

Using the Corpus of Spoken Afrikaans to generate an Afrikaans chatbot

Bayan Abu Shawar* and Eric Atwell

School of Computing, University of Leeds, Leeds LS2 9JT, England

** Corresponding author, e-mail: bshawar@comp.leeds.ac.uk*

Abstract: This paper presents two chatbot systems, ALICE and Elizabeth, illustrating the dialogue knowledge representation and pattern matching techniques of each. We discuss the problems which arise when using the Corpus of Spoken Afrikaans (Korpus Gesproke Afrikaans) to retrain the ALICE chatbot system with human dialogue examples. A Java program to convert from dialog transcripts to the AIML linguistic knowledge representation formalism provides a basic implementation of corpus-based chatbot training. The Java program used the Afrikaans dialogue corpus texts to generate two versions of the Afrikaans chatbot.

1. Introduction

“Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them” (ACM SIGCHI, 2002). It is an area of Information Technology where researchers and developers try to facilitate human communication with computers by using natural language processing. One important technology in this area is the chatbot: a conversational software agent, which interacts with users using natural language. The idea of chatbot systems originated in the Massachusetts Institute of Technology (Weizenbaum, 1966, 1967), where Weizenbaum implemented the Eliza chatbot to emulate a psychotherapist. Since that time, a lot of conversational systems have been developed to serve in different domains such as: customer service, educational guidance, web site help, and for fun. In general, all chatbot responses depend on the linguistic knowledge stored and the pattern-matching algorithm used. In this paper we will review two chatbot systems with different linguistic knowledge representation formalisms, and present an approach to linguistic knowledge acquisition via Machine Learning from a corpus. Our approach uses a Java software system to read a dialogue transcript from an Afrikaans human-to-human conversation, and extract dialogue language patterns to retrain a chatbot in order to make it converse like an

Afrikaans speaker. The approach seemed straightforward at the outset, but we encountered a lot of problems and drawbacks; we discuss these and propose potential directions for further research..

The remainder of section one introduces the problem of linguistic knowledge acquisition, and why Machine Learning may be an answer for languages with few existing NLP resources. Sections two and three outline the linguistic knowledge representation and pattern matching algorithms of two chatbot systems: ALICE (ALICE, 2002; Abu Shawar & Atwell, 2002, 2003) and Elizabeth (Millican, 2002; Abu Shawar & Atwell, 2002, 2003). Both systems were adapted from the ELIZA program (Weizenbaum, 1966, 1967), which emulated a psychotherapist. ALICE was found to be better suited for training using dialogue corpora because of its simple pattern templates and simple matching technique, based on the AIML linguistic knowledge representation formalism. The Potchefstroom University Korpus Gesproke Afrikaans (Corpus of Spoken Afrikaans) (Van Rooy, 2003), which contains transcripts of a range of conversations between more than two speakers, was used to generate an Afrikaans version of ALICE. Section four presents the Java program that converts a dialogue text to AIML format, and how it was applied to the Korpus Gesproke Afrikaans. Several problems found in analysis of this corpus are discussed

in this section. This formalisation has helped us to see the main characteristics that must be found in dialogue corpora in order to use them for training a chatbot, as discussed in section five. Section six reviews the evaluation of two Afrikaans chatbots that were generated using the Java program: Afrikaana (Afrikaana, 2003), which speaks just Afrikaans, and a bilingual version speaking English and Afrikaans, named AVRA (AVRA, 2003).

1.1 The linguistic knowledge acquisition bottleneck

In the field of artificial intelligence or knowledge based systems, knowledge acquisition is the capturing and encoding of human expert knowledge in a computer-usable form, to be used in 'intelligent' software. Natural Language Processing (NLP) systems require linguistic knowledge, formal models of the natural language to be processed. This linguistic knowledge generally ought to be formalised and encoded by someone who is both an expert in the language in question, and in appropriate AI formalisms. However, this combination of expertise is scarce outside the main European Union languages: not enough speakers of southern African languages are also knowledge engineers. A possible alternative is to try to separate the roles of linguist and knowledge engineer, and replace the latter with a machine learning approach (Atwell *et al.*, 2003). If southern African linguists can supply appropriate corpus samples of the language(s) in question, it may be possible to automate knowledge acquisition, using software to extract linguistic patterns to use in NLP systems.

Ideally, to machine learn a language model for a specific NLP application, we need a training corpus, which includes sufficient representative examples of the target linguistic behaviour. In practice, we may have to compromise, and use what is readily available rather than the ideal corpus for our needs. For example, many large-vocabulary speech recognition systems for European languages use language models trained with a corpus of newspaper texts; this is not because these systems are built specifically to recognise newspaper text read out loud (!), but because newspaper text is readily available in large quantities, and is judged to be reasonably close to the language style or genre of dictation. To train a chatbot to

converse in Afrikaans, ideally we wanted a large corpus with many examples of Afrikaans speakers interacting with an existing chatbot in Afrikaans. Unfortunately no such corpus was available, but instead we were kindly offered use of a set of transcripts of human group conversations, the Korpus Gesproke Afrikaans (Van Rooy, 2003). We were not sure at the outset whether this text-type was sufficiently similar to our target; at least some of the problems discussed in sections five and six can be explained in terms of inappropriateness of the training corpus. However overall we judged this to be a successful experiment in linguistic knowledge acquisition via machine learning, in which linguistic and knowledge acquisition expertise were clearly separated: the linguists supplying the training corpus were not aware of ALICE or AIML formalisms; and we had (almost) no prior understanding of Afrikaans.

