INTELIGENCIA ARTIFICIAL

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

Tutorial: Práctica 3

Agentes conversacionales

(Agenda de contactos y eventos)

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

UNIVERSIDAD DE GRANADA

Curso 2016-2017

1. Introducción

En este tutorial se mostrará una posible forma de almacenar la información de la agenda de contactos y eventos, conjuntamente con trozos de código AIML 2.0 que explique su utilización. Cada estudiante puede elegir la forma en que desee almacenar la información y no estará obligado a utilizar ni las estructuras de datos ni el código asociado que se muestre en este tutorial. Podéis considerar este tutorial como un punto de partida para comenzar con el desarrollo de la práctica 3.

2. Almacenamiento de la agenda

En necesario realizar una implementación del agente conversacional que sea transparente para el usuario, es decir, si bien el usuario indica la dirección de un contacto como "Calle Turning 8", internamente hace falta almacenarlo como una sola palabra (por ejemplo uniendo cada palabra con un guión bajo => "Calle_Turing_8"). Esto mismo sucederá con las fechas, las horas y demás datos. Para ellos tienen a su disposición reglas auxiliares en el fichero `utilidades.aiml`.

2.1 Agenda de contactos

Para almacenar la información de cada contacto (apellido, nombre, dirección y telefono) se utilizará tres diccionarios (maps en AIML): `nombre`, `direccion` y `telefono`. Estos diccionarios utilizarán el apellido como clave, ya que según indica el guión, el apellido no se puede repetir. Es decir, si quiero agregar los datos una persona cuyo apellido es Sanchez, su nombre es Maria, su dirección es Calle Turing 8 y su numero de teléfono es 666000444, se creará una entrada en cada diccionario de la siguiente manera:

nombre	direccion	telefono
Sanchez:Maria	Sanchez:Calle_Turing_8	Sanchez:666000444

Recordad que no pueden utilizarse espacios en blanco, ni signos de puntuación.

2.2 Agenda de eventos

Para la agenda de eventos se utilizarán dos diccionarios: `evento` y `evper`. En el primero se guardará la información sobre la fecha, hora y título del evento; mientras que en el segundo se guardará la lista de contactos asociado a cada evento. Por ejemplo, si quiero agregar un evento el

día 20/06/2017 a las 9:00 titulado "Examen" con la asistencia de Sanchez, cada diccionario quedaría de la siguiente manera:

evento
20/06/2017:null null null null null null null null null null null null null null null null null null Examen null null null null null null null null null null null null null null null null null null null null null null null null null null null null null

evper
20/06/2017_0900:Sanchez

Como se puede apreciar el diccionario `evento` utiliza como clave la fecha en donde se produce el evento y luego tiene una lista de 48 posiciones que indican los intervalos de 30 minutos de un día. Esta lista comienza en la hora 00:00, siguiendo por 00:30, 01:00, 01:30 y así siguiendo. Por tanto las 09:00 corresponden a la posición 19 de esa lista. Las 00:00 corresponden a la posición 1 (la primera) y las 23:30 corresponde a la 48 (la última).

El diccionario `evper` utiliza como clave la fecha y la hora juntas y tiene asociada una lista de contactos (contactos que deben existir en la agenda de contactos). Notar que la lista de contactos puede tener diferente tamaño para cada evento. En caso de no haber un contacto asociado no habrá entrada en el diccionario `evper`.

Estas estructuras de datos han sido seleccionadas para resolver la práctica, aun cuando pueden no ser las óptimas o las más intuitivas. Cada estudiante valorará la/s idea/s presentadas y en base a éstas modificará las estructuras de datos que crea conveniente o creará nuevas que se adapten mejor a su propia forma de pensar y codificar.

3. Extensión del lenguaje AIML

El lenguaje AIML 2.0 tiene algunas restricciones a la hora de poder trabajar con conjuntos y diccionarios, así también 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.

3.1 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 permita:

- Cargar en una variable los valores de un conjunto:

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

- Guardar en un conjunto nuevos valores:

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

3.2 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 permita:

- 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>
```

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

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

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

- 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>
```

De esta manera es posible crear contactos y eventos en la agenda y poder reutilizarlos cada vez que iniciemos el bot.

