Universidad de Granada
Departamento de Ciencias de la
Computación e Inteligencia Artificial

DECSAI

ANEXO 2
Descripción del fichero "utilidades.aiml"

Junto con el software entregado para la realización de la práctica, se incluye un fichero llamado "utilidades.aiml" en el que se incluyen algunas reglas genéricas que pueden resultarle útiles para realizar con mayor comodidad su práctica.

En este anexo se describen las reglas incluidas en este fichero, con algunos ejemplos que ilustran su uso. Las utilidades proporcionadas son las siguientes:

1. Operaciones para manejar listas.
2. Cómo generar un número aleatorio dentro de un rango.
3. Otra forma de implementar un bucle.
4. Cómo comparar dos cadenas.

A2.1. Operaciones para manejar listas de nombres

En AIML se trabaja exclusivamente con cadenas de caracteres y con procesamiento simbólico. De forma que cada variable global o local que se usa es siempre de tipo cadena de caracteres. Por esta razón, es importante tener algunas funciones que faciliten el trabajo con este tipo de dato.

En el fichero "utilidades.aiml" se han incluido algunas de las operaciones más frecuentes que pueden ser útiles para el desarrollo de los problemas de 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 lista sin 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 la posición "number".
- **ADDITEM [palabra] IN [listaPalabras]:** Añade "palabra" al comienzo de la lista "listaPalabras", sólo si "palabra" no estaba antes en "listaPalabras". Si la palabra ya está en la lista, la operación no hace nada.

Este conjunto de operaciones no se usa directamente en las operaciones de E/S de los agentes, sino que está concebido para trabajar sobre variables, ya sean globales 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 cómo se pueden usar las distintas operaciones, supongamos que existe una variable global llamada list_fruit que almacena una lista de frutas.

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

Aprovecharemos la pregunta "conoces algunas frutas?":

```
<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: Actualizar una variable global con datos proporcionados por el usuario.

Una regla que aprenda nuevas frutas a partir 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 seguido por esta regla es bastante intuitivo:

1. Indica en la variable local existe si lo que se especifica en <star/> es una de las frutas que ya conoce, las almacenadas en la variable global list_fruit. Para determinar la presencia de esa fruta en la lista hemos recurrido a **FINDITEM**.
2. La variable existe contendrá un 0 si no se encuentra la fruta indicada en la lista y, en ese caso, lo que hace es añadirla al principio de la lista list_fruit y mostrar un mensaje al interlocutor indicando que se acordará de que es una fruta. Para añadirla a nuestra lista se utiliza **ADDITEM**.
3. Si la variable existe almacena un valor distinto de 0, entonces dicho valor indicará la posición de la lista en la que ya se encuentra la fruta indicada por el usuario.. En este caso, nos limitamos a mostrar un mensaje apropiado.

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 programador competente se daría cuenta qué es más simple crear un nueva regla que invoque la anterior y cambiando el 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> desempeña un papel semejante al de la invocación de un método en un lenguaje de programación tradicional.

Ejemplo 3: Permitir la corrección de errores en una lista.

Supongamos que, en un momento dado, por error, el interlocutor dijo *"la mesa es una fruta"* y nuestro agente añadió *mesa* a la lista de frutas. Nuestro interlocutor, poco después, se da cuenta de su error y quiere corregirlo, por lo que nos dice *"fue un error, la mesa no es una fruta"*. La siguiente regla nos permitirá rectificar el error:

```
<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 del *Ejemplo 2*: Se busca si <star/> está en la lista de frutas mediante **FINDITEM**, almacenado su posición en pos. Si pos contiene el valor 0, entonces no estaba en nuestra lista. En caso contrario, eliminamos un elemento de la lista mediante **REMOVEITEM,** que elimina la palabra de posición pos de list_fruit, y la lista que se obtiene como resultado la almacenamos en list_fruit.

Igual que antes, deberíamos incluir una regla adicional que admita patrones 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: Introducir múltiples valores en una lista

