Making Speech Look Like Text in the Regulus Development Environment

Elisabeth Kron
3 St Margarets Road, Cambridge CB3 0LT, England
elisabethkron@yahoo.co.uk

Manny Rayner, Marianne Santaholma, Pierrette Bouillon, Agnes Lisowska
University of Geneva, TIM/ISSCO, 40 bvd du Pont-d'Arve
CH-1211 Geneva 4, Switzerland
Emmanuel.Rayner@issco.unige.ch
Marianne.Santaholma@eti.unige.ch
Pierrette.Bouillon@issco.unige.ch
Agnes.Lisowska@issco.unige.ch

Abstract

We present an overview of the development environment for Regulus, an Open Source platform for construction of grammar-based speech-enabled systems, focussing on recent work whose goal has been to introduce uniformity between text and speech views of Regulus-based applications. We argue the advantages of being able to switch quickly between text and speech modalities in interactive and offline testing, and describe how the new functionalities enable rapid prototyping of spoken dialogue systems and speech translators.

1 Introduction

Sex is not love, as Madonna points out at the beginning of her 1992 book *Sex*, and love is not sex. None the less, even people who agree with Madonna often find it convenient to pretend that these two concepts are synonymous, or at least closely related. Similarly, although text is not speech, and speech is not text, it is often convenient to pretend that they are both just different aspects of the same thing.

In this paper, we will explore the similarities and differences between text and speech, in the concrete setting of Regulus, a development environment for grammar based spoken dialogue systems. Our basic goal will be to make text and speech processing as similar as possible from the point of view of the developer. Specifically, we arrange things so that the developer is able to develop her system using a text view; she will write text-based rules, and initially test the system using text input and output. At any point, she will be able to switch to a speech view, compiling the text-based processing rules into corresponding speech-based versions, and test the resulting speech-based system using speech input and output.

Paradoxically, the reason why it is so important to be able to switch seamlessly between text and speech viewpoints is that text and speech are in fact *not* the same. For example, a pervasive problem in speech recognition is that of easily confusable pairs of words. This type of problem is often apparent after just a few minutes when running the system in speech mode (the recogniser keeps recognising one word as the other), but is invisible in text mode. More subtly, some grammar problems can be obvious in text mode, but hard to see in speech mode. For instance, articles like "the" and "a" are short, and usually pronounced unstressed, which means that recognisers can be reasonably forgiving about whether or not to hypothesise them when they are required or not required by the recognition grammar. In text mode, it will immediately be clear if the grammar requires an article in a given NP context: incorrect variants will fail to parse. In speech mode, the symptoms are far less obvious, and typically amount to no more than a degradation in recognition performance.

The rest of the paper is structured as follows. Sections 2 and 3 provide background on the Regulus platform and development cycle respectively. Section 4 describes speech and text support in the interactive development environment, and 5 describes how the framework simplifies the task of

Coling 2008: Proceedings of the workshop on Grammar Engineering Across Frameworks, pages 9–16
Manchester, August 2008

switching between modalities in regression testing. Section 6 concludes.

2 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]. In terms of ideas (though not code), Regulus is a descendent of SRI International's CLE and Gemini platforms (Alshawi, 1992; Dowding et al., 1993); other related systems are LKB (Copestake, 2002), XLE (Crouch et al., 2008) and UNIANCE (Bos, 2002).

Regulus has already been used to build several large applications. Prominent examples are Geneva University's MedSLT medical speech translator (Bouillon et al., 2005), NASA's Clarissa procedure browser (Rayner et al., 2005) and Ford Research's experimental SDS in-car spoken dialogue system, which was awarded first prize at the 2007 Ford internal demo fair. Regulus is described at length in (Rayner et al., 2006), the first half of which consists of an extended tutorial introduction. The release includes a command-line development environment, extensive online documentation, and 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. These recognition packages produced by Regulus can be invoked through the Regulus SpeechServer ("Regserver"), which provides an interface to the underlying Nuance recognition engine. The value added by the Regserver is to provide a view of the recognition process based on the Regulus unification grammar framework. In particular, recognition results, originally produced in the Nuance recognition platform's internal format, are reformatted into the semantic notation used by the Regulus grammar formalism.

