

NATURAL LANGUAGE UNDERSTANDING IN THE DACST-AST DIALOGUE SYSTEM

T.R. Niesler[†] and J.C. Roux[‡]

trn@dsp.sun.ac.za jcr@maties.sun.ac.za

[†]Department of Electrical Engineering

[‡]Research Unit for Experimental Phonology University of Stellenbosch
Private Bag X1, Matieland, Stellenbosch 7602, South Africa

ABSTRACT

This paper describes the natural language understanding (NLU) component of the DACST-AST spoken dialogue system. We adopt a finite-state architecture and develop a syntax that allows these finite-state networks to be defined in a modular fashion. Meaning is associated with a particular path through a network by embedding semantic tags at appropriate points in its definition. The understanding process consists of a parsing operation that determines whether a user utterance is contained within a given finite-state network. The semantic tags associated with the path resulting from a successful parse represent the information that has been “understood” from the user utterance.

1. INTRODUCTION

The DACST-AST project is supported by the South African Department for Arts, Culture, Science and Technology (DACST) and has among its long-term aims the development of resources and expertise in the field of human language technology within the South African context. Human language technology has an important role to play in moving South Africa’s strongly multi-lingual and multi-ethnic society into the information age by improving the access and flow of information between individuals and private or public institutions. With 11 official languages and an abundance of accents and dialects, the successful development and deployment of speech technology presents many challenges.

Initial activity within the DACST-AST project has focused on the gathering, transcription and validation of speech resources for 5 of the 11 languages and on the design and development of speech and language processing algorithms to allow the realisation of spoken dialogue systems [4]. This paper describes the development of the natural language understanding (NLU) component, which is a key subsystem of a dialogue system.

2. SPOKEN DIALOGUE SYSTEMS AND NATURAL LANGUAGE UNDERSTANDING

A **dialogue** may be described as an interaction between two parties in which information is communicated between the parties in a number of sequential turns¹. When the method of communication is speech, we refer to a **spoken dialogue**. A machine designed to maintain a spoken dialogue with a person is a **spoken dialogue system**. Since no other means of communication other than speech will be considered here, the term “dialogue system” will be taken to mean “spoken dialogue system”. Hence each turn of a spoken dialogue consists either of a system or a user utterance.

¹A **turn** refers to a single uninterrupted transfer of information from one party to the other.

The task of natural language understanding within the context of a dialogue system is to extract **meaning** from a user utterance. On the basis of this meaning, the dialogue system must decide on how to proceed with the dialogue. A simplified diagram illustrating the architecture of the DACST-AST dialogue system is shown in figure 1.

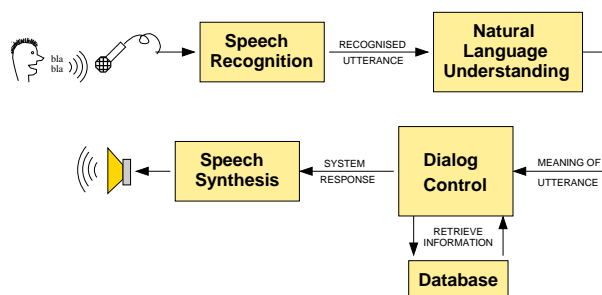


Figure 1: Architecture of the DACST-AST dialog system.

To illustrate the operation of the system in figure 1, consider as a particular example a dialogue system with which the user can make a reservation at a certain hotel. With reference to figure 1, a typical dialogue might proceed as follows. First the user utters a sentence or phrase corresponding to his or her turn in the dialogue. This utterance is transcribed into text by the speech recognition component. Let us assume for now that the user’s utterance is:

“Could I have a single room for tonight please”

and that this has been correctly transcribed by the speech recogniser. The recognised text is interpreted by the natural language understanding component, and its meaning passed on to the dialogue controller. For our example, this meaning would be (i) that a booking has been requested, (ii) that it is for a single room and (iii) that it is for the same day. The understanding component must extract these three items of information from the recognised user utterance. Based on this inferred meaning, the dialogue controller decides on an appropriate next action. This may be the retrieval of some requested information from a database, for example to check whether there are in fact any single rooms available that evening. Alternatively the system may request from the user some further information that it requires to perform its task, for example it may ask the user whether he or she would like a room with sea view or one with mountain view. Once the system has decided on the appropriate next action, it responds to the user via the speech synthesis module.