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 frutas de una en una. Sería interesante que pudiera darnos una lista de frutas y que el agente fuera capaz de admitir frases del tipo "la manzana, el platano, el melon y la ciruela son frutas", en las que aparece un artículo delante del nombre de cada fruta ("el" o "la") y una conjunción antes del último elemento ("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>
```

Vayamos paso a paso:

1. Se 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. En función del valor de esta variable, que podrá tomar 4 valores diferentes (“y”, “la”, “el” u otro valor), distinguimos distintos casos:

a. "y" (justo antes de terminar de la lista): La palabra después de "y" será un artículo que podemos ignorar y la siguiente será el nombre de la fruta. Por esa razón, usamos **SELECTITEM** para tomar el tercer elemento de la lista, que almacenamos en ítem, para después invocar a la regla que de la forma *"la ... es una fruta"*, dónde … es el valor de ítem. Al ser la última fruta de la lista, se sale del ciclo. Por eso, a diferencia del resto de los casos, no terminamos con <loop/>.

b. "la" o "el" (al presentar un nuevo elemento de la lista): En este caso, ignoramos el artículo y pasamos a evaluar la siguiente palabra, que es la que contiene el nombre de la fruta. Mediante **TOP** extraemos el primer elemento de lo que queda en la lista y lo almacenamos en ítem, mientras que con **REMAIN** quitamos el primer elemento de la lista. Como en este caso aún no hemos terminado nuestra enumeración, seguimos en un bucle con <loop/>.

c. Caso por defecto (si no es una "y", ni un "el", ni un "la", lo que se tiene en ítem será el nombre de una fruta): En este caso, se invoca la regla *"la ... es una fruta"*, que se añadirá si no existe ya en la lista y se continúa igual que en el apartado (b); es decir, con **TOP**, **REMAIN** y <loop/>.

3. Terminado el bucle, el procesamiento de la cadena ha terminado y se muestra al interlocutor una frase que tenga sentido en la conversación.

A2.2. RANDOM

RANDOM [number] y devuelve un número aleatorio entre 1 y number.

Ejemplo 5: Elegir aleatoriamente un elemento de una lista

Se plantea una regla simple que, ante la entrada de "Dime una fruta", hace que el agente conversacional devuelva aleatoriamente un elemento de la lista de frutas que tiene almacenadas en la variable list_fruit.

```
<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 elementos de la lista y lo almacena en la variable cantidad.
3. A partir de cantidad, que indica el número de frutas elegibles, se usa **RANDOM** para generar un número aleatorio entre 1 y cantidad, número que se almacena en pos.
4. Se selecciona la palabra que ocupa la posición pos de lista y se almacena en elegida utilizando **SELECTITEM.**
5. Se devuelve como respuesta el valor de la variable elegida.

A2.3. ITERATE / NEXT

Con el par ITERATE/NEXT implementamos una versión más convencional de un bucle 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 viene guiado por NEXT, que devuelve el siguiente valor de la lista. Tanto ITERATE como NEXT devuelven la cadena "end" cuando se termina de recorrer una lista.

Ejemplo 6: Mostrar todos los elementos de una lista.

En principio, esta regla sería tan simple como devolver el valor de la variable list_fruit, pero nosotros vamos a complicarlo un poco para aprender a utilizar ITERATE/NEXT:

```
<category>
<pattern> Dime todas las frutas que conoces </pattern>
<template>
   Estas son las frutas que conozco
   <think>
     <set var="item">
        <srai> ITERATE <get var="list_fruit"/> </srai>
     </set>
     <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>
```

A2.4 COMPARE

COMPARE [palabra] WITH [palabra], y devuelve "YES" si son iguales las palabras y "NO" si no lo son.

Ejemplo 7: Comparación de palabras

Vamos a realizar una implementación alternativa para resolver el problema del ejemplo 4. En este caso, directamente se indica una lista de términos que no son frutas y queremos, por un lado, actualizar la variable list_fruit y, por otro, dar una respuesta diferente dependiendo de si había alguna "no fruta" incluida en nuestra lista de frutas conocidas.

```
<category>
<pattern> no son frutas *</pattern>
<template>
    <think>
      <set var="cantidad">
        <srai>COUNT <get var="list_fruit"/></srai>
      </set>
      <set var="item">
        <srai>ITERATE <get var="lista"/></srai>
      </set>

    <condition var="item">
      <li value="y">
        <set var="item"><srai>NEXT</srai></set>
        <set var="item"><srai>NEXT</srai></set>
        <srai> no es una fruta la <get var="item"/> </srai>
      </li>

      <li value="la">
        <set var="item"><srai>NEXT</srai></set>
        <loop/>
      </li>

      <li value="el">
        <set var="item"><srai>NEXT</srai></set>
        <loop/>
      </li>

      <li>
        <srai> no es una fruta la <get var="item"/> </srai>
```

```
        <set var="item"><srai>NEXT</srai></set>
        <loop/>
      </li>
    </condition>

    <set var="cantidad2">
      <srai>COUNT <get name="list_fruit"/></srai>
    </set>

    <set var="iguales">
      <srai>
         COMPARE <get var="cantidad"/> WITH <get var="cantidad2"/>
      </srai>
    </set>
    </think>

    <condition var="iguales">
    <li value="YES"> Ya sabia que ninguna de esas era un fruta </li>
    <li> Pues vaya, he tenido que corregir algunos errores </li>
    </condition>
</template>
</category>
```

Este último ejemplo es fácil de seguir si se ha entendido bien todo lo descrito en este anexo, así que dejaremos que lo intente por sí mismo.

En la zona de descargas de la asignatura puede encontrar el fichero “anexo2.aiml” con la resolución de todos estos ejemplos.

OBSERVACIONES FINALES

¿Qué ocurrirá si las reglas del ejemplo 3, en vez de incluir el patrón “no es una fruta la *”, tuvieran un patrón semejante al del ejemplo 2, “la * no es una fruta”? ¿Qué se tendría que modificar para hacer compatibles las dos posibilidades anteriores? ¿Se atreve a corregirlo?