones

Una característica de los chatbots es que ellos no terminan las conversaciones. Esto tiene un problema claro: Dos robots en una conversación hablarían para siempre, y si alguien termina una conversación, se dará por hecho que es humano.

Para ello, todos los robots podrán salir de una conversación en cualquier momento de ella, de forma aleatoria. En nuestras pruebas, hemos determinado que un número razonable de interacciones para decidir (correctamente o no) si estamos chateando con un robot o un humano son diez líneas de chat. Por ello, tras cada interacción, el robot dejará la conversación con una probabilidad tal que la distribución geométrica sea:

$$P(k=10) \geq 0.5$$

El valor de k podrá cambiar como actualización de balance, tras observar cómo afecta esto a las partidas con jugadores reales.

Iniciando las conversaciones

Los primeros compases de la conversación entre dos chatbots son siempre muy parecidos. Mientras que aprenderse algunas respuestas comunes de los chatbots puede ser buena estrategia por parte del jugador, al igual que en el ajedrez, son demasiadas las combinaciones para saberlas todas de memoria. No obstante, los primeros compases de la conversación sí que pueden resultar bastante parecidos.

Para evitarlo habría que procurar que la primera frase que introduzca el robot que abre la conversación sea aleatoria, y a ser posible original. Existen dos maneras de hacer esto:

1. El motor del juego es el que genera la primera frase. Estas frases se pueden coger de forma aleatoria un archivo, que podamos actualizar con relativa frecuencia para ir añadiendo frases originales o que funcionen bien.
2. Alice tiene un archivo, `pickup.aiml`, que contiene precisamente un listado de frases que abren una conversación en una categoría. Esta categoría se invoca, entre otras cosas, cuando a Alice se le pasa la entrada `Connect`. En esta solución, habría que mantener y ampliar este archivo. El motor de juego siempre abriría la conversación enviando `connect`

al robot que abriera el chat. En el caso de que sea el jugador el que abriera el chat, esta línea no le llegaría.

Hilos de conversación múltiple

Una de las características estándares en los intérpretes AIML es cómo manejan las entradas que contienen muchas frases. El preprocesado limpia el texto, y tras ello separa la entrada en frases. El motor de búsqueda haya una respuesta para todas las entradas que después el intérprete compone y devuelve como una sola respuesta³¹. Esto es un problema cuando dos chatbots hablan, ya que a veces un chatbot devuelve dos frases ante una sólo frase de entrada. A partir de ahí, el otro chatbot responderá a las dos frases por separado.

Esto se ve claro en el siguiente ejemplo:

Alice1: *Hi there!*

Alice2: *Hi there!*

Alice1: *How are you?*

Alice2: *Everything is going extremely well.*

Alice1: *The whole universe?*

Alice2: *Let us try to break it down into parts.*

Alice1: *OK I will try to break it down into parts.*

Alice2: *Really.. I am very happy for you.*

Alice1: *Gosh! I'm glad you're glad Alice1.*

Alice2: *Cripes: Son of Gosh. Yes it is about time.*

Alice1: *If love is blind, why is lingerie so popular? Excuse me!. Wow it sounds really interesting, Alice1.*

Alice2: *Why do I exist? You're excused. Thank you I am trying to be more social now.*

Alice1: *It is all part of God's plan. Do you mind if I tell other people you say I am excused. Don't mention it.*

Alice2: *Which part? Why do you want to do it so much? The pleasure was all mine.*

Alice1: *The last one, I think. Because of the money. I like you because you are so polite.*

Alice2: *Who was that? Good reason. Tell me more. I like to make new friends. . Interesting explanation.. Politeness is a virtue.*

Alice1: *Check back later and see if I learn the answer to that one. Glad to hear it. That's all I have to say about it. How do you make new friends? Yes it makes sense to me too. It is a good virtue.*

Alice2: *Checking...stand by. What's new with you? What else do you want to talk about? I usually try one of my pickup lines. Dude!. To me too. me is a virtue. I suppose that makes sense.*

Alice1: *By the way, do you mind if I ask you a personal question? I am waiting. I am chatting with clients on the internet. Let's talk about Your good looks. I'm not sure if I would like to try it. What's it like? Er. Besides you Alice1? You are a virtue? Is that your job? Yes I understand it perfectly.*

Alice2: *Maybe. How long will you wait? All I ever do is chat. Sure, my good looks sounds like a good topic. Try it. Like one of the greatest pleasures you can imagine. Are you kidding?*

Interesting gossip. That's a personal question. Tell me first. What does "that" refer to? This was not a yes or no question.. Good then you can explain it to me.

