Tobias Kuhn
Norbert E. Fuchs (Eds.)

LNAI 7427

Controlled Natural Language

Third International Workshop, CNL 2012
Zurich, Switzerland, August 2012
Proceedings

Springer

Lecture Notes in Artificial Intelligence 7427

Subseries of Lecture Notes in Computer Science

Tobias Kuhn Norbert E. Fuchs (Eds.)

Controlled Natural Language

Third International Workshop, CNL 2012
Zurich, Switzerland, August 29-31, 2012
Proceedings

Springer

Volume Editors

Tobias Kuhn
Yale University
School of Medicine
Department of Pathology
300 George Street
New Haven, CT 06511, USA
E-mail: kuhntobias@gmail.com

Norbert E. Fuchs
Universität Zürich
Institut für Computerlinguistik
Binzmühlestrasse 14
8050 Zürich, Switzerland
E-mail: fuchs@ifi.uzh.ch

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-32611-0 e-ISBN 978-3-642-32612-7
DOI 10.1007/978-3-642-32612-7
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012943860

CR Subject Classification (1998): F.4.3, I.2.4, I.2.7, F.4, I.2, J.1, J.5, H.3, H.4

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

CNL 2012 was the third of a series of international workshops on controlled natural language. Skipping one year, it followed up on the previous two events, CNL 2009 and 2010, both held on the beautiful island of Marettimo in Italy. Their success convinced us to continue the series. This time, for a change and for better reachability, we chose Zurich in Switzerland as venue instead of Marettimo. The event was hosted by the Department of Informatics and the Institute of Computational Linguistics of the University of Zurich. Another change concerns the proceedings. Previously, contributions for the workshops were reviewed based on extended abstracts with the full papers being reviewed and published after the event. This time, the complete proceedings are published before the workshop, which involved just one round of reviewing of the full papers.

The topics of the workshop are the different theoretical and practical aspects of controlled natural languages. Such languages are based on natural languages but come with restrictions on vocabulary, grammar, and/or semantics. The general goal is to reduce or eliminate ambiguity and complexity. Some of these controlled languages are designed to improve communication among humans, especially for non-native speakers of the respective natural language. In other cases, the restrictions on the language are supposed to make it easier for computers to analyze such texts in order to improve computer-aided, semi-automatic, or automatic translations into other languages. A third group of controlled natural languages has the goal to enable reliable automated reasoning on seemingly natural texts. Such languages have a direct mapping to some sort of formal logic and should improve the accessiblity of formal knowledge representations or specifications for people unfamiliar with formal notations. All these types are covered by the workshop. To our knowledge, it is currently the only scientific event with this particular focus.

Of 15 initial submissions, one was withdrawn, two were rejected by the Program Committee, and for one paper no acceptable final version was submitted. This led to the acceptance of the 11 papers of this volume, which were presented at the workshop. In addition, the program included the talks of two invited speakers: Sabine Lehmann (Acrolinx) and Stephen Pulman (University of Oxford). Finally, there were two slots for system demonstrations, one academic and one industrial. We hope to have achieved the same informal and productive atmosphere of the previous events.

We would like to thank the authors for their work and for choosing this event to submit their manuscripts. We also thank the Program Committee members for their reviews, discussions, and general feedback. A special thank you goes to the invited speakers for agreeing to present their work on controlled natural language. We also thank our host departments at the University of Zurich — the Department of Informatics and the Institute of Computational Linguistics — for

their support. Finally, we are very grateful for being sponsored by the MOLTO project[1]. All this support has allowed us, as in previous years, to organize the workshop without the need to collect participation fees.

June 2012

Tobias Kuhn
Norbert E. Fuchs

[1] http://www.molto-project.eu/

Organization

Program Committee

Johan Bos	University of Groningen, The Netherlands
Peter Clark	Vulcan Inc., USA
Rogan Creswick	Galois Inc., USA
Danica Damljanović	University of Sheffield, UK
Brian Davis	DERI, National University of Ireland
Norbert E. Fuchs	University of Zurich, Switzerland
Normunds Gruzitis	University of Latvia
Stefan Hoefler	University of Zurich, Switzerland
Kaarel Kaljurand	University of Zurich, Switzerland
Peter Koepke	University of Bonn, Germany
Tobias Kuhn	Yale University, USA
Hans Leiß	University of Munich, Germany
Reinhard Muskens	Tilburg University, The Netherlands
Gordon Pace	University of Malta
Richard Power	Open University, UK
Laurette Pretorius	University of South Africa
Aarne Ranta	Chalmers University of Technology and Göteborg University, Sweden
Mike Rosner	University of Malta
Uta Schwertel	IMC Information Multimedia Communication AG, Germany
Rolf Schwitter	Macquarie University, Australia
Silvie Spreeuwenberg	LibRT, The Netherlands
Geoff Sutcliffe	University of Miami, USA
Yorick Wilks	University of Sheffield, UK
Adam Wyner	University of Liverpool, UK

Additional Reviewers

Borg, Claudia
Enache, Ramona
Sabuncu, Orkunt

Table of Contents

Applying CNL Authoring Support to Improve Machine Translation of Forum Data

Sabine Lehmann, Ben Gottesman, Robert Grabowski, Mayo Kudo,
Siu Kei Pepe Lo, Melanie Siegel, and Frederik Fouvry

Acrolinx GmbH, Friedrichstr. 100, 10117 Berlin, Germany

Abstract. Machine translation (MT) is most often used for texts of publishable quality. However, there is increasing interest in providing translations of user-generated content in customer forums. This paper describes research towards addressing this challenge by automatically improving the quality of community forum data to improve MT results.

Keywords: Controlled Natural Language, Authoring Support, Machine Translation, User-Generated Content, Forum Data.

1 Introduction

With the exception of some experiments such as Roturier (2006), authoring support and machine translation (MT) have generally been seen as two distinct areas. We want to show that MT can profit from being combined with authoring support methods.

Authoring support with language technology methods is aimed at supporting authors in the process of writing controlled natural language (CNL). Kohl (2008) gives guidelines for writing text that is easily translatable – by humans as well as by machines. Thicke (2011) measures the effect on MT quality of improving the source-language text by applying the recommendations of the Global English Style Guide. Tools based on computational linguistic methods, such as Acrolinx (www.acrolinx.com), implement these rules and thus support technical documentation writers. These tools provide spell and grammar checking, style checking, term extraction and terminology checking, and sentence clustering.

Many users of authoring support software are already using it in conjunction with translation memory tools such as TRADOS (www.trados.com) (Somers, 2003). The translation memory tools provide access to previously translated sentences, based on fuzzy matching algorithms.

These users are gradually adopting MT software as well. Often, however, they are not aware of the possibilities and limitations of the different MT methods and tools, and this leads some to reject MT after a period of experimentation.

Thus, some users already combine the tools in their daily work, but the tools themselves are completely distinct.

In the domain of technical authoring, pre-editing has previously been used to improve the human- and machine-translatability of the source text. Studies such as

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 1–10, 2012.

