A Prototype Natural Language Interface for Animation Systems

Diana Inkpen and Darren Kipp
University of Ottawa, School of information Technology and Engineering
diana@site.uottawa.ca, dkipp076@uottawa.ca

Abstract

We present a prototype implementation of a natural language interface to an animation system. The interface provides the means for a human user to issue commands in natural language to an avatar in a virtual reality environment. The purpose of our system is to convert the input text into commands in an animation script language and execute them. Our system uses a general-purpose parser and a domain-specific semantic interpreter based on pattern matching.

1. Introduction

This paper presents a prototype implementation of a natural language interface to an animation system. The main components of the system are: a parser, a semantic interpreter, and a command interpreter. The architecture of the system is presented in Figure 1 and explained in detail in section 3. The parser is a general-purpose natural-language parser [3]; it transforms each input sentence into a parse tree. The semantic interpreter takes as input the parse tree and generates commands in an animation script language. The command interpreter executes the animation script using the animation module, which in our case is very simple; it will be replaced by an anthropomorphic avatar [5] in future work.

The semantic interpreter is the core of our system. It directs sub-trees of the parse tree to the appropriate sub-modules. The verb phrases are sent to the "action processor", which locates the main verb and identifies the command to be generated. The other sub-trees are sent to the "details processor", which uses a pattern matcher to identify values for attributes of the commands, in the sub-trees. When certain attribute values are not specified, default values are used. For example, if the avatar is told to run without specifying how fast, a default speed is used. When a sentence contains a conjunction of two verb phrases, two consecutive commands are generated. Their attributes come from the details processor, with attributes in the sub-tree of each verb phrase allowed to overwrite attributes from higher levels in the parse tree. This allows us to capture the correct syntactic and semantic dependencies.

An example of input and output to the system is the following (the output format is explained in details in section 6):

Input: John, walk five steps to the right.
Output: walk speed=5 direction=right repetition=5;

The system is designed to be easily extended to accept other types of sentences, without modifying the code, but only the text files used as parameters by the semantic interpreter. More patterns (phrase and sentence structures) can be added to the parameter file of the details processor. New commands can be created by adding more verbs and their synonyms to the parameter file of the action processor. In order to accommodate more than one avatar, another attribute can be added to all commands, to store the name of the avatar.

2. Related work

There is a lot of related work in natural language interfaces. The two main directions are: more or less ad-hoc systems for specific domains (see [1] [2] for

two surveys), or more-principled interfaces to databases [6], that translate the natural language input into SQL queries.

One work similar to ours with respect to the application domain is [4], but there the emphasis was placed on the animation rather than the natural language interface. The system described in [4] used a simplified grammar for its input commands. Our system uses a general grammar and relies on the semantic analyzer to interpret the resulting parse tree.

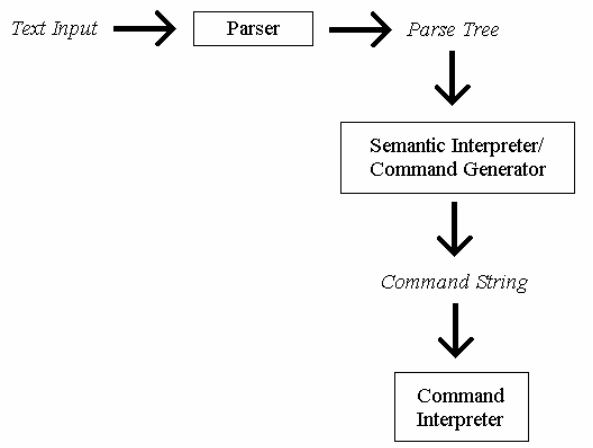

Figure 1: Command interpretation process

3. General Design Principles

Our system was designed to be an extendable system for converting natural language into a script language for animation. The semantic analyzer is completely parameterized using input files. Any new information added to the domain-specific lexicon or to the phrase inventory is placed within these parameter files. The animation module can be easily replaced with a more elaborate animation module. The system was implemented in Java.

The purpose of the system is to convert text sentences into a script language and execute it. This process is completed using three main steps (see Figure 1). The first step of the process converts the input text into a parse tree. The second step of the process converts the parse tree into a script-like command language. The final step is to interpret and execute the command language and produce the animation.

4. The syntactic analysis

The first module performs the parsing (the syntactic analysis) of the input sentence. This module accepts a string input and returns a tree data structure. See Figure 2 for an example of a resulting parse tree. The parser [3] that we used was developed by the Natural Language Processing (NLP) group at Stanford University. It comes with a collection of Java packages for handling the data structures. The NLP group periodically produces new versions of their Java parser. These could be plugged into our animation system very easily. It would also be possible to use another parser with similar functionality to the Stanford parser, by