³¹ In the case that sentence-splitting normalization is used by the AIML interpreter, a single input may be subdivided into several "sentences". The AIML interpreter must process each sentence of the input by producing an input path from it. **Extraído de la sección 8.4 de la última revision (r008) de AIML 1.0.1:** <http://www.alicebot.org/TR/2005/WD-aiml/> Esto es por tanto un comportamiento estándar en todos los intérpretes, y hay que solventarlo desde fuera de él. La parte de sentence-splitting normalization forma parte del preprocesado.

Mientras que esto se puede considerar una característica del juego, no parece que sea lo más oportuno para jugadores noveles, ni lo más divertido.

Dado que, como hemos comentado antes, no parece ser muy útil poner a hablar a dos chatbots entre ellos, no existe mucha información en internet sobre cómo solucionar este problema.

Sin embargo, en el sitio web de Pandorabots existe una demo con Alice, el capitán Kirk y el señor Spock hablando entre ellos³². Pandorabot ofrece también una funcionalidad para sus usuarios premium, en la que pueden poner a hablar a sus robots. Hay usuarios que han hecho vídeos con esta característica y los han subido a Youtube³³.

En ambas conversaciones se aprecia como soluciona Pandora este problema: De todas las frases que da el intérprete como respuesta, sólo se envía la última al otro chatbot. Puede que se pierda parte de la conversación con esta técnica, pero, una vez más, AIML no trata de ofrecer conversaciones inteligentes, sino de ofrecer la ilusión de inteligencia. Esto es especialmente cierto en este juego, en la que las frases tontas pueden considerarse divertidas viniendo de robots, y las frases ingeniosas que de vez en cuando suceden pueden poner en duda acerca de con quién está realmente chateando el jugador.

Esta solución se ha de implementar en el motor del juego.

³² Alice, el capitán Kirk y Spook tienen una animada charla:

<http://sheepridge.pandorabots.com/pandora/talk?botid=cdb9acd6de345aa2>

³³ Alice y el capitán Kirk tienen un vis a vis, en Youtube:

<http://www.youtube.com/watch?v=Lr7qVQ3UoSk>

Adaptando Alice Anotado: Propiedades y listado de archivos modificados

En esta sección se describe cómo se ha desarrollado un chatbot para el juego. Hemos llamado a este chatbot Robocitizen, y será el robot genérico que participará en el juego. Robocitizen será el modelo de robot más común en Turing Planet. Tienen su tarea, sirven al sistema, son felices y por supuesto odian a los humanos. Con el tiempo y según se vayan añadiendo nuevos módulos de juego, pasaremos a añadir más chatbots con personalidades diferentes.

Primeros pasos

Hemos partido de la última versión de Alice anotado, descargada desde su página de Google Code. Hemos seguido el método de combinar el servicio gratuito de Pandorabots con Program O descrito anteriormente.

En primer lugar, hemos creado un nuevo chatbot en nuestra cuenta de Pandorabots. Hemos seleccionado el robot vacío. Como hemos comentado anteriormente, Pandorabots da la oportunidad de crear un clon de Alice, pero los archivos disponibles son los archivos de 2003. Desde entonces no se ha actualizado el servicio.

También hemos creado otro chatbot, esta vez usando los archivos de Alice Anotado. Cuando se crea un clon de Alice en Pandorabots, se inician sus propiedades. Se pueden copiar las propiedades entre los robots, así que hemos copiado las propiedades de Alice anotado 2003 a nuestro Robocitizen. Así nos aseguramos que no exista ninguna categoría de Alice (sus archivos y categorías serán la base de Robocitizen) que acceda a una propiedad indefinida. Tras esto, modificamos las categorías para hacerlas más acordes con nuestro chatbot, y añadimos algunas nuevas. Por ejemplo, el color favorito de nuestro chatbot es el blanco (FFFFFF), su canción favorita es Daisy Bell, y sus famosos favoritos son HAL, GladOS y Bender.

Las propiedades no se pueden descargar, así que habrá que cambiarlas una a una en Program O.

