INTELIGENCIA ARTIFICIAL

E.T.S. de Ingenierías Informática y de Telecomunicación

Práctica 1

Agentes Conversacionales

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD DE GRANADA
Curso 2017-2018

1. Objetivo

El objetivo de la primera práctica de la asignatura es familiarizarse con el uso del lenguaje AIML. Ese lenguaje ha sido especialmente diseñado para el desarrollo de bases de conocimiento para la construcción de agentes conversacionales.

Un agente conversacional es un agente software, diseñado con el objetivo de mejorar la comunicación entre los seres humanos y las máquinas, haciendo que estas últimas sean capaces de manejar conceptos del lenguaje natural (palabras, frases) como símbolos y reglas que actúan sobre estos símbolos.

Los agentes conversacionales tienen multitud de aplicaciones, siendo su principal labor en estos últimos años la de asistente de los seres humanos. Es muy conocido **SIRI**, el agente conversacional incluido en los dispositivos desarrollados por la empresa ***Apple***. La relevancia de este tipo de agentes se pone de manifiesto en el creciente número de dispositivos y páginas de empresas que los incluyen, como por ejemplo, **GOOGLE NOW** una aplicación de características similares a SIRI desarrollada por ***Google*** para dispositivos Android, o **CORTANA** la aplicación desarrollada por ***Microsoft***, o los asistentes incluidos en las páginas web de empresas muy importantes como ***Ikea***. La propia ***Universidad de Granada*** tiene su asistente llamado **Elvira** que te ayuda a encontrar información relativa a la Universidad.

AIML no es el único lenguaje definido para el desarrollo de agentes conversacionales, ya que podemos encontrar otros como *API.AI* usado para el desarrollo de **Google Assistant** (una aplicación Android que incorpora una gran flexibilidad debida a su capacidad para aprender), pero sí que es el más estándar (aunque no lo es completamente) y del que es más fácil encontrar información.

En el primer seminario de la asignatura se ha impartido un tutorial sobre el uso de este lenguaje, indicando cuáles son las sentencias más habituales e ilustrando su uso con algunos ejemplos y ejercicios. El **objetivo de esta primera práctica** consiste en definir bases de conocimiento en AIML para establecer algunas conversaciones sobre temas concretos, usando ***program-ab***, un intérprete para el lenguaje ***AIML 2.0*** (una de las últimas versiones del lenguaje) de código abierto y que se puede descargar desde la página web de la asignatura (acceso identificado de ***decsai.ugr.es***).

2. Detalles de la práctica

Como se ha indicado anteriormente, la práctica está orientada al estudio del lenguaje AIML y a mostrar sus posibilidades como medio para desarrollar agentes conversacionales. En concreto, se pide construir un agente conversacional que sea capaz de seguir una conversación sobre dos temas: (a) responder preguntas sobre la asignatura Inteligencia Artificial, y (b) desarrollar un juego que llamaremos "¿En qué fruta estoy pensando?", que consiste en adivinar la fruta en la que alguien ha pensado en base a una serie de pistas que se van dando. A continuación detallamos más cada uno de estos dos temas.

2.1. Asistente de la asignatura Inteligencia Artificial

El objetivo de esta parte es describir el conocimiento necesario usando el lenguaje AIML para que un agente conversacional sea capaz de responder a las preguntas más habituales sobre aspectos del seguimiento y evaluación de la asignatura Inteligencia Artificial. El agente desarrollado por el alumno deberá poder dar respuesta a preguntas tales como: "¿Cuánto cuenta la parte teórica sobre la nota global de la asignatura?", "¿Cuál es la calificación de cada práctica?", "¿Tengo que presentarme a todas las prácticas?", "¿Cómo cuenta la asistencia para la evaluación?", "¿Qué se considera participar en clase de prácticas?",…

La información necesaria para responder a estas preguntas se encuentra disponible en el método de evaluación de la asignatura (guía docente), así como en la información adicional que proporcionan los profesores tanto en clase de teoría como en clase de prácticas. No sólo se pide que sepa responder a la pregunta en su formulación original, sino que sea capaz de responder correctamente variantes de la misma pregunta. Es decir, si la pregunta original es "¿Cómo cuenta la asistencia para la evaluación?", debe también saber responder a otras dos variantes qué vosotros propongáis, como por ejemplo "¿Qué valor tiene la asistencia en la evaluación?", "¿Cómo se considera la asistencia en la evaluación?".

2.2. Juego "¿En qué fruta estoy pensando?"

"¿En qué fruta estoy pensando?" es un juego bipersonal simple que consiste en que uno de los participantes debe adivinar la fruta en la que está pensado el otro participante (dentro de un conjunto fijo de frutas) a través de un conjunto de pistas que el primero va obteniendo haciendo preguntas al segundo. Las preguntas deben versar sobre el color, tamaño y sabor de la fruta, así como de su aporte en las vitaminas A, B y C.

La lista de frutas junto con sus propiedades se presentan en la siguiente tabla:

	Color	Tamaño	Sabor	Vitamina A	Vitamina B	Vitamina C
Fresa	Rojo	Pequeño	Semiácido	No	Si	Si
Manzana	Amarillo, Rojo, Verde	Mediano	Semiácido	No	Si	Si
Naranja	Naranja	Mediano	Ácido	No	Si	Si
Kiwi	Verde, Amarillo	Pequeño	Ácido	No	No	Si
Mandarina	Naranja	Pequeño	Ácido	Si	No	No
Piña	Amarillo, Verde	Grande	Ácido	No	Si	Si
Mango	Verde	Grande	Semiácido	Si	Si	Si
Melocotón	Amarillo, Naranja	Mediano	Semiácido	Si	Si	No
Higo	Verde, Morado	Pequeño	Dulce	No	Si	Si
Ciruela	Amarillo, Morado	Pequeño	Dulce	Si	Si	Si
Pera	Verde, Amarillo	Mediano	Dulce	No	Si	Si
Plátano	Amarillo	Mediano	Dulce	No	Si	Si
Coco	Marrón	Grande	Neutro	No	Si	Si
Aguacate	Verde	Mediano	Neutro	No	Si	Si
Nuez	Marrón	Pequeño	Neutro	No	Si	No
Cereza	Rojo	Pequeño	Semiácido	Si	No	Si

Se destaca que el color de la fruta indicado en la tabla anterior debe considerarse como posibilidad. Es decir, para melocotón, el color {“Amarillo”, “Naranja”} indica que melocotón puede ser amarillo o puede ser naranja.