2. ALICE

ALICE (ALICE, 2002; Abu Shawar & Atwell 2002, 2003), the Artificial Linguistic Internet Computer Entity, is a software robot or program that you can chat with using natural language. ALICE knowledge about English conversation patterns is stored in AIML files. AIML (Wallace, 2003), or Artificial Intelligence Mark-up Language, is a derivative of Extensible Mark-up Language (XML). It was developed by the Alicebot free software community during 1995-2000 to enable developers to input dialogue pattern knowledge into chatbots based on the A.L.I.C.E free software technology.

AIML consists of data objects called AIML objects, which are made up of units called topics and categories. The topic is an optional top-level element; it has a name attribute and a set of categories related to that topic. Categories are the basic unit of knowledge in AIML. Each category is a rule for matching an input and converting to an output, and consists of a pattern that represents the user input and a template that implies the ALICE robot answer. The AIML pattern is simple, consisting only of words, spaces, and the wildcard symbols _ and *. The words may consist of letters and numerals, but no other characters. Words are separated by a single space, and the wildcard characters function like words. The pattern language is case invariant.

2.1 Types of categories in ALICE

There are three types of categories: atomic categories, default categories, and recursive categories.

a. *Atomic categories*: are those with patterns that do not have wildcard symbols, `_` and `*`, e.g.:

```
<category>
<pattern>10 DOLLARS</pattern>
<template> WOW, that is cheap.</tem-
plate>
</category>
```

In the above category, if the user inputs: 10 dollars, then ALICE answers: WOW, that is cheap.

b. *Default categories*: are those with patterns having wildcard symbols `*` or `_`. The wildcard symbols match any input but they differ in their alphabetical order. Assuming the previous input 10 DOLLARS, if the robot does not find the previous category with an atomic pattern, then it will try to find a category with a default pattern such as:

```
<category>
<pattern>10 * </pattern>
<template>It is ten.</template>
</category>
```

So ALICE answers: It is ten.

c. *Recursive categories*: are those with templates having `<sr>` and `<srai>` tags, which refers to simply recursive artificial intelligence and symbolic reduction. Recursive categories have many applications: symbolic reduction that reduces complex grammatical forms to simpler ones; divide and conquer that splits an input into two or more subparts, and combines the responses to each; and dealing with synonyms by mapping different ways of saying the same thing to the same reply.

c.1 Symbolic reduction

```
<category>
<pattern>DO YOU KNOW WHAT THE *
IS</pattern>
<template><srai>What is <star/></srai>
</template>
</category>
```

In this example `<srai>` is used to reduce the input to simpler form "what is *".

c.2 Divide and conquer

```
<category>
<pattern>YES * </pattern>
<template><srai>YES</srai><sr/><tem-
```

```
plate>
</category>
```

The input is partitioned into two parts, 'yes' and the second part; `*` is matched with the `<sr/>` tag.

```
<sr/> = <srai><star/></srai>
```

c.3 Synonyms

```
<category>
<pattern>HALO</pattern>
<template><srai>Hello</srai></tem-
plate>
</category>
```

The input is mapped to another form, which has the same meaning.

2.2 Preparation for pattern matching in ALICE

Before starting the pattern-matching algorithm, each input to the AIML interpreter must pass through two processes: a normalization process, and producing an input path from each sentence.

1. *Normalization process* involves three phases. Firstly, substitution normalization is a heuristic applied to an input that attempts to retain information in the input that would otherwise be lost during the sentence splitting or pattern fitting normalization. Secondly, sentence-splitting normalization splits the input into two or more sentences using rules like 'break sentence at periods'. Thirdly, pattern-fitting normalization removes punctuation from the input, and converts the input to uppercase. The normalization process is applied in this order and at least the pattern fitting normalization must be done.

Example: If the input is: "I do not know. Do you, or will you, have a robots.txt file?"

a. Substitution Normalized form is:

"I do not know. Do you, or will you, have a robots dot txt file?"

b. Sentence Splitting Normalized Form is:

"I do not know."
"Do you, or will you, have a robots dot txt file"

c. Pattern fitting Normalized Form is:

"I DO NOT KNOW"
"DO YOU OR WILL YOU HAVE A ROBOTS DOT TXT FILE"

2. *Producing Input Path*, for each sentence an input path of the following form is produced. Normalized-Input<that>Tvalue<topic>

Pvalue. The input path has three parts. Firstly, normalized input. Secondly, <that> tag which will be followed by a Tvalue that holds the previous robot answer if it exists or * otherwise. Thirdly, <topic> tag which will be followed by Pvalue that holds the topic name if it exists or * otherwise. Topic tag is an optional top-level element having a name attribute and holding all categories related to that attribute name.

