Evaluation of Recent Speech Grammar Standardization Efforts

Tom Brøndsted

Center for PersonKommunikation
Institute of Electronic Systems
University of Aalborg, Denmark
tb@cpk.auc.dk

Abstract

The “Voice Browser “ activity within the W3C consortium addresses the need for standards for speech grammars, dialogue descriptions etc. in distributed systems. This paper discusses the consortium’s recent speech grammar working draft specification. The W3C specification is based on the Java[tm] Speech Grammar Format (JSGF) defined by Sun Microsystems and is - with all good and bad qualities - characterized by traditions of *formal language* theory. In a constructive spirit, we suggest some possible improvements based on *natural language* theory. The suggestions concern compound feature-based semantic presentations, lexicon structure, ambiguity, and exhaustive parsing,

1. Introduction

Distributed spoken dialogue systems, e.g remote server-based architectures where end-users access web-services through handheld devices, presuppose standards for communication protocols, grammars, dialogue descriptions etc. As regards formats, standardization efforts have taken place within the “Voice Browser“ activity of the W3C consortium [1]. So far, this has resulted in specifications for “Speech Recognition Grammar”, “Stochastic Language Models”, “Speech Synthesis Markup Language”, and “Natural Language Semantics Markup Language”. Of the core formalisms, the “Dialogue Markup Language” still has to be specified.

The present paper discusses the “Speech Recognition Grammar” working draft specification [2] which is largely a XML-based version of the Java[tm] Speech Grammar Format (JSGF) defined by Sun Microsystems [3]. The main idea behind JSGF and W3C’s XML-based counterpart is to control the entire process of speech understanding (speech in, semantics out) with *a single grammar*. This is also the idea underlying a suite of Natural Language Processing tools developed at Center for PersonKommunikation [4]. As opposed to systems that use *two grammars*, one for recognition (e.g. an *n*-gram) and another for syntactic and semantic parsing, a system based on a double-functioning grammar is easier to maintain.

The W3C speech grammar specification, and the JSGF model underlying it, is clearly characterized by traditions of *formal language* theory. In this paper, we will introduce viewpoints from the related tradition of *natural language* theory. Historically, the two disciplines have a common base in the Chomsky-tradition. However, many algorithms and paradigms developed within the theory of natural languages are not commonly known in formal language theory (and vice versa). For instance, the W3C working group discusses the problem of expressing morphological agreement in grammar rules and notes that it "is not aware of any existing grammar format that supports this kind of morphological inference by a speech recognizer for a grammar" [2]. In natural language theory, such a grammar format has been known for 15-20 years. The format is called "feature based" or "unification" grammar and will be discussed below.

Based on own experiences with an approach aiming at controlling speech understanding via such a feature based unification grammar formalism, we discuss the JSGF model (section 2) with focus on semantic representation (2.1), "missing" lexicon structure (2.2), and ambiguity and exhaustive parsing (2.3). The "alternative" unification-based approach is briefly outlined (section 3) answering some of the most obvious questions to ask from the point of view of formal language theory: Equivalence to finite state grammars (3.1) and efficient parsing algorithms (3.2). The conclusion (section 4) draws the attention to some drawbacks of the unification-based approach compared to the JSGF-model.

2. The JSGF-model

JSGF (and its XML-based counterpart specified by W3C) is an EBNF-equivalent “traditional style” grammar including the normal unary and binary operators: iteration (zero or more, one or more), optionality, 'and', 'or'. In addition JSGF includes: (1) A Java-adapted style for specifying grammar URLs, importing grammars or grammar rules, specifying rules as being public or private etc. (2) “semantic tags” intended mainly for handling situations where two or more words "mean" the same thing (synonymy, multilinguality, e.g. "hi", "hello", "guten Tag", "bon jour"), and (3) weights enabling implementation of simple n-grams or statistical "fullgrams".

The core of the interface for retrieving results from JSGF-controlled recognition (defined in the Java package javax.speech.recognition) includes (1) A 1-best/n-best-list; and for each item in the list: (a) a list of tokens (“words”, i.e. the sentence/utterance), (b) a list of semantic tags, (c) the name of JSGF grammar accepting input, and (d) the name of public rule (top-symbol, axiom) accepting input.