En el juego entre humanos, cada jugador adopta los dos papeles, el de pensar una fruta que el otro debe acertar y a la vez intentar acertar la fruta que el adversario ha pensado, haciéndose preguntas uno a otro a razón de una pregunta por turno.

En nuestro caso, vamos a considerar dos juegos distintos: (a) Uno en el que el humano trata de acertar la fruta pensada por la máquina (y la máquina sólo se dedica a responder preguntas), y (b) Un segundo juego independiente al anterior, donde la máquina es la que hace las preguntas para tratar de adivinar, a través de las respuestas del humano, la fruta que ha pensado. Este segundo juego requiere más habilidad para manejar el conocimiento que el primero, y dicha complejidad puede ser graduada en función de cómo de inteligente deseamos que sea el agente jugando. Por esa razón hemos considerado, a su vez, dos posibilidades de resolución:

- En la primera, menos compleja, se pide que el agente vaya haciendo preguntas y que, en algún momento, termine dando una respuesta sobre la fruta pensada por el humano.
- En una segunda, más compleja, se pide que el agente no realice las preguntas sin orden establecido, sino que siga una estrategia que implique que, a cada pregunta que haga, al menos se descarte una de las frutas entre las posibles hasta ese momento. Además, el sistema debe detectar cuándo el jugador humano trata de engañarlo, en caso de que esto pudiese ocurrir (por ejemplo, cuando el humano responde erróneamente a preguntas para hacer que su adversario no gane la partida).

A continuación describimos brevemente algunas de las peculiaridades de lo que se pide en cada versión del juego.

2.2.1. El humano trata de acertar la fruta pensada por el agente

En esta versión del juego, el humano hace las preguntas para tratar de adivinar la fruta, y el agente está encargado de responder a esas preguntas. Las preguntas que realiza el humano tendrán una estructura semejante a la siguiente: “¿La fruta es Roja?”, “¿Es

pequeña?", "¿Tiene vitamina C?", "¿Es dulce?", de manera que la respuesta esperada por el agente sea binaria; es decir, "sí" o "no". Además, el agente debe responder preguntas que ayuden al humano a conocer las frutas (por ejemplo "¿El plátano es amarillo?", "¿Es el higo rojo?",…).

El juego termina cuando el humano introduce una afirmación del tipo "Es la naranja" o "La fruta en la que estabas pensando es el aguacate".

2.2.2. El agente trata de acertar la fruta pensada por el humano

En esta segunda modalidad del juego, es el agente conversacional el que hace las preguntas y el usuario humano el que las responde de forma afirmativa o negativa, en función de que la fruta en el que él ha pensado tenga o no el rasgo en cuestión. Además, esta modalidad implica haber completado la anterior, por lo que resulta necesario introducir un trozo de conversación relativo para preguntar al humano qué papel va a jugar en el juego (si hace las preguntas o si las responde). Sólo entraríamos en este modo de juego en el caso en el que responda que él decide la fruta y el ordenador debe adivinarlo. El formato de las preguntas es semejante a las del apartado anterior, con la diferencia de que ahora es el agente el que debe formularlas en base a las preguntas previamente realizadas. Es decir, el agente no debe preguntar dos veces "¿Tiene los ojos claros?", de manera que debe recordar sobre qué cosas ya preguntó anteriormente.

El juego termina cuando el agente devuelve en la conversación una afirmación que incluye a una de las frutas. Debe esperar que el usuario le conteste si ha acertado o no y reaccionar ante esa respuesta.

2.2.3. El agente resuelve de forma inteligente la fruta pensado por el humano.

Esta variante del juego es una mejora de la modalidad anterior, con mejoras en las capacidades comunicativas y de razonamiento, adoptando un comportamiento más cercano al humano y, por consiguiente, más inteligente.

¿En qué se ha de notar esa inteligencia? Fundamentalmente, en dos aspectos:

- El primero de ellos, en el que la secuencia de preguntas que realiza el agente sigue una estrategia para intentar descubrir la fruta con el menor número de preguntas posible.
- En el segundo, el agente debe ser capaz de detectar si el humano le está engañando.

En relación al primero, no vamos a pedir que en cada turno se haga la mejor pregunta posible bajo algún criterio objetivo basado en una medida de información, pero sí que la respuesta a la pregunta sea útil; es decir, cualquiera que sea la respuesta que el usuario dé, el conjunto de frutas posibles se reduce.

En relación al engaño que puede tratar de hacer el humano, éste se detecta bien porque durante la secuencia del juego el número de frutas que satisfacen los rasgos conocidos hasta el momento es 0, o bien cuando la respuesta dada por el agente no coincide con la fruta pensada por el humano. En este último caso, el agente debe argumentar la razón por la cual no es posible que la solución sea la que da el humano.

3. Evaluación

3.1 Definición de los niveles de dificultad

Hemos diseñado un modelo de evaluación para que el alumno decida con qué intensidad y a qué nivel desea implicarse en su elaboración. Obviamente, a mayor nivel de implicación, el alumno opta a una mayor calificación en la práctica. Por tanto hemos definido 4 niveles en la entrega de esta primera práctica que son, de menor a mayor implicación, los siguientes:

(a) **Nivel 0**: Entrega del conocimiento necesario para responder a las preguntas sobre la asignatura Inteligencia Artificial. En este caso, el alumno puede optar hasta cuatro puntos sobre diez.

(b) **Nivel 1**: El apartado (a) más la parte del juego de "¿En qué fruta estoy pensando?" en el que el humano debe acertar la fruta pensada por el agente conversacional. En este caso, el alumno pude optar hasta una calificación de seis puntos sobre diez.