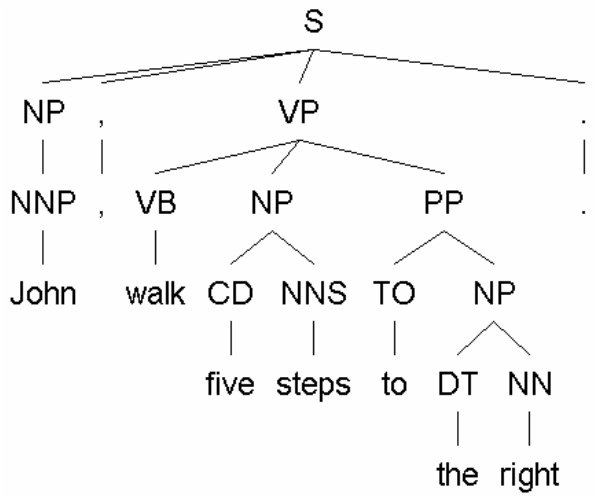

Figure 2: Example of parse tree for "John, walk five steps to the right."

creating additional classes to interface it to the rest of the software.

The Stanford parser has been in existence for several years and currently works very well. The only significant problems encountered with the parser involved very small, semi-incomplete sentences with assumed subjects. These included examples such as "Jump." This was overcome by adding the avatar's name (in our case the avatar is named "John") to all the sentences. Although it is possible to train the parser using your own data, the default probabilistic grammar which was provided with the parser was used in our project.

Another drawback of this parser was that it required a significant amount of memory. It was therefore necessary to restrict the size of the heap (to about 800Mb) when launching the Java program. This restriction limited the length of sentences which it was

able to parse. This limitation did not cause any problems in our application.

5. The semantic analysis

The major emphasis of our work was on the semantic analyzer, whose architecture is detailed in Figure 3. The semantic analyzer was designed to accept a tree structure containing the parse tree and return the command or the sequence of commands in the form of a string. It uses five separate sub-modules.

The first module is called the command processor. This module accepts the input tree and returns the output string. Aside from I/O operations, it serves to direct the sub-trees from the first level of the sentence tree to the appropriate modules (either the details processor or the action processor). This module functions by first processing non verb phrases by sending these sub-trees to the details processor. All resulting details are collected in a list and sent to the action processor along with the verb phrase. This mechanism allows details from higher levels to propagate down into the verb phrase. The action processor returns one or more commands in the form of a string.

The second module is the details processor. This module has the responsibility of extracting command details from sections of tree which are not verb phrases. If this module encounters a verb phrase it sends the verb phrase to the action processor for handling.

The details processor uses a pattern matcher to extract meaning from sections of sub-trees. If the details processor matches a section of tree it returns the resulting property. Otherwise, it must drill deeper in the tree in order to attempt to find the correct property.

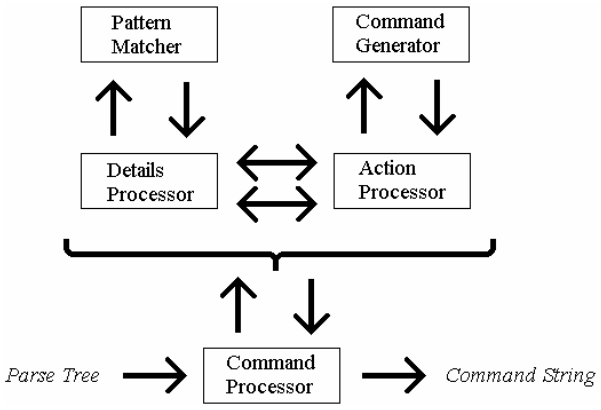

Figure 3: The semantic analyzer

The third module is the action processor. The action processor is responsible for processing verb phrases. The action processor works by first locating the verb within the verb phrase. Once found, the verb is compared with a list of possible verbs (and their synonyms) representing the set of actions which the avatar is capable of. Once a matching action is found, a set of defaults associated with that action is returned. The action processor then utilizes the services of the details processor to obtain any additional details about the action from other parts of the verb phrase. All the information associated with the actions is provided in a text input file.

Once all information about a particular command has been obtained, the command attributes must be resolved. Each command attributes is resolved individually. The command attributes potentially come from up to three locations.

Default attributes are returned by the command generator. These attributes serve only as defaults and can be overwritten by attributes returned by the details processor. The remaining two locations come from the details processor. The lower level of these attributes come from portions of the parse tree which are above or equal to the verb phrase sub-tree. These attributes override the command defaults but can, themselves, still be changed by attributes from portions of the sentence which appear within the verb phrase. Attributes from within the verb phrase are the highest level and override all others.

The final situation the action processor must handle is a sentence containing a conjunction within the first level of a verb phrase. This situation results in two verb phrases separated by a coordinating conjunction. In such a circumstance, each of the verb phrases is processed independently. Each of the verb phrases inherits the same details from higher portions of the tree but can interpret and possibly override them independent of each other. Separate commands are generated by each of the verb phrases. The result is two separate commands generated for the input sentence.

6. The command interpreter

An example of command to be interpreted is the following, where the attributes of the two commands, “walk” and “turn”, are speed, direction, and repetition (duration):

walk speed=9 direction=left repetition=5;
turn speed=5 direction=reverse repetition=1;

The input sentence is this case was: “John, run to the left and then turn back”. This is an example of a sentence containing two coordinated verb phrases, for which the semantic interpreter produced two consecutive commands.

