A Development Environment for Building Grammar-Based Speech-Enabled Applications

Elisabeth Kron[1], **Manny Rayner**[1,2], **Marianne Santaholma**[1], **Pierrette Bouillon**[1]

[1] **University of Geneva, TIM/ISSCO**
40 bvd du Pont-d'Arve
CH-1211 Geneva 4, Switzerland
elisabethkron@yahoo.co.uk
Marianne.Santaholma@eti.unige.ch
Pierrette.Bouillon@issco.unige.ch

[2] **Powerset, Inc.**
475 Brannan Street
San Francisco, CA 94107
manny@powerset.com

Abstract

We present a development environment for Regulus, a toolkit for building unification grammar-based speech-enabled systems, focussing on new functionality added over the last year. In particular, we will show an initial version of a GUI-based top-level for the development environment, a tool that supports graphical debugging of unification grammars by cutting and pasting of derivation trees, and various functionalities that support systematic development of speech translation and spoken dialogue applications built using Regulus.

1 The Regulus platform

The Regulus platform is a comprehensive toolkit for developing grammar-based speech-enabled systems that can be run on the commercially available Nuance recognition environment. The platform has been developed by an Open Source consortium, the main partners of which have been NASA Ames Research Center and Geneva University, and is freely available for download from the SourceForge website[1]. Regulus has been used to build several large systems, including Geneva University's MedSLT medical speech translator (Bouillon et al., 2005) and NASA's Clarissa procedure browser (Rayner et al., 2005b)[2].

Regulus is described at length in (Rayner et al., 2006), the first half of which consists of an extended tutorial introduction. The release also includes extensive online documentation, including several example applications.

The core functionality offered by Regulus is compilation of typed unification grammars into parsers, generators, and Nuance-formatted CFG language models, and hence also into Nuance recognition packages. Small unification grammars can be compiled directly into executable forms. The central idea of Regulus, however, is to base as much of the development work as possible on large, domain-independent resource grammars. A resource grammar for English is available from the Regulus website; similar grammars for several other languages have been developed under the MedSLT project at Geneva University, and can be downloaded from the MedSLT SourceForge website[3].

Large resource grammars of this kind are overgeneral as they stand, and it is not possible to compile them directly into efficient recognisers or generators. The platform, however, provides tools, driven by small corpora of examples, that can be used to create specialised versions of these general grammars using the Explanation Based Learning (EBL) algorithm. We have shown in a series of experiments that suitably specialised grammars can be compiled into efficient executable forms. In particular, recognisers built in this way are very competitive with ones created using statistical training methods (Rayner et al., 2005a).

The Regulus platform also supplies a framework for using the compiled resources — parsers, generators and recognisers — to build speech translation and spoken dialogue applications. The environment currently supports 75 different commands,

[1] http://sourceforge.net/projects/regulus/
[2] http://ic.arc.nasa.gov/projects/clarissa/
[3] http://sourceforge.net/projects/medslt

Proceedings of SPEECHGRAM 2007, pages 49–52,
Prague, Czech Republic, June 2007.

which can be used to carry out a range of functions including compilation of grammars into various forms, debugging of grammars and compiled resources, and testing of applications. The environment exists in two forms. The simpler one, which has been available from the start of the project, is a command-line interface embedded within the SICStus Prolog top-level. The focus will however be on a new GUI-based environment, which has been under development since late 2006, and which offers a more user-friendly graphical/menu-based view of the underlying functionality.

In the rest of the paper, we outline how Regulus supports development both at the level of grammars (Section 2), and at the level of the applications that can be built using the executable forms derived from them (Section 3).

2 Developing unification grammars

The Regulus grammar development toolset borrows ideas from several other systems, in particular the SRI Core Language Engine (CLE) and the Xerox Language Engine (XLE). The basic functionalities required are uncontroversial. As usual, the Regulus environment lets the user parse example sentences to create derivation trees and logical forms; in the other direction, if the grammar has also been compiled into a generator, the user can take a logical form and use it to generate a surface string and another derivation tree. Once a derivation tree has been created, either through parsing or through generation, it is possible to examine individual nodes to view the information associated with each one. Currently, this information consists of the syntactic features, the piece of logical form built up at the node, and the grammar rule or lexical entry used to create it.

The Regulus environment also provides a more elaborate debugging tool, which extends the earlier "grammar stepper" implemented under the CLE project. Typically, a grammar development problem has the following form. The user finds a bad sentence B which fails to get a correct parse; however, there are several apparently similar or related sentences $G_1 ... G_n$ which do get correct parses. In most cases, the explanation is that some rule which would appear in the intended parse for B has an incorrect feature-assignment.