(c) **Nivel 2**: El apartado (b) más la parte del juego de "¿En qué fruta estoy pensando?" en el que el agente conversacional trata de adivinar la fruta pensada por el ser humano. Para esta opción, el alumno puede obtener hasta una calificación de ocho sobre diez.

(d) **Nivel 3**: El apartado (b) más la parte del juego de "¿En qué fruta estoy pensando?" en el que el agente conversacional trata de acertar usando una estrategia inteligente la fruta pensada por el humano, siendo capaz de detectar si el humano lo intenta engañar. Este es el nivel de mayor dificultad e implicación y el alumno opta a la valoración de diez sobre diez.

No se considerarán aquellas entregas que no puedan ubicarse en uno de estos niveles; es decir, estas son las únicas posibilidades de entrega.

Un ejemplo de entrega inválida, es aquella donde el alumno entrega el conocimiento para resolver el juego "¿En qué fruta estoy pensando?" en el que el agente piensa la fruta, pero no se entrega el que responde a las cuestiones sobre la asignatura de Inteligencia Artificial, ya que esta última es obligatoria en todos los niveles.

3.2 ¿Cómo se evaluará cada uno de los problemas?

Adjunto a este guión, se establece una fecha límite para la entrega de la práctica. Antes de dicha fecha, el alumno debe subir el material que más adelante se indica conteniendo el conocimiento necesario para resolver los problemas hasta el nivel que ha decidido. En una segunda fecha posterior, que también se indica en este guión y que corresponderá con un día del horario habitual del grupo de prácticas se realizará la defensa de la práctica.

Hemos definido una forma de defensa para cada uno de los niveles definidos en la entrega. A continuación se indica cómo se evaluará cada nivel:

Nivel 0: En el anexo 1 de este guión se proponen 30 preguntas relativas a la asignatura Inteligencia Artificial. El alumno debe definir el conocimiento necesario

para responder correctamente a cada una de ellas. Además, se han definido otras preguntas que proponen respuestas semejantes a las que se proporcionan en las propuestas, pero que no están a disposición del alumno (básicamente, otras formas de preguntar lo mismo). Durante el proceso de defensa, teniendo en ejecución su agente conversacional, el alumno introducirá 2 preguntas que le indicará el profesor entre las 30 propuestas y otras 2 preguntas de entre las no conocidas. Para poder optar a tener puntuación en este nivel, el agente deberá responder correctamente a las 2 primeras preguntas, y superará este nivel si responde correctamente al menos a una de las no conocidas. En el caso de no responder correctamente ninguna de las 2 últimas preguntas, el profesor le pedirá al alumno que incorpore a su conocimiento una de esas 2 preguntas y se verificará que lo ha hecho correctamente. En este mismo proceso, se verificará las otras opciones el estudiante había planteado para una de estas preguntas. Si lo hace correctamente, superará este nivel y obtendrá una calificación de 4. Si no lo sabe hacer, su calificación será de 2 puntos.

En cualquier caso, el profesor siempre tiene la potestad de pedir al alumno que introduzca nuevo conocimiento o que le explique alguna parte de su código para asegurarse de que supera este nivel.

Nivel 1: Para llegar a este nivel en la defensa, debe haberse superado el nivel anterior. En este caso, la evaluación es simple: Se pedirá al alumno que juegue una partida siguiendo las indicaciones que le propone el profesor. Si el resultado es coherente con las preguntas formuladas, el alumno superará este nivel y añadirá

- 1 punto más a su calificación, si sólo ha considerado en el atributo “color” el primero de los colores. **Su calificación final será de 5 puntos y no podrá optar a los siguientes niveles.**
- 2 puntos más a su calificación, si ha considerado todos los colores en el atributo “color”. Su calificación será de 6 puntos y tiene la posibilidad de elegir uno de los dos siguientes niveles.

En caso de una resolución del juego no satisfactoria, la defensa termina y la calificación del alumno será de 4 puntos.

Nivel 2: Al igual que antes, debe haberse superado el nivel 1 con una puntuación de 6 puntos para pasar a evaluar este nivel, y del mismo modo la forma de evaluarlo es jugar una partida donde el profesor es el que pensará la fruta y debe ser el agente conversacional el que lo acierte. Si el agente mantiene la conversación y la secuencia de preguntas hasta llegar a la solución correcta, el alumno superará este nivel, y añadirá 2 puntos a su calificación. Si no es así, la defensa habrá terminado y la calificación del alumno será de 6 puntos.

Nivel 3: Para acceder a este nivel, el agente presentado para su evaluación ha debido superar el nivel 1 con una puntuación de 6 puntos y el alumno debe indicar al profesor que opta a este nivel frente al nivel 2. Es importante esta decisión por parte del alumno, ya que si no se supera el nivel 3, el alumno obtendrá la calificación de un 6 (menos calificación que superar el Nivel 2). Por el contrario, si se supera totalmente este nivel se puede obtener hasta un 10 en la calificación. Para evaluar este nivel, se jugará 2 veces con el agente. Por consiguiente, al terminar una partida, este debe volver a preguntarte si quieres volver a jugar. La primera partida será una partida normal, dónde el profesor pensará una fruta, el agente irá proponiendo preguntas y el alumno justificará qué ha hecho para asegurarse que todas las preguntas permiten descartar al menos a una fruta. En una segunda partida, se intentará engañar al agente y éste lo debe detectar. Si no lo detecta, el nivel no está superado y se le otorgará al alumno la calificación de 6. Si lo supera, la calificación del alumno oscilará entre 8 y 10 en función de los elementos introducidos en el conocimiento para detectar las contradicciones y asegurar que las preguntas siempre permiten reducir el número de frutas. Se tendrá muy en cuenta para optar a la más alta calificación, el diseño realizado por el estudiante, en el sentido que el juego sea fácilmente ampliable tanto en el número de nuevas frutas que se puedan incorporar (melón, sandía, ...) como en el número de nuevas preguntas que se pudieran incluir (tales como, “mineral principal que aporta” o “de dónde es originaria” la fruta).

