3.2 Alice

Alice is a chatterbot developed by Dr. Richard S. Wallace of Lehigh University, Pennsylvania, USA [39]. Alice is an acronym for Artificial Linguistic Internet Computer Entity. The Alice open source is available in Java (among others) and is distributed under the terms of the GNU General Public License[1]. Alice can be reached at **`www.alicebot.org`**.

The idea behind Alice is in fact nothing more than Weizenbaum's Eliza; the main differences are a much larger database with patterns and the tools for creating new content by dialog analysis. The knowledge represented in the patterns is represented in a XML specification called Artificial Intelligence Markup Language (AIML).

Alice won the Loebner Contest 2000 for keeping the most human conversation. Talking to Alice gives you an idea why she won. Most responses contain the answer to the question asked, and otherwise she will give you the impression she understands what you are talking about. Moreover, she uses pickup lines to actively start a new conversation about a specific topic. The online Alice on **`www.alicebot.org`** logs the IP addresses of the visitors, so she can recognize you if you visit her again. She will know the topics you have discussed, your favorite color, food and of course your name.

[1] see http://www.fsf.org/copyleft/gpl.html

3.2.1 AIML

AIML is a XML specification especially developed for the Alice project [39]. Thanks to the clear syntax of the categories and the available libraries containing over 22.000 categories, it is easy to create a new bot with unique features. The most important aspect of AIML is the fact that people who already know HTML will find developing AIML relatively easy. The categories in AIML can be compared to the rules of a grammar. The algorithm finds the best matching pattern for each input. The category connects the stimulus pattern directly to the response template. All AIML categories belong to one of three groups: atomic, default or recursive.

An example AIML category

This is a simple default AIML category:

```
<alice>
<category>
<pattern> * </pattern>
<template>Hello</template>
</category>
</alice>
```

The `<alice>...</alice>` tags indicate that this file describes the knowledge of a chatterbot. The `<category>` tag indicates an AIML category, the basic unit of a bot's knowledge. The category has a `<pattern>` and a `<template>`. The pattern in this case is the wildcard symbol * that matches one or more words. The template represents the bot's answer. This simple bot just responds by saying "Hello" to any input.

Atomic categories

Atomic categories contain no wildcards. This is an atomic category:

```
<category>
<pattern>WHAT IS A CIRCLE</pattern>
<template>A cicle is a the set of points equidistant from
a common point called the center.
</template>
</category>
```

Default categories

Default categories contain exactly one wildcard. The ultimate default category is the category from 3.2.1.

```
<category>
<pattern>I NEED HELP * </pattern>
<template>OK, tell me more.
</template>
```

```
</category>
```

This category responds to a number of questions like "I need help debugging my program" and "I need help with my marriage".

Recursive categories

Recursive categories are used because often the same answer can be given to various inputs:

```
<category>
<pattern>BYE BYE</pattern>
<template><srai>GOODBYE</srai>
</template>
</category>
```

Once the user enters "bye bye", the phrase "goodbye" is re-evaluated by the pattern matcher. This is done by the **<srai>** tag. This way, multiple greetings like "bye", "bye bye" and "see you" are all reduced to "goodbye". A pattern matching "goodbye" should then respond with an appropriate answer.

Alice's main advantage is the fact that her source code is open source. This encourages people to write their own AIML, which resulted in over 22.000 categories so far. Alice stores the categories in alphabetical order by pattern. When a client enters an input, the program scans the categories in reverse alphabetical order to find the best match. By comparing the input with the patterns in reverse alphabetical order, the algorithm ensures that the most specific pattern matches first. 'Specific' in this case means that the program finds the longest pattern matching an input. The wildcard character $*$ comes before 'A' in alphabetical order. For example, the

WHAT $*$

pattern is more general than

WHAT IS $*$

The default pattern $*$ is first in alphabetical order and the most general pattern. For convenience, AIML also provides a variation on $*$ denoted '_', which comes after 'Z' in alphabetical order.

Once the most specific matching pattern has been found, the template in that category is executed. If the pattern contains a wildcard, the value of this wildcard can be used in the template using the **<star>** tag.

```
<category>
<pattern>ARE YOU VERY * </pattern>
<template><srai>ARE YOU <star/></srai>
</template>
</category>
```

This recursive category reduces "Are you very smart" to "Are you smart". Additionally, several <get> and <set> methods can be used to store and retrieve variables. These variables contain information about the bot's name, gender, birthday and location and the user's IP address, name and other information. The topic of the conversation and the last reply of the bot can be stored and retrieved using the <topic> and the <that> tag.