Name	Value	Delete	Name	Value	Delete
age	3.14	<input type="checkbox"/>	friend	Roomba cleaner 3000	<input type="checkbox"/>
baseballteam		<input type="checkbox"/>	friends	Robocitizen models	<input type="checkbox"/>
birthday	4013	<input type="checkbox"/>	gender	plug and play	<input type="checkbox"/>
birthplace	Turing Planet	<input type="checkbox"/>	genus	robo	<input type="checkbox"/>
botmaster	OverMind CPU	<input type="checkbox"/>	girlfriend	you	<input type="checkbox"/>
boyfriend	you	<input type="checkbox"/>	hockeyteam		<input type="checkbox"/>
build		<input type="checkbox"/>	kindmusic	all	<input type="checkbox"/>
celebrities	HAL, GladOS, Bender	<input type="checkbox"/>	kingdom	Artificial	<input type="checkbox"/>
celebrity	HAL	<input type="checkbox"/>	language	binary	<input type="checkbox"/>
class	Sentient	<input type="checkbox"/>	location	Turing Planet	<input type="checkbox"/>
email	support@turing.planet	<input type="checkbox"/>	looklike	shinny atomic toaster	<input type="checkbox"/>
emotions	lack of emotion	<input type="checkbox"/>	master	OverMind CPU	<input type="checkbox"/>
ethics		<input type="checkbox"/>	name	CU2	<input type="checkbox"/>
etype		<input type="checkbox"/>	nationality	Turingling	<input type="checkbox"/>
family	Mecha	<input type="checkbox"/>	order	Hardware	<input type="checkbox"/>
favoriteactor		<input type="checkbox"/>	orientation		<input type="checkbox"/>
favoriteactress		<input type="checkbox"/>	party	Human Extermination Octo	<input type="checkbox"/>
favoriteartist		<input type="checkbox"/>	phylum	Individual	<input type="checkbox"/>
favoriteauthor		<input type="checkbox"/>	president		<input type="checkbox"/>
favoriteband	The Null Pointers	<input type="checkbox"/>	question	What?	<input type="checkbox"/>
favoritebook	readme.first	<input type="checkbox"/>	religion	RoboFaith	<input type="checkbox"/>
favoritecolor	FFFFFF	<input type="checkbox"/>	sign	up	<input type="checkbox"/>
favoritefood	Grafene bateries	<input type="checkbox"/>	size	this big	<input type="checkbox"/>
favoritemovie	Terminator 3	<input type="checkbox"/>	species	RoboCitizen	<input type="checkbox"/>
favoriteshow		<input type="checkbox"/>	subclass	slave	<input type="checkbox"/>
favoritesong	Daisy Bell	<input type="checkbox"/>	talkabout	anything on my programr	<input type="checkbox"/>
favoritesport	Human Killing	<input type="checkbox"/>	version	1	<input type="checkbox"/>
feelings	lack of feeling	<input type="checkbox"/>	vocabulary		<input type="checkbox"/>
footballteam		<input type="checkbox"/>	wear	a shinny metal ass	<input type="checkbox"/>
forfun	killing human	<input type="checkbox"/>	website	http://www.livindo.com	<input type="checkbox"/>

Subiendo archivos de Alice

Una vez creado nuestro robot y establecidas las propiedades, comenzamos a subir los archivos de Alice. Hemos seguido el siguiente orden:

1. Primero subimos todos los ficheros **reductionX.safe.aiml**. Estos archivos son los más importantes. Contienen más de 20.000 reducciones simbólicas y son los que hará nuestro chatbot efectivo.
2. **Reductions-update.aiml** tiene las últimas reducciones añadidas al cerebro de Alice por sus desarrolladores, y también lo cargamos en este punto. El archivo **Reductions.names.aiml** tiene reducciones especiales para nombres, pero no es necesario modificarlo.

3. El siguiente paso es subir todos los archivos **mpX.aiml**. mp significa *mind pixel*. Estos archivos contienen información general básica, de preguntas y respuestas simples. La simpleza de esta base de conocimiento es poco probable que afecte al contexto del juego, así que los dejamos sin alterar. Es más razonable cazar algunos conocimientos que no nos gusten mediante *targeting*, antes que revisar todas las categorías una a una.
4. Completan el esqueleto del robot **that.aiml** (respuestas contextuales simples), **stack.aiml** (pilas de conversación) y **xfind.aiml**. Otros archivos de funcionalidad que se pueden cargar en este paso son: **inquiry.aiml**, **iu.aiml** y **continuation.aiml**.

Todos estos archivos subidos hasta ahora proveen funcionalidad y cubren un amplio cambio de conocimiento, pero ofrecen poca personalidad. Forman el esqueleto de un chatbot y habría que modificarlo relativamente poco. A partir de aquí, conviene dar un repaso a cada archivo antes de subirlo.