3.3. ¿Qué hay que entregar?

Antes de que termine el plazo de entrega fijado en este guión, el alumno deberá subir a la plataforma de la asignatura en decsai.ugr.es, un archivo comprimido con zip llamado "practica1.zip". Este archivo debe tener la misma estructura en carpetas que cuelga de la carpeta "mybot", donde obviamente las carpetas "aiml", "aimlf", "sets" y "maps" contienen los ficheros necesarios para resolver la práctica al nivel que ha decidido el alumno.

3.4. Observaciones Finales

Esta **práctica es INDIVIDUAL** y trata de establecer la capacidad del alumno para desarrollar una base de conocimiento usando el lenguaje AIML. El profesorado para asegurar la originalidad de cada una de las entregas, someterá a estas a un procedimiento de detección de copias.

En el caso de detectar prácticas copiadas, los involucrados (tanto el que se copió como el que se ha dejado copiar) tendrán suspensa la asignatura. Por esta razón, recomendamos que en ningún caso se intercambie código entre los alumnos. No servirá como justificación del parecido entre dos prácticas el argumento *"es que la hemos hecho juntos y por eso son tan parecidas"*, o "como estudiamos juntos, pues...", ya que como se ha dicho antes, **las prácticas son INDIVIDUALES**.

Como se ha comentado previamente, el objetivo de la defensa de prácticas es evaluar la capacidad del alumno para enfrentarse a este problema. Por consiguiente, se asume que todo el código que aparece en su práctica ha sido introducido por él por alguna razón y que dicho alumno domina perfectamente el código que entrega. Así, si durante cualquier momento del proceso de defensa el alumno no justifica adecuadamente algo de lo que aparece en su código, la práctica se considerará copiada y tendrá suspensa la asignatura. Por esta razón, aconsejamos que el alumno no incluya nada en su código que no sea capaz de explicar qué misión cumple dentro de su práctica y que revise el código con anterioridad a la defensa de prácticas.

Por último, las prácticas presentadas en tiempo y forma, pero no defendidas por el alumno, se considerarán como no entregadas y el alumno obtendrá la calificación de 0. El supuesto

anterior se aplica a aquellas prácticas no involucradas en un proceso de copia. En este último caso, el alumno tendrá suspensa la asignatura.

4. Desarrollo temporal de la práctica

La práctica primera seguirá el siguiente desarrollo temporal:

a) **Del 21 de febrero al 23 de febrero**: Presentación de la práctica.

b) **Semanas del 26 de Febrero:** Desarrollo de la práctica en clase.

c) **Semana del 2 de Abril**: Defensa de la práctica que se realizará en el día y hora de la sesión de prácticas que le corresponde habitualmente al alumno. Si por algún motivo, el alumno no pudiera asistir a su sesión esa semana, debe avisar al profesor de prácticas para que le asigne a otro grupo de prácticas de la asignatura para esa misma semana.

d) **La fecha tope para la entrega** será **el Domingo 18 de Marzo antes de las 23:00 horas.**

NOTA: La semana del 5 de Marzo se dedicará a resolver los ejercicios de la relación de problemas sobre agentes reactivos, por tanto, las sesiones de dicha semana no estarán dedicadas a resolver esta práctica. La semana del 6 de Marzo es Semana Santa y no hay clase.

ANEXO 1

Las 30 preguntas sobre la asignatura de Inteligencia Artificial

(Aclaración: Se eliminan los símbolos de interrogación y las tildes para evitar problemas de codificación de caracteres en cada ordenador)

1. COMO CONSIGO SUPERAR LA PARTE PRACTICA
2. COMO SE EVALUA EN LA CONVOCATORIA ORDINARIA
3. HAY EXAMEN DE PRACTICAS EN LA CONVOCATORIA EXTRAORDINARIA
4. QUE PASA SI NO PUEDO ASISTIR A CLASE REGULARMENTE
5. HAY NOTA MINIMA PARA APROBAR
6. LAS PRACTICAS SON OBLIGATORIAS
7. CUANTO VALE CADA PRACTICA CON RESPECTO A LA NOTA FINAL
8. CUANTO VALE LA PARTICIPACION CON RESPECTO A LA NOTA FINAL
9. COMO SE CONSIGUEN PUNTOS POR PARTICIPACION
10. LOS TRABAJOS SON INDIVIDUALES
11. SE PUEDEN HACER LOS TRABAJOS EN GRUPO
12. HAY QUE ENTREGAR LOS EJERCICIOS DE LAS RELACIONES DE PROBLEMAS
13. CUANTO VALE EL PRIMER EXAMEN DE TEORIA
14. PUEDO RECUPERAR UNA PRACTICA O EXAMEN
15. QUE PASA SI ME COPIO DE OTRO ESTUDIANTE
16. QUE PASA SI ME DEJO COPIAR DE OTRO ESTUDIANTE
17. Y SI SUSPENDO EN LA CONVOCATORIA ORDINARIA

18. CUANTO VALE LA PRACTICA Y LA TEORIA
19. CUAL ES EL HORARIO DE TUTORIAS DEL PROFESOR DE PRACTICAS
20. SE PUEDE APROBAR LA PARTE PRACTICA DE LA ASIGNATURA SI NO PRESENTO UNA PRACTICA
21. CUAL ES LA NOTA MINIMA QUE TENGO QUE OBTENER EN CADA PARTE PARA HACER MEDIA
22. QUE PUNTUACION OBTENGO EN LA PRACTICA 1 SI OPTO POR DEFENDER EL NIVEL 3 Y NO LO SUPERO
23. PUEDO ENTREGAR LA PRACTICA 1 SIN HACER LA TAREA RELATIVA AL NIVEL 0
24. QUE TENGO QUE HACER SI NO PUEDO ASISTIR A LA SESION DE PRACTICAS EN LA QUE ME CORRESPONDE HACER LA DEFENSA DE MI PRACTICA 1
25. CUANTAS PRUEBAS TEORICAS TENDRE A LO LARGO DEL CURSO
26. QUE VALE LA PRIMERA PRUEBA TEORICA
27. EN QUE CONSISTE EL EXAMEN DE EJERCICIOS
28. QUE DEBERÍA HACER SI NO ESTOY SEGURO DE PODER SUPERAR EL NIVEL 3 DE LA PRÁCTICA 1
29. YO PROGRAMO MUY BIEN EN C++, PODRIA HACER LA PRACTICA 1 EN ESE LENGUAJE
30. ME ENCANTAN LOS AGENTES CONVERSACIONALES Y SOY UN EXPERTO EN SU CONSTRUCCIÓN PODRIA USAR UN AGENTE CONVERSACIONAL PARA QUE VAYAN EN MI LUGAR A HACER LA DEFENSA DE PRACTICAS

