# A Graphical Tool for Handling Rule Grammars in Java Speech Grammar Format

**Kallirroi Georgila, Nikos Fakotakis, George Kokkinakis**

Wire Communications Laboratory
Electrical and Computer Engineering Dept.
University of Patras
265 00 Rion, Patras, Greece
{rgeorgil, fakotaki, gkokkin}@wcl.ee.upatras.gr

**Abstract**

This paper describes a graphical tool used for generating and depicting rule grammars in the Java Speech Grammar Format (JSGF), which has been developed in the framework of the EC-funded research project GEMINI (Generic Environment for Multilingual Interactive Natural Interfaces, IST-2001-32343). A vocabulary builder component that produces the phonetic transcription of the words included in the grammar file is also incorporated into the tool. Currently, the tool supports embedded grapheme-to-phoneme conversion only for Greek in SAMPA format. However, a language-independent function is included that enables the user to write context-dependent rules for symbol conversions (both grapheme-to-phoneme and phoneme-to-grapheme). Manual vs. tool-based handling of grammars are compared and evaluated in terms of time required for grammar creation and efficiency.

## Introduction

The goal of this work is to describe a graphical tool used for generating and depicting rule grammars in the Java Speech Grammar Format (JSGF). Developed by Sun Microsystems, JSGF defines a platform-independent, vendor-independent way of describing one type of grammar, a rule grammar (also known as a command and control grammar or regular grammar). JSGF uses a textual representation that is readable and editable by both developers and computers, and can be included in Java source code. JSGF is simple and easy to understand and thus has become a standard (Sun Microsystems, 1998).

This tool has been developed in the framework of the EC-funded research project GEMINI (Generic Environment for Multilingual Interactive Natural Interfaces, IST-2001-32343) (www.gemini-project.org). The project has two main objectives: First, the development of a flexible Application Generation Platform (AGP) able to produce user-friendly mixed-initiative interactive multilingual and multi-modal dialogue interfaces to databases with a minimum of human effort, and second, the demonstration of the platform's efficiency through the development of two different applications based on this platform.

The AGP will exploit the structured information contained in databases of information services and sets of sample dialogues, to create customized dialogue models (i.e. dialogue scripts, grammars and lexicons) for specific information services with the minimum of human effort.

The tool presented in this paper is part of the AGP and handles grammars for the speech modality. Moreover, a vocabulary builder component that produces the phonetic transcription of the words included in the grammar file is also incorporated into the tool. Currently, the tool supports embedded grapheme-to-phoneme conversion only for Greek in SAMPA format. However, a language-independent function is included that enables the user to write context-dependent rules for symbol conversions (both grapheme-to-phoneme and phoneme-to-grapheme).

In the following sections, the tool's functionality will be described in detail. In addition, the generation of grammars using the tool will be compared and evaluated against their manual development in terms of time and efficiency.

## Java Speech Grammar Format

A rule grammar specifies the types of utterances a user might say (a spoken utterance is similar to a written sentence). For example, a simple grammar for giving the arrival city in a flight reservation system would include sentences like "I would like to travel to Lisbon", "my destination is Paris", etc.

A single file defines a single grammar. The definition grammar contains two parts: the grammar header and the grammar body. The grammar header includes a self-identifying header, declares the name of the grammar and optionally declares imports of rules from other grammars. The body defines the rules of the grammar, some of which may be public, that is, can be accessed by other grammar files. The two patterns of rule definitions are:

<ruleName> = ruleExpansion ;
public <ruleName> = ruleExpansion ;

The rule expansion defines how the rule may be spoken. It is a logical combination of tokens (text that may be spoken) and references to other rules. An expansion defines how a rule is expanded when it is spoken – a single rule may expand into many spoken words plus other rules which are themselves expanded.