There is extensive support within the Regulus toolkit for development of both speech translation and spoken dialogue applications. Spoken dialogue applications (Rayner et al., 2006, Chapter 5) use a rule-based side-effect free state update model similar in spirit to that described in (Larsson and Traum, 2000). Very briefly, there are three types of rules: state update rules, input management rules, and output management rules. State update rules take as input the current state, and a "dialogue move"; they produce as output a new state, and an "abstract action". Dialogue moves are abstract representations of system inputs; these inputs can either be logical forms produced by the grammar, or non-speech inputs (for example, mouse-clicks in a GUI). Similarly, abstract actions are, as the name suggests, abstract representations of the concrete actions the dialogue system will perform, for example speaking or updating a visual display. Input management rules map system inputs to dialogue moves; output management rules map abstract actions to system outputs.

Speech translation applications are also rule-based, using an interlingua model (Rayner et al., 2006, Chapter 6). The developer writes a second grammar for the target language, using Regulus tools to compile it into a generator; mappings from source representation to interlingua, and from interlingua to target representation, are defined by sets of translation rules. The interlingua itself is specified using a third Regulus grammar (Bouillon et al., 2008).

To summarise, the core of a Regulus application consists of several different linguistically oriented rule-sets, some of which can be interpreted in either a text or a speech modality, and all of which need to interact correctly together. In the next section, we describe how this determines the nature of the Regulus development cycle.

3 The Regulus development cycle

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, linguistically motivated 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[2]. Regulus contains

[1] `http://sourceforge.net/projects/regulus/`

[2] `http://sourceforge.net/projects/medslt`

an extensive set of tools that permit specialised domain-specific grammars to be extracted from the larger resource grammars, using example-based methods driven by small corpora (Rayner et al., 2006, Chapter 7). At the beginning of a project, these corpora can consist of just a few dozen examples; for a mature application, they will typically have grown to something between a few hundred and a couple of thousand sentences. Specialised grammars can be compiled by Regulus into efficient recognisers and generators.

As should be apparent from the preceding description, the Regulus architecture is designed to empower linguists to the maximum possible extent, in terms of increasing their ability directly to build speech enabled systems; the greater part of the core development teams in the large Regulus projects mentioned in Section 1 have indeed come from linguistics backgrounds. Experience with Regulus has however shown that linguists are not quite as autonomous as they are meant to be, and in particular are reluctant to work directly with the speech view of the application. There are several reasons.

First, non-toy Regulus projects require a range of competences, including both software engineering and linguistics. In practice, linguist rule-writers have not been able to test their rules in the speech view without writing glue code, scripts, and other infrastructure required to tie together the various generated components. These are not necessarily things that they want to spend their time doing. The consequence can easily be that the linguists end up working exclusively in the text view, and over-refine the text versions of the rule-sets. From a project management viewpoint, this results in bad prioritisation decisions, since there are more pressing issues to address in the speech view.

A second reason why linguist rule-writers have been unhappy working in the speech view is the lack of reproducibility associated with speech input. One can type "John loves Mary" into a text-processing system any number of times, and expect to get the same result. It is much less reasonable to expect to get the same result each time if one *says* "John loves Mary" to a speech recogniser. Often, anomalous results occur, but cannot be debugged in a systematic fashion, leading to general frustration. The result, once again, is that linguists have preferred to stick with the text view, where they feel at home.

Yet another reason why rule-writers tend to limit themselves to the text view is simply the large number of top-level commands and intermediate compilation results. The current Regulus command-line environment includes over 110 different commands, and compilation from the initial resource grammar to the final Nuance recognition package involves creating a sequence of five compilation steps, each of which requires the output created by the preceding one. This makes it difficult for novice users to get their bearings, and increases their cognitive load. Additionally, once the commands for the text view have been mastered, there is a certain temptation to consider that these are enough, since the text and speech views can reasonably be perceived as fairly similar.