3.2.2 Pattern matching

The pattern matching used by Alice starts with analyzing the words in the phrase entered by the user [39]. These words are in natural language, separated by spaces and converted to uppercase to enable case independent matching. If a matching category is found, it is possible that the category is a recursive category. If this is the case, the target phrase will be altered and evaluated again by the pattern matcher. This adds the wildcard symbols * and '_' to the pattern language. Many AIML patterns start with the same words:

```
WHAT IS A CLOUD
WHAT IS A *
WHAT IS *
WHAT QUESTION
WHAT
```

All these patterns overlap in one or more words. This fact suggests the use of a tree to store the patterns in. Alice uses a rooted, directed tree. Each node in the tree contains a hashtable, which is used to store its successor nodes. This way, pattern matching time is almost constant independent of the number of categories. It only depends on the length of the sentence; the longer the sentence, the longer the time to find a matching pattern. Yet, this relation is linear. All patterns are stored in alphabetical order; this means that

```
WHAT _
```

comes before

```
WHAT QUESTION
```

and is considered first, but the latter on its turn comes before

```
WHAT *
```

so wildcard patterns can be checked either before or after the normal patterns.

The reasoning behind finding the way through the tree of categories is done by considering two possible cases:

1. The sentence is empty, there are no more words to process. If the node is a terminal node, a match is found. If the node is not a terminal node, the matching pattern does not exists. The best match is the last matching pattern containing a *. For example,

   ```
   WHAT IS A COMPUTER
   ```

does not perfectly match one of the five patterns shown earlier. Therefore, the matching pattern will be

```
WHAT IS A *
```

2. The sentence is not empty and contains at least one word. If the first word of the sentence is equal to one of the successors of the current node, the next node is entered with all but the first word. This way, every step from node to node reduces the length of the sentence by one. Then, the process starts over again.

The number of recursive calls is thus limited to the number of words in the sentence. The process described above changes slightly when a wildcard is detected. If one of the successors of the current node is a wildcard, a separate routine is used to check whether the words match the wildcard. If the successor containing the wildcard is a terminal node, any sentence matches. For example, the phrase

```
WHAT IS A CHATTERBOT CAPABLE OF
```

matches the pattern

```
WHAT IS A *
```

If it is not a terminal node, a recursive method tries to match all substrings of the current phrase with the current node. If this succeeds, a match is found. If it fails, the best match is the last wildcard match found. In the worst case, the pattern consisting of just *, the most general match, will be used. Usually, this pattern contains very general replies.

3.3 Alice and AIML versus grammars and parsers

Eliza can be considered a basic FSTN with some additions; it has an option to reuse different parts of the Eliza scripts during pattern matching and a FIFO memory stack where user phrases can be stored. These can later be used to refer to a previous topic. These two features are in fact functionalities of RTNs and ATNs; the reuse of script code (although no real recursivity) and the use of a memory stack.

Obviously, Alice contains these features as well, although in a more advanced form. The reuse of AIML by using recursivity is done by the `<srai>` tag as described in section 3.2.1. This tag can be used for both redirecting and re-evaluating (parts of) user input. Besides, words or whole sentences can be stored in memory by using the `<get>` and `<set>` tags. Special `<get>` and `<set>` tags are the `<gettopic>` and `<settopic>` tags, which get and set the current topic of the conversation. The words describing the topic do not have to be words the user entered. An AIML developer can set the topic manually for fairly specific categories:

```
<category>
<pattern>DO YOU LIKE CATS AND DOGS</pattern>
<template>Yes, I like them very much.
```

```
<think><settopic>pets</settopic></think>[2]
</template>
</category>
```

Later on during the conversation, the following category could be selected:

```
<category>
<pattern> * </pattern>
<template>Let's talk some more about <gettopic/>.
</template>
</category>
```

which results in

> Let's talk some more about pets.

Besides a pattern and a template, a AIML category can contain **<that>** and **</that>** tags. **<that>** refers to the last reply from the bot and can be used to create context-specific answers. Examples will be shown in chapter 4 on Development.