For the aforementioned destination paradigm the corresponding JSGF file would be as follows:

#JSGF V1.0 ISO8859-1 en_Uk;
grammar journey;

<want> = I (want to | would like to | must)
<travel> = fly | go | travel
<polite> = please | if possible
<arrival> = Lisbon {destination="Lisbon"}| Paris {destination= "Paris"} | Athens {destination= "Athens"}

<want_dest> = <want> <travel> <to> | my destination is
<destination> = [<want_dest>] <arrival> <polite>*

Note that rule names are contained within angle brackets. A vertical bar '|' marks alternatives, '()' parentheses are used for grouping, and '[]' for optional grouping. A rule expansion followed by the asterisk symbol (Kleene star) indicates that the expansion may be spoken zero or more times whereas a plus symbol (not depicted in the above example) indicates that the expansion may be spoken one ore more times. Quotes can be used to surround multi-word tokens and special symbols, e.g. the "New York" subway. A multi-word token is useful when the pronunciation of words varies because of the context. Multi-word tokens can also be used to simplify the processing of results, for example, getting single-token results for "New York", "Sydney" and "Rio de Janeiro". Tags, that is, strings delimited by curly braces '{}' provide a mechanism for grammar developers to attach application-specific information to parts of rule definitions. Tag attachments do not affect the recognition of a grammar. Instead, the tags are attached to the result object returned by the recogniser to an application (Sun Microsystems, 1998). Thus when Lisbon is recognised the application variable 'destination' will be filled with the value "Lisbon".

## Grammar Tool Description

For each grammar the internal structure in which all the information is stored is as follows:

Grammar:
- Grammar name
- Grammar type
- Grammar language
- List of rules

Rule:
- Rule name
- Rule type
- List of attributes
- List of tokens

Rule type: *Public* || *Private*

Attribute:
- Attribute name
- Attribute type

Attribute type: *Integer* || *String* || *Boolean*

Token:
- List of contents
- Contents combination type
- List of semantics

Contents combination type: *AND* || *OR*

Semantic:
- Attribute name
- Attribute value

Attribute value:
- Value
- Attribute value

Content:
- Name
- Optional flag ( *true* || *false*)
- Kleene star flag (*true* || *false*)
- Plus symbol flag (*true* || *false*)
- Quotes flag (*true* || *false*)
- Content value

Content value: *Token* || *String* || *Rule*

To illustrate the functionality of the tool we will show how the example rule grammar in section "Java Speech Grammar Format" may be generated. The first step is to give a name to the grammar and then select its type and language. Currently only "JSGF" type is supported. However, the grammar file is first saved in XML format and then transformed to JSGF. This makes it easier to incorporate in the future other formats as well. Four languages are supported English, Greek, German, and Spanish.

The next step is to create a new rule, select its type "public" or "private" and its attributes where semantic information will be stored.

Moreover, the tokens of the rule must be created. The tokens of a rule are combined with "OR". The rule <want> is formed by one token that has two contents, the string "I" and a token "(want to | would like to | must)" that consists of 3 strings (see figures 1, 2 and 3). More

