INTELIGENCIA ARTIFICIAL

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

Práctica 1
ANEXO 3

Agentes Conversacionales

Descripción del fichero “utilidades.aiml”

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
UNIVERSIDAD DE GRANADA
Curso 2019-2020

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.

3.1. 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.

E.T.S. de Ingenierías Informática y de Telecomunicación. C/ Daniel Saucedo Aranda s/n. 18071. Granada - Tlf. +34 958 244019 - http://decsai.ugr.es

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":

E.T.S. de Ingenierías Informática y de Telecomunicación. C/ Daniel Saucedo Aranda s/n. 18071. Granada - Tlf. +34 958 244019 - http://decsai.ugr.es

```
<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

E.T.S. de Ingenierías Informática y de Telecomunicación. C/ Daniel Saucedo Aranda s/n. 18071. Granada - Tlf. +34 958 244019 - http://decsai.ugr.es

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*".

3.2. 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.

3.3. 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.