W/AGE : The Windsor Attribute Grammar Programming Environment

Richard A. Frost
School of Computer Science, University of Windsor, Windsor, Ontario, Canada N9B 3P4
richard@cs.uwindsor.ca

Abstract

Advances in speech-recognition and wireless-communication technology have generated a growing interest in hands-free, eyes-free, access to, and navigation between, remote data sources. Owing to the fact that conventional query languages such as SQL cannot be spoken, there is a need to develop natural-language processors to be used in conjunction with speech interfaces. Ideally, it should be possible for non-programmers to create their own speech-accessible natural-language applications that they can easily add to a SpeechWeb, in a similar manner to the creation of web pages using html. The W/AGE programming environment is a step towards this goal.

1. Introduction

Off-the-shelf speech-recognition technology and emerging standards such as the voice mark-up language VXML [9], have made it relatively easy to build speech-recognition front ends to remote applications. However, these front ends have only been used to access relatively simple applications through command-like languages.

More complex applications, such as database query processors, cannot yet be easily accessed through speech. This is a result of the fact that conventional query languages such as SQL cannot be spoken. Consequently, there is a need to build natural-language processors in order to integrate complex applications with the emerging speech-recognition technology.

Unfortunately, the construction of natural-language processors from scratch is very difficult even for experienced programmers. In order to overcome this problem, the Windsor Attribute Grammar programming Environment (W/AGE) has been developed in order to allow non-programmers to construct natural-language processors as executable specifications of grammars which define the syntax and semantics of the input language.

The skills required to use W/AGE do not require any programming experience, they are more related to understanding " grammar"as taught in high school.

W/AGE has been used to define a number of language-processors. One of these processors, called " solarman" , is an interface to a database containing facts about the planets, the moons that orbit them, and the people who discovered those moons. Example queries are:

```
Which of the moons that were discovered by
                              Hall orbit Mars?
Did Hall discover every moon that
                                 orbits Mars?
How many moons were discovered by Hall
                                  or Kuiper?
Which planets are orbited by a moon that
                 was discovered by Galileo?
Is every planet orbited by a moon?
Which planets are orbited by two moons?
Who was the discoverer of Phobos?
Which of the moons that orbit Mars were
                       discovered by Hall
```

Solar man can answer over 120,000 sensible questions and is constructed as an 800 executable specification of the input query language. An interface to the solar man query processor and a complete listing of the executable specification can be accessed at:

```
http://www.cs.uwindsor.ca/users/r/richard
                    /miranda/wage_demo.html
```

2. The structure of a W/AGE program

A W/AGE program takes a list of characters as input and returns a list of characters as output. In order to create a W/AGE program, the user has to define the language of the input strings. This definition includes the following:

1. A set of rules defining the syntax of the input language. These rules are written using standard Backus Naur Notation (BNF). For example:

```
|| snouncla ::= cnoun
               | adjs cnoun

|| adjs     ::= adj
               | adj adjs
```

Proceedings of the IEEE 2002 Symposia on Human Centric Computing Languages and Environments (HCC'02)
0-7695-1644-0/02 $17.00 © 2002 IEEE

2. A list of types of the semantic attributes (meanings) that are to be computed for different types of words and phrases of the input language. For example:

```
attribute  ::=
    SENT_VAL            bool
  | NOUNCLA_VAL         set_of_entities
```

3. A dictionary defining the words used in the input language. For each word, the user indicates the syntactic category, and the meaning of the word. For example:

```
("moon","cnoun",
            [NOUNCLA_VAL set_of_moon])
```

4. A set of syntax rules defining how compound phrases are made from simpler components. For example, a rule for simple noun_phrases might be as follows:

```
snouncla =
  cnoun
  $orelse
  structure (s1 adjs ++ s2 cnoun)
```

5. A set of semantic rules that define how the meaning of a compound phrase is computed from the meanings of its components. Each semantic rule is associated with a syntax rule. For example:

```
snouncla =
  cnoun
  $orelse
  structure (s1 adjs ++ s2 cnoun)
  [a_rule 1( NOUNCLA_VAL $of lhs )
         EQ intrsct1[  ADJ_VAL  $of  s1,
                    NOUNCLA_VAL  $of  s2]]
```

6. A set of definitions of the semantic operators/functions used in the semantic rules. For example:

```
intrsct1 [ADJ_VAL x, NOUNCLA_VAL y]
          = NOUNCLA_VAL (intersect x y)
```

7. A set of definitions of data. For example:

```
set_of_moon  = [18..53]
```

8. A set of definitions linking the integers representing entities to strings that can be used for output. For example:

```
("luna", 18)
```

The resulting set of definitions constitutes a program which, when executed, will take a string as input and, if the string is a member of the language defined by the syntax rules, will output a string that is computed using the semantic rules.

3. How W/AGE differs from other approaches

W/AGE programs are completely declarative and the order of the definitions does not matter. Also, there are no variables and there is no notion of updating any data structure. The user simply writes equations that define more complex structures in terms of simpler structures. The user does not have to worry about the flow of control through the program, and does not have to worry about the scope of variables or their current value.

The declarative nature of W/AGE makes it quite different from approaches in which natural-language processors are built in languages such as C or Java where a substantial part of the computation is based on the procedural paradigm. Even if the parser generators Yacc or CUP [3] are used in these languages, the semantics has to be coded procedurally and this requires sophisticated programming skills.

The W/AGE approach is somewhat related to the use of definite-clause grammars (DCG' s)[1] in the Prolog logic-programming language. However, DCGs only provide structure for defining syntax rules, and the coding of semantics within this structure is quite complex and certainly beyond the scope of anyone who has not had substantial experience in using Prolog.