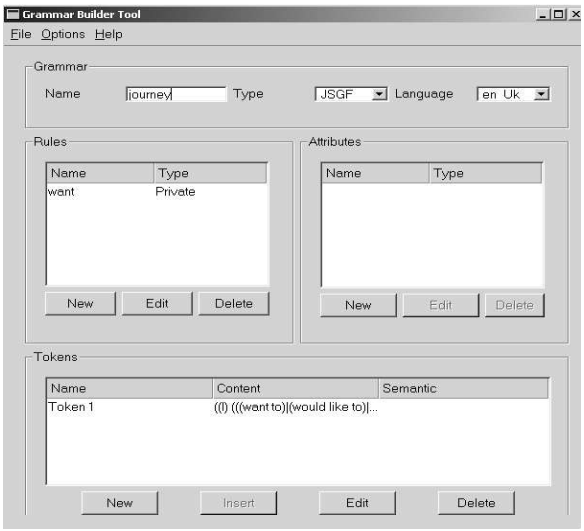


Figure 1: Rules consist of tokens

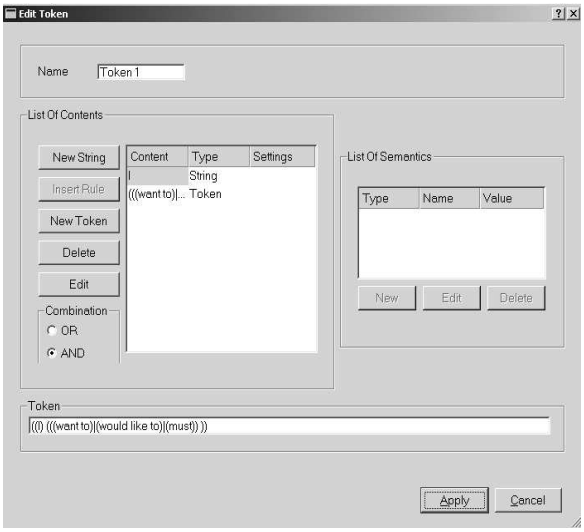


Figure 2: A token is formed as part of a rule

specifically, a token may include strings, other tokens or rules. In addition, there is an option to choose whether the contents of the token should be linked with an "AND" or an "OR". The grammar developer is free to type words and strings of words or insert them from vocabularies that can be loaded from "Menu → Options". Multiple vocabularies can be active simultaneously (see Figure 4). There are no semantic attributes for the rule <want>.

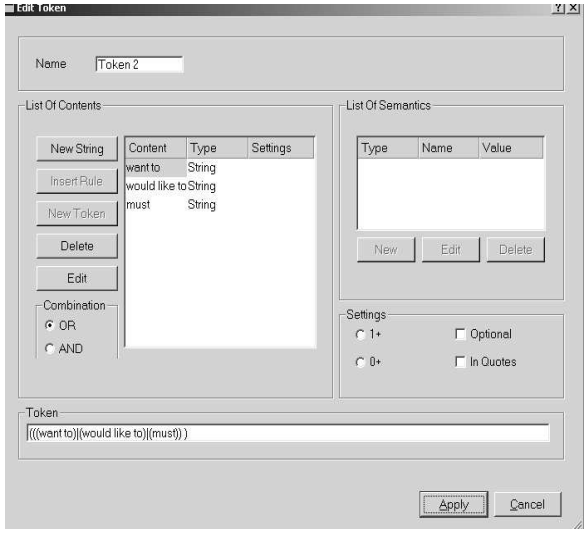


Figure 3: A token may be formed as part of another token

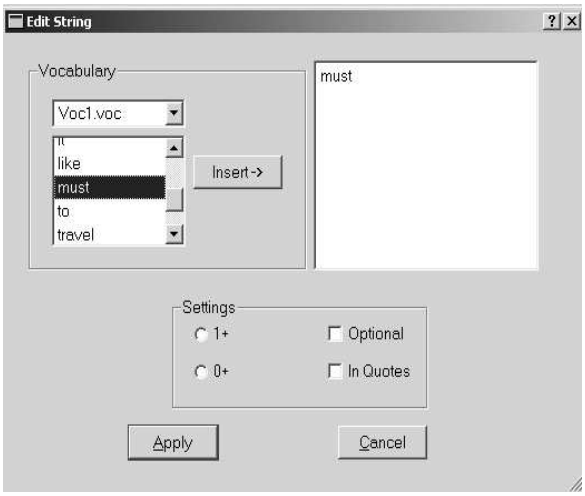


Figure 4: Words are inserted either manually or using a loaded vocabulary

The contents of a token, that is, strings, rules or other tokens may be tagged as optional or enclosed in quotes. They may also be expanded zero or more times (Kleene star symbol in JSGF) or one or more times (plus symbol in JSGF).

In the same way the rules <travel>, <polite> and <want_dest> are formed. The rule <arrival> has one semantic attribute i.e. destination, and as many tokens as the cities that are included in the grammar. In most cases, city names, surnames, airline companies names and other similar information is stored in databases and may consist of hundreds of records. Thus to help the grammar developer incorporate all this information in one step, the tool supports the "Insert Tokens" option. The user can load a file that contains all database records, link it to the correct semantic attribute and decide on the type of the tokens (optional, in quotes, expanded zero or more times, expanded one or more times). Figure 5 depicts the insertion from file procedure and Figure 6 the resulting structure.