Example:

If the normalized input is: "YES", previous robot output is: "DO YOU LIKE CHEESE", and there is no topic name, then the input path will be: "YES <that> DO YOU LIKE CHEESE <topic> *".

2.3 ALICE pattern matching algorithm

The AIML interpreter tries to match word by word to obtain the longest pattern match, as this is normally the best one. This behaviour can be described in terms of the Graphmaster set of files and directories, which has a set of nodes called nodemappers and branches representing the first words of all patterns and wildcard symbols.

Assume the user input starts with word X and the root of this tree structure is a folder of the file system that contains all patterns and templates, the pattern matching algorithm uses depth first search techniques:

1. If the folder has a subfolder which starts with underscore then turn to, "_", scan through it to match all words suffixed X, if no match then:
2. Go back to folder, try to find a subfolder which starts with word X, if so turn to "X", scan for matching the tail of X, if no match then:
3. Go back to the folder, try to find a subfolder starting with star notation, if so, turn to "*", try all remaining suffixes of input following "X" to see if one matches. If no match was found, change directory back to the parent of this folder, and put "X" back on the head of the input.

When a match is found, the process stops, and the template that belongs to that category is processed by the interpreter to construct the output.

Example: assume the following categories:

(1) <category>

```
<pattern>_ WHAT IS 2 AND 2</pattern>
<template> <sr/> <srai>WHAT IS 2 AND 2</srai> </template>
</category>
```

(2) <category>

```
<pattern>WHAT IS 2 *</pattern>
<template> <random> <li>Two.</li>
<li>Four.</li> <li>Six.</li> <li>12.</li> </random>
</template>
</category>
```

(3) <category>

```
<pattern>HALO</pattern>
<template> <srai>HELLO</srai> </template>
</category>
```

(4) <category>

```
<pattern>HELLO</pattern>
<template> <random> <li>Well hello there!</li> <li>Hi there!</li>
<li>Hi there. I was just wanting to talk</li>
<li>Hello there !</li> </random>
</template>
</category>
```

If the user Input is: "halo what is 2 and 2 ?", then the ALICE output will be: "Hi there! Four." The process is as follows:

1. This input will match category (1) that breaks it into two sentences:
Sentence one: represented by <sr/> tag that matches (_) which is the word HALO.
Sentence two: WHAT IS 2 AND 2.
2. An atomic pattern is found for HALO and substituted by HELLO in category (3) and then HELLO matches again with category (4); the answer will be selected randomly from the template list of category (4). Here it deals with synonyms since halo and hello has the same output.
3. The interpreter tries to match WHAT IS 2 AND 2, no atomic pattern is found, so a backtracking technique is applied, scanning to find WHAT IS 2 AND *; again no match, another backtracking tries to find WHAT IS 2 *, a match found in category (2) and a response is selected randomly from the list.
4. The AIML interpreter will combine these answers together and display them.

3. Elizabeth

Elizabeth is an adaptation of the Eliza program, in which the various selection, substitution, and phrase storage mechanisms have been

enhanced and generalized to increase both flexibility and (potential) adaptability. Knowledge is stored as a script in a text file, where each line starts with a script command notation. These notations are single characters, one for each rule-type:

W: Welcome message Q: Quitting message
 N: No match V: Void input
 I: Input transformation /: Comment
 K: Key word pattern R: Key word response
 M: Memorise phrase N: No match
 O: Output transformation &: Action to be performed

Each script command has an index code that is generated automatically. It can also be indexed using a user special code.

3.1 Elizabeth's script file format

A script file may contain at most four parts. Part one, script command lines holding welcome, void and no keyword messages. Part two, input transformation rules, which start with "I" and maps input to another form to be compatible with the defined keywords. Part three, Output transformation rules, which start with "O" and changes personal pronouns to be appropriate as a response. Part four, keyword patterns, which start with "K" to denote keyword pattern to be matched, followed by robot response denoted by "R". There are two type of keyword patterns: simple patterns: matching only single words, and composite patterns: matching a sentence or phrase, words, string, letter, anything and nothing. For example:

```
/ The script begins with Welcome, void, no key-
word and quit messages
W HELLO, I AM ELIZABETH. WHAT DO YOU
LIKE TO TALK ABOUT?
V CAN'T YOU THINK OF ANYTHING TO
SAY?
V ARE YOU ALWAYS SHY?
N TELL ME WHAT YOU LIKE DOING?
Q GOODBYE! COME BACK SOON.
/ Input transformation rules
I MUM => mother
I DAD => father
/Output transformation rules
O my => YOUR
O YOU IS => YOU ARE
/keyword transformations
K MOTHER
K FATHER
R TELL ME MORE ABOUT YOUR FAMILY.
R ARE YOU THE YOUNGEST IN YOUR
```

FAMILY?

K I LIKE [string]ING

R HAVE YOU [string]ED AT ALL RECENTLY?