Due to the many ways in which a particular message may be communicated by human language, the user’s input may be expected to be highly variable. The NLU system must extract the important information from such natural language input and

present it to the dialogue management subsystem in a predictable manner. Considering again our example user utterance, the natural language understanding system might return the extracted information in the following form.

```
request = "make reservation";
room_type = "single";
arrival_date = "today";
```

This information, whose format is regular and well-defined, is used by the dialogue management subsystem to decide on the most appropriate response to the user.

3. FINITE-STATE NATURAL LANGUAGE UNDERSTANDING

The DACST-AST system makes use of a *finite-state* natural language understanding network, as currently successfully employed by a number of experimental as well as commercially deployed dialogue systems [1], [2], [3], [5]. In this approach the set of (partial) sentences that can be processed (i.e. "understood") by the system is precisely defined by a graph whose links (edges) correspond to words or sets of words. The set of sentences that the system will accept is given by the set of all the unique paths through the network. For example, consider the following finite-state network.

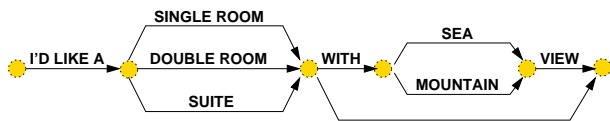


Figure 2: An example of a finite-state network .

This network is a compact representation of a total of nine different utterances. In practice, the finite-state network would be much more complex and the number of utterances it covers much larger. In order to extract information using the finite-state network, an appropriate interpretation must be explicitly assigned to the information bearing words. In the DACST-AST system this is achieved by allowing every link of the finite-state network to be associated with the assignment of an appropriate value to a particular variable, as illustrated in figure 3.

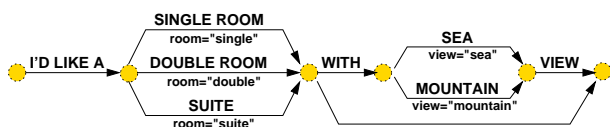


Figure 3: The network of figure 2 with variable assignments.

Now any path through the network has associated with it a number of variable assignments. When a user utterance is presented to the natural language understanding system, a parsing algorithm first determines whether there is a path through the network corresponding to the same sequence of words present in the utterance. If such a path is found, then the variable assignments associated with the links along this path are made. For example, had the user uttered the sentence:

"I'd like a double room with sea view"

the parsing algorithm would find that this word sequence is indeed present in the finite-state network illustrated in figure 3 and that the corresponding path through this network has associated with it the following two variable assignments.

```
room = "double";
view = "sea";
```

The names and final values of these variables constitute the result of the natural language understanding process.

4. SPECIFYING FINITE-STATE UNDERSTANDING NETWORKS

For any particular application, the structure of the finite-state network described in section 3 must be specified by the developer so that the network reflects the set of word sequences and associated variable assignments that must be covered. In the DACST-AST system, this is achieved by means of a regular grammar. Hence the structure of the finite-state network is defined in a plain text file according to a syntax that will be described in the remainder of this section.

4.1. Set of alternatives

The delimiters "{", "|", and "}" are used to indicate a set of alternative words or phrases. For example, the definition:

I'd like a {single room | double room | suite} please;

defines the network shown in figure 4.

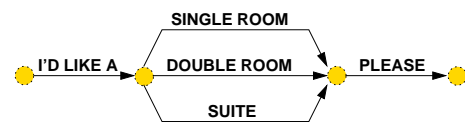


Figure 4: Network with set of alternatives.

These delimiters can be nested. Hence the same grammar could have been defined as:

I'd like a {{single | double} room | suite} please;

The finite-state network corresponding to this definition is shown in figure 5.