ANEXO 2

Durante la presentación de la práctica se ha hecho una descripción de las componentes más importantes del lenguaje AIML 2.0. En este anexo, se describen algunas funcionalidades adicionales que se han incluido en este lenguaje para aumentar su expresividad.

Extensión del lenguaje AIML

El lenguaje AIML 2.0 tiene algunas restricciones a la hora de poder trabajar con conjuntos y diccionarios, así también para poder utilizar patrones <pattern> con expresiones regulares. Es por ello que se ha desarrollado en un Trabajo Fin de Grado (realizado por Benjamín Vega Herrera) una versión extendida que contempla nuevas características que hacen más llevadera la programación en este lenguaje.

A2.1 Conjuntos en AIML

Los conjuntos en AIML 2.0 se acceden mediante la etiqueta <set>, pero solamente se pueden utilizar dentro del <pattern> y no en el <template>, ya que se confundiría con la etiqueta <set> para guardar valores en variables. Para darle mayor flexibilidad se ha desarrollado una versión extendida del lenguaje que permite:

- Cargar en una variable los valores de un conjunto:

```
<set var="valores"><readset>telefono</readset></set>
```

- Guardar en un conjunto un nuevo valor:

```
<addtoset>
  <name>datos</name>
  <key>email</key>
</addtoset>
```

donde <name> hace referencia al nombre del fichero que almacena el "set", en este caso, el fichero se llama "datos.txt" y está en la carpeta "set", y <key> hace referencia al nuevo valor que se desea incluir, en este caso, "email". Es un

conjunto, por tanto, si “email” ya existiera, no se incluye este valor de nuevo en el “set”.

- Eliminar en un conjunto un valor:

```
<removefromset>
  <name>datos</name>
  <key>email</key>
</removefromset>
```

donde <name> hace referencia al nombre del fichero que almacena el “set”, en este caso, el fichero se llama “datos.txt” y está en la carpeta “set”, y <key> hace referencia al nuevo valor que se desea incluir, en este caso, “email”.

A2.2 Diccionarios en AIML

Los diccionarios en AIML 2.0 se acceden mediante la etiqueta <map>. Esta etiqueta puede utilizarse solamente dentro del <template>. Por ejemplo, puedo intentar guardar en una variable local llamada “eltel" el valor asociado al diccionario *telefono* utilizando *Sanchez* como clave de la siguiente manera (las dos formas son equivalentes):

```
<set var=“eltel”><map name=“telefono”>Sanchez</map></set>
<set var=“eltel”><map><name>telefono</name>Sanchez</map></set>
```
[1]

Lamentablemente los diccionarios en AIML 2.0 son fijos, es decir, no se pueden modificar desde dentro del AIML. Es por ello que se ha desarrollado una versión extendida del lenguaje que permite:

- Cargar en una variable las claves de un diccionario (así luego puedo averiguar si existe una cierta clave en un “map”):

```
<set var=“claves”><readkeys>telefono</readkeys></set>
```

1 Esta segunda versión resulta útil cuando el nombre del diccionario se encuentra almacenado en una variable.

- Guardar en un diccionario un nuevo par (clave, valor):

```
<addtomap>
  <name>telefono</name>
  <key>Perez</key>
  <value>111222333</value>
</addtomap>
```

  donde <name> hace referencia al nombre del fichero que almacena el "map", en este caso, el fichero se llama "telefono.txt" y está en la carpeta "map", <key> hace referencia a la nueva clave que se desea incluir, en este caso, "Perez", y <value> es el valor que se asocia con esa clave. Si la clave ya existe, esta operación no modifica el "map".

- Modificar el valor asociado a una clave de un diccionario:

```
<modifymap>
  <name>telefono</name>
  <key>Sanchez</key>
  <value>111222333</value>
</modifymap>
```

- Borrar del diccionario una par (clave, valor):

```
<removefrommap>
  <name>telefono</name>
  <key>Sanchez</key>
</removefrommap>
```

  en este último caso, no es necesario indicar el campo <value>.

A2.3 Nuevas opciones para <pattern>

Se ha agregado la posibilidad de utilizar algunas expresiones regulares más comunes dentro de la etiqueta <pattern>. A continuación se listan y ejemplifican cada una de ellas:

- **Palabras opcionales**: Muchas veces nos interesa poder permitir la aparición de una palabra opcional en una expresión regular (en otros lenguajes se utiliza el símbolo "?" seguido de lo que se quiere hacer opcional). En AIML se ha optado por incluir la palabra opcional entre paréntesis. Por ejemplo, el patrón

```
<pattern>tengo (muchas) posibilidades</pattern>
```

se enlazará correctamente con las frases: "tengo posibilidades" y con "tengo muchas posiblidades". Incluso se pueden utilizar varias veces en una misma frase:

```
<pattern>un numero al azar (uno) (dos) (tres)</pattern>
```

se enlazará correctamente con "un numero al azar", "un numero al azar uno", "un numero al azar uno dos", "un numero al azar uno dos tres", "un numero al azar uno tres" y "un numero al azar dos tres".