If the user input is: "I like swimming", robot answer will be: "HAVE YOU SWIMMED AT ALL RECENTLY?"

3.2 Elizabeth's pattern matching algorithm

Before starting the matching process, an active text is generated for each input by: converting user input to lower case, inserting spaces between words and punctuation, and removing some characters from the input, except: ! " ' () , - . 0..9 ; ; ? a..z

The matching process involves five stages. Stage one matches with input transformation rules. Stage two matches with keyword patterns. Stage three matches with output transformation rules. Stage four matches with void or no keyword messages. Stage five performs any dynamic process.

Stage one:

input transformation rules, all input rules are applied in turn to the active text, if a match occurs with the left hand side of the rule then substitute it by the right hand side, and record or apply any dynamic process found. If the same rule is applicable to the active text more than 10 times in succession, then it is applied just once.

Stage two:

keyword pattern matching, all keywords are applied in turn until one matches or there is no match at all. The response is given according to the first match, so if there is a list of responses, one of them is selected randomly and the index code of this response will be recorded in order not to use it if the same match occurs in the next session. If there is a dynamic process, it will be applied or recorded to be performed in stage five. If no match at all is found then go to phase four.

Stage three:

output transformation rules, includes two steps: apply the same algorithm used in stage one. Change the active text to upper case.

Stage four:

if no match occurs with the keyword patterns then either the active text is empty, and one of the void messages will be selected randomly, or no keyword found, and one of the

no keyword messages will be selected.

Stage five:

perform any dynamic processes, performs the recorded actions if any exists.

4. Software implementation

The above analyses of ALICE and Elizabeth focus on the linguistic knowledge representation formalisms used by each system, rather than the algorithm implementation details, since the aim of our comparison was to select a system to retrain using the Korpus Gesproke Afrikaans. We concluded that ALICE is better suited to corpus-based retraining because of the greater simplicity of the AIML pattern notation used in its pattern-matching algorithm; and because AIML is closer to the XML markup format used in the annotated corpus — information about speaker turn-taking, overlap, etc. is encoded in <XML-tags> (see examples below). In addition, ALICE is available in Java via a Web interface, whereas Elizabeth is implemented in Delphi and must be installed on individual user PCs, making web-based evaluation of ALICE more straightforward. However, we have also developed a Java program to translate AIML categories into a subset of Elizabeth-

style rules, so in principle we are able to develop an Elizabeth 'equivalent' of an Afrikaans-speaking ALICE.

In order to retrain ALICE, we developed a Java program to read a dialogue from a corpus and convert it to AIML format. The Afrikaans dialogue corpora were used to generate an Afrikaans chatbot. The program algorithm and the corpus analysis problems raised during implementation are discussed in detail in the following sections.

4.1 Program description

The program algorithm outlined in Figure 1 works as follows: all corpus files are read from an index database to extract texts from each file and insert it into a vector. Each vector element passes through a sequence of phases. It is first pre-processed where all useless parts are filtered, and all vector elements are reiterated and prepared to originate pattern and template sequentially. Second the vector is restructured where all different patterns with the same template are classified as srai (symbolic reduction) categories, and all different templates related to the same pattern are grouped as an atomic category with random list. Finally the

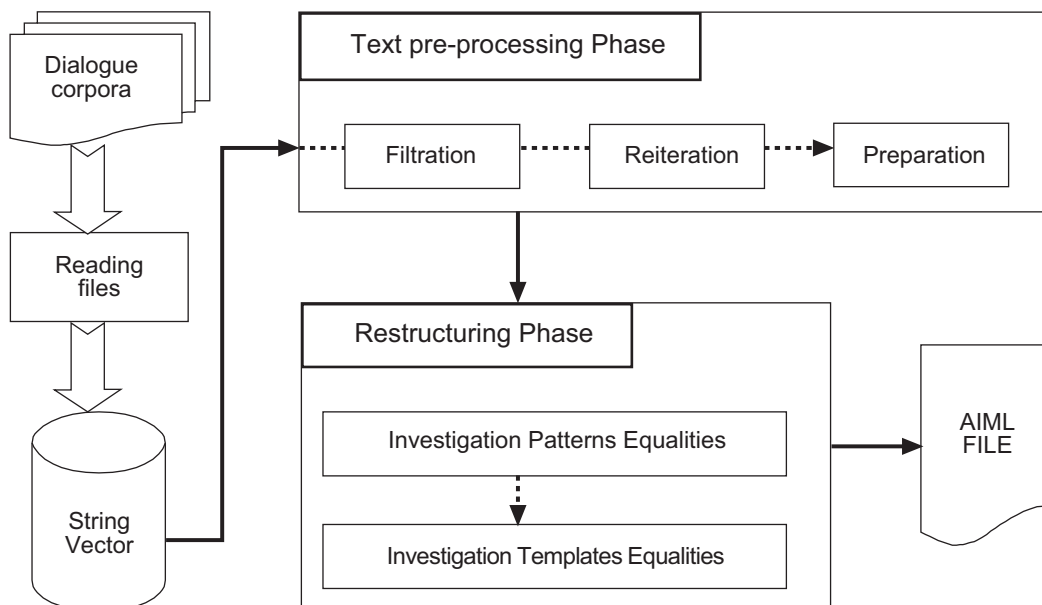


Figure 1: The program algorithm

AIML vector is transferred to an AIML file.