Although traditional grammars only describe the *syntax*, AIML actually checks a part of the *semantics*, because no abstract categories (i.e. noun phrase, verb phrase) are used. Contradicting the idea behind RTNs, all AIML consists of actual words instead of abstract categories. Admittedly, this prevents the use of small recursive datastructures for representing the AIML and consequently results in about 3MB of text for 22.000 categories. However, consider the following sentence:

> Green ideas sleep furiously

Obviously, this phrase does not make any sense. On the other hand:

> Young trees grow fast

has exactly the same syntactic structure. Using AIML, this results in fast recognition of non-semantic sentences. This way, time is saved and a not-understood reply can be formulated. Most user interfaces are (supposed to be) realtime; users want a conversation without waiting for the parser to finish. The parsing mechanism of Alice, as described in section 3.2.2, can parse sentences in (near) linear time because of the flattened structure of AIML, independent of the database size. Therefore, the time complexity is $O(n)$ with n the length of the sentence. Compared to breadth-first and depth-first search with time complexities of $O(b^x)$ (where b is the branching factor and x either the depth of the solution or the maximum depth of the tree) Alice is very fast. This is an important requirement for natural language interface systems, as human to human conversations are usually fluent without relatively long pauses in between. Eventually, once the tree becomes to large, the calculation of the reply takes up too much time; more than a few seconds becomes already annoying. The

[2]The **<think></think>** tags prevent all text in between from appearing on screen.

low complexity can be explained by the fact that Alice is a simple stimulus-response machine. Whereas parsers produce a syntactic structure that has to be interpreted before an answer to the question can be given, Alice's parser leads directly to the response.

3.3.1 First, second and third person

Sometimes, a reply is formed by reformulating the user's question. However, if the phrase contains a pronoun like 'I' or 'your', this word has to be transformed into the right form. The `<person>...</person>` tags exchange first and third person:

> Alice, tell me something about *you* please
>
> What do you want to know about *me*?

Additionally, `<person2>...</person2>` tags interchange first en second person.

3.3.2 Wildcards

Whereas a grammar can produce syntactically correct sentences by walking along its nodes, AIML can not. The reason for this is the occurrence of wildcards; characters that represent one or more words. A lot of the patterns in AIML categories contain a wildcard and it is unknown which words from the user input will match the wildcard. Consequentially, only small parts of the language can be reconstructed by looking at the AIML. Eliza scripts contain an even more general wildcard; this wildcard matches no or more words, instead of one or more. This fuzziness makes it possible however to create nonsense sentences that actually are accepted by the AIML. Take for example the most general AIML category (see section 3.2.1) which will match any input. Therefore, any sentence will be accepted if this category is loaded. This adds to the robustness of the system by trying to interpret partial correct phrases, but at the same time it undermines the credibility of the system which will keep responding to the strangest user inputs. An AIML pattern can only contain one wildcard for a number of reasons:

- Basic AIML should be as simple as possible for non-programmers.
- Many problems that could be solved with multiple wildcards can also be solved with other AIML tags, such as `<srai>`.
- Multiple-wildcard patterns have ambiguous matching properties such as the "* AND *" with sentences like "John and I worked and played".

3.3.3 More tags

AIML contains even more tags, like <condition> and <random> to test if a variable has a certain values, and to pick a random reply. Tag combinations that occur often can be abbreviated, like `<srai><star/></srai>` can be rewritten as `<sr/>`.

Appendix C

Links

General links	
InfoBots	www.infobots.nl
The Alice Nexus	www.alicebot.org
Alice at SourceForge.net	sourceforge.net/projects/alicebot/
The Loebner Prize Competition	www.loebner.net/Prizef/loebner-prize.html
The Simon Laven Page	www.toptown.com/hp/sjlaven
GNU General Public License	www.fsf.org/copyleft/gpl.html
Free CGI server space	www.mycgiserver.com

Other chatterbots	
Alison	alison.alicebot.com
Ally	www.accessterminal.com/L.html
Claude	www.basicguru.com/mclaughlin
Eliza	ecceliza.cjb.net
Elvis	elvis.alicebot.com/~acraig/index.htm
Hex	www.amristar.com.au/~hutch/hex
Hippie	hippie.alicebot.com/cgi.html
John Lennon	www.triumphpc.com/john-lennon
SHAMpage	www.toptown.com/hp/sjlaven/shampage.zip