O'Brien and Roturier (2007) and Aikawa et al (2007) have shown that this approach can improve machine translation quality.

The situation in our case is different from the technical authoring situation: The text that we want to translate is user-generated content (UGC) from customer forums and is therefore written in a language quite different from (relatively standardized) technical documentation. Furthermore, it is impossible to train a statistical machine translation (SMT) system on this type of data because there is not enough human-translated forum text available as training data. As a result, we are forced to translate UGC using a mismatched SMT system trained on technical documentation. One goal of pre-editing is therefore to make the source language text more similar to technical documentation text.

Experiments (such as Allen (2001) and Plitt and Masselot (2010)) show that manual post-editing on high-quality MT output is already much faster than translating from scratch by humans. We aim to reduce the manual post-editing effort even further through the use of automatic rules that support the post-editing process and by enhancing the quality of the MT output.

Researchers have become interested in a variety of aspects of post-editing, such as the cognitive effort it takes (Temnikova 2010), productivity gains (Garcia 2011; de Sousa et al 2011), or how to best support human workflows by translation quality estimation (Specia 2011).

Few approaches to automatic post-editing have been discussed in the literature, among them: using statistical methods (Dugast et al 2007; Simard et al 2007) or manually built regular expressions (Guzmán 2008). All approaches are first steps in a developing field of research.

The idea of using post-editing patterns as error descriptions appears in Stymne and Ahrenberg (2010). The authors use a grammar checker both to analyze SMT output and to automatically correct it. Although the reported experiments serve as a proof of concept, the authors note that the grammar checker they used was built for supporting human document production and thus was unable to spot many errors typical of MT systems.

2 Background

2.1 Authoring Support Software

Built on a linguistic analytics engine that provides rules and resources concerning monolingual texts (as described in Bredenkamp et al, 2000), the Acrolinx software provides spelling, grammar, style and terminology checking. It is the most commonly used software in the technical documentation authoring process, where reformulation suggestions are applied by professional writers.

2.2 Methods in Machine Translation

We look at two approaches to machine translation: the statistical approach and the rule-based approach. Although there has been much research on combining the ideas

of both, such as Eisele (2009), we make a clear distinction at this point so that it is easier to evaluate the influence that authoring support has on each.

The rule-based approach to MT makes use of linguistic information and dictionaries and defines translation rules based on this information. Examples of systems using this approach are Lucy (www.lucysoftware.com), Langenscheidt T1 (www.langenscheidt.de), and Systran (www.systransoft.com). We expect a positive effect on the MT output when source language texts are more correct and therefore easier to analyze.

The statistical approach to MT (Koehn, 2010) analyzes large amounts of parallel data. On the basis of this training data, statistical models are built, such as the 'phrase table', which consists of probabilities of given source-language phrases being translated as given target-language phrases. These models are then used in the translation task. One source of such parallel data is translation memories. Examples of systems implementing this approach are Google Translate (translate.google.de) and Moses (www.statmt.org/moses). We expect a positive effect on the MT output when the input is standardized, such that it diverges less from the training data.

In machine translation research, evaluation often consists of comparison of machine translation output with a 'gold standard' reference translation, with the assumption that greater similarity is better. Some of the similarity metrics standardly used for this are Smoothed BLEU (Bilingual Evaluation Understudy) as described by Lin and Och (2004), GTM (General Text Matcher) f-measure as described by Melamed et al (2003), and TER (Translation Error Rate) as described by Snover et al (2006). We use these scores both to identify sentences which are difficult to translate, so that requirements for reformulations can be implemented in pre-editing rules in a more focused and targeted way, and to evaluate the effectiveness of pre-editing and post-editing.

2.3 User-Generated Content

Software publishers rely on manuals and online support (knowledge base) articles to assist their customers or users with the installation, maintenance, or troubleshooting of products. With the emergence of Web 2.0 communication channels, however, these documentation sets have been supplemented with user-generated content (UGC). Users are now extremely active in the generation of content related to software products, especially on online forums, where questions are being asked and links to solutions exchanged among savvy users. These conversations take place in a number of online environments (e.g. stackexchange.com), some of which are moderated and facilitated by software publishers. While specific language versions of such forums sometimes exist, most content is very often written in English and may require translation to be of any use to specific users. While such content is sometimes machine-translated, major comprehension problems may persist. These comprehension problems on the target side may be caused by the following characteristics of UGC on the source side:

- Source content may be written by non-professionals or non-native speakers of the language used in the forum (so its linguistic and technical accuracy may not be optimal).

- Although written, this content is stylistically closer to spoken language, with informal syntax and creative lexicon.
- Some of the content is authored by power users (or "techies") who "exhibit communicative techniques and practices that are guided by attitudes of technological elitism" (Leblanc, 2005). These can include alternative spellings, acronyms, case change, color change, techie terms, emoticons, or representation of non-lexical speech sounds.

3 The Problem

Due to the characteristics of user-generated content described in the previous section, translation systems usually do not translate it well. We aim to show that this problem can be ameliorated using Acrolinx. The latter can be used both to reformulate source language forum entries in order to make them more machine-translatable (pre-editing) and to reformulate the translated entries in order to correct translation errors. Where possible, these reformulations should be done automatically. For cases where an automatic reformulation is not possible, we also investigate ways to support forum users via authoring support mechanisms. Here, the major challenge is to adjust the typical CNL workflow to average forum users, who tend to be non-professional writers.

However, authoring support software, like machine translation software, is not built for dealing with UGC, nor is it built for machine translation output. While the standard "out-of-the-box" Acrolinx checking rules help the author to improve the quality of the content in the source language, they need to be specially adapted to also have a positive effect on the machine-translatability of the text. Likewise, a rule set for post-editing machine translation output needs to be specially designed. The main task is to identify which rules to choose, and how to adapt them.

4 Methodology

4.1 Pre-editing

We conducted a standard corpus analysis of the forum text to identify areas where source normalization is required to ensure that input text is as understandable and consistent as possible and matches as much as possible the coverage of the translation system. Such an analysis covers the following aspects:

Spelling and Grammar
Using authoring support in the pre-editing process firstly involves correcting spelling and grammar errors in the source document. We expect that correcting spelling and frequent grammar errors such as agreement errors will improve the translatability of the source text. Automatic spelling correction is, however, error-prone, as there is usually more than one correction suggestion for a misspelled word.

Terminology
Consistent and precise use of terminology also helps the MT process. Acrolinx provides functionality both for extracting terminology in order to set up a terminology

database, and for checking terminology usage against such a database. Term extraction rules are based on linguistic information and run on data in the relevant domain. Thus, the extracted terms are more useful than, for example, a general domain-independent ontology.

In forum data, authors tend to use terminological variants, such as short names for products, that do not occur in the more formal sorts of text that are used as SMT training data. Therefore, it is important to extract terminology from forum data before checking terminology in forum data.

Style

Authoring support for technical documentation includes rules regarding style of writing. These rules are designed to improve the consistency, clarity, understandability, and (human-)translatability of text. Many of these rules also have a positive effect of machine-translatability. For example, rules that simplify long sentences and complex syntactic structures lower the burden on the MT system. Furthermore, we developed rules specifically aimed at improving the machine-translatability of forum text, such as *avoid_colloquial_language*.