In the next two sections, we describe an enhanced development environment for Regulus, which addresses the key problems we have just sketched. From the point of view of the linguist rule-writer, we want speech-based development to feel more like text-based development.

4 Speech and text in the online development environment

The Regulus GUI (Kron et al., 2007) is intended as a complete redesign of the development environment, which simultaneously attacks all of the central issues. Commands are organised in a structured set of functionality-based windows, each of which has an appropriate set of drop-down menus. Following normal GUI design practice (Dix et al., 1998, Chapters 3 and 4); (Jacko and Sears, 2003, Chapter 13), only currently meaningful commands are executable in each menu, with the others shown greyed out.

Both compile-time and run-time speech-related functionality can be invoked directly from the command menus, with no need for external scripts, Makefiles or glue code. Focussing for the moment on the specific case of developing a speech translation application, the rule-writer will initially write and debug her rules in text mode. She will be able to manipulate grammar rules and derivation trees using the Stepper window (Figure 1; cf. also (Kron et al., 2007)), and load and test translation rules in the Translate window (Figure 2). As soon as the grammar is consistent, it can at any point be compiled into a Nuance recognition package using the command menus. The resulting recogniser, together with other speech resources (license man-

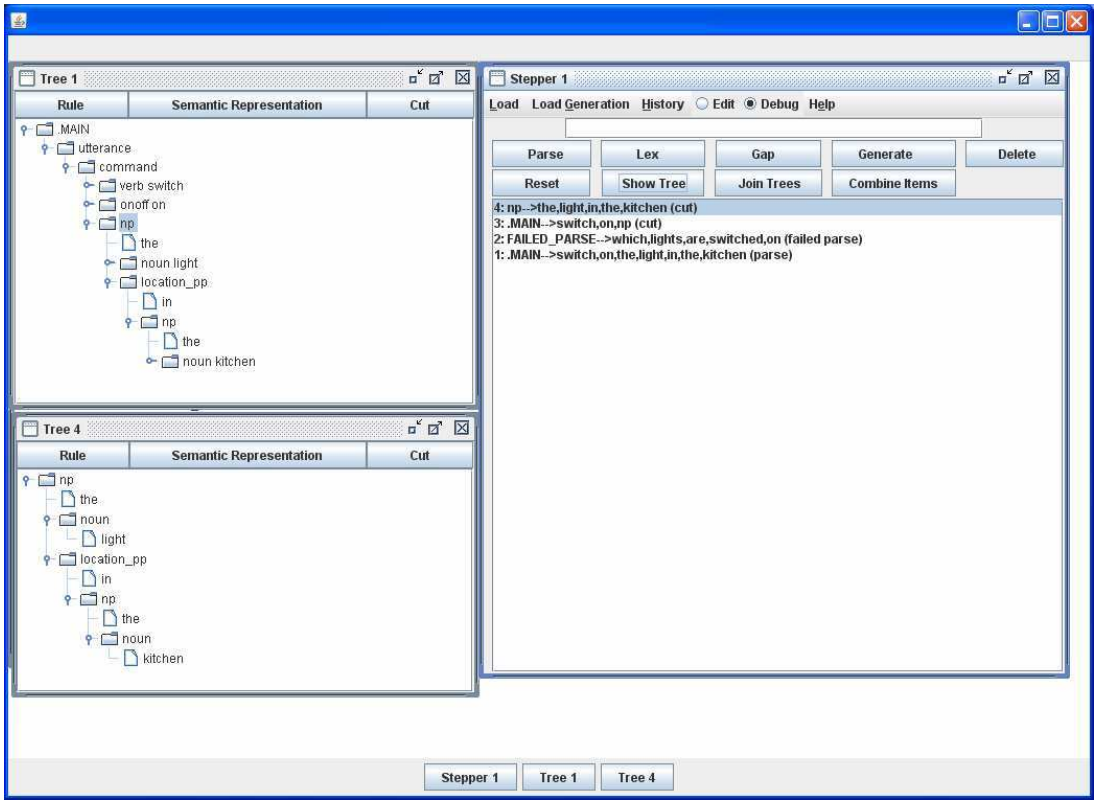

Figure 1: Using the Stepper window to browse trees in the Toy1 grammar from (Rayner et al., 2006, Chapter 4). The upper left window shows the analysis tree for "switch on the light in the kitchen"; the lower left window shows one of the subtrees created by cutting the first tree at the higher NP node. Cut subtrees can be recombined for debugging purposes (Kron et al., 2007).