All commands are returned in the form of a string. The command begins with the command action followed by each attribute, denoted by the attribute name, the equal sign, and the attribute value. Each attribute name/value pair is separated by a space. The command is completed with a semicolon. In the case where several commands are generated each command is separated by a semicolon. The command attributes are permitted to appear in any order.

7. The animation module

The animation module is responsible for both interpreting the commands and drawing the avatar to the screen. Several commands within a string are easily distinguished since each command ends with a semicolon. Once separated, each command can be interpreted separately.

Each individual command is processed by first obtaining the action from the front of the command. The remainder of the command provides the attributes that describe the specifics of the animation.

The simplified avatar (Figure 4), a stick with head, arms and legs, is a placeholder that will be replaced with a realistic avatar [5].

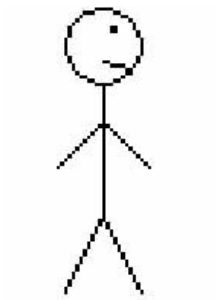

Figure 4: The simplified avatar

The animation module is the only hard coded part of the current system. It uses separated code to execute each command. The avatar is drawn by modifying class variables, so that the position of the avatar is changed each time the screen is redrawn. The ability to add arm and leg movement is also included, but is not used yet.

8. Results

Most predictable sentences were able to be processed fairly successfully. As with most natural language systems, the main difficulty is that the natural language used by the user is highly variable and can be highly ambiguous. In some respects, however, the expectation placed on computer systems is also extremely high. In general computer system systems are expected to work very precisely. Because of this, computers are often expected to outperform humans. This is, however, unrealistic in case of natural language processing because the computers do not have real intelligence and lack world knowledge. Moreover, there are situations that are truly ambiguous even for humans.

Since the system uses editable parameter files for matching commands and portions of the parse tree, new words, phrases and sentence structures are most easily handled by adding them to these text files. The input files allow for new synonyms of command verbs as well as new tree structures to be converted into command attributes.

9. Conclusions

There are a number of difficulties with natural language systems. Such difficulties can be reasonably overcome by restricting the domain of the problem. There are still, however, several problems occurring when there is considerable variation in the sentence structure.

The system was built in such a way that new commands and properties could easily be added to the system. This also allowed easy modification for new sentence structures and phrases. Our system made use of a generalized parser and grammar rather than a specific one. This design left the task of extracting meaning from the sentences to the semantic analyzer. This allowed for greater sentence variety without having to make modifications to the grammar used. Instead, new patterns could simply be added to the text parameter files.

A great deal of potential exists for natural language interfaces for various types of applications. The main solution is to restrict the domain enough to allow system not to be excessively hampered by the massive variety of natural human language.

10. Future work

One direction of future work is to add speech recognition capabilities so that the commands can be issued by voice. The speech recognition module will

include a domain model in order to reduce the number of recognition errors.

Another direction of future work is to replace our simple animation module with an existing full-scale animation module [5].

Our semantic interpreter can be easily extended to accommodate new commands, including arm and leg movement for example. The command generator needs to be adapted to generate commands and attributes for these commands according to the specific animation script language.

In future work our system could be expanded to include additional avatars. This would allow a user to be able to command two or more avatars separately. In such a case, a name parameter would be required. Each avatar will need to have a separate space of state and environment variables.. The input sentences need to be interpreted in the given context. Pronouns and other deictic expressions that refer to objects or avatars will need to be resolved.

The addition of new objects and avatars would probably also require a push-down automata to store previous commands. This would also a use to use commands such as:

Now do the same with the red ball.

Such a command would require knowledge of what was previously done and how it relates to the new object, in this case, the red ball. Such information could be stored by previous commands on a stack.

11. References

[1] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch, “Natural Language Interfaces to Databases: an Introduction”, Journal of Language Engineering", 1(1), pp. 29-81, 1995

[2] R. C. Perrault and B. J. Grosz, "Natural Language Interfaces", Annual Review of Computer Science, volume 1, J.F. Traub, editor, pp. 435-452, Annual Reviews Inc., Palo Alto, CA, 1986.

[3] D. Klein and C. D. Manning, “Accurate Unlexicalized Parsing”, The 41th Meeting of the Association for Computational Linguistics (ACL 2003), Sapporo, Japan, 2003.

[4] D. C. Petriu, X. L. Yang, and T. E. Whalen “Behaviour-Based Script Language for Anthropomorphic Avatar Animation in Virtual Environments”, International Symposium on Virtual and Intelligent Measurement Systems (VIMS 2002,), Mt. Alyeska Resort, USA, 2002.

[5] M. D. Petriu, N. D. Georganas, and T. E. Whalen “Development of a Humanoid Avatar in Java3D”, IEEE International Workshop on Haptic, Audio and Visual Environments and their Applications (HAVE 2003), Ottawa, ON, Canada, 2003.

[6] A. Popescu, O. Etzioni, H. Kautz, “Towards a Theory of Natural Language Interfaces to Databases”, IUI 2003 Maimi, Florida USA, 2003.