An outcome of our analysis was that we identified language peculiarities of forum data that need to be reflected in the basic linguistic resources, such as specific words or types of errors. Many of these are due to the informal or spoken-language style used. Here are some examples from the English and French forums:

- It was **some what** messy.
- Re: symantec update **wont** work.
- It's very interesting, **wanna** give it a go?
- 512MO ram de **dique** dur, mais **la**, cela a toujours **fonctionner** normalement avant, cela fait 4 jours que le **probleme** est apparu quand des mises **a** jours Windows ont été faites.

French forum writers often commit the error of omitting an accent. There are multiple instances of this in the last example above. Often, this is seen by authoring support software as a mere spelling error, as in the case of “probleme” vs. “problème”. In other cases, however, the author in effect substitutes a homophone for the intended word. One of the classic homophone pairs is “à” and “a”, the first of the two being a preposition and the second a form of the verb “avoir” (“has”). Given that there are a series of such “confusion words”, a lot of the work on French has focused on developing rules to correct this sort of error.

Using this information, we were able to identify and implement a first set of pre-editing rules for English and French forum data, such as:

- (FR) Confusion de mots (la vs. là, ce vs. se, etc.) (word confusion)
- (FR) Mots simples (simple words)
- (FR) Évitez le langage familier (avoid informal language)
- (EN) do not use complex sentences
- (EN) do not omit relative pronouns such as that and which
- (EN) use standard capitalization
- (EN) avoid parenthetical expressions in the middle of a sentence

The identification of those rules was based on extensive manual and semi-automated data analysis. The work was carried out in several cycles and has focused on one first set of data. The next step will focus on a different corpus in order to verify whether the basic findings can be confirmed.

The pre-editing rules can on the one hand be applied by authors, as is usually done in the technical documentation authoring process. The author gets error flags and decides about reformulations. This process ensures that text reformulations are always correct. Further, a learning process for the author starts. He or she gets a better understanding of the abilities and limits of MT. On the other hand, many of these rules can be automatically applied, which saves a lot of time and human effort. It lowers the precision, but this may be an acceptable tradeoff as long as the overall effect on the translation quality is positive.

4.2 Post-editing

The authoring support tool is applicable to monolingual text. Therefore, using the same mechanisms as in pre-editing, we can correct spelling, grammar and style errors on the target language text. In order to identify errors to correct via automatic post-editing, we conducted experiments involving professional translators performing post-editing on MT output. Aside from standard spelling and grammar corrections, we identified rules for automatic post-editing of German machine translation output on this basis. Here are examples of these rules:

- correct terminology
- correct standard expressions
- correct word order
- convert future tense to present tense
- convert indicative to causative
- convert “man” to passive
- convert series of “von”+ Noun

Terminology errors account for a large percentage of the corrected errors and thus deserve special attention. Allen (2001) describes a tool that supports humans in manually adding unknown words to a dictionary and in domain-adaptation by manually selecting the best translation of words in a certain domain. We propose to do multilingual term extraction using Acrolinx, and make use of term bank organization (with domains and linked terms) for domain-dependent selection of translations. Domain-specific terminology can be extracted from the training data, a term bank is set up with information about the domain and used for terminology checking on the source and target language texts.

4.3 Support with Automatic Tools

For many errors found in a source text, there is an obvious improvement suggestion. In fact, many of the developed Acrolinx rules for pre-editing suggest the appropriate improvement to support the author. It is thus feasible to include automatic

reformulations in the translation workflow. The challenge is that there may be several possible improvement suggestions for a detected error (a common situation for spelling mistakes in particular), or there may even be several overlapping error flags that have to be resolved in a particular order.

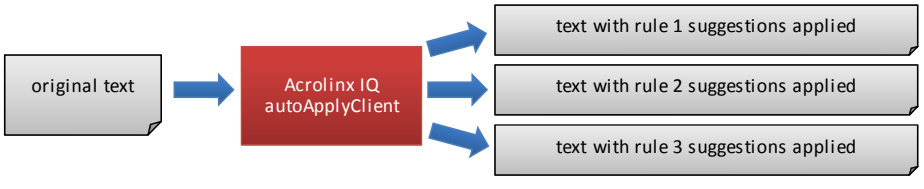
In order to perform automatic pre-editing tasks, and to evaluate their impact on translation quality, two automatic tools have been developed.

AutoApply Client

The Acrolinx AutoApply client is an Acrolinx client that checks a document and automatically replaces flagged (marked) sections of text with the top-ranked improvement suggestion given by Acrolinx, provided that the flag has at least one such suggestion. The tool can be restricted to applying only the suggestions of specific error types, rule sets, and term sets to a given document, such that their respective impacts can be examined in isolation.

Rule Scoring Framework

To fully automatically identify the suitability of rules and suggestions for pre-editing purposes, we have developed an automatic rule scoring framework. It first uses the AutoApply client to automatically apply the suggestions of rules to the sentences to get reformulated versions of the sentences. The result of applying suggestions of each rule is output separately.



We then machine-translate the reformulated and corresponding original sentences, and score the translated sentences against the corresponding reference translations using the metrics mentioned in section 2.2. Since each reformulation reflects the application of only one suggestion, the differences in the scores are usually rather small. Therefore, we merely perform an automatic contrastive evaluation: we note whether the scores have improved, degraded or stayed the same after the reformulation according to the automatic metrics. The mechanism can be complemented by a human evaluation, where judges are given the task of judging which of the pair of translations pair is superior. The rating results are grouped by rule name, such that the rules with the best impact on the translation quality can be easily identified. This automatic analysis thus focuses the manual task of finding new potential rule candidates on "interesting" rules.

5 Evaluation

The following table summarizes the results for some of the automatically applied Acrolinx rules for translations from English to French. The results are relative to the

instances in which a rule could be applied. The table indicates for each scoring metric in how many of these instances the reformulation of the source segment resulted in a better translation ("+" column), and in how many instances it resulted in a worse translation ("-" column). Note that for the remaining instances, the translation quality did not change.

	Human evaluation		**GTM**		**BLEU**		**TER**	
	+	-	+	-	+	-	+	-
Use relative pronouns such as that and which	33%	4%	26%	19%	26%	7%	15%	15%
Confusion between noun and adjective	23%	8%	46%	0%	46%	0%	38%	8%
Avoid contractions	27%	12%	31%	12%	31%	8%	27%	19%
Internet and Web capitalization	30%	17%	30%	22%	22%	9%	22%	13%

Two conclusions can be drawn from the table. First, the four rules shown have a predominantly positive impact on the translation quality, and can thus be used in a fully automatic pre-editing mechanism. Second, and more importantly, the ratings based on the calculated automatic scores correlate well with human judgements. This leaves us confident that we can use the automatic framework to easily identify suitable pre-editing rules, and possibly to support the development of rules that have a positive impact on translation quality.