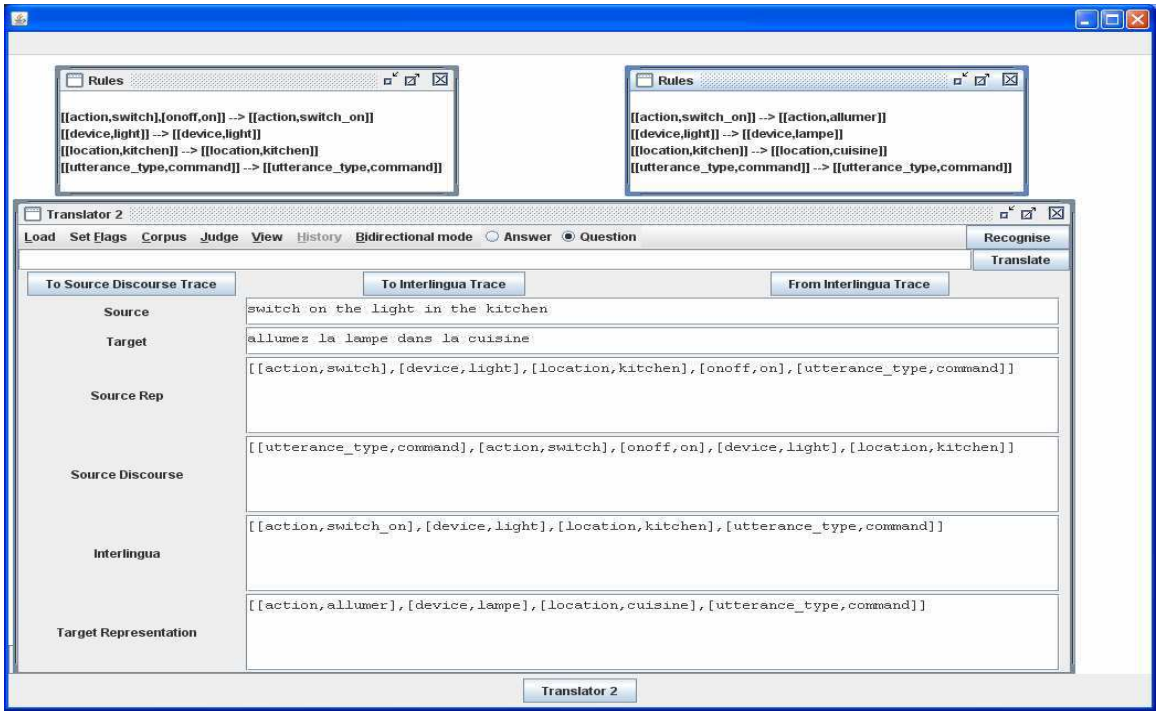

Figure 2: Using the Translate window to test the toy English → French translation application from (Rayner et al., 2006, Chapter 6). The to- and from-interlingua rules used in the example are shown in the two pop-up windows at the top of the figure.

```
regulus_config(regulus_grammar,
               [toy1_grammars(toy1_declarations),
                toy1_grammars(toy1_rules),
                toy1_grammars(toy1_lexicon)]).
regulus_config(top_level_cat, '.MAIN').
regulus_config(nuance_grammar, toy1_runtime(recogniser)).

regulus_config(to_interlingua_rules,
               toy1_prolog('eng_to_interlingua.pl')).
regulus_config(from_interlingua_rules,
               toy1_prolog('interlingua_to_fre.pl')).
regulus_config(generation_rules, toy1_runtime('generator.pl')).

regulus_config(nuance_language_pack,
               'English.America').
regulus_config(nuance_compile_params, ['-auto_pron', '-dont_flatten']).
regulus_config(translation_rec_params,
               [package=toy1_runtime(recogniser), grammar='.MAIN']).
regulus_config(tts_command,
    'vocalizer -num_channels 1 -voice juliedeschamps -voices_from_disk').
```

Figure 3: Config file for a toy English → French speech translation application, showing items relevant to the speech view. Some declarations have been omitted for expositional reasons.

ager, TTS engine etc), can then be started using a single menu command.