A simple strategy for investigating problems of this kind is just to examine the structures of B and $G_1 ... G_n$ by eye, and attempt to determine what the crucial difference is. An experienced developer, who is closely familiar with the structure of the grammar, will quite often be able to solve the problem in this way, at least in simple cases. "Solving by inspection" is not, however, very systematic, and with complex rule bugs it can be hard even for experts to find the offending feature assignment. The larger the grammar becomes, especially in terms of the average number of features per category, the more challenging the *ad hoc* debugging approach becomes.

A more systematic strategy was pioneered in the CLE grammar stepper. The developer begins by looking at the working examples $G_1 ... G_n$, to determine what the intended correct structure would be for B. They then build up the corresponding structure for the bad example, starting at the bottom with the lexical items and manually selecting the rules used to combine them. At some point, a unification will fail, and this will normally reveal the bad feature assignment. The problem is that manual bottom-up construction of the derivation tree is very time-consuming, since even quite simple trees will usually have at least a dozen nodes.

The improved strategy used in the Regulus grammar stepper relies on the fact that the $G_1 ... G_n$ can usually be constructed to include all the individual pieces of the intended derivation tree for B, since in most cases the feature mis-match arises when combining two subtrees which are each internally consistent. We exploit this fact by allowing the developer to build up the tree for B by cutting up the trees for $G_1 ... G_n$ into smaller pieces, and then attempting to recombine them. Most often, it is enough to take two of the G_i, cut an appropriate subtree out of each one, and try to unify them together; this means that the developer can construct the tree for B with only five operations (two parses, two cuts, and a join), rather than requiring one operation for each node in B, as in the bottom-up approach.

A common pattern is that B and G_1 are identical, except for one noun-phrase constituent NP, and G_2 consists of NP on its own. To take an example from the MedSLT domain, B could be "does the morning

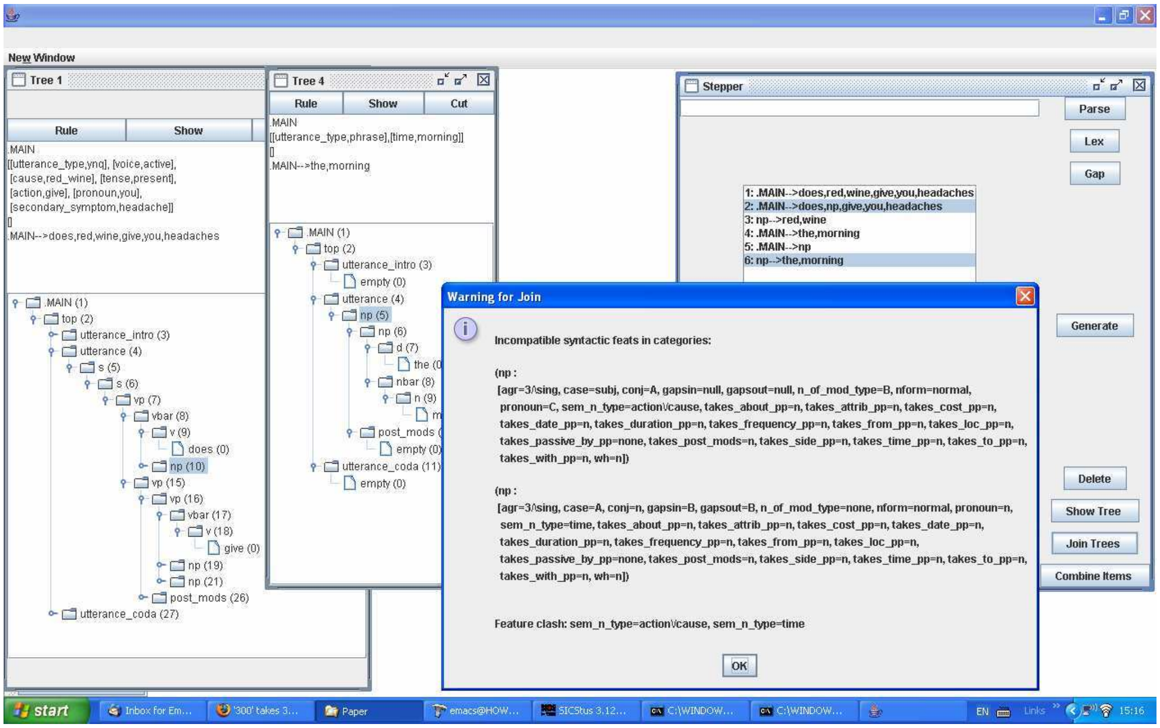

Figure 1: Example of using the grammar stepper to discover a feature mismatch. The window on the right headed "Stepper" presents the list of available trees, together with the controls. The windows headed "Tree 1" and "Tree 4" present the trees for item 1 ("does red wine give you headaches") and item 4 ("the morning"). The popup window on the lower right presents the feature mismatch information.

give you headaches?", G_1 the similar sentence "does red wine give you headaches?" and G_2 the single NP "the morning". We cut out the first NP subtree from G_1 to produce what is in effect a tree with an NP "slash category", that can be rendered as "does NP give you headaches?"; call this G_1'. We then cut out the single NP subtree (this accounts for most, but not all, of the derivation) from G_2, to produce G_2'. By attempting to unify G_2' with the NP "hole" left in G_1', we can determine the exact nature of the feature mismatch. We discover that the problem is in the sortal features: the value of the sortal feature on G_2' is `time`, but the corresponding feature-value in the NP "hole" is `action\/cause`.