- **Subconjunto de palabras**: En algunos casos el uso del "*" es demasiado amplio, y nos gustaría poder restringirlo a un conjunto más pequeño. En AIML se ha optado por utilizar corchetes para indicar esto, de forma similar a otros lenguajes. Por ejemplo, si solo quiero permitir que puedan aparecer dos palabras (si o no) escribiría el siguiente patrón:

```
<pattern>el me dijo que [si no] queria</pattern>
```

esto permite que se enlace correctamente solo dos posibles frases: "el me dijo que si quería" o bien "el me dijo que no quería".

- **Prefijos y sufijos**: Muchas veces nos resulta útil poder colocar la raíz de un verbo e indicar que todas las posibles conjugaciones también se redirijan a la misma porción de código. Para ello se ha incluido la posibilidad de utilizar sufijos mediante el carácter “+”.

```
<pattern>^ guard+ ^</pattern>
<template>Guardando dato</template>
```

De esta manera se puede llegar al trozo de código asociado a ese patrón cuando el usuario ingrese diferentes frases, ej: “por favor guardame este dato”, “no te olvides de guardar este dato”, “ya me estas guardando este dato”, etc. De forma similar se pueden utilizar prefijos anteponiendo el “+” a la palabra deseada.

```
<pattern>^ +ndo ^</pattern>
<template>Arreando que es gerundio</template>
```

ANEXO 3

Descripción del fichero “utilidades.aiml”

Dentro del software entregado para la realización de la práctica 1, se incluye un fichero llamado “utilidades.aiml” que incorpora algunas reglas genéricas que permitirán al alumno realizar con mayor comodidad su práctica.

En este anexo se describen las reglas incluidas en este fichero con algunos ejemplos para ilustrar su uso.

Las utilidades son las siguientes:

1. Operaciones para manejar una lista de nombres
2. Generar un número aleatorio en un rango
3. Otra forma de hacer un ciclo y Comparar dos cadenas

Ahora describiremos cada una de ellas.

A3.1. Operaciones para manejar lista de palabras

En AIML se trabaja exclusivamente con cadenas de caracteres y con procesamiento simbólico. Así, cada variable global o local que se usa es siempre de tipo cadena de caracteres. Por esa razón, es importante tener definidas funcionalidades dentro del lenguaje que faciliten el trabajo con este tipo de dato.

En el fichero “utilidades.aiml” hemos incluido algunas de las operaciones más frecuentes y que pueden tener uso para el desarrollo de los problemas que se piden en esta práctica. En concreto, se han incluido las siguientes operaciones:

- **TOP**: Dada una lista de palabras, devuelve la primera palabra de esa lista.
- **REMAIN**: Dada una lista de palabras, devuelve la misma lista de palabras quitando la primera palabra.
- **COUNT**: Dada una lista de palabras, devuelve el número de palabras que contiene.

- **FINDITEM [palabra] IN [listaPalabras]:** Determina si "palabra" es una palabra incluida en "listaPalabras".
- **SELECITEM [number] IN [listaPalabras]:** Devuelve la palabra que ocupa la posición "number" en "listaPalabras".
- **REMOVEITEM [number] IN [listaPalabras]:** Elimina de "listaPalabras" la palabra de posición "number".
- **ADDITEM [palabra] IN [listaPalabras]:** Añade "palabra" a principio "listaPalabras", sólo si "palabra" no estaba antes en "listaPalabras". En este segundo caso, la operación no hace nada.
- **CODE [listaPalabra]:** Transforma la lista de palabras en una sola palabra sustituyendo los espacios en blanco por "_". Si no hay espacios en blancos, esta función no cambia nada en el argumento.
- **DECODE [Palabra]:** Transforma la palabra en una lista de palabras, sustituyendo los "_" por los espacios en blanco. Si no hay "_", esta función no provoca ningún cambio.
- **DELETREA [Palabra]:** Construye una lista de palabras, donde cada palabra está formada por cada letra de la palabra que se pasa por argumento.

Este repertorio de operaciones no tiene sentido si se utiliza como salida directa del agente, sino que tiene como función procesar de una manera más elaborada la salida que se ofrecerá, y está concebido para trabajar sobre variables, ya sean globales (predicados) o locales.

Una aclaración antes de pasar a ilustrar su uso con algunos ejemplos: En las definiciones anteriores se ha hecho uso de conceptos como "palabra" y "lista de palabras" para matizar la intención con la que se han construido estas herramientas pero, en realidad, en todos los casos, los parámetros son cadenas de caracteres. Cuando se hace referencia a "palabra" se quiere indicar que es una secuencia de símbolos que está entre 2 separadores (entendiendo aquí separador por uno o varios espacios en blanco). Cuando se hace referencia a "lista de palabras", se indica que contiene una secuencia de palabras separadas por espacios en blanco.

Para ilustrar como se pueden usar, supongamos que existe una variable global llamada list_fruit que almacena una secuencia de frutas.

Ejemplo 1: Darle algunos valores a esa variable global mediante la definición de una regla.

Para esto, aprovecharemos la pregunta “¿Conoces algunas frutas?”, para darle un valor inicial a esa variable.

```
<category>
<pattern> conoces algunas frutas</pattern>
<template>
    <think>
      <set name="list_fruit">fresa cereza naranja mandarina</set>
    </think>
    si, alguna conozco.
</template>
</category>
```

Ejemplo 2: Queremos ir actualizando esa variable global con nuevas frutas que nos pueda ir proporcionando el usuario.