In accordance with the usual Regulus design philosophy of declaring all the resources associated with a given application in its config file, the speech resources are also specified here. Figure 3 shows part of the config file for a toy translation application, in particular listing all the declarations relevant to the speech view. If we needed to change the speech resources, this would be done just by modifying the last four lines. For example, the config file as shown specifies construction of a recogniser using acoustic models appropriate to American English. We could change this to British English by replacing the entry

```
regulus_config(nuance_language_pack,
               'English.America').
```

with

```
regulus_config(nuance_language_pack,
               'English.UK').
```

When the speech resources have been loaded, the Translate window can take input equally easily in text or speech mode; the Translate button processes written text from the input pane, while the Recognise button asks for spoken input. In each case, the input is passed through the same processing stages of source-to-interlingua and interlingua-to-target translation, followed by target-language generation. If a TTS engine or a set of recorded target language wavfiles is specified, they are used to realise the final result in spoken form (Figure 4).

Every spoken utterance submitted to recognition is logged as a SPHERE-headed wavfile, in a time-stamped directory started at the beginning of the current session; this directory also contains a meta-data file, which associates each recorded wavfile with the recognition result it produced. The Translate window's History menu is constructed using the meta-data file, and allows the user to select any recorded utterance, and re-run it through the system as though it were a new speech input. The consequence is that speech input becomes just as reproducible as text, with corresponding gains for interactive debugging in speech mode.

5 Speech and text in regression testing

In earlier versions of the Regulus development environment (Rayner et al., 2006, §6.6), regression testing in speech mode was all based on Nuance's `batchrec` utility, which permits offline recognition of a set of recorded wavfiles. A test suite for spoken regression testing consequently consisted of a list of wavfiles. These were first passed through `batchrec`; outputs were then post-processed into Regulus form, and finally passed through Regulus speech understanding modules, such as translation or dialogue management.

As Regulus applications grow in complexity, this model has become increasingly inadequate, since system input is very frequently not just a list of monolingual speech events. In a multimodal dialogue system, input can consist of either speech or screen events (text/mouse-clicks); context is generally important, and the events have to be processed in the order in which they occurred. Dialogue systems which control real or simulated robots, like the Wheelchair application of (Hockey