4.1.1 Text pre-processing phase

This phase involves three modules: filtration, reiteration and preparation for the text.

A. Filtration process

The filtering process is applied to remove overlaps between two speakers, and other useless (for us) linguistic annotation, deleting the parts of conversation that are repeated. We decide to remove the overlap because it will not happen during human-machine conversation, and because it is not evident how to analyse overlap in terms of pattern and template.

Example:

Original corpus speech transcript:

```
<spreker2> jissie toe wie't hulle wie't hulle
gespeel <oorveuel>
<spreker2> altwee ke~ o Elandskraal
</spreker2>
<spreker1> Elandskraal Elandskraal
</spreker1> </oorveuel> Elandskraal o
</spreker2>
```

After removing overlap:

```
<spreker2>jissie toe wie't hulle wie't hulle
gespeel altwee ke~ o Elandskraal
Elandskraal o</spreker2>
```

Removing useless linguistic annotation, which includes:

1. Removing "filler" speech that is written between <fil> and </fil> tags.
2. Removing any unclear text, marked up between * and +.
3. Removing text between parentheses (extralinguistic annotations) .
4. Removing speakers tags <spreker1> etc.

Example:

```
<spreker1> teen <fil> uh uuhm </fil>
Proteapark </spreker1>
<spreker2> *a+ gewen </spreker2>
```

After removing filler and unclear annotations:

```
<spreker1> teen Proteapark </spreker1>
<spreker2> gewen </spreker2>
```

Many problems encountered in this process related to the original corpus content and markup. The most important problems were:

1. Turn taking was not straightforward; for example, it was not clear what pattern and template sequences to extract from nested overlapping.
2. Spelling errors in the annotations, using "oorveuel" instead of "oorveuel" to indicate

the overlap session.

3. Spelling errors in the annotations, using "ff" instead of "fil" to indicate the filler speech.
4. XML syntax errors in opening and closing speaker tags, e.g. using <spreker1> instead of </spreker1> to indicate the end of speaker1's turn.
5. Spelling/syntax error in annotating the speaker e.g. by using <spreker 1> instead of <spreker1>
6. Extra-linguistic annotation which is not part of the dialogue, marked up between "*" and "+".
7. Extra-linguistic annotation in parentheses, denoting an unclear word, which ideally should be replaced by a linguist with an appropriate word,
8. Markup syntax errors in using * and forgetting to close it by +.
9. Not the same numbers of open and close tags especially in nested overlapping turns, and speaker-number tags.
10. Long monologs: a speaker taking a very long turn.

We solved corpus markup analysis problems by:

1. Ignoring overlap sections, as shown above
2. Replacing all "oorveuel" with "oorveuel",
3. Replacing all "ff" with "fil".
4. Correcting the XML to add "/" to closing tags where this is missing
5. All "spreker+space" tags are replaced with "spreker", removing the space.
6. Removing any text between "*", and "+"
7. Removing any text between parentheses; we realise that this results in an ungrammatical sentence, but finding a suitable replacement requires linguistic knowledge.
8. In the case of using "*" and forgetting to close it by "+", any string preceded by * is removed until first blank found.
9. Ignoring the whole turn that involved nested overlapping sessions without sufficient numbers of open and closed tags.
10. Long turns are kept verbatim as pattern and then template; this results in some unexpectedly long and involved replies to user input!

B. Reiteration process

Chatting with a computer involves just two speakers: one is the user and the other is the chatbot. In the Afrikaans corpus texts we have

more than two speakers, so we cannot recognize which will provide the pattern to match the user input and which will yield the template to generate the chatbot answer. In order to solve this problem we decided to recur each element of the speech vector except the first to be treated as a pattern in one turn and as a template in the next.

C. Preparation process

The aim of this process is to generate pattern and template from each pair of sequential elements. Since the first element in every vector is the zeroth element, even subscripts represent the patterns and odd ones hold the templates. To achieve this, a normalization process is carried out: all elements are tokenised to be sure that there is one space between each word, and all special characters are replaced with appropriate alphabets, i.e. “ô”, and “ê” are replaced by “o” and “e” (standard UK/US keyboards do not include vowels with diacritics, so their removal simplifies chatbot testing) Then all patterns are changed to capital letters and punctuations are removed from these patterns to make it the same as the AIML pattern/template formats (capitalisation and punctuation are notoriously variable in chatbot usage, so it is simpler to ignore these). All patterns templates are sorted alphabetically according to the patterns.