The core facility of JSGF for generating semantics lies in the tagging scheme. However, as tagging has shortcomings even in simple systems, a more enhanced “action tag” concept has been proposed [5] [6]. Action tags can be compared (and is most likely inspired by!) the "semantic actions" of a yacc-parser. The scripting language for implementing action tags is ECMA.

Our discussion will include the action tag proposal as well.

2.1. Semantic representation in JSGF

The JSGF-model suggests that the semantic representation level generated by tags or action tags can be used as language-independent meaning-representations. This is an extremely

important issue, since

1. *Cross-language portability*: A language independent meaning representation allows applications to be ported e.g. from English to German simply by substituting the English grammars with German ones. The actual dialogue description (e.g. implemented in the “Dialogue Markup Language” still to be specified by W3C) needs no modifications.

2. *Distribution*: The JSGF-model allows grammars to be URLs. This points in the direction of a future scenario where separate components of spoken language applications are implemented and maintained at various places. In such a context, we need a language independent representation to specify what a specific grammar (or class of grammars, e.g date-grammars in any language) must return.

Ideally, this requires a very powerful representation formalism, e.g. a formalism conforming to 1st order predicate logic or "stronger". However, JSGF addresses simple dialogue systems. The current JSGF version suggests simple tagging lists, whereas the action tag proposal implies nested structures that can be considered a tiny subset of 1st order predicate logic.

2.1.1. Tagging list scheme

The scheme can be compared to part-of-speech tagging approaches in natural language processing. However, the JSGF-scheme is different in two respects:

1. In NLP, tagging is normally used as pre-processing for actual syntactic parsing (generating hierarchic tree structures); in JSGF the flat lists of tags is the final result.

2. In NLP, tagging is used for resolving lexical ambiguities; in JSGF tagging is mainly a replacement mechanism for (cross-language) synonyms (hi=hi, hello=hi, guten Tag=hi etc.)

The obvious shortcomings are:

1. Lists of tags reflect the word (token) order of the spoken utterance. It is not possible to generate the same lists of tags for e.g. "June the 1st" and "the 1st of June".

2. In case of utterances where two or more sub-phrases are accepted by the same syntactic rule, e.g. "from <date> to <date>", lists of tags become complicated and require post-processing.

Witout doubt, most systems using the tagging-scheme of JSGF-model will end up with (ad-hoc) routines for post-processing of tags that are similar to syntactic parsing in conventional NLP systems.

2.1.2. Action Tag proposal

The action tag proposal has a clear resemblance to the "semantic actions" in a yacc- (or bison-) parser. Each syntactic rule can be assigned an action building e.g. a (nested) feature-value paired structure and typically incorporating previously built structures from daughter nodes. Action tags are implemented using the ECMA scripting language allowing almost "anything" to happen within an action: Construction of arrays and objects, slot-filling, numerical operations, manipulation of strings etc.

Further, the proposed scheme can be compared to the ability of a unification-based formalism to "percolate" features from child nodes to parent nodes (see section 3.). In both cases the sentence (utterance) is not seen merely as a sequence of words/tokens but as a hierarchical constituent structure (a parse tree).

ECMAScript was originally designed for use in a graphical web browser environment. However, in many XML-based formalisms including VoiceXML, the script-language is used as escape-level when miscellaneous computing is required. In JSGF, ECMA is an alien substance. When embedded in the formalism, the usual problems like compile-time validation and runtime exceptions arise.

In the unification approach described in section 3., the nested structures are created within the parsing paradigm as a side effect. However, formally the unification concept cannot generate any representation that is not possible with action tags implemented in ECMA.

2.2. Lexicon

Like most grammar formalisms used in formal language design, JSGF involves no actual "lexicon". All rules are "syntactic" and the lexicon is embedded in these rules as terminals, e.g.

```
<noun> orange | ducks;
<adjective> orange | big;
<verb> ducks | works;
```

In natural language theory, the lexicon is usually separated from the syntactic rules, e.g.

```
{lex=orange, cat=noun}
{lex=orange, cat=adjective}
 etc.
```

where the lexical rules also define semantic features (objects for "action tags") to be percolated up through the parse tree.