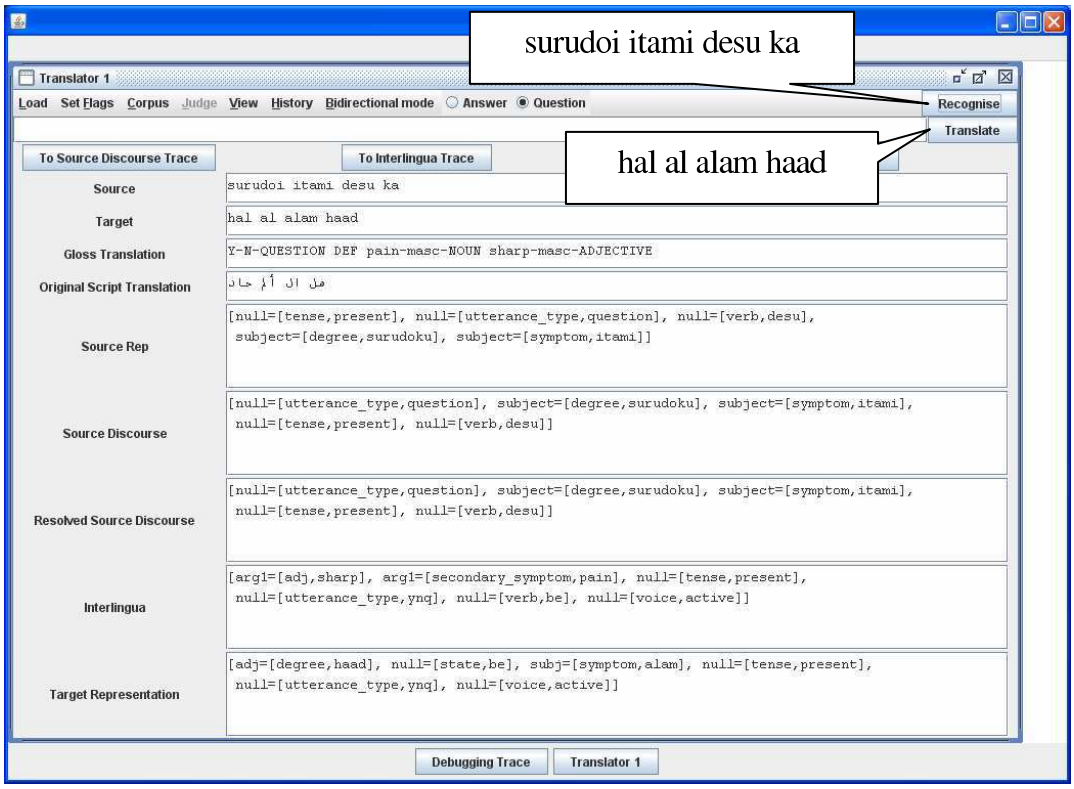

Figure 4: Speech to speech translation from the GUI, using a Japanese to Arabic translator built from MedSLT components (Bouillon et al., 2008). The user presses the Recognise button (top right), speaks in Japanese, and receives a spoken translation in Arabic together with screen display of various processing results. The application is defined by a config file which combines a Japanese recogniser and analysis grammar, Japanese to Interlingua and Interlingua to Arabic translation rules, an Arabic generation grammar, and recorded Arabic wavfiles used to construct a spoken result.

and Miller, 2007) will also receive asynchronous inputs from the robot control and monitoring process; once again, all inputs have to be processed in the appropriate temporal order. A third example is contextual bidirectional speech translation (Bouillon et al., 2007). Here, the problem is slightly different — we have only speech inputs, but they are for two different languages. The basic issue, however, remains the same, since inputs have to be processed in the right order to maintain the correct context at each point.

With examples like these in mind, we have also effected a complete redesign of the Regulus environment's regression testing facilities. A test suite is now allowed to consist of a list of items of any type — text, wavfile, or non-speech input — in any order. Instead of trying to fit processing into the constraints imposed by the `batchrec` utility, offline processing now starts up speech resources in the same way as the interactive environment, and submits each item for appropriate processing in the order in which it occurs. By adhering to the principle that text and speech should be treated uniformly, we arrive at a framework which is simpler, less error-prone (the underlying code is less fragile) and above all much more flexible.

6 Summary and conclusions

The new functionality offered by the redesigned Regulus top-level is not strikingly deep. In the context of any given application, it could all have been duplicated by reasonably simple scripts, which linked together existing Regulus components. Indeed, much of this new functionality is implemented using code derived precisely from such scripts. Our observation, however, has been that few developers have actually taken the time to write these scripts, and that when they have been developed inside one project they have usually not migrated to other ones. One of the things we have done, essentially, is to generalise previously *ad hoc* application-dependent functionality, and make it part of the top-level development environment. The other main achievements of the new Regulus top-level are to organise the existing functionality in a more systematic way, so that it is easier to find commands, and to package it all as a normal-looking Swing-based GUI.

Although none of these items sound dramatic, they make a large difference to the platform's over-

all usability, and to the development cycle it supports. In effect, the Regulus top-level becomes a generic speech-enabled application, into which developers can plug their grammars, rule-sets and derived components. Applications can be tested in the speech view much earlier, giving a correspondingly better chance of catching bad design decisions before they become entrenched. The mechanisms used to enable this functionality do not depend on any special properties of Regulus, and could readily be implemented in other grammar-based development platforms, such as Gemini and UNIANCE, which support compilation of feature grammars into grammar-based language models.