Example:

```
<spreker2> mōre my tannie </spreker2>
<spreker5> hello Marida </spreker5>
```

After preparation process:

```
<pattern>MORE MY TANNIE</pattern>
<template>hello Marida</template>
```

Since all categories are atomic ones, the user must input the same pattern exactly to be matched, which will limit the possibility of getting an answer. Default categories were added to expand the matching patterns by using two approaches:

1. First word approach.
2. Least frequent word approach.

1. First word approach:

This approach is based on the generalisation that the first word of an utterance may be a good clue to an appropriate response: if we cannot match the whole input utterance, then at least we can try matching just the first word. For each atomic pattern, we generated a default

version that holds the first word followed by star to match any text, and then associated it with the same atomic template.

Example:

```
<category>
<pattern>EENDAG AS JY GROOT IS SAL
JY</pattern>
<template>maar ons sal ek sal oorkom</tem-
plate>
</category>
```

The first word default category is:

```
<category>
<pattern>EENDAG *</pattern>
<template>maar ons sal ek sal oorkom</tem-
plate>
</category>
```

2. Least frequent word approach:

This approach is a variant of the above, loosely based on information theory. Instead of assuming the first word of an utterance is most “significant”, we look for the word in the utterance with the highest “information content”, the word that is most specific to this utterance compared to other utterances in the corpus. This should be the word that has the lowest frequency in the rest of the corpus.

We tokenised the whole corpus into word-tokens, to produce a list of all words, along with the frequency of each word. Sorting this list in ascending order yields the overall least frequent word list. Then we tokenised each pattern and compared it with the Corpus word-frequency list to get the least frequent word in this pattern. Four default categories are added which holding this least-frequent word in different position of the pattern: start, middle, last or just the least frequent word alone.

We choose the least frequent approach to generate the default categories, because usually in human dialogues the intent of the speakers is hiding in the least-frequent, highest-information word.

Example:

```
<category>
<pattern>JA MAN HULLE HET GEKOOK OU
MAAT</pattern>
<template>het Franna weer drie
gedruk</template>
</category>
```

The least frequent word for the previous pattern is GEKOOK, so the default categories are:

First: holding the least frequent word only:

```
<category>
<pattern>GEKOOK</pattern>
<template>het Franna weer drie
gedruk</template>
</category>
```

Second: holding the least frequent word at the start of a pattern:

```
<category>
<pattern>GEKOOK * </pattern>
<template>het Franna weer drie
gedruk</template>
</category>
```

Third: holding the least frequent word at the middle of the pattern:

```
<category>
<pattern>* GEKOOK * </pattern>
<template>het Franna weer drie gedruk
</template>
</category>
```

Fourth: holding the least frequent word at the end of the pattern:

```
<category>
<pattern>* GEKOOK </pattern>
<template>het Franna weer drie
gedruk</template>
</category>
```

Within the AIML format, these four categories are the only way to allow the word to appear in a wildcard position in a sentence.

4.1.2 Restructuring phase

Given a large training corpus, the same utterance may be said more than once, with different replies; or one reply may be the answer to two or more different utterances. If we add extra default categories to allow the chatbot to answer partial matches, there are many more repeated templates and patterns in our AIML file. The file must be restructured to collate these repetitions.

Atomic categories that are generated from the previous process are investigated in two directions: templates and patterns investigation. All categories with the same pattern and different templates are grouped to generate one category holding that pattern and new template with a sorted random list of different choices of the original templates. On the other hand all different templates related to the same pattern are used to generate one atomic category of the pattern and template and other categories that map each pattern to another one using symbolic reduction templates.

Example 1: Patterns equalities investigation:

Original categories:

```
<category>
<pattern>DIS REG</pattern>
<template> ek sal dit by Jeanine-hulle kry
asseblief</template>
</category>
```

```
<category>
<pattern>DIS REG</pattern>
<template> raait wat nog</template>
</category>
```

Grouped as:

```
<category>
<pattern>DIS REG</pattern>
<template>
<random>
<li>ek sal dit by Jeanine-hulle kry asse-
blief</li>
<li>raait wat nog</li>
</random>
</template>
</category>
```

Example 2: Templates equalities investigation:

Original categories:

```
<category>
<pattern>DINGES MARION SE VIR MY
TWICKENHAM WAS HULLE HOME
GROUND</pattern>
<template>London Irish</template>
</category>
```

```
<category>
<pattern>EN THE LONDON IRISH</pattern>
<template> London Irish </template>
</category>
```

Mapping to:

```
<pattern>EN THE LONDON IRISH</pattern>
<template><srai>DINGES MARION SE VIR
MY TWICKENHAM WAS HULLE HOME
GROUND</srai></template>
</category>
```

After these processes the vector holding categories in AIML format, the final process is copying this vector to an AIML file, then use this file to retrain ALICE.

5. Ideal corpus characteristics for retraining a chatbot

The Corpus of Spoken Afrikaans was transcribed from recorded conversations between more than two speakers. It was used to retrain

ALICE using the programs described above. We encountered a number of problems, analogous to the problems we found in our analysis of English spoken corpus texts in the Dialogue Diversity Corpus (DDC) (Abu-Shawar and Atwell, 2003). We classify the problem sources into two categories: the first is from human conversation characteristics; the second is from human transcription problems.

5.1 Human conversation characteristics

- a. Irregular turn taking and overlapping
- b. Long turns, amounting to monologues.
- c. More than two speakers.