There are a number of reasons for separating lexicon from syntax: "Linguistic tradition", parsing algorithm (reducing the number of different terminals in syntactic rules to a small number of word classes enables efficient top-down prediction in the Early chart parsing paradigm, see section 3.2), etc.

However, voice-based services are typically database-access systems with a strong dependency between lexicon and the application. If the database of the system is updated, the lexicon must be modified whereas the syntactic rules remain unaffected. Further, when using a compound feature based lexicon, the lexical rules can in many cases automatically be derived from the underlying database accessed by the user. The following example can automatically be derived from the attributed database available for a service provider selling flight tickets

{lex=Manchester,cat=city,country=UK,airport=MAN}.
{lex=Leeds,cat=city,country=UK,airport=LBA}.
{lex=London,cat=city,country=UK,airport=LHR}.
{lex=London,cat=city,country=UK,airport=LGW}.
{lex=London,cat=city,country=UK,airport=STN}.
{lex=Heathrow,cat=city,country=UK,airport=LHR}.
{lex=Aalborg,cat=city,country=DK,airport=AAL}.
.....

This approach is explored in [7].

2.3. Ambiguity and exhaustive parsing

In formal language design, ambiguity is normally avoided. The most likely reasons are:

1. The tradition from programming language design; programming languages are never ambiguous.

2. Parsing algorithm (e.g. a yacc-parser cannot handle ambiguous grammars)

The typical way to solve ambiguity problems is to define precedence rules. This is the solution suggested in the W3C speech grammar specification (section 2.8 in [2]). However, there are strong arguments for allowing ambiguity:

1. Natural language *is* ambiguous. Even the word "London" is ambiguous in the example in section 2.2. (which airport in London?)

2. In a distributed scenario where grammars or even grammar fragments are designed and maintained by various sub-providers (see section 2.1) ambiguity can arise at any time

3. If parts of grammars, e.g. lexical rules, are derived automatically from databases (section 2.2) ambiguity can arise at any time

4. There are efficient algorithms for exhaustive parsing (among these the Early chart parsing paradigm described in section 3.2)

5. In a concrete application, it is no more "difficult" to handle alternative results from a parser than an n-best list from a recogniser or multiple results from a database query.

3. Unification-based approach

Unification grammars are not commonly known among specialists in formal languages (though the standard textbooks usually mention the approach with a few lines). The core of the unification grammar formalism e.g. as introduced in [8] [9] can briefly be described starting from the well-known EBNF-scheme:

1. In EBNF, the core structure is the *symbol* (a "string"), e.g. a syntactic rule consists of a left-hand-side symbol and a sequence of right-hand-side symbols. In unification grammars the corresponding core structure is the *feature set* (a set of features each of which consists of a value assigned to an attribute)

2. In EBNF, the fundamental operation performed on two symbols is *comparison*. In unification grammars the operation performed on two feature sets is *unification*. Comparison is a Boolean function, whereas unification returns a *new* feature set if successful.

Traditionally, the structure sharing mechanism is brought out as the superior facility of unification grammars over traditional symbol-based grammars. Rules may incorporate variables pointing to the same structure. This enables the simple solution to the problem of describing agreement (see section 1.), e.g.

{cat=sentence} [
    {cat=pronoun, number=$A, person=$B}
    {cat=verb, number=$A, person=$B }
].

Further, variables can be used to percolate (nested) features bottom-up in a way similar to the action tag proposal of the JSGF-model. However, the percolation takes place within the parsing paradigm as a result of the unification operations.

3.1. Regular grammar equivalence.

The Viterbi-decoding scheme of state-of-the-art speech recognition technology presupposes a finite state grammar network. Hence, the suitability of a unification grammar (or any other formalism) for control of speech understanding depends on its ability to conform to a weakly equivalent regular grammar. The conditional regular grammar equivalence of unification grammars can be deduced from the statements

1. A unification grammar where features take values taken from a finite set can be rewritten as a weakly equivalent context-free grammar utilizing a finite set of symbols.

2. A context-free grammar that involves no embedded recursions (in formal language theory: A_nB_n problems) can be rewritten as a weakly equivalent finite state grammar.