W/AGE is an attribute-grammar programming environment and is closely related to other attribute-grammar programming systems such as those surveyed by Paaki [12]. However, most of these other systems were developed to facilitate the construction of compilers and are not well suited to the construction of natural-language processors. In particular, they cannot accommodate ambiguity, have no notion of a dictionary, cannot be used to define words in terms of other words or phrases and have no mechanism for transforming one syntactic structure to another through the definitions of Chomsky-like re-write rules. We show later that W/AGE accommodates all of these tasks.

The major difference between W/AGE and all other attribute grammar programming systems is that W/AGE code is directly executable. The attribute grammar is not compiled into code that is embedded in a conventional programming language, which is the approach used in most attribute grammar and parser-generator systems. In W/AGE every production rule in the grammar can be executed independently of all others except those which are used in its definition. This modularity facilitates construction and experimentation with the processor, and re-use of components.

4. Use of special features to define a rich language in terms of a core language

Rather than define the meanings of all words directly by listing their attributes, it is appropriate in some applications to treat only some of the words in this way. The meanings of other words can be defined by relating them to expressions that have equivalent meanings provided that these expressions include only those words and syntactic constructs that have already been defined. For example:

```
("anyone", "indefpron",
           meaning_of detph  "a person")
```

The ability to define the meaning of words in terms of the meaning of other phrases, and to rewrite phrases into others, are very powerful features of the W/AGE environment. One can begin by defining a processor for a " core" language in which all semantic rules are given explicitly, and then extend this to a richer language through the use of these features. The major advantage of this is that the core language can be chosen to facilitate the definition of semantic rules. More complex syntactic constructs, whose associated semantic rules may be problematic, can then be defined in terms of the core language.

5. Technical aspects of W/AGE

W/AGE is constructed as an extension to the pure functional programing language Miranda [13]. The underlying parser, which is hidden from the W/AGE user, is based on a top-down recursive-descent backtracking search strategy. As discussed by Koskimies [8], this approach provides significantly more modularity that alternative techniques.

Straightforward implementation of top-down parsing has exponential worst-case complexity. Frost and Szydlowski [7] have shown how memoization [11] can be used to reduce this complexity to cubic whilst maintaining pure functional propertiesof the parser.

The parsing strategy is also based on a recursive-programming technique developed by Burge [2] and subsequently refined by Wadler [14] in which each component language processor returns a list of results which are each processed by the subsequent component processor in turn.

Owing to the delayed (non-strict) computation in Miranda, no attribute values are calculated until the parser has completed its task, even though the semantic attribute rules are closely associated with the syntax rules that direct the parser. This is necessary in order to process complex queries in reasonable time.

W/AGE is being used to construct a number of natural-language front-ends to distributed data sources that are being added to SpeechWeb [6]. These data sources are accessible through the Internet from remote user-independent speech interfaces.

6. References

1. Abramson, H. and Dahl, V. *Logic Grammars* Springer-Verlag. pages 101– 102., 1989.
2. Burge, W. H., *Recursive Programming Techniques*, Addison-Wesley Publishing Company, Reading Massachusetts, 1975.
3. Cup (Constructor of Useful Parsers) `http://www.cs.princeton.edu /~appel/modern/java/CUP/CUPman.html`
4. Dowty, D. R., Wall, R. E. and Peters, S. *Introduction to Montague Semantics*. D.Reidel Publishing Company, Dordrecht, Boston, Lancaster, Tokyo, 1981.
5. Frost, R. A. and Boulos, P., " An Efficient Compositional Semantics for Natural– Language Database Queries with Arbitrarily– Nested Quantification and Negation" , accepted for the *Fifteenth Canadian Conference on Artificial Intelligence*, Calgary May 2002.
6. Frost, R. A.. and Chitte, S. " A New Approach for Providing Natural-Language Speech Access to Large Knowledge Bases" ,*Proc. of PACLING ' 99,The Conference of the Pacific Association for Computational Linguistics,* 1999, pp. 82– 90.
7. Frost, R. A. and Szydlowski, B. " Memoizing purely-functional top-down backtracking language processors" ,*Science of Computer Programming* (27), 1996, pp.263– 288.
8. Koskimies, K. Lazy recursive descent parsing for modular language implementation. *Software Practice and Experience*, 20 (8) 749– 772, 1990.
9. Lucas, B. " VoiceXML for Web-based distributed conversational applications" , *Communications of the ACM*, Vol.43, No.9, 2000, pp. 53– 57.
10. Montague, R.in *Formal Philosophy: Selected Papers of Richard Montague*, edited by R. H. Thomason. Yale University Press, New Haven CT, 1974.
11. Norvig, P. Techniques for automatic memoisation with applications to context-free parsing, *Computational Linguistics* 17 (1) pp. 91 - 98, 1991.
12. Paaki, J. " Attribute Grammar Paradigms — a High-Level Methodology in Language Implementation" ,*ACM Computing Surveys*, Vol. 27, No. 2, 1995, pp. 196– 256.
13. Turner, D. A lazy functional programming language with polymorphic types. *Proc. IFIP Int. Conf. on Functional Programming Language and Computer Architecture*, Nancy, France. Springer Lecture Notes 201, 1985.
14. Wadler, P. How to replace failure by a list of successes. In *Functional Programming Languages and Computer Architectures*, edited J. P. Jouannaud. Lecture Notes in Computer Science 201, Springer-Verlag, Hedelberg, pp. 1131985.