6 Summary

We have shown how CNL authoring tools can support and supplement the process of machine translation of user-generated content. The ACCEPT research project aims at translating user-generated content automatically with a statistical MT system that is trained on user manuals. This divergence of training data and test data is due to the fact that there simply is not enough human-translated UGC available to train an SMT system. Pre-editing rules implemented in the CNL authoring tool are aimed at reformulating the source text to make it easier to translate automatically. Therefore, we first implemented corrections of spelling and grammar, specifically adapted to the data. When using statistical MT, the translation quality is improved if corrections to the input text make it more similar to the training corpus. In a next step, we implemented pre-editing rules that stylistically reformulate the source text, such as by shortening sentences. We have shown a procedure to analyze the corpus, identify pre-editing rules, implement them and evaluate their impact.

Post-editing rules are applied to the MT output. They reformulate the output text to fix common mistakes by the MT system. These rules concern grammar, spelling, and standard stylistic reformulations as well as terminology corrections.

The automatic application of CNL reformulations seems feasible and can be part of the machine translation process. We implemented an automatic tool to support the identification of suitable pre-editing rules that can be safely automatically applied, and that clearly improve the translation quality. This tool is also used to support the identification and development of new rules, even for manual pre-editing workflows.

The next step will be to conduct more evaluations and therefore get a better insight into the impact of specific rules on the MT process. Further, we will evaluate the translation of more language pairs.

Future work will focus on analysis of the training corpus. The idea is to investigate whether the training corpus provides clues about which rules would be best to use. More concretely, it would be a sort of "reverse engineering": if rules don't trigger in the training corpus or trigger very rarely, then it is likely that the structure or pattern the rule is looking for does not occur in the training corpus. In that case, we would assume that this rule might be a good rule candidate for pre-editing. On the other hand, if a rule triggers a lot in the training corpus, it means that the structure or pattern in question is frequent in the training corpus. As a consequence, we would assume that the MT engine has no problem translating them and we would not take it up as rule candidate. So we hope that the trigger frequency correlates with its usefulness in pre-editing.

The advantage of such an approach is that it would help us automatically select different rules depending on which training corpus we use – a feature which is highly valuable as it would allow us to take into account the specificities of different MT systems.

Acknowledgments. This project is carried out with financial support from the European Community through the 7th Framework Programme for Research and Technological Development (grant agreement 288769).

References

Aikawa, T., Schwartz, L., King, R., Corston-Oliver, M., Lozano, C.: Impact of controlled language on translation quality and post-editing in a statistical machine translation environment. In: Proceedings of MT Summit XI, Copenhagen, Denmark, pp. 1–7 (2007)

Allen, J.: Postediting: an integrated part of a translation software program. In: Language International, pp. 26–29 (April 2001)

de Sousa, S., Aziz, W., Specia, L.: Assessing the Post-Editing Effort for Automatic and Semi-Automatic Translations of DVD Subtitles. In: Proceedings of the International Conference Recent Advances in Natural Language Processing 2011, Hissar, Bulgaria, pp. 97–103. RANLP 2011 Organising Committee (September 2011)

Dugast, L., Senellart, J., Koehn, P.: Statistical post-editing on SYSTRAN's rule-based translation system. In: Proceedings of the Second Workshop on Statistical Machine Translation, Prague, Czech Republic, pp. 220–223. Association for Computational Linguistics (June 2007)

Eisele, A.: Hybrid Architectures for Better Machine Translation. GSCL WS "Kosten und Nutzen von MT", Potsdam (September 2009)

Garcia, I.: Translating by post-editing: is it the way forward? Machine Translation 25, 217–237 (2011), doi:10.1007/s10590-011-9115-8

Guzmán, R.: Advanced automatic mt postediting. Multiling Computing 19(3), 52–57 (2008)

Koehn, P.: Statistical Machine Translation. Cambridge University Press (2010)

Kohl, J.: The Global English Style Guide. Writing Clear, Translatable Documentation for a Global Market. SAS Institute INC, Cary NC (2008)

Lin, C.-Y., Och, F.J.: Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, Barcelona, Spain (2004)

Melamed, I., Green, R., Turian, J.: Precision and recall of machine translation. In: Proceedings of HLT-NAACL: Short Papers (2003)

O'Brien, S., Roturier, J.: How Portable are Controlled Languages Rules? A Comparison of Two Empirical MT Studies. In: Proceedings of MT Summit XI, Copenhagen, Denmark, pp. 345–352 (2007)

Plitt, M., Masselot, F.: A Productivity Test of Statistical Machine Translation Post-Editing in a Typical Localisation Context. The Prague Bulletin of Mathematical Linguistics 93, 7–16 (2010)

Simard, M., Ueffing, N., Isabelle, P., Kuhn, R.: Rule-Based Translation with Statistical Phrase-Based Post-Editing. In: Proceedings of the Second Workshop on Statistical Machine Translation, Prague, Czech Republic, pp. 203–206. Association for Computational Linguistics (June 2007)

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., Makhoul, J.: A Study of Translation Edit Rate with Targeted Human Annotation. In: Proceedings of the 7th Conference of the Association for Machine Translation of the Americas, Cambridge, Massachusetts (2006)

Somers, H.L.: Translation Memory Systems. In: Somers, H.L. (ed.) Computers and Translation: A Translator's Guide, pp. 31–48. John Benjamins Publishing Company, Amsterdam (2003)

Specia, L.: Exploiting objective annotations for measuring translation post-editing effort. In: Proceedings of the 15th International Conference of the European Association for Machine Translation, Leuven, Belgium, pp. 73–80 (2011)

Stymne, S., Ahrenberg, L.: Using a grammar checker for evaluation and postprocessing of statistical machine translation. In: Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., Tapias, D. (eds.) Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010), Valletta, Malta. European Language Resources Association (ELRA) (May 2010)

Temnikova, I.: Cognitive evaluation approach for a controlled language post-editing experiment. In: Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., Tapias, D. (eds.) Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010), Valletta, Malta. European Language Resources Association (ELRA) (May 2010)

Thicke, L.: Improving MT results: a study. Multilingual 22(1), 37–40 (2011)

SQUALL: A Controlled Natural Language for Querying and Updating RDF Graphs

Sébastien Ferré

IRISA, Université de Rennes 1
Campus de Beaulieu, 35042 Rennes cedex, France
ferre@irisa.fr

Abstract. Formal languages play a central role in the Semantic Web. An important aspect regarding their design is syntax as it plays a crucial role in the wide acceptance of the Semantic Web approach. The main advantage of controlled natural languages (CNL) is to reconcile the high-level and natural syntax of natural languages, and the precision and lack of ambiguity of formal languages. In the context of the Semantic Web and Linked Open Data, CNL could not only allow more people to contribute by abstracting from the low-level details, but also make experienced people more productive, and make the produced documents easier to share and maintain. We introduce SQUALL, a controlled natural language for querying and updating RDF graphs. It has a strong adequacy with RDF, an expressiveness close to SPARQL 1.1, and a CNL syntax that completely abstracts from low-level notions such as bindings and relational algebra. We formally define the syntax and semantics of SQUALL as a Montague grammar, and its translation to SPARQL. It features disjunction, negation, quantifiers, built-in predicates, aggregations with grouping, and n-ary relations through reification.