Vamos a construir una regla qué aprenda nuevas frutas a través de afirmaciones del usuario del tipo “la manzana es una fruta” o “el membrillo es una fruta”.

```
<category>
<pattern>la * es una fruta</pattern>
<template>
    <think>
      <set var="existe">
        <srai>FINDITEM <star/> IN <get name="list_fruit"/></srai>
      </set>
    </think>
    <condition var="existe">
      <li value="0">
        <think>
          <set name="list_fruit">
             <srai>
                ADDITEM <star/> IN <get name="list_fruit"/>
             </srai>
          </set>
       </think>
            Recordare que <star/> es una fruta.
       </li>
       <li>
            Ya sabia que <star/> es una fruta.
       </li>
    </condition>
</template>
</category>
```

El proceso de esta regla es bastante intuitivo,

1. Asigna en la variable local existe si lo que se pasa en <star/> es una de las frutas que ya conoce y se encuentran almacenadas en la variable global list_fruit. Para determinar la existencia de esa fruta hemos hecho uso de **FINDITEM**.
2. La variable existe almacena un 0, si no encuentra esa fruta en la lista y en ese caso lo que hace es añadirla al principio de la lista list_fruit, y ofrece el mensaje al interlocutor de que recordará esa fruta. Para añadir la nueva fruta hemos hecho uso de **ADDITEM**.
3. Si la variable existe almacena un valor distinto de 0, significa que en la posición almacenada por esa variable se encuentra la fruta. En este caso, no modifica la lista list_fruit, y lanza el mensaje de que ya sabía que eso era una fruta.

La regla anterior recoge el caso de un patrón del tipo *"la… es una fruta"*, pero nos gustaría recoger el caso también del patrón "el… es una fruta". La primera intención sería copiar esta regla, pegarla debajo en el editor y cambiar el "la" por un "el". Esto funcionaría, pero un experto programador en AIML se daría cuenta qué es más simple crear un nueva regla que invoque a la anterior y cambiando el nuevo patrón. El resultado sería el siguiente:

```
<category>
<pattern>el * es una fruta</pattern>
<template>
    <srai>la <star/> es una fruta</srai>
</template>
</category>
```

Como se puede observar <srai> hace un papel semejante al de la invocación de un método en un lenguaje de programación convencional, y aquí aprovechamos ese recurso.

Ejemplo 3: Ahora vamos a ampliar nuestro repertorio de reglas para permitir corregir algún error en nuestra lista de frutas. Supongamos que en un momento dado, por error, el interlocutor dijo: *"la mesa es una fruta"* y nuestro agente añadió *mesa* a la lista, pero el interlocutor poco después se da cuenta de su error y quiere corregirlo y nos dice *"fue un error, la mesa no es una fruta"*. Incluiremos una regla para permitir que esto se pueda hacer:

```
<category>
<pattern>no es una fruta la *</pattern>
<template>
    <think>
      <set var="pos">
        <srai>FINDITEM <star/> IN <get name="list_fruit"/></srai>
      </set>
    </think>
    <condition var="pos">
      <li value="0"> Como puedes pensar que considere eso una fruta
      </li>
      <li>
        <think>
          <set name="list_fruit">
           <srai>
             REMOVEITEM <get var="pos"/> IN <get name="list_fruit"/>
           </srai>
          </set>
       </think>
           Menos mal que me lo dices, yo creia que era una fruta.
        </li>
    </condition>
</template>
</category>
```

El proceso es semejante al que se ilustra en el *Ejemplo 2*, se busca si <star/> está en la lista de frutas mediante **FINDITEM** y almacenado su valor en pos. Si pos vale 0 entonces eso no se había considerado como fruta. En otro caso, el valor está dentro de la lista de frutas y hay que eliminarlo. Para eso usamos **REMOVEITEM** que elimina la palabra de posición pos de list_fruit y el resultado se reasigna a list_fruit.

De igual modo que en el ejemplo anterior, podemos incluir la regla que considera los patrones que son de la forma "… el… no es una fruta":

```
<category>
<pattern>no es una fruta el *</pattern>
<template>
    <srai> no es una fruta la <star/></srai>
</template>
</category>
```

Ejemplo 4: Ya podemos actualizar nuestra lista de frutas insertando y eliminando elementos de la misma, pero el interlocutor sólo nos puede dar la información para recordar la fruta de una en una. Sería interesante que pudiera darnos una lista de frutas que el agente fuera capaz de recordar del tipo "*la manzana, el platano, el melon y la ciruela son frutas*", donde aparecen delante de cada fruta un "el" o un "la" y antes de dar la última fruta aparece una "y". Obviamente,

deberíamos usar las reglas definidas anteriormente que nos permiten modificar la lista.

```
<category>
<pattern> * son frutas</pattern>
<template>
    <think>
      <set var="lista"><star/></set>
      <set var="item">
        <srai>TOP <get var="lista"/></srai>
      </set>

    <condition var="item">
      <li value="y">
        <set var="item">
          <srai>SELECTITEM 3 IN <get var="lista"/></srai>
        </set>
        <srai> la <get var="item"/> es una fruta </srai>
      </li>

      <li value="la">
        <set var="lista">
          <srai>REMAIN <get var="lista"/></srai>
        </set>
        <set var="item">
          <srai>TOP <get var="lista"/></srai>
        </set>
        <loop/>
      </li>

      <li value="el">
        <set var="lista">
          <srai>REMAIN <get var="lista"/></srai>
        </set>
        <set var="item">
          <srai>TOP <get var="lista"/></srai>
        </set>
        <loop/>
      </li>

      <li>
        <srai> la <get var="item"/> es una fruta </srai>
        <set var="lista">
          <srai>REMAIN <get var="lista"/></srai>
        </set>
        <set var="item">
          <srai>TOP <get var="lista"/></srai>
        </set>
        <loop/>
```

```
      </li>
    </condition>
  </think>
  Recordare todas estas frutas
</template>
</category>
```

Vamos poco a poco:

1. La primero acción almacena el contenido de la parte variable del patrón en la variable local lista.
2. Se extrae la primera palabra de lista y se almacena en la variable ítem.
3. Ahora se plantea una estructura condicional en función del valor de ítem, que consideramos que puede tomar 4 valores diferentes: “y”, “la”, “el” u otro valor. En el caso de tomar el valor:
    a. “y”, entonces estamos justo antes de terminar la secuencia de palabras. La siguiente palabra a “y” es un artículo que podemos ignorar y la siguiente es el nombre de la fruta. Por esa razón, usamos **SELECTITEM** para tomarla tercera palabra de la lista, que almacenamos en ítem, para después invocar a la regla que es de la forma *“la… es una fruta”*, dónde… es el valor de ítem. Esta es la última fruta de la lista, por tanto, se sale del ciclo. (por eso, a diferencia del resto de los casos, no termina con un <loop/>.
    b. “la” o “el”, para las dos situaciones tengo que hacer lo mismo. En este caso, ignorar el artículo y pasar a evaluar la siguiente palabra que es la que contiene el nombre de la fruta. Esta operación se hace con la conjunción de **TOP** que extrae el primero de lo que queda en la lista y lo almacena en ítem, y **REMAIN** que devuelve la lista menos el primero, que se vuelve a almacenar en lista. Como en este caso la secuencia de palabras no ha terminado, se tiene que ciclar, por eso aparece <loop/>.
    c. el caso por defecto. Si no es una “y”, ni un “el”, ni un “la”, lo que tiene en ítem es el nombre de una fruta. En este caso, se invoca al igual que en el apartado (a) a la regla *“la… es una fruta”* que la añadirá si no existe en la lista, y pasa a la siguiente palabra de la misma forma que se indica en el apartado (b), es decir, con la conjunción de **TOP** y **REMAIN** y al igual que antes, quedan palabras por procesar, por tanto se cicla con <loop/>.
4. Terminado el ciclo el procesado de la cadena ha terminado y propone al interlocutor una frase en el sentido de que recordara las frutas.

En estos cuatro ejemplos hemos mostrado el uso de todas y cada una de las operaciones para el manejo de listas de palabras.

Las tres últimas operaciones son CODE, DECODE y DELETREA. Aquí pondremos un ejemplo de uso de cada una de ellas. Supongamos que hay frutas con nombre compuesto, por ejemplo, "nuez de macadamia" y queremos insertarla en nuestra lista de frutas. Si la insertamos tal cual, en los procesos descritos anteriormente de contrar, seleccionar, eliminar, etc. esta nueva única fruta va a ser considerada como si fueran tres "nuez", "de", "macadamia". Para resolver esto, podemos usar las funciones CODE y DECODE. La primera de ella, aplicada a la fruta anterior

```
<set var = "item" > </srai>
  <srai> CODE nuez de macadamia</srai>
</set>
```

nos devuelve en la variable local "item" el valor "*nuez_de_macadamia*" como una única palabra.

La operación contraria es DECODE que dada una palabra que representa una lista de palabras pero separadas en lugar de espacios en blanco por caracteres "_" la trasforma en una verdadera lista de palabras. Así,

```
<set var = "item" > </srai>
  <srai> DECODE nuez_de_macadamia</srai>
</set>
```

devuelve en la variable "item" la cadena "*nuez de macadamia*".

Por último, DELETREA como su nombre indica, transforma una palabra o lista de palabras en una lista de palabras formada por las letras que componen la palabra pasada como argumento.

El resultado de

```
<set var = "item" > </srai>
  <srai> DELETREA nuez de macadamia</srai>
</set>
```

es que la variable "item" contiene la secuencia "*n u e z d e m a c a d a m i a*".

A3.2. RANDOM

La sintaxis de esta acción es **RANDOM [number]** y devuelve un número aleatorio entre 1 y number. Aquí mostramos un ejemplo de uso de RANDOM.

Ejemplo 5: Se plantea una regla simple que ante la entrada de "Dime una fruta", el agente conversacional devuelve aleatoriamente una de entre la lista de frutas que tiene almacenadas en la variable list_fruit.

La descripción de dicha regla sería la siguiente:

```
<category>
<pattern> Dime una fruta </pattern>
<template>
   <think>
     <set var="lista"> <get var="list_fruit"/> </set>
     <set var="cantidad"><srai>COUNT <get var="lista"/></srai></set>
     <set var="pos"><srai>RANDOM <get var="cantidad"/></srai></set>
     <set var="elegida">
          <srai>
            SELECTITEM <get var="pos"/> IN <get var="lista"/>
          </srai>
     </set>
   </think>
   <get var="elegida"/>
</template>
</category>
```

El proceso de respuesta a la pregunta sería el siguiente:

1. Asigna a la variable lista una secuencia de nombres de frutas.
2. Mediante **COUNT** determina el número de frutas introducidas en lista y se la asigna a la variable cantidad.
3. A partir de cantidad, qué indica el número de frutas elegibles, usa **RANDOM** para generar un número aleatorio entre 1 y cantidad que se almacena en pos.
4. Selecciona la palabra que ocupa la posición pos de lista y la almacena en elegida usando **SELECTITEM.**
5. Devuelve como respuesta el valor de la variable elegida.

A3.3. ITERATE / NEXT y COMPARE

Con el par ITERATE/NEXT tratamos de hacer una versión más convencional de un ciclo que itera sobre una lista de palabras. El proceso es simple: ITERATE se usa sólo una vez al principio del ciclo, y permite situarse sobre la primera palabra de la lista de palabras. El resto del proceso está guiado por NEXT, que devuelve el siguiente valor de la lista. Tanto ITERATE como NEXT devuelve la cadena "end" cuando se termina de recorren la lista de palabras.

Ejemplo 6: Construir una regla que devuelva todas las frutas que conoce el agente.

En principio, esta regla sería tan simple como devolver el valor de la variable list_fruit, pero nosotros vamos a complicarlo un poco para usar estas acciones. Una posible implementación es la siguiente:

```
<category>
<pattern> Dime todas las frutas que conoces </pattern>
<template>
   Estas son las frutas que conozco
   <think>
     <set var="item">
        <srai> ITERATE <get name="list_fruit"/> </srai>
     </set>
   </think>
     <condition var="item">
        <li value="end"></li>
        <li> <get var="item"/>
              <think>
                <set var="item">
                  <srai>NEXT</srai>
                </set>
              </think>
              <loop/>
        </li>
     </condition>
</template>
</category>
```

Creemos que este último ejemplo es fácil de seguir, si se ha entendido todo lo que se ha descrito anteriormente en este Anexo 3, así que dejaremos que lo intentéis por vosotros mismos.