3.3 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>
```

4. Algunos apuntes de utilidad

4.1 Creando reglas auxiliares

Es muy útil crear ciertas reglas auxiliares que permitan que el código AIML dentro de un <template> sea lo menos farragoso posible. Por ejemplo, sería aconsejable tener una regla auxiliar para verificar si un contacto existe o no en mi agenda. Esta regla podría ser utilizada al realizar la regla que almacene un nuevo contacto (para verificar que no exista ya otro con el mismo nombre), para agregar un contacto a un evento (para verificar que exista en nuestra agenda), etc.

También resulta interesante crear reglas genéricas que permitan, por ejemplo, crear un evento en una fecha puntual y luego realizar otras reglas más pequeñas que llamen a la primera que puedan resolver el "hoy" o "mañana". Es decir, si consigo pasar el "hoy" a una fecha fija de la forma D/M/AAAA puedo entonces llamar a la regla genérica y así evito duplicar el código con la consiguiente complejidad de mantenerlo siempre actualizado.

4.2 Usando el "topic" y el "that"

Recordemos el uso de cada uno de estos términos:

- **that**: esta etiqueta permite crear una nueva regla basada en la ultima frase generada. Esta etiqueta es muy útil para preguntas de tipo Si/No en donde depende de la pregunta realizada hare una cosa u otra.

```
<category>
    <pattern> ^ borra+ ^ contacto ^ </pattern>
    <template>Esta usted seguro?</template>
</category>
```

```
<category>
    <pattern>si</pattern>
    <that>Esta usted seguro?</that>
    <template>Ok. Contacto borrado.</template>
</category>

<category>
   <pattern>no</pattern>
   <that>Esta usted seguro?</that>
   <template>Ok. Contacto no borrado.</template>
</category>
```

- **topic**: esta etiqueta permite agrupar reglas bajo una misma temática. Se podría, por ejemplo, utilizar de forma conjunta con la etiqueta anterior:

```
<category>
  <pattern> ^ borra+ ^ contacto ^ </pattern>
  <template>Esta usted seguro?<set name="topic">contacto</set></template>
</category>

<category>
  <pattern> ^ borra+ ^ evento ^ </pattern>
  <template>Esta usted seguro?<set name="topic">evento</set></template>
</category>

<topic name="contacto">
  <category>
     <pattern>si</pattern>
     <that>Esta usted seguro?</that>
      <template>Ok. Contacto borrado.</template>
  </category>

  <category>
     <pattern>no</pattern>
     <that>Esta usted seguro?</that>
      <template>Ok. Contacto no borrado.</template>
  </category>
</topic>
```

```
<topic name="evento">
   <category>
      <pattern>si</pattern>
      <that>Esta usted seguro?</that>
       <template>Ok. Evento borrado.</template>
   </category>

   <category>
      <pattern>no</pattern>
      <that>Esta usted seguro?</that>
       <template>Ok. Evento no borrado.</template>
   </category>
</topic>
```

4.3 Verificando la información

Es necesario recordar que es imprescindible verificar la información antes de agregar un contacto o un evento, por ejemplo, que una fecha sea válida. También es necesario que antes de recuperar un dato de una lista se verifique primero que está el dato en esa lista.

4.4 Utilizando condicionales

Las estructuras condicionales `<condition>` son sencillas de utilizar, pero aun así existen algunos trucos que pueden servimos de ayuda. Si la condición que quiero utilizar corresponde a una igualdad no hay ningún problema, pero cuando corresponde a una desigualdad puede parecer más complejo de lo que realmente es. Por ejemplo, si quiero que se imprima un OK solo cuando el valor de una variable es distinto a 0:

```
<condition var="variable">
      <li value="0"></li>
      <li>OK</li>
</condition>
```

4.5 Utilizando ciclos

El uso de ciclos en AIML es a veces algo complejo, pero puede hacerse más sencillo con alguna ayuda. Por ejemplo, si me interesa buscar si hay un evento en una hora determinada en un día determinado, puedo iterar sobre la lista de valores del diccionario cuya clave sea la fecha

indicada o bien puedo hacer uso de alguna de las reglas auxiliares en el fichero `utilidades.aiml` que se incluye con el material de la práctica. Para ello se utilizará un nuevo diccionario llamado `horas` que indica para cada hora la posición en la lista.