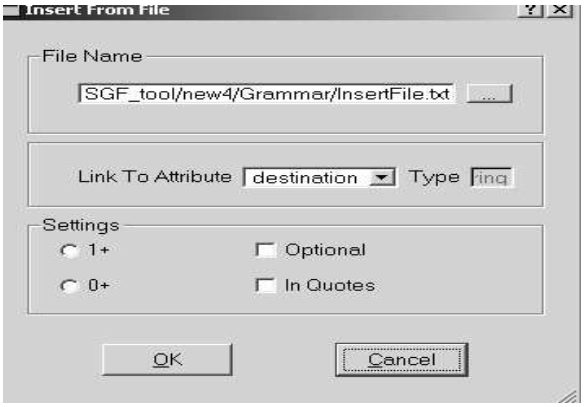


Figure 5: Insertion of multiple tokens simultaneously

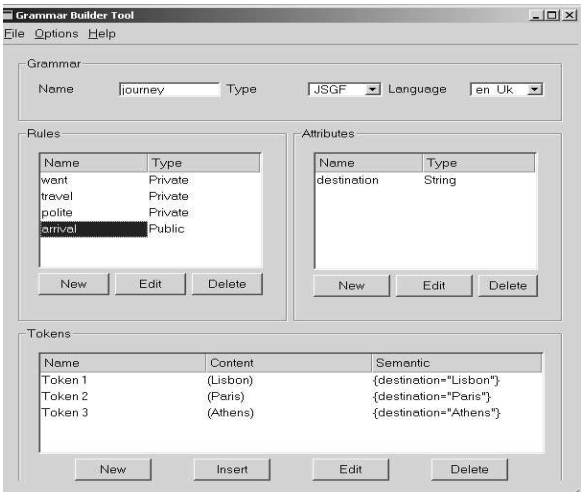


Figure 6: Multiple tokens loaded with their semantic attribute

In Figure 7, it is shown how the rule <destination> is generated. The rule has been slightly modified compared to the example in the Java Speech Grammar Format section to illustrate another available option, that is, to have a semantic attribute of one rule that refers to the semantic attribute of another rule. Thus the rule <destination> is reformed as follows:

<destination> = ([<want_dest>] <arrival> <polite>*) {final_destination = arrival.destination}

The redundant parentheses that appear in most examples assist in building the internal grammar structure and are discarded when the final JSGF file is saved. Moreover, the user is allowed to preview the complete grammar structure before saving it as well as load an already available grammar file.

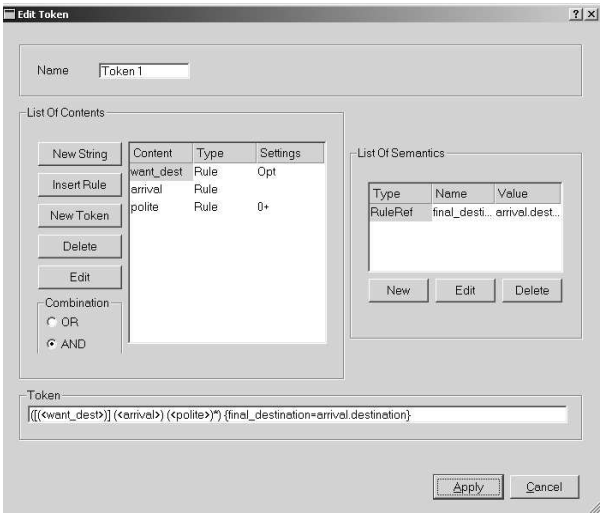


Figure 7: The semantic attribute of one rule refers to the semantic attribute of another rule

## Vocabulary Builder

A vocabulary builder component that produces the phonetic transcription of the words included in the grammar file is also incorporated into the tool. Currently, the tool supports embedded grapheme-to-phoneme conversion only for Greek in SAMPA format (SAM-PA). However, a language-independent function is included that enables the user to write context-dependent rules for symbol conversions (both grapheme-to-phoneme and phoneme-to-grapheme).