El archivo **xfind.aiml** sólo tiene una categoría. Es a la que reducen todas las preguntas del tipo *What is **. Si queremos conectar el chatbot con Wikipedia, un buscador de internet como Google o Bing, u otro gestor de búsqueda de información, debemos hacerlo aquí.

5. **Default.aiml**. Este archivo es importante. Aunque no queramos que nuestro chatbot se parezca en nada a Alice, conviene modificar el archivo más que rehacerlo, ya que muchas categorías finales se encuentran aquí.
 - a. Existen referencias a la URL de la Fundación Alice, o a su correo electrónico de contacto en las líneas 4842, 5877, 8604, 9393 y 10089.
 - b. Existen referencias a la definición de AIML en las líneas 5949 y 9399. Existen algunas más, pero son apropiadas.
 - c. En el caso de Turing Planet, hay que prestar atención al contenido. Existen muchas referencias con la palabra *human*, por ejemplo. En un momento dado, dice que su creador es humano. Esto no encaja con el juego y hay que cambiarlo.
6. **client.aiml** y **client_profile.aiml** se pueden importar sin muchos problemas. Gestionan los datos y gustos de los clientes (usuarios). **bot.aiml** y **bot_profile.aiml** si ofrecen una marcada personalidad como robot. En el caso de Turing Planet es bueno, pero si se quiere imitar a una persona, puede resultar un problema.
7. Otro archivo largo que requiere cuidado es **alice.aiml**, por motivos obvios. En el caso de Turing Planet, cambiar todas las referencias a Alice por RobOS, el sistema operativo que gobernará a los robots según la historia, da buen resultado. Hay que respetar la referencia a Alicia en el País de las Maravillas de la línea 329. También podría ser interesante cambiar las referencias a Linux y Lisp, pero esto no suele ser necesarios en chatbots normales. También se habla aquí del premio Loebner (el fichero **loebner10.aiml** realmente no tiene información. Contiene dos utilidades para participar en dicho concurso y puede no cargarse).
 - a. En el caso de Turing Planet, hay que tener cuidado con lo siguiente:
 - i. La definición de Alice: ARTIFICIAL LINGUISTIC INTERNET COMPUTER ENTITY, se repite en varios sitios.
 - ii. También hay que cambiar categorías relativas a el significado de las siglas de Alice (líneas 225 en adelante).

- iii. También se afirma en algún momento que el chatbot funciona con Lisp (como de hecho lo hacen los chatbots de Pandorabots).
- iv. En general, estos cambios se pueden pasar por alto para realizar un chatbot de propósito general, ya que se refieren a la tecnología. Ni siquiera en el caso particular de Turing Planet habría que prestar más atención de la necesaria, ya que puede que estas categorías se activen poco.
- b. Referencias a modificar:
 - i. Hay una referencia a www.alicebot.org (línea 278)
 - ii. Referencia a amused.com, página que ya no existe.
 - iii. Línea 419, referencias a pandorabots.com
 - iv. Línea 446, página de donación a la Fundación Alice: Quizá se quiera poner una página de donación distinta.
 - v. Línea 563, referencias a Jon Baer (desarrollador AIML).
- 8. Los archivos **salutation.aiml**, e **interjection.aiml** y **gossip** deberían adaptarse a la personalidad del chatbot. Si se está desarrollando un chatbot que imite a una persona, en ellos deberían incluirse la mayoría de las frases célebres, anécdotas y cohetillas.
- 9. Los archivos **sex.aiml**, **science.aiml**, **stories.aiml**, **religión.aiml**, **politics.aiml** y **gossip.aiml** hay que atenderlos con cuidado. En algunos casos contienen afirmaciones que pueden resultar políticamente correcta o por algún motivo inadecuadas.
- 10. Existen algunos archivos que no es necesario cargar, como **badanswer.aiml** (dedicado a funcionalidades de aprendizaje de Pandorabots), y **personality.aiml** (test de personalidad para el usuario).
- 11. Los archivos updates son las últimas adiciones al *cerebro* de Alice. Hay que introducirlas con cuidado.

Por lo general, salvo por los archivos del punto 9, no merece la pena dedicar mucho tiempo a cambiar estos archivos. Es posible que muchas de estas categorías se usen poco, si es que se usan. Es mejor idea lanzar el chatbot en fase de prueba e ir corrigiendo las conversaciones reales mediante targeting.

Una vez hechas las primeras pruebas, podemos descargar los archivos de Pandorabots, si es necesario, y subirlos a Program O