Figure 1 contains a screenshot of the development environment in the example above, showing the state when the feature mismatch is revealed. A detailed example, including screenshots for each step, is included in the online Regulus GUI tutorial[4].

[4] Available in the file doc/RegulusGUITutorial.pdf from the SourceForge Regulus website

3 Developing applications

The Regulus platform contains support for both speech translation and spoken dialogue applications. In each case, it is possible to run the development top-loop in a mode appropriate to the type of application, including carrying out systematic regression testing using both text and speech input. For both types of application, the platform assumes a uniform architecture with pre-specified levels of representation.

Due to shortage of space, and because it is the better-developed of the two, we focus on speech translation. The framework is interlingua-based, and also permits simple context-based translation involving resolution of ellipsis[5]. Processing goes through the following sequence of representations:

1. Spoken utterance in source language.

2. Recognised words in source language.

[5] Although it is often possible to translate ellipsis as ellipsis in closely related language pairs, this is usually not correct in more widely separated ones.

3. Source logical form. Source logical form and all other levels of representation are (almost) flat lists of attribute/value pairs.

4. "Source discourse representation". A regularised version of the source logical form, suitable for carrying out ellipsis resolution.

5. "Resolved source discourse representation". The output resulting from carrying out any necessary ellipsis processing on the source discourse representation. Typically this will add material from the preceding context representation to create a representation of a complete clause.

6. Interlingua. A language-independent version of the representation.

7. Target logical form.

8. Surface words in target language.

The transformations from source logical form to source discourse representation, from resolved source discourse representation to interlingua, and from interlinga to target logical form are defined using translation rules which map lists of attribute/value pairs to lists of attribute/value pairs. The translation trace includes all the levels of representation listed above, the translation rules used at each stage, and other information omitted here. The "translation mode" window provided by the development environment makes all these fields available in a structured form which allows the user to select for display only those that are currently of interest. The framework for spoken dialogue systems is similar, except that in the last three steps "Interlingua" is replaced by "Dialogue move", "Target logical form" by "Abstract response", and "Surface words in target language" by "Concrete response".

The platform contains tools for performing systematic regression testing of both speech translation and spoken dialogue applications, using both text and speech input. Input in the required modality is taken from a specified file and passed through all stages of processing, with the output being written to another file. The user is able to annotate the results with respect to correctness (the GUI presents a simple menu-based interface for doing this) and save the judgements permanently, so that they can be reused for future runs.

The most interesting aspects of the framework involve development of spoken dialogue systems. With many other spoken dialogue systems, the effect of a dialogue move is distributed throughout the program state, and true regression testing is very difficult. Here, our side-effect free approach to dialogue management means that the DM can be tested straightforwardly as an isolated component, since the context is fully encapsulated as an object. The theoretical issues involved are explored further in (Rayner and Hockey, 2004).

References

[Bouillon et al.2005] P. Bouillon, M. Rayner, N. Chatzichrisafis, B.A. Hockey, M. Santaholma, M. Starlander, Y. Nakao, K. Kanzaki, and H. Isahara. 2005. A generic multi-lingual open source platform for limited-domain medical speech translation. In *Proceedings of the 10th Conference of the European Association for Machine Translation (EAMT)*, pages 50–58, Budapest, Hungary.

[Rayner and Hockey2004] M. Rayner and B.A. Hockey. 2004. Side effect free dialogue management in a voice enabled procedure browser. In *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP)*, Jeju Island, Korea.

[Rayner et al.2005a] M. Rayner, P. Bouillon, N. Chatzichrisafis, B.A. Hockey, M. Santaholma, M. Starlander, H. Isahara, K. Kanzaki, and Y. Nakao. 2005a. Methodology for comparing grammar-based and robust approaches to speech understanding. In *Proceedings of the 9th International Conference on Spoken Language Processing (ICSLP)*, pages 1103–1107, Lisboa, Portugal.

[Rayner et al.2005b] M. Rayner, B.A. Hockey, J.M. Renders, N. Chatzichrisafis, and K. Farrell. 2005b. A voice enabled procedure browser for the international space station. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (interactive poster and demo track)*, Ann Arbor, MI.

[Rayner et al.2006] M. Rayner, B.A. Hockey, and P. Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Press, Chicago.