The structure of the rules is as follows: $L_1$, $L_2$, …, $L_k$, $S$, $R_1$, $R_2$, …, $R_n$ where $L_i$ i=1, …, $k$ is the left context of the rule, $S$ is the class, which includes the symbols or symbol combinations for conversion, and $R_p$ $p$=1, …, $n$ is the right context of the rule. The values of $k$ and $n$ could vary according to the language and the way the designer of the rules has decided to form them. A similar rule definition mechanism has also been used for building phonological rules to enhance recognition performance (Georgila et al., 2002). An example is as follows ($k$=1 and $n$=1):

-, *ND* *d*, *w#*

This rule says that the phonetic transcription of *ND* is *d* when an empty string is preceded and the class *#w* follows. A class may include one or more symbols or symbol combinations. That is, if *#w*=(*A*, *E*, *I*, *O*, *U*) this means that when *ND* is preceded by an empty string and followed by *A*, *E*, *I*, *O* or *U* it will be converted to *d*. Note that in the above example upper case letters indicate the graphemic form and lower case ones denote the phonetic transcription. This rule mechanism works well for the Greek language in which there are not many grapheme-to-phoneme or phoneme-to-grapheme conversion rules. However, it has not been tried for other languages with more complex phonological features.

## Evaluation

Three JSGF grammar developers were asked to use and evaluate the tool. They had to build grammars for 15 dialogue states both manually and with the aid of the tool. For 5 of these dialogue states they were provided with grammar files of similar tasks. In 3 of the remaining 10 dialogue states they were asked to update half-completed grammars. Thus they had to build from scratch 7 grammars. Grammars varied in complexity from very simple to quite complicated ones. Then for each of the grammars they were asked to fill a questionnaire with the following "Yes/No" questions:

*Did you have to build the grammar from scratch?*
*Did you find it easier to build the grammar using the tool rather than manually?*
*Did you make any mistakes in the manual grammar building?*
*Did it take you more time to create the grammar manually?*
*Was a grammar for a similar task available?*

The developers answered that they found it easier to build grammars manually when their complexity was low and when they were provided with a grammar file in a similar task. This is because it was easier to make changes manually in the text file of the similar task grammar. However, when they were asked to deal with grammars of new tasks they all agreed that the use of the tool proved more efficient than manual creation in both update and from scratch building. Finally, using the tool prevented them from making mistakes that sometimes are hard to locate when the grammar has been developed manually.

## Summary and Conclusions

This work describes a graphical tool used for handling rule grammars in the Java Speech Grammar Format (JSGF), which has been developed in the framework of the EC-funded research project GEMINI (Generic Environment for Multilingual Interactive Natural Interfaces, IST-2001-32343). Grammar files are first saved in XML format and then transformed to JSGF. This makes it easier to incorporate other formats as well in the future.

A vocabulary builder component that produces the phonetic transcription of the words included in the grammar file is also incorporated into the tool. Currently, the tool supports embedded grapheme-to-phoneme conversion only for Greek in SAMPA format. However, a language-independent function is included that enables the user to write context-dependent rules for symbol conversions (both grapheme-to-phoneme and phoneme-to-grapheme). Evaluation on manual vs. tool-based handling of grammars proved the efficiency of the tool in terms of time required for grammar generation and correctness.

## References

Georgila, K., Fakotakis, N., Kokkinakis, G. (2002). Large Vocabulary Search Space Reduction Employing Directed Acyclic Word Graphs and Phonological Rules. International Journal of Speech Technology, Kluwer Academic Publishers, Vol. 5, No. 4, pp. 355-370.

SAM-PA, Standards, Assessment, and Methods: Phonetic Aplhabets. http://phon.acl.ac.uk/home/sampa/home.htm

Sun Microsystems (1998). Java Speech Grammar Format Specification Version 1.0.