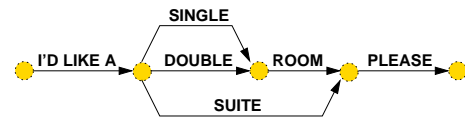


Figure 5: Network with nested set of alternatives.

4.2. Optional phrases

The delimiters "[" and "]" indicate that the enclosed word or phrase is optional. For example, the definition:

I'd like a {single room | double room | suite} [please];

indicates that the final word "please" is optional, and defines the network shown in figure 6.

As before, nesting is possible. Using the regular syntax defined so far, the network in figure 2 may be described by the following definition.

I'd like a {single room | double room | suite} [with {sea | mountain} view];

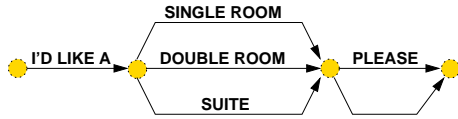


Figure 6: Network with set of alternatives and optional word.

4.3. One or more repetitions

The delimiters “<” and “>” indicate that the enclosed word or phrase may be repeated one or more times. For example, the definition:

```
<I'd like> a {single room | double room | suite};
```

indicates that the phrase “I’d like” can be repeated, to allow for false starts for example. The network corresponding to this definition is shown in figure 7.

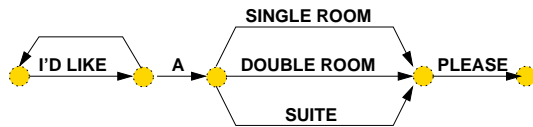


Figure 7: Network with one or more repetitions.

4.4. Variable assignments

A variable assignment is associated with a link in the understanding network by means of the syntax:

```
linkname : varname = “varvalue”;
```

Hence the assignments as indicated in figure 3 would be described by the following definition.

```
I'd like a {single room : room = “single” |
double room : room = “double” |
suite : room = “suite”} [with {sea : view = “sea”|
mountain : view = “mountain” } view ];
```

Here there are two variables named “room” and “view” respectively. In the DACST-AST natural language understanding definition, all variables must be declared prior to use and hence the complete network definition is shown below.

```
variables
{
string : room;
string : view;
}
grammar
{
I'd like a {single room : room = “single” |
double room : room = “double” |
suite : room = “suite”} [with {sea : view = “sea”|
mountain : view = “mountain” } view ];
}
```

The variables and then the network are declared by the keywords *variables* and *grammar* respectively. In the *variables* declaration, the two variables “room” and “view” are both defined to be strings. The *grammar* declaration then defines the finite-state network as before.

4.5. Subgrammars

In order to allow modular design and definition of natural language understanding grammars, the DACST-AST software allows a grammar to call another by means of an appropriate call within the *grammar* declaration section. The grammar initiating such a call is termed the *parent grammar*, and the grammar being called the *subgrammar*. When a grammar has no parent grammar it is termed the *root grammar*. A subgrammar is defined separately and is hence a fully independent grammar in its own right. Information is passed from the subgrammar to its parent grammar by means of return variables. These are defined in the *return* declaration, as illustrated below.

```
variables
{
string : type;
}
grammar
{
{single room : type = “single” |
double room : type = “double” |
suite : type = “suite”}
}
return
{
type;
}
```

This grammar is a subsection of that defined at the end of section 4.4, dealing only with the room type. The *return* declaration indicates that, on completion of a successful parse with this grammar, the value of the variable “type” will be passed back to the parent grammar. Multiple variables can be passed back by listing their names in the *variables* declaration separated by commas. No information is passed from the parent grammar to the subgrammar. In order for one grammar to employ another as a subgrammar, the latter must be declared in the *subgrammar* declaration section of the former. For example,

```
variables
{
string : room;
string : view;
}
subgrammar
{
roomtype : getroomtype;
}
grammar
{
I'd like a @getroomtype:room
[with {sea : view = “sea”|
mountain : view = “mountain” } view ];
}
{
type,view;
}
```

Here the subgrammar “getroomtype” has been declared to be of type “roomtype”, which is assumed to be the name assigned to the grammar definition considered at the start of this section². In the *grammar* declaration, a call is made to “getroomtype” and its return variable (which we remember to be the value of the variable “type”) is passed into the local variable “room”. Finally, on exit, this grammar returns the values of the variables “type” and “view”.