5.2 Human transcription syntax problems

- a. Syntax errors in using mark up to annotate the dialogue such as:
 - Spelling errors in opening and closing tags, which leads to different structure format in dealing with them like <oorveuel>, and <oorlveuel> to denote the overlapping.
 - Not the same number of open and close tags especially in nested overlapping, and speaker recognition turns.
 - Syntax errors leading to difficulties in recognizing when speakers begin and end turns.
- b. Syntax errors in adding extra-linguistic annotations
 - Opening a string with “*” and forgetting to close it by “+”
 - Syntax errors in using filler tags <fl> and <fil>
 - Using parentheses to denote the unclear words, which must be replaced with another appropriate word by a linguist.

We solved these problems in our analyses of the Spoken Corpus of Afrikaans and the Dialogue Diversity Corpus by deciding different approaches and methods, but ideally we would prefer general characteristics to deal with all dialogue corpora. The optimal characteristics of dialogue corpora that are really needed to retrain a chatbot system are:

- a. Two speakers, ideally human and chatbot.
- b. Structured format.
- c. Short, obvious turns without overlapping, and without any unnecessary notes, expressions or other symbols that are not

used when writing a text.

Even such ‘idealised’ transcripts may still lead to a chatbot which does not seem entirely ‘natural’: although we aim to mimic the natural conversation between humans, the chatbot is constrained to chatting via typing, and the way we write is different from the way we speak. We believe that chatting is an intermediate stage between spoken and written dialogues.

6. Evaluation

Our aim was to develop a chatbot, which speaks the Afrikaans language. Two version of ALICE were generated using the Afrikaans corpus texts. The first is called Afrikaana (Afrikaana, 2003), and speaks in the Afrikaans language only. We noticed occasional “code-switching” into English in the Corpus: as far as we know, all Afrikaans speakers also speak English, and may switch or drop into (and out of) English in casual conversation. So, we decided to try a second version of ALICE, called AVRA. AVRA (AVRA, 2003) is a bilingual version combining the standard ALICE AIML files that are written in English and the Afrikaana AIML file that is written just in Afrikaans, so users can chat with this version using two languages. We used the Padorobot web-hosting service (Pandorobot, 2003) to make our chatbots available for use over the World Wide Web. We are grateful to Afrikaans-speaking researchers at Potchefstroom University who tried out our prototype versions of Afrikaana and AVRA. The first prototypes were based only on literal pattern matching against corpus utterances: we had not implemented the first word approach and least-frequent word approach to add “wildcard” default categories. Our Afrikaans-speaking evaluators found these first prototypes disappointing and frustrating: it turned out that few of their attempts at conversation found exact matches in the training corpus, so Afrikaana replied with a default “ja” most of the time, and AVRA generally fell back on her “ALICE half” and responded in English. However, expanding the AIML pattern matching using the first-word and least-frequent-word approaches yielded more favourable feedback: our informants found the conversations less repetitive and more interesting. Responses were sometimes weird and apparently irrelevant, but at least they were not always just “ja”...

The reasons behind most of the users' feedback can be related to three issues. Firstly the dialogue corpus context does not cover a wide range of domains, so Afrikaana can only "talk about" the domain of the training corpus. Secondly, the repeated approach that we used to solve the problem of determining the pattern and the template in case of more than two speakers may lead to incoherent transcripts. Thirdly, our machine-learned models have not included linguistic analysis markup, such as grammatical, semantic or dialogue-act annotations (Atwell, 1996; Atwell *et al.*, 2000), as ALICE/AIML makes no use of such linguistic knowledge in generating conversation responses. However, the Elizabeth chatbot does in theory allow for more sophisticated conversation modelling including such linguistic features (see Abu Shawar & Atwell, 2002).

7. Conclusions

Human-computer conversation is an area of natural language processing (NLP) technology, where developers try to facilitate communication with the computer using users' natural language in order to mimic human communication. Porting NLP systems to new languages is a labour-intensive task, and the right mix of expertise may be hard to find. A software system based on Machine Learning from a training corpus was developed for this challenge. Two chatbot systems, ALICE and Elizabeth, were reviewed in this paper. We decided to train ALICE rather than Elizabeth to learn from human dialogue corpora, because the AIML format is closer to the markup format used in annotated corpora such as the Corpus of Spoken Afrikaans; and because of the simplicity of ALICE's mechanisms for generating patterns/templates and applying pattern matching techniques.

The main goal was to develop a general program that can read from human dialogue corpus texts to retrain ALICE to converse in the language of the training corpus. Many problems were raised while learning from the Afrikaans corpus texts, explained in terms of the number of speakers, long monologues, overlapping, and problems in interpreting markup and linguistic annotations. Two versions of

ALICE were generated using the Corpus of Spoken Afrikaans: Afrikaana is based exclusively on the Afrikaans training corpus; and AVRA combines the Afrikaana model with the original English ALICE model, to give a bilingual Afrikaans/English chatbot. Informal user trials indicated that our first learning approach, requiring an exact match between input sentence and corpus utterance, was too restrictive, resulting in poor responses. We then extended the pattern matching to allow matches against first word or least-frequent word; this resulted in Afrikaans-speaking chatbots which users found more acceptable.