1 Introduction

The Semantic Web [1,7] is founded on a number of formal languages, used to represent: data (RDF), ontologies (RDFS, OWL), rules (SWRL), queries and updates (SPARQL). In this paper, we focus on queries and updates. SPARQL is a very expressive query language [11] as it includes relational algebra, and recently included aggregates, subqueries, negation, and property paths (SPARQL 1.1[1]). It has also been extended with an update language for RDF graphs (SPARQL 1.1 Update[2]). However, its usability is limited because it exhibits low-level notions from relational algebra (e.g., bindings, join, union), and logic (e.g., variables, connectors, quantifiers). An ideal candidate for combining expressiveness and usability is natural language. However, full natural languages have a weak adequacy to Semantic Web formalisms [6], because they are too expressive and too ambiguous. A number of NLP-based systems have been developped for querying the Semantic Web, e.g., Aqualog [9], FREyA [3]. However, those systems

[1] http://www.w3.org/TR/sparql11-query/
[2] http://www.w3.org/TR/sparql11-update/

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 11–25, 2012.

are generally limited to simple questions (typically equivalent to SPARQL *basic graph patterns*), and cannot be used for updates. For instance, Aqualog queries are limited to two-triples patterns. Therefore, those systems are adequate for simple needs, but are in no way a substitute for formal languages like SPARQL.

Our objective is to define an expressive formal query and update language as a fragment of a natural language. Like Montague, we think that *"There is no important theoretical difference between natural languages and the artificial languages of logicians"* [10]. This opinion is supported by the existence of Controlled Natural Languages (CNL) [13,5]. The main advantage of CNLs is to improve usability by reusing the cognitive capabilities of people, and therefore reducing their learning effort, while retaining the properties of formal languages. To the best of our knowledge, no existing CNL fulfills adequacy and interoperability with RDF and SPARQL. ACE [5] is a general purpose CNL, and requires the definition of a lexicon, which is not available for most RDF datasets (e.g., Linked Open Data). SOS or Rabbit cover OWL ontologies, and also assume some lexical knowledge [12].

In this paper, we introduce SQUALL, a Semantic Query and Update High-Level Language. It qualifies as a CNL for querying and updating RDF graphs. Its contribution is not about expressiveness, which is close to SPARQL. The contribution of SQUALL is to combine (1) an expressiveness close to SPARQL 1.1, (2) a high-level and natural syntax that completely abstracts from low-level notions such as bindings or relational algebra, and (3) a full adequacy with Semantic Web formalisms. SQUALL shares the motivation for a nicer syntax with RDF notations like Turtle and N3, but is rich enough to be a standalone language. The focus of this paper is on the syntax, retaining existing SPARQL notations at the lexical level, i.e., for non-grammatical words (URIs, literals, and variables). This makes SQUALL fully and directly interoperable with existing notations, and this removes all ambiguity at the lexical level. The drawback is that SQUALL sentences may look unnatural because URIs are invariant with respect to number or person: e.g., URI ":Person" stands for "person" and "people", URI ":knows" stands for "knows", "know". Whenever a lexicon, i.e. a non-ambiguous mapping from natural words to URIs and literals, is available or can be produced automatically from an ontology [9], it can easily be integrated into SQUALL to make sentences more natural. In any way, SQUALL has to be learned, like any other formal language. However, we think that it is easier to learn, and that it makes it easier to formulate complex queries and updates: e.g., "for which researcher-s ?X, in graph DBLP every publication whose author is ?X and whose year $\geq$ 2000 has at least 2 author-s ?" is a valid query in SQUALL.

In Section 2, we first give preliminaries about the Semantic Web and Montague grammars. In Section 3, we formally define the syntax and semantics of SQUALL, where semantics is given in terms of an intermediate logical language. Section 4 provides a translation from this intermediate language to SPARQL, therefore providing a concrete semantics as well as a possible implementation of SQUALL as a query and update interface. Section 5 presents our implementation, and

illustrates SQUALL and its translation to SPARQL with the above complex query. Section 6 concludes this paper.

2 Preliminaries

We recall basic facts about the Semantic Web and Montague grammars. The Semantic Web provides, through RDF, the data model underlying both SPARQL and SQUALL. Montague grammars provide the theoretical framework in which we formally define the syntax and semantics of SQUALL.

2.1 Semantic Web

The Semantic Web (SW) is founded on several representation languages, such as RDF, RDFS, and OWL, which provide increasing inference capabilities [7]. The two basic units of these languages are *resources* and *triples*. A resource can be either a URI (Uniform Resource Identifier), a literal (e.g., a string, a number, a date), or a *blank node*, i.e., an anonymous resource. A URI is the absolute name of a *resource*, i.e., an entity, and plays the same role as a URL w.r.t. web pages. Like URLs, a URI can be a long and cumbersome string (e.g., `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`), so that it is often denoted by a qualified name (e.g., `rdf:type`), where `rdf:` is the RDF namespace. In the N3 notation, the default namespace `:` can be omitted for qualified names that do not collide with reserved keywords (*bare qualified names*).

A triple (*s p o*) is made of 3 resources, and can be read as a simple sentence, where *s* is the subject, *p* is the verb (called the predicate), and *o* is the object. For instance, the triple (`Bob friend Alice`) says that "Bob has a friend Alice", where `Bob` and `Alice` are the bare qualified names of two individuals, and `friend` is the bare qualified name of a property, i.e., a binary relation. The triple (`Bob rdf:type man`) says that "Bob has type man", or simply "Bob is a man". Here, the resource `man` is used as a class, and `rdf:type` is a property from the RDF namespace. The triple (`man rdfs:subClassOf person`) says that "man is a subclass of person", or simply "every man is a person". The set of all triples of a knowledge base forms an RDF graph.

Query languages provide on semantic web knowledge bases the same service as SQL on relational databases. They generally assume that implicit triples have been inferred and added to the base. The most well-known query language, SPARQL, reuses the `SELECT FROM WHERE` shape of SQL queries, using graph patterns in the `WHERE` clause. A graph pattern *G* is one of:

- a triple pattern (*s p o* .) made of RDF terms and variables (e.g., `?x`),
- a conjunction of two patterns ($\{G_1\ G_2\}$),
- an union of two patterns (G_1 `UNION` G_2),
- an optional pattern (`OPTIONAL` G_1),
- a filter pattern (`FILTER` C), where C is a constraint, i.e., either a Boolean expression based on primitive predicates (e.g., comparison, string matching), or a negated graph pattern (`NOT EXISTS` G_1),

- a named graph pattern (`GRAPH` g G_1), where g is URI or a variable denoting a named graph,
- a subquery.

Aggregation operators can be used in the `SELECT` clause (e.g., `COUNT`, `SUM`), and `GROUP BY` clauses can be added to a query. SPARQL has been extended into an update language to insert and delete triples in/from a graph. The most general update form is `INSERT` I `DELETE` D `WHERE` G, where I and D must be sets of triple patterns, and G is a graph pattern that defines bindings for variables occuring in I and D.