At risk of stating the obvious, it is also worth pointing out that many users, particularly younger ones who have grown up using Windows and Mac environments, expect as a matter of course that development platforms will be GUI-based rather than command-line. Addressing this issue, and simplifying the transition between text- and speech-based, views has the pleasant consequence of improving Regulus as a vehicle for introducing linguistics students to speech technology. An initial Regulus-based course at the University of Santa Cruz, focussing on spoken dialogue systems, is described in (Hockey and Christian, 2008); a similar one, but oriented towards speech translation and using the new top-level described here, is currently under way at the University of Geneva. We expect to present this in detail in a later paper.

References

Alshawi, H., editor. 1992. *The Core Language Engine*. MIT Press, Cambridge, Massachusetts.

Bos, J. 2002. Compilation of unification grammars with compositional semantics to speech recognition packages. In *Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan.

Bouillon, P., M. Rayner, N. Chatzichrisafis, B.A. Hockey, M. Santaholma, M. Starlander, Y. Nakao, K. Kanzaki, and H. Isahara. 2005. A generic multilingual 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.

Bouillon, P., G. Flores, M. Starlander, N. Chatzichrisafis, M. Santaholma, N. Tsourakis, M. Rayner, and B.A. Hockey. 2007. A bidirectional grammar-based medical speech translator. In *Proceedings of the ACL Workshop on Grammar-based Approaches to Spoken Language Processing*, pages 41–48, Prague, Czech Republic.

Bouillon, P., S. Halimi, Y. Nakao, K. Kanzaki, H. Isahara, N. Tsourakis, M. Starlander, B.A. Hockey, and M. Rayner. 2008. Developing non-european translation pairs in a medium-vocabulary medical speech translation system. In *Proceedings of LREC 2008*, Marrakesh, Morocco.

Copestake, A. 2002. *Implementing Typed Feature Structure Grammars*. CSLI Press, Chicago.

Crouch, R., M. Dalrymple, R. Kaplan, T. King, J. Maxwell, and P. Newman, 2008. *XLE Documentation*. http://www2.parc.com/isl/groups/nltt/xle/doc. As of 29 Apr 2008.

Dix, A., J.E. Finlay, G.D. Abowd, and R. Beale, editors. 1998. *Human Computer Interaction. Second ed.* Prentice Hall, England.

Dowding, J., M. Gawron, D. Appelt, L. Cherny, R. Moore, and D. Moran. 1993. Gemini: A natural language system for spoken language understanding. In *Proceedings of the Thirty-First Annual Meeting of the Association for Computational Linguistics*.

Hockey, B.A. and G. Christian. 2008. Zero to spoken dialogue system in one quarter: Teaching computational linguistics to linguists using regulus. In *Proceedings of the Third ACL Workshop on Teaching Computational Linguistics (TeachCL-08)*, Columbus, OH.

Hockey, B.A. and D. Miller. 2007. A demonstration of a conversationally guided smart wheelchair. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 243–244, Denver, CO.

Jacko, J.A. and A. Sears, editors. 2003. *The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications*. Lawerence Erlbaum Associates, Mahwah, New Jersey.

Kron, E., M. Rayner, P. Bouillon, and M. Santaholma. 2007. A development environment for building grammar-based speech-enabled applications. In *Proceedings of the ACL Workshop on Grammar-based Approaches to Spoken Language Processing*, pages 49–52, Prague, Czech Republic.

Larsson, S. and D. Traum. 2000. Information state and dialogue management in the TRINDI dialogue move engine toolkit. *Natural Language Engineering, Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering*, pages 323–340.

Rayner, M., B.A. Hockey, J.M. Renders, N. Chatzichrisafis, and K. Farrell. 2005. 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, M., B.A. Hockey, and P. Bouillon. 2006. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Press, Chicago.