Statement 1. is true for the unification formalism used in [4]. In practical terms, the grammar converter of the system computes the set of possible values for all variables by observing the percolation paths in the grammar. For instance, the possible values of $A in the feature set

{cat=noun, number=$A}

may be 'singular' and 'plural'.

Statement 2. imposes restrictions on recursions in the grammar and is valid also for grammars of the JSGF-type. The JSGF-model explicitly prohibits not only embedded recursions but also left-recursions. The latter may be due to the background of the designers in formal language theory (many efficient parsing algorithms like shift-reduce parsers exclude left recursion). The system described in [4] rewrites left and right recursions to fully equivalent iterations ("loops") and performs

approximations in the rare cases where embedded recursions are encountered.

3.2. Algorihms

Many efficient parsing algorithms developed to be used with context-free grammars can be modified to work also on unification grammars. A widely used paradigm is variants of the Early chart parsing algorithm [10]. Unlike e.g. the shift-reduce paradigm, this algorithm is *general* (involves no restrictions as regards recursion) and has a theoretical worst-case behavior of $O(N^3)$ where N is the number of tokens in the utterance.

4. Conclusion

We have suggested some improvements to the W3C speech grammar specification based on the traditions of natural language theory and own experiences with a unification-based approach. Feature-based representations of semantics (which is already included in the action tag proposal), lexical rules separated from syntax, support for ambiguous grammars by means of exhaustive parsing paradigms.

The evaluation does not insist on the unification-based approach being preferable to the JSGF-model (with action tags).

1. Any java-programmer can with ease implement simple grammars in JSGF and any web-designer masters the ECMA scripting language For traditionally trained programmers, unification grammar is not easily accessible.

2. Deriving finite state grammars from unification grammars can be critical if features have large sets of possible values.

3. Often semantic representations generated by a unification-grammar need post-processing in the form of simple numerical operations etc. (e.g. when parsing natural numbers). Such operations are note available within the unification paradigm.

The unification-based approach has been used together with a design environment where the actual unification scheme was hidden for the user. This approach is described in [11] [12].

5. References

[1] W3C: Voice Browser Activity, http://www.w3.org/Voice/

[2] W3C: *Speech Recognition Grammar for the W3C Speech Interface Framework*. W3C Working Draft 3 January 2001, http://www.w3.org/TR/speech-grammar/

[3] The JavaTM Speech API http://java.sun.com/products/java-media/speech/

[4] T. Brøndsted: The CPK NLP Suite for Language Understanding. *Proc. EUROSPEECH, European Conference on Speech Communication and Tehnology*, Budapest, 1999. http://cpk.auc.dk/~tb/articles/eursp99.pdf

[5] B. Lucas, W. Walker, A. Hunt: *ECMAScript Action Tags for JSGF*. Proposal September 1999. http://x-language.tripod.com/ECMAScriptActionTagsforJSGF.html

[6] Sun Microsystems: *Action Tags for JSGF, version 0.5* Internal communication with Sun Microsystems, Palo Alto, CA, 1998

[7] T. Brøndsted: The Linguistic Components of the REWARD Dialogue Creation Environment and Run Time System. *Proceedings from the 4th IEEE Workshop on Interactive Voice Technology for Telecommunications Applications*, Turino 1998

[8] [9] S.M. Shieber: Using Restriction to Extend Parsing Algorithms for Complex-feature-based Formalims". *Proceedings of the 22nd Annual Meeting of the Association for Computational Linguistics*, Chicago 1985, pp.145-152.

[9] S.M. Shieber: An Introduction to Unification based Approaches to Grammar. *CSLI Lecture Notes Series, Number 4, Stanford CA 1986*

[10] J. Earley: An efficient context-free parsing algorithm. *Communications of the ACM*, Vol. 13

[11] T. Brøndsted: Implementing a Task Specific Grammar for Recognition and Parsing. *ESCA Tutorial and Research Workshop: Interactive Dialogue in Multi-Modal Systems*. Kloster Irsee, 1999, http://kom.auc.dk/~tb/articles/klosterirr99b.pdf

[12] T. Brøndsted: Non-Expert Access to Unification based Speech Understanding. *Proc. ICSLP*, Sydney 1998, http://cpk.auc.dk/~tb/articles/icslp98a.pdf