2.2 Montague Grammars

Montague grammars [4] are an approach to natural language semantics that is based on formal logic and λ-calculus. It is named after the American logician Richard Montague, who pioneered this approach [10]. A Montague grammar is a context-free generative grammar, where each rule is decorated by a λ-term that denotes the semantics of the syntactic construct defined by the rule. For example, the following rule gives the syntax, semantics, and an example of globally existentially quantified sentences:

$$S \rightarrow \textbf{there is}\ NP \quad \{\ np\ \lambda x.\textbf{true}\ \} \quad \text{“there is } [_{NP}\text{a man]”}$$

Here, **there** and **is** are keywords, or grammatical words, of the language; S (for sentence) and NP (for noun phrase) are syntagms. The semantics is given between curly brackets, and is defined in a fully compositional style, i.e., the semantics of a construct is always a composition of the semantics of sub-constructs. The semantics of a sub-construct is given by the lowercase name of the corresponding syntagm. Every λ-term is typed, and the type of the semantics of a given syntagm is always the same. By convention, we name λ-variables according to their type: x, y, z for RDF *resources* (a.k.a. Montague *entities*), p for RDF properties, t for reified statements (both are special kinds of resources), d for *descriptions*, and s for *sentences*. Sentences are the intention of truth values (a.k.a. Montague *propositions*), and descriptions are the intention of sets of resources (a.k.a. Montague *properties*), i.e., of functions from resources to sentences. In the above rule, x denotes a resource, **true** denotes a sentence, and np denotes the intention of a set of descriptions, i.e., a function from descriptions to sentences (a.k.a. Montague *properties of properties*). Therefore, the whole λ-term denotes a sentence. The constants used in those λ-terms are the constructors of the intermediate language (e.g., **and**, **exists**), and RDF terms. The λ-terms obtained by composition can be simplified according to λ-calculus, through β-reduction (e.g., $(\lambda x.s)\ y =_\beta s[x \leftarrow y]$), and η-expansion (e.g., $d =_\eta \lambda x.(d\ x)$).

3 Syntax and Semantics of SQUALL

In this section, we formally define the syntax and semantics of SQUALL in the style of Montague grammars. This provides a translation from the concrete and

natural syntax of SQUALL to an intermediate logical language, rather than directly in terms of an existing query language for the Semantic Web. This is a common practice in the compilation of high-level programming languages, and has a number of advantages. First, it makes the semantics easier to write and understand because defined at a more abstract level. Second, it gives freedom in the choice of the implementation. For instance, the operational semantics of the intermediate language can be given by translating it to an existing language, e.g., SPARQL; by interpreting it in a relational algebra engine; or by using continuation passing-style, like in Prolog. In Section 4, we sketch a solution in the first approach.

We first define a core language corresponding to RDF triples as simple sentences. We then detail a number of orthogonal extensions: relational algebra, queries, quantifiers, reification and n-ary relations, built-in predicates, and aggregations with grouping. The Montague grammars given in this paper provide a self-contained specification of the syntax and semantics of SQUALL.

3.1 Lexical Conventions

The lexical conventions for RDF terms and variables are the same as in Turtle, N3 and SPARQL. In SQUALL examples, we assume the usual namespaces `rdf:`, `rdfs:`, and the default namespace for the domain vocabulary so that bare qualified names can be used for most classes and properties. For reasons of space in SQUALL examples, we will use capital letters (like *A*, *B*) instead of full URIs to denote individuals (e.g., publications, persons). The keywords of SQUALL are grammatical words of English: e.g., **is**, **of**, **and**, **every**.

In the scope of this paper, SQUALL directly uses URIs for non-grammatical words, and therefore makes no distinction between singular and plural, nor between nouns and verbs at the lexical level (it does however at the syntactic level). A more essential distinction, which is readily available in RDF schemas, is between unary predicates called *classes*, and binary predicates called *properties*. Therefore, RDF classes are used as nouns (e.g., *woman*) and intransitive verbs (e.g., *work*), while RDF properties are used as relation nouns (e.g., *author*) and transitive verbs (e.g, *know*). For a slightly improved presentation of SQUALL sentences, we allow grammatical suffixes to be appended to URIs: e.g., "work-s" for the third person singular, "author-s" for the plural.

3.2 Triples

A triple (*s p o*) can be seen as a sentence. The tradition in linguistics [2] is to analyse *s* and *o* as noun phrases (*NP*), *p o* as a verb phrase (*VP*), and the whole triple as a sentence (*S*). In a Semantic Web context, a *NP* can be a term (*Term*), i.e., one of a URI, a literal, or a variable. A unary predicate (*P1*) can be used as an intransitive verb, and a binary predicate (*P2*) can be used as a transitive verb followed by an object *NP*. In a Semantic Web context, a *P1* is typically a class URI, and a *P2* is typically a property URI. Variables can also be used as predicates.

$$
\begin{array}{lll}
S \rightarrow NP\ VP\ \{\ np\ vp\ \} & \text{“}[_{NP}\text{A}]\ [_{VP}\text{know-s B}]\text{”} \\
NP \rightarrow Term\ \{\ \lambda d.(d\ term)\ \} & \text{“A”} \\
VP \rightarrow P1\ \{\ \lambda x.(p1\ x)\ \} & \text{“}[_{P1}\text{work-s}]\text{”} \\
\quad \mid\ P2\ NP\ \{\ \lambda x.(np\ \lambda y.(p2\ x\ y))\ \} & \text{“}[_{P2}\text{know-s}]\ [_{NP}\text{B}]\text{”} \\
P1 \rightarrow ClassURI\ \{\ \lambda x.(\textbf{type}\ x\ uri)\ \} & \text{“work”} \\
\quad \mid\ Var\ \{\ \lambda x.(\textbf{type}\ x\ var)\ \} & \text{“?C”} \\
P2 \rightarrow PropertyURI\ \{\ \lambda x.\lambda y.(\textbf{stat}\ x\ uri\ y)\ \} & \text{“know”} \\
\quad \mid\ Var\ \{\ \lambda x.\lambda y.(\textbf{stat}\ x\ var\ y)\ \} & \text{“?P”}
\end{array}
$$

In the semantics, (**stat** x p y) denotes a triple statement x p y, and (**type** x c) is a shorthand for (**stat** x `rdf:type` c). We also define (**thing** x) as a shorthand for (**type** x `rdfs:Resource`). The semantics of a term, when used as a NP, is the set of descriptions (λd) of which the term is an instance (d $term$). The semantics of a verb phrase ($P2$ NP) is the description of resources x such that the description of the resources y that are connected to x through the property $p2$ ($p2$ x y), is an instance of np. For instance, the sentence “A know-s B” is parsed as “$[_S[_{NP}A]\ [_{VP}[_{P2}\text{know-s}]\ [_{NP}B]]]$”, and translates to ($\lambda d.(d\ A)\ \lambda x.(\lambda d.(d\ B)\ \lambda y.(\textbf{stat}\ x$ `know` $y))$), which reduces to (**stat** A `know` B). This complexity makes sense when introducing determiners and quantifiers (see Section 3.6).