5. INTERACTION WITH OTHER SYSTEM COMPONENTS

Now that the finite-state natural language understanding process and its definition syntax has been described, the communication between the speech recogniser, natural language understanding and dialogue control components can be considered in greater detail. The block diagram in figure 8 illustrates what is described in the following.

The speech recogniser accepts audio input from a microphone or telephone card, and delivers the recognition result to the natural language understanding component in the form of a text string. Since only those word sequences that are covered by the particular finite-state network used by the understanding component can be parsed and hence “understood”, the speech recogniser makes use of this same finite-state network to constrain its recognition search. In this way it is guaranteed that all utterances output by the recognition process can also be parsed by the understanding component.

The speech recogniser is under the control of the natural language understanding component, which determines the commencement and termination of the recognition process. Once the recognition and understanding processes are complete, the understanding module passes the names and values of all variables returned from the root understanding grammar back to the dialogue control module.

The dialogue controller determines the most appropriate next action to take on the basis of the dialogue history as well as the newly obtained information from the natural language understanding component. In doing so the dialogue controller therefore also identifies the natural language understanding grammar that will be appropriate for processing the user’s response to the system’s next action. The identity of this grammar as well as the signal to start the next recognition and understanding phase is passed from the dialogue controller to the natural language understanding component.

6. SUMMARY AND ONGOING WORK

This paper has described the architecture of the natural language understanding component that is under development as part of the DACST-AST project. Natural language understanding is achieved by finite-state networks that are defined by means of a regular grammar whose syntax has been described. Modular design and development is made possible by allowing calls from one finite-state network to another with appropriate variable passing. While the finite-state parser and the understanding architecture as described here have been developed, their integration with the speech recognition and dialogue management components is still in progress and will be reported on in future.

²The name of a grammar type is reflected by the name of the file in which it is defined.

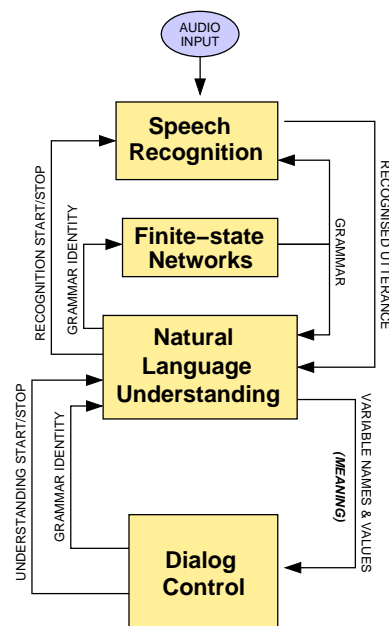


Figure 8: Interaction between system components.

7. REFERENCES

- [1] Barnard, E; Halberstadt, A; Kotelly, C; Phillips, M; *A consistent approach to designing spoken-dialog systems*, Proceedings of the Automatic Speech Recognition and Understanding Workshop, Keystone, Colorado, December 1999, pp. 363-368.
- [2] Carlson, R; Hunnicutt, S; *Generic and domain-specific aspects of the Waxholm NLP and dialog modules*, Proceedings of the International Conference on Spoken Language Processing, Philadelphia, 1996, pp. 677-680.
- [3] Jurafsky, D; Wooters, C; Tajchman, G; Segal, J; Stolcke, A; Fosler, E; Morgan, N; *The Berkeley Restaurant Project*, Proceedings of the International Conference on Spoken Language Processing, Yokohama, 1994, pp. 2139-2142.
- [4] Louw, P.H; Roux, J.C; Botha, E; *African Speech Technology (AST) Telephone Speech Databases: Corpus design and contents*. Proceedings of Eurospeech 2001, Aalborg, Denmark, 2001, pp. 2055-8.
- [5] McTear, M.F; *Modelling spoken dialogues with state transition diagrams: experiences with the CSLU toolkit*, Proceedings of the International Conference on Spoken Language Processing, Sydney, 1998.