Further research is needed to incorporate higher-level linguistic knowledge into the chatbot. The AIML formalism allows for recording of wider Context and Topic, but we have yet to include this in the categories "learnt": we need to extend our software to extract context and topic from the training corpus. We would like to find further useful applications for corpus-trained chatbots, for example in language teaching (DAVE, 2003). We would also like to try retraining ALICE with other corpora, including other languages of Southern Africa. We hope that Southern African linguists will develop corpus resources for a range of languages, so we can develop a wider range of chatbots.

Acknowledgements — Thanks to Richard Wallace and the ALICE AI Foundation for free access to ALICE and AIML technology; and to the Pandorabots website for hosting Afrikaana and AVRA, allowing trials over the World Wide Web. We are also grateful to Bertus van Rooy and Gerhard van Huyssteen for allowing us free access to the Potchefstroom University Korpus Gesproke Afrikaans (Corpus of Spoken Afrikaans) (Van Rooy 2003) for our research.

Thanks also to all Afrikaans-speaking researchers at Potchefstroom University who tried out our prototype versions of Afrikaana and AVRA. Gerhard van Huyssteen and Sulene Pilon suggested we rename the Afrikaans chatbot KARIKE instead of Afrikaana in future research. KARIKE is an abbreviation of *Kunsmatige Afrikaanse Rekenaar- en Internetkletsbot-entiteit*, which means Artificial Afrikaans Computer and Internet Chatbot Entity in English, so it is equivalent to ALICE. Also Gerhard van Huyssteen provided us with some greeting dialogue to improve Afrikaana.

9. References

- ALICE.** 2002. *A.L.I.C.E. AI Foundation*. Available at: <http://www.alicebot.org/> or <http://alicebot.franz.com/>
- Abu Shawar B & Atwell E.** 2002. *A comparison between ALICE and Elizabeth chatbot systems*. Research Report 2002.19, School of Computing, University of Leeds. Available at: ftp://ftp.comp.leeds.ac.uk/scs/doc/reports/2002/2002_19.pdf
- Abu Shawar B & Atwell E.** 2003. Using dialogue corpora to retrain a chatbot system. In: Archer D, Rayson P, Wilson A & McEnery T (eds) *Proceedings of CL2003: International Conference on Corpus Linguistics*. Lancaster University UK. pp. 681–690. Available at: <http://www.comp.leeds.ac.uk/eric/cl2003>
- ACM SIGCHI.** 2002. Curricula for human-computer interaction. Available at: <http://sigchi.org/>
- Afrikaana.** 2003. Afrikaans-speaking chatbot. Available at: <http://www.pandorabots.com/pandora/talk?botid=eba8f4dc9e3406b8>
- Atwell E.** 1996. Machine Learning from corpus resources for speech and handwriting recognition. In: Thomas J & Short M (eds) *Using Corpora for Language Research: Studies in the Honour of Geoffrey Leech*. Harlow, Longman. pp. 151–166.
- Atwell E, Demetriou G, Hughes J, Schiffrin A, Souter C & Wilcock S.** 2000. A comparative evaluation of modern English corpus grammatical annotation schemes. *ICAME Journal* 24: 7–23. Available at: <http://www.hd.uib.no/icame/ij24/atwell.pdf>
- Atwell E, Abu Shawar B, Babych B, Elliott D, Elliott J, Gent P, Hartley A, Hu XR, Medori J, Oba T, Roberts A, Scharoff S & Souter C.** 2003. Corpus linguistics, machine learning and evaluation: Views from Leeds. Research Report 2003.02, School of Computing, University of Leeds. Available at: ftp://ftp.comp.leeds.ac.uk/scs/doc/reports/2003/2003_02.pdf
- AVRA.** 2003. Afrikaans-English bilingual chatbot. Available at: <http://www.pandorabots.com/pandora/talk?botid=daf612c52e3406bb>
- DAVE.** 2003. DAVE English as a second language chatbot. Available at: <http://www.alicebot.org/dave.html>
- Millican P.** 2002. Elizabeth's home page. Available at: <http://www.etext.leeds.ac.uk/elizabeth>
- Pandorabot.** 2003. Pandorabot chatbot hosting service. Available at: <http://www.pandorabots.com/pandora>
- Van Rooy B.** 2002. *Transkripsiehandleiding van die Korpus Gesproke Afrikaans*. (Transcription Manual of the Corpus Spoken Afrikaans.) Potchefstroom, South Africa: Potchefstroom University.
- Wallace R.** 2003. The elements of AIML style. ALICE AI Foundation. Available at: <http://www.alicebot.org/> or <http://alicebot.franz.com/>
- Weizenbaum J.** 1966. ELIZA-A computer program for the study of natural language communication between man and machine. *Communications of the ACM* 10(8): 36–45.
- Weizenbaum J.** 1967. Contextual understanding by computers. *Communications of the ACM* 10(8): 474–480