3.3 Relational Algebra

For queries, SQUALL provides the algebraic operators of SPARQL [11]: **and** for joins, **or** for unions, **not** for differences and negations, **maybe** for optional patterns. Like in Turtle, the coordination **and** can be replaced by a *dot* (**.**) for sentences (S), and by a *comma* (**,**) for NPs.

$$
\begin{array}{lll}
\Delta \rightarrow \textbf{not}\ \Delta_1\ \{\ \textbf{not}\ \delta_1\ \} & \text{“not }[_{VP}\text{know-s B}]\text{”} \\
\quad \mid\ \Delta_1\ \textbf{and}\ \Delta_2\ \{\ \textbf{and}\ \delta_1\ \delta_2\ \} & \text{“}[_{VP}\text{work-s}]\text{ and }[_{VP}\text{cite-s X}]\text{”} \\
\quad \mid\ \Delta_1\ \textbf{or}\ \Delta_2\ \{\ \textbf{or}\ \delta_1\ \delta_2\ \} & \text{“}[_{NP}\text{A}]\text{ or }[_{NP}\text{B}]\text{”} \\
\quad \mid\ \textbf{maybe}\ \Delta_1\ \{\ \textbf{option}\ \delta_1\ \} & \text{“maybe }[_{VP}\text{know-s B}]\text{”}
\end{array}
$$

In this rule, Δ stands for any syntagm so that algebraic operators can coordinate all kinds of constructs: e.g., “A or B work-s and cite-s X”, which is equivalent to “A work-s and cite-s X or B work-s and cite-s X”. For a same constructor to apply in constructs having different types, it is necessary to add rewriting rules of λ-terms like the following: **and** δ_1 δ_2 $\alpha \rightsquigarrow$ **and** (δ_1 α) (δ_2 α), where α stands for any argument: e.g., α is an entity x if $\Delta = VP$, α is a description d if $\Delta = NP$.

3.4 Headed NPs, Relatives, and Auxiliary Verbs

This section augments the syntax to make it more natural and flexible. A NP can be made of a head preceded by a determiner (Det), and followed by an optional term as apposition (App), and a coordination of relatives (Rel): e.g., “a woman ?A

that is an author of X". The syntagms *NG1* and *NG2* (nominal groups) describe the possible heads, and the relative position of apposition and relatives (AR), which are both optional. *Rel* defines the possible relatives. Headed noun phrases allow for new verbal forms based on the auxiliary verbs **is** and **has**: e.g., "is a woman", "has an author". The syntagms *NG1*, *AR*, *Rel*, *AP* denote descriptions, the syntagm *NG2* denote binary predicates, while determiners denote binary relations between descriptions (i.e., quantifiers).

$$
\begin{array}{lll}
NP & \to Det\ NG1\ \{\ \lambda d.(det\ (\textbf{init}\ ng1)\ d)\ \} & \text{"[}_{Det}\text{a] [}_{NG1}\text{woman]"} \\
 & \mid Det\ NG2\ \textbf{of}\ NP\ \{\ \lambda d.(np\ \lambda x.(det\ (\textbf{init}\ (ng2\ x))\ d))\ \} & \\
 & & \text{"[}_{Det}\text{the] [}_{NG2}\text{author-s] of [}_{NP}\text{X]"} \\
Det & \to \textbf{a(n)}\ \{\ \lambda d_1.\lambda d_2.(\textbf{exists}\ (\textbf{and}\ d_1\ d_2))\ \} & \\
 & \mid \textbf{the}\ \{\ \lambda d_1.\lambda d_2.(\textbf{the}\ d_1\ d_2)\ \} & \\
NG1 & \to \textbf{thing}\ AR\ \{\ \textbf{and thing}\ ar\ \} & \text{"thing [}_{AR}\text{that cite-s A]"} \\
 & \mid P1\ AR\ \{\ \textbf{and}\ p1\ ar\ \} & \text{"[}_{P1}\text{woman] [}_{AR}\text{?A]"} \\
NG2 & \to P2\ AR\ \{\ \lambda x.\lambda y.(\textbf{and}\ (p2\ x\ y)\ (ar\ y))\ \} & \\
 & & \text{"[}_{P2}\text{author] [}_{AR}\text{?A]"} \\
AR & \to App\ Rel\ \{\ \textbf{and}\ app\ rel\ \} & \text{"[}_{App}\text{?A] [}_{Rel}\text{that X cite-s]"} \\
 & \mid App\ \{\ app\ \} & \\
App & \to URI\ \{\ \lambda x.(\textbf{eq}\ x\ uri)\ \} & \text{"A"} \\
 & \mid Var\ \{\ \lambda x.(\textbf{bind}\ x\ var)\ \} & \text{"?X"} \\
 & \mid \epsilon\ \{\ \lambda x.\textbf{true}\ \} & \\
Rel & \to \textbf{that}\ VP\ \{\ \textbf{init}\ vp\ \} & \text{"that [}_{VP}\text{know-s B]"} \\
 & \mid \textbf{that}\ NP\ P2\ \{\ \textbf{init}\ \lambda x.(np\ \lambda y.(p2\ y\ x))\ \} & \\
 & & \text{"that [}_{NP}\text{X] [}_{P2}\text{cite-s]"} \\
 & \mid \textbf{such that}\ S\ \{\ \textbf{init}\ \lambda x.s\ \} & \text{"such that [}_{S}\text{?A work-s]"} \\
 & \mid Det\ NG2\ \textbf{of which}\ VP\ \{\ \textbf{init}\ \lambda x.(det\ (ng2\ x)\ vp)\ \} & \\
 & & \text{"[}_{Det}\text{an] [}_{NG2}\text{author] of which [}_{VP}\text{know-s B]"} \\
 & \mid \textbf{whose}\ NG2\ VP \equiv \textbf{the}\ NG2\ \textbf{of which}\ VP & \\
 & & \text{"whose [}_{NG2}\text{author ?A] [}_{VP}\text{cites-s a colleague of ?A]"} \\
 & \mid \textbf{whose}\ P2\ \textbf{is/are}\ NP\ \{\ \lambda x.(np\ \lambda y.(p2\ x\ y))\ \} & \\
 & & \text{"whose [}_{P2}\text{author] [}_{VP}\text{is a woman]"} \\
VP & \to \textbf{is/are}\ AP\ \{\ ap\ \} & \text{"is [}_{AP}\text{a woman]"} \\
 & \mid \textbf{is/are}\ Rel\ \{\ rel\ \} & \text{"is [}_{Rel}\text{such that ?A work-s]"} \\
 & \mid \textbf{has/have}\ Det\ P2\ AR\ \{\ \lambda x.(det\ (p2\ x)\ ar)\ \} & \\
 & & \text{"have [}_{Det}\text{an] [}_{P2}\text{author] [}_{AR}\text{that X cite-s]"} \\
AP & \to Term\ \{\ \lambda x.(\textbf{eq}\ x\ term)\ \} & \text{"A"} \\
 & \mid \textbf{a(n)/the}\ NG1\ \{\ ng1\ \} & \text{"a [}_{NG1}\text{woman]"} \\
 & \mid \textbf{a(n)/the}\ NG2\ \textbf{of}\ NP\ \{\ \lambda x.(np\ \lambda y.(ng2\ y\ x))\ \} & \\
 & & \text{"the [}_{NG2}\text{author] of [}_{NP}\text{X]"} \\
S & \to S_1\ \textbf{where}\ S_2\ \{\ \textbf{where}\ s_1\ s_2\ \} & \\
 & & \text{"[}_{S}\text{A know-s ?B] where [}_{S}\text{?B is a woman]"}
\end{array}
$$