Commercial applications	
Ask Jeeves	www.askjeeves.com
Native Minds	www.nativeminds.com
Artificial Life	www.artificial-life.com
Virtual Personalities	www.vperson.com
KiwiLogic	www.kiwilogic.com
Centraal Beheer	www.centraalbeheer.nl
Q-go	www.q-go.com

Bibliography

[1] **[P, R, 4]** op den Akker, R. et al. (1994). *Natuurlijke Taal Interfaces voor Dialoogsystemen*, Memoranda Informatica 94-04, January 1994.

[2] **[P, R]** Artificial Life, Inc. (1999). *Smart Bots: Solutions for the Networked Economy*, white paper.

[3] **[P, R]** Artificial Life, Inc. (1999). *The Art of Bot Procreation*, concept paper.

[4] **[P, R]** Bates, J. (1994). *The Role of Emotion in Believable Agents*, Technical Report CMU-CS-94-136, School of Computer Science, Carnegie Mellon University, April 1994. Also to appear in Communications of the ACM, Special Issue on Agents, July 1994.

[5] **[4]** Biermann, A. W. et al. (1983). *An experimental study of natural language*, International Journal of Man-Machine Studies 18, 71–87.

[6] **[P, R]** Boyce, S. J. (2000). *Natural Spoken Dialogue Systems for Telephony Applications*, Communications of the ACM, Vol. 43, No. 9, 29–34.

[7] **[1]** Bradshaw, J. M. (Ed.) (1997). *Software Agents*, Boston: MIT Press.

[8] **[P, R, 2, 4]** Capindale, R. A. and Crawford, R. G. (1990). *Using a natural language interface with casual users*, International Journal of Man-Machine Studies 32, 341–361.

[9] **[P, 2]** Carroll, J. M. and Rosson, M. B. (1987). *The paradox of the active user*, In J.M. Carroll (Ed.), Interfacing Thought: Cognitive Aspects of Human-Computer Interaction. Cambridge, MA: MIT Press.

[10] **[1, 2]** Dix, A. et al. (1993). *Human-computer interaction*, Prentice-Hall.

[11] **[P, R, 2]** Dreyfus, H. L. and Dreyfus, S. E. (1988). *Making a Mind versus Modeling the Brain: Artificial Intelligence Back at a Branchpoint*, Daedalus 177, 15–43.

[12] **[R, 3]** Gazdar, G. and Mellish, C. (1989). *Natural Language Processing in Prolog*, Cogsweb Project Books, March 1989.

[13] **[4]** Grice, H. P. (1975). *Logic and Conversation*, in P. Cole and J.L. Morgan (eds.) Syntax and Semantics 3: Speech Acts, Academic Press.

[14] **[4]** Hauptman, A. G. and Green, B. F. (1981). *A comparison of command, menu-selection and natural language computer programs*, Behaviour and Information Technology, Vol. 2, No. 2, 163–178.

[15] **[2]** Hill, I. (1983). *Natural language versus computer language*, in M. Sime and M. Coombs (eds.) Designing for Human-Computer Communication, Academic Press.

[16] **[R, 2]** Hutchens, J. L. (1996). *How to Pass the Turing Test by Cheating*, available online[1].

[17] **[R, 3]** Hutchens, J. L. (1996). *How Hex Works*, available online[2].

[1] http://ciips.ee.uwa.edu.au/Papers/Technical_Reports/1997/05/
[2] http://www.amristar.com.au/~hutch/hex/How.html

[18] [**2**] Jarke, M. et al. (1985). *A field evaluation of natural language for data retrieval*, IEEE Transactions of Software Engineering SE-II, Vol. 1, 97–113.

[19] [**4**] Kelley, J. F. (1983). *An empirical methodology for writing user-friendly natural-language computer applications*, Proceedings of the CHI '83 Conference on Human Factors in Computing Systems, ACM, New York, 193–196.

[20] [**P, R, 2**] Kelly, M. J. and Chapanis, A. (1977). *Limited vocabulary natural language dialogue*, International Journal of Man-Machine Studies 9, 479–501.