```
<!– Determina si hay un evento en una fecha y hora determinadas, devuelve el nombre del evento en caso afirmativo o null en caso contrario –>
<category>
<pattern>evento * *</pattern>
<template>
  <think>
    <!– Guardo los parametros de entrada –>
    <set var="fecha"><star/></set>
    <set var="hora"><star index="2"/></set>

    <!– Verifico que existan eventos para esa fecha –>
    <set var="fechas"><readkeys>evento</readkeys></set>
    <set var="hay">
      <srai>FINDITEM <get var="fecha"/> IN <get var="fechas"/></srai>
    </set>
   </think>
   <condition var="hay">
     <li value="0">null</li><!– No hay eventos –>
     <li>
       <think>
       <!– Recupero del diccionario la lista de valores –>
       <set var="lista"><map name="evento"><get var="fecha"/></map></set>
       <set var="pos"><map name="hora"></map></set>
      </think>
      <srai>SELECTITEM <get var="pos"/> IN <get var="lista"/></srai>
    </li>
  </condition>
</template>
</category>
```

4.6 Utilizando números

No es nada fácil utilizar números en AMIL. No es posible (de forma sencilla) sumar, restar, multiplicar y dividir dos números. Tampoco es intuitivo comparar dos valores por mayor o menor. Lo único que se puede hacer con los números es sumar de a 1 o restar de a 1 utilizando dos

diccionarios: `successor` y `predecessor`. Por ejemplo si quiero realizar un ciclo de 1 hasta 10 puedo hacerlo así:

```
<set var="i">0</set>
<condition var="i">
      <li value="10">FIN</li>
      <li>
            <set var="i">
                  <map name="successor"><get var="i"/></map>
            </set>
            <loop/>
      </li>
</condition>
```

Y si quiero realizar un ciclo de 10 a 1 puedo hacerlo de manera análoga:

```
<set var="i">10</set>
<condition var="i">
      <li value="1">FIN</li>
      <li>
            <set var="i">
                  <map name="predecessor"><get var="i"/></map>
            </set>
            <loop/>
      </li>
</condition>
```

4.7 Utilizando fechas

La manipulación de fechas se realiza mediante la etiqueta <date>. Aquí os mostramos algunos ejemplos:

- **Fecha actual**: `<date jformat="d/M/yyyy" />`
- **Fecha actual**: `<date jformat="dd/MM/yyyy" />`
- **Hora actual**: `<date jformat="hh:mm" />`
- **Hora actual**: `<date jformat="HH:mm" />`
- **Solo la hora actual**: `<date jformat="hh" />`
- **Minuto actual**: `<date jformat="mm" />`

4.8 Utilizando conjuntos y diccionarios auxiliares

Los conjuntos y diccionarios son herramientas muy útil, se pueden crear de manera fija y acceder a ellos en el código AIML.

Por ejemplo, podrían ser útiles conjuntos que indicaran:

- Los días en un mes
- Los meses en un año
- Los años válidos (ej:, 2017, 2018, …, 2017)

Por ejemplo, podrían ser útiles diccionarios que indicaran:

- Si una hora particular es de mañana, tarde o noche.
- Si un minuto en particular esta antes o después de las 00 o 30 (04 -> 00, 37 -> 30).
- El numero del mes (enero -> 1, diciembre -> 12).

5. Observaciones finales

La programación en AIML suele tener una curva de aprendizaje un poco lenta al comienzo, pero cuando avanzáis un poco en la creación de reglas todo se vuelve más sencillo y más intuitivo. Empezad con tiempo e id probando las reglas nuevas y las viejas ¡también!… tened en cuenta que pueden haber patrones que "entren en conflicto" y unos tengan más prioridad que otros, causando que una regla que antes funcionaba a la perfección deja de hacerlo al incorporar nuevas reglas…