Introduction to AIML

Witold Paluszyński
Department of Cybernetics and Robotics
Faculty of Electronics
Wrocław University of Technology
http://kcir.pwr.edu.pl/~witold/

2014

Background

AIML is an XML-based description language designed for creating natural language software agents. It was developed by Richard Wallace starting in 1995 and was the basis for the conversation agent A.L.I.C.E. ("Artificial Linguistic Internet Computer Entity"), which won the annual Loebner Prize Competition in Artificial Intelligence three times (2000, 2001, and 2004), and was also the Chatterbox Challenge Champion in 2004.

The A.L.I.C.E. AIML set was released under the GNU GPL license and the development of the language was continued with the participation of the free software community.

This document describes the basic elements of AIML corresponding mostly to version 1 of AIML. In 2013 work has been started on version 2 of AIML. As of March 2014 a working draft is available.

Rule-based language

AIML is a rule-based language, which means the program is created as a collection of rules. This represents the so-called **data-driven** programming paradigm, which is different from **sequential programming**.

The **rules** are small entities consisting of two parts: **condition** and **action**. The system works by selecting one rule which has its condition satisfied, and then executing the action of the selected rule. More precisely, an instantiation of the rule's action is executed, since during the evaluation of the rule's condition, some variables can be assigned values, which instantiates the action part of the rule.

The execution of some rule's action is called its **firing**. The process of selecting and firing a rule forms the basic work cycle of the system, performed by a language interpreter. In general, the interpreter of a rule-based system may repeat the cycle some specified number of times, or as long as there is at least one rule, which has the condition part satisfied. The AIML interpreter fires exactly one rule (if at all possible), and may fire additional rules if so instructed under the **recursion mechanism** (see the `srai` tag below).

The ordering of firing rules

More than one rule can have their conditions satisfied at any time. If so happens, it is called a **conflict**. The interpreter must have a **strategy** for resolving conflicts, so that exactly one rule is selected for firing. If no rule has the condition satisfied, the system stops, or fails.

It is important to note, that in most rule-based systems, the order in which the rules are written in the program has no influence on the selection of the rules for firing. In other words, ordering of the rules in the program does not affect the order of their execution. The sequence of the fired rules is only determined by their content, and the data provided to the rules.

The AIML program

An AIML program is an XML document consisting of the elements defined in the AIML schema. It must consist exactly one `aiml` element:

```
<aiml>
</aiml>
```

Theoretically, a file might just contain a plain `aiml` tag like that. In practice, many AIML interpreters might require a more complete XML header pointing to the AIML schema location:

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="1.0.1"
 xmlns="http://alicebot.org/2001/AIML-1.0.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://alicebot.org/2001/AIML-1.0.1
                http://aitools.org/aiml/schema/AIML.xsd">
</aiml>
```

The elements of a rule

A rule is defined with the `category` element, so the AIML document should contain a sequence of the `category` elements. Each one should contain one `pattern` element and one `template` element, which define the condition and action parts, respectively:

```
<aiml>
    <category>
        <pattern>HELLO</pattern>
        <template>Hi, how are you?</template>
    </category>
</aiml>
```

The above program will answer "Hi, how are you?" if the user first says "HELLO". If the user says anything else, the program will fail, since there is no rule possible to select.

The wildcards and the star tag

The condition pattern can contain wildcards * which allow writing more general patterns:

```
<category>
    <pattern>MY NAME IS *</pattern>
    <template>OK, nice to meet you.</template>
</category>
```

The `star` tag allows using in the template the text matched to the wildcard in the pattern:

```
<category>
    <pattern>MY NAME IS *</pattern>
    <template>OK, nice to meet you <star/></template>
</category>
```

The srai tag and the recursion

It often happens, that the same user question or phrase can be stated in different ways, but should be handled in the same way. In such cases, the `srai` tag can be used to refer to a specific rule from another rule:

```
<category>
    <pattern>How do I get to the train station?</pattern>
    <template>Go to the main street and then turn right.
    </template>
</category>

<category>
    <pattern>How do I get to the supermarket?</pattern>
    <template>Follow the main street out of town.</template>
</category>

<category>
    <pattern>What is the way *</pattern>
    <template><srai>How do I get <star/></srai></template>
</category>
```


Randomized actions

The action part of a rule can select one of a several responses randomly:

```
<category>
    <pattern>How are you doing?</pattern>
    <template>
        <random>
            <li>I'm fine.</li>
            <li>OK, I am doing alright.</li>
            <li>Quite well, thanks.</li>
            <li>Good, good, and you?</li>
        </random>
    </template>
</category>
```

Global variables

AIML allows the program to use global variables. Any rule can assign a text value to any variable, and it can be referenced in any rule:

```
<category>
    <pattern>MY NAME IS *</pattern>
    <template>OK, nice to meet you
              <set name="userName"><star/></set> .
    </template>
</category>

<category>
    <pattern>MY NAME IS *</pattern>
    <template>OK, nice to meet you.
        <think><set name="userName"><star/></set></think>
    </template>
</category>
```

The `think` tag in the last rule allows its content to be processed like the first one, but prevents it from being merged to the output string and being displayed.

Using global variables in conditionals

A value assigned to a global variable can be placed in any output test using the `get` tag. It can also be used in a conditional expression:

```
<category>
    <pattern>WOULD YOU LIKE TO DANCE WITH ME?</pattern>
    <template>
        Sure, it will be a pleasure.
        <condition name="userName" value="">
            Can you tell me your name?
        </condition>
    </template>
</category>
```

In the above rule, the template checks, in addition to answering the question, that the program has learned the name of the user. Empty value of the `userName` variable means, that the variable has not been set. In such case, the program asks for the name. If the user provides it, the previous rule will fire, which will record the name.

Grouping rules using the topic tag

It is possible to control the topic of the conversation using the `topic` tag. The `topic` element(s) must occur at the top level of the program, alongside other rules (which do not belong to any topic). The topic can be changed at any time, effectively moving to a different group of rules.

```
<category>
    <pattern>Can I buy a train ticket?</pattern>
    <template>Yes, this is the ticket office.</template>
</category>

<category>
    <pattern>Can I buy a ticket to * ?</pattern>
    <template>Yes, you can.
        <think><set name="userDestination"><star/></set>
               <set name="topic">PAYMENT</set></think>
        <srai>How can i pay?</srai>
    </template>
</category>
```



```
<topic name="PAYMENT">

    <category>
        <pattern>How can i pay?</pattern>
        <template>You can pay cash or use a credit card.
        </template>
    </category>

</topic>
```

AIML interpreters — Program D

There exist many AIML interpreters — program that execute the AIML work cycle when loaded with a set of AIML categories (from one or more files). For some reason, most are named as Program X where X is some letter, like: Program D, Program E, Program M, Program O, Program P, Program Q, Program R, Program V, Program Y, Program #, etc. They are written in a wide selection of programming languages and offer different range of additional functionality, beside the basic AIML interpretation.

A simple but well-established interpreter written in Java is Program D. It has a text version which can be run in a text terminal, and a window GUI which opens a window showing the dialog.

After starting the text version, a new program can be loaded with:

```
/load filename
```

The file can be placed in the ProgramD directory, or specified with a full disk path. After successfully loading a program, the user can start the dialog.

Important references

The "home page" of AIML — a repository of resources related to AIML:

`http://www.alicebot.org/aiml.html`

The AIML tutorial by it author Richard S. Wallace:

`http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html`