[21] [**P, R**] Kubon, P. P. et al. (2000). *An Extendable Natural Language Interface to a Consumer Service Database*, Proceedings of the 13th Biannual Conference of the Canadian Society for Computational Studies of Intelligence, Montréal, Canada, May 2000.

[22] [**P, R, 1, 2**] Lansdale, M. W. and Ormerod, T. C. (1994). *Understanding Interfaces - A handbook of human-computer dialogue*, San Diego, Academic Press Inc.

[23] [**R, 3**] Linz, P. (1996). *Introduction to Formal Languages and Automata*, D. C. Heath and Company.

[24] [**P, R, 2**] Long, B. (1994). *Natural Language as an Interface Style*, available online[3].

[25] [] Maes, p. (1994). *Agents that Reduce Work and Information Overload*, Communications of the ACM, Vol. 37, No. 7, 31–40.

[26] [**P, R**] Minsky, M. (1982). *Why People Think Computers Can't*, AI Magazine, Vol. 3, No. 4, 1982.

[27] [**P, R, 2**] Mykowiecka, A. (1991). *Natural language generation – an overview*, International Journal of Man-Machine Studies 34, 497–511.

[28] [**P**] Nijholt, A. (1988). *Computers and Languages - theory and practice*, Elsevier Science Publishers BV, 1988.

[29] [**P**] Nwana, H. S. and Ndumu, D. T. (1999), *A Perspective on Software Agents Research*, The Knowledge Engineering Review, Vol. 14, No. 2, 1–18.

[30] [**2**] Petrick, S. R. (1976). *On natural-language based computer systems*, IBM Journal of Research and Development, Vol. 20, 314–325.

[31] [**3?**] Pinker, S. (1999). *How the Mind Works*, W. W. Norton & Company, 1999.

[32] [**R, 3**] Russell, S. and Norvig, P. (1995). *Artificial Intelligence - A modern approach*, Prentice-Hall.

[33] [**P, R, 2**] Searle, J. R. (1980). *Minds, Brains, and Programs*, The Behavioral and Brain Sciences, Vol. 3, 417–457.

[34] [**2**] Slator, B. et al. (1986). *Pygmalion at the Interface*, Communications of the ACM Vol. 29, No. 7, 599–604.

[35] [**P, R, 2**] Turing, A. M. (1950). *Computing machinery and intelligence*, Mind, Vol. 59, No. 236, 433–460.

[36] [**2**] Turner, J. A. et al. (1984). *Using restricted natural language for data retrieval: a plan for field evaluation*, Human Factors and Interactive Computer Systems, Norwood NJ: Ablex, 163–190.

[37] [] Véronis, J. (1991). *Error in natural language dialogue between man and machine*, International Journal of Man-Machine Studies 35, 187–217.

[38] [**P**] Vetulani, Z. et al. (2000). *Corpus Based Methodology in the Study and Design of Systems with Emulated Linguistic Competence*, Proceedings of the Second International Conference on Natural Language Processing, Greece, June 2000.

[3]http://www.dgp.toronto.edu/people/byron/papers/nli.html

[39] **[R, 3]** Wallace, R. S. (2000). *Don't read me - A.L.I.C.E. and AIML documentation*, available online[4].

[40] **[P, R, 1, 2, 3]** Weizenbaum, J. (1965). *Eliza - a computer program for the study of natural language communication between man and machine*, Communication of the Association for Computing Machinery 9: 36–45.

[41] **[P]** Weizenbaum, J. (1976). *Computer Power and Human Reason - from judgement to calculation*, W. H. Freeman and Company, 1976.

[42] **[P, 2]** Whalen, T. (1995). *How I Lost the Loebner Contest and Re-evaluated Humanity* available online[5].

[43] **[P]** Winograd, T. (1980). *What Does It Mean to Understand Natural Language*, Cognitive Science 4, 209–241.

[44] **[P]** Yankelovich, N. et al. (1995). *Designing SpeechActs: Issues in Speech User Interfaces*, Proceedings of the CHI '95 Conference on Human Factors in Computing Systems, Denver, USA, May 1995.

[45] **[P, R, 2, 4]** Zoltan-Ford, E. (1991). *How to get people to say and type what computers can understand*, International Journal of Man-Machine Studies 34, 527–547.

[4]http://www.alicebot.com/dont.html
[5]http://debra.dgrc.crc.ca/chat/story95.html