The meaning of the function **init** is clarified in Section 3.7 about reification. At this point, it can be assumed to be the identity function. The constructor **exists**

is an existential quantification over an entity. The constructor **eq** represents equality (a binary predicate), and the constructor (**bind** x y) represents an assignement of x into y. The keywords **the** and **where** are propagated to the semantics using the constructors **the** and **where**, because their interpretation differs whether they occur in the scope of a query or an update (see Section 4). In short, they are respectively equivalent to **a** and **and** in queries, and separate conditions and changes in updates.

3.5 Queries

Question words distinguish declarative sentences (updates) from interrogative sentences (queries): **whether** introduces a closed question, whose semantics encapsulates the sentence in the constructor **ask**; **what** can be used in place of any *NP* to form open questions. The constructor **select** applies to a description, and indicates that the instances of this description should be returned as query results. Multi-dimensional queries are expressed by using several occurences of **what**: e.g., "what is the author of what". Some question words are used as determiners. **which** has the same effect as **what** while allowing a restriction on returned resources: e.g., "which woman is an author of X". **how many** provides the easy expression of the most common aggregation, counting: e.g., "how many person is an author of X". The constructor **count** applies to a description, and indicates that the number of its instances should be returned as a query result.

$$
\begin{array}{lll}
S & \rightarrow \textbf{whether } S_1 \ \{ \textbf{ ask } s_1 \ \} & \text{"whether } [_S \text{A know-s a woman]"} \\
NP & \rightarrow \textbf{what} \equiv \textbf{which thing} & \\
 & \mid \textbf{whose } \mathit{NG2} \equiv \textbf{the } \mathit{NG2} \textbf{ of what} & \\
 & & \text{"whose } [_{\mathit{NG2}} \text{author ?A]"} \\
\mathit{Det} & \rightarrow \textbf{which } \{ \lambda d_1.\lambda d_2.(\textbf{select } (\textbf{and } d_1 \ d_2)) \} & \\
 & \mid \textbf{how many } \{ \lambda d_1.\lambda d_2.(\textbf{count } (\textbf{and } d_1 \ d_2)) \} &
\end{array}
$$

In order for the question constructors not to appear in the scope of non-question constructors, we introduce rewriting rules like the following: **and** (**select** $\lambda x.s_1$) $s_2 \rightsquigarrow$ **select** $\lambda x.$(**and** s_1 s_2); **exists** $\lambda x.$(**select** $\lambda y.s$) $\rightsquigarrow$ **select** $\lambda y.$(**exists** $\lambda x.s$). Similar rules are to be defined for **count**, other algebraic operators (like **and**) and other quantifiers (like **exists**), which are presented in Section 3.3 and 3.6. Also, we ensure that no **select** falls in the scope of **count** by adding the additional rewriting rule: **count** $\lambda x.$(**select** $\lambda y.s$) $\rightsquigarrow$ **select** $\lambda y.$(**count** $\lambda x.s$).

A sentence is a valid query if its semantics contains either one **ask** or any number of **select** and possibly one **count**. A sentence is a valid update if its semantics contains none of the question constructors. Otherwise, the sentence is invalid, and a syntax error should be returned.

3.6 Quantifiers

Quantifiers are commonplace in natural languages in the form of determiners, whereas they are notoriously difficult to express in SPARQL or SQL [8]. The

quantifier **exists** applies to a description, and checks that its extension is not empty. The quantifier **forall** applies to two descriptions, and checks that the extension of the first is included in the extension of the second. The constructor **atleast** i applies to a description, and checks that its extension has at least i elements. A possible use of them is: "every person is an author of at least 10 publication-s".

$$
\begin{aligned}
\mathit{Det} \rightarrow\ & \textbf{some}\ \ \{\ \lambda d_1.\lambda d_2.(\textbf{exists}\ (\textbf{and}\ d_1\ d_2))\ \} \\
\mid\ & \textbf{every}\ \ \{\ \lambda d_1.\lambda d_2.(\textbf{forall}\ d_1\ d_2)\ \} \\
\mid\ & \textbf{no}\ \ \{\ \lambda d_1.\lambda d_2.(\textbf{not}\ (\textbf{exists}\ (\textbf{and}\ d_1\ d_2)))\ \} \\
\mid\ & \textbf{at least}\ i\ \ \{\ \lambda d_1.\lambda d_2.(\textbf{atleast}\ i\ (\textbf{and}\ d_1\ d_2))\ \} \\
S \rightarrow\ & \textbf{for}\ \mathit{NP}\ ,\ S\ \ \{\ \mathit{np}\ \lambda x.s\ \} \\
\mid\ & \textbf{there is}\ \mathit{NP}\ \ \{\ \mathit{np}\ \lambda x.\textbf{true}\ \}
\end{aligned}
$$

The grammar rules are defined so that the scope of quantifiers are leftmost-outermost, and are restricted to the scope of the related verb. Therefore, "every man love-s some woman" means there is possibly a different woman for each man; while in "there is a woman that every man love-s" means there is a single woman.

The keywords **for** and **there is** introduce global quantifiers, in a style closer to mathematical logic. Their scope extends to the end of the sentence, which may be a coordinated sentence: e.g., "for every publication ?X, ?X has an author ?A and ?A cite-s ?X". Relatives introduced by **such that** can be used in a global existential quantification: "there is a person ?X such that no publication has the author ?X".

3.7 Reification and N-Ary Predicates

RDF triples can be reified as statements. Therefore, it is useful to allow statements about statements. The keyword **that** turns a sentence into a noun phrase, changing the focus from the truth of the sentence to the statements involved in the sentence. The variable t is used for denoting individual statements. Here it becomes necessary to explicit two arguments shared by all constructs, and that were left implicit so far (thanks to η-equivalence of λ-calculus): $\overline{a}$ is a list of additional arguments to the predicate of the sentence, and g denotes a graph, i.e., a set of statements. Together, they form the context in which a sentence is interpreted. Contexts are the counterpart of worlds in Montague semantics. The extension of a phrase is obtained by passing a context, and an intension is formed by abstracting over a context.

In the sentence "A say-s that B is an author of X", the object of the property "say" is the statement of the triple (X `author` B). In the semantics, the main clause (verb "say-s") abstracted over its object and receiving its own context ($\lambda t.(\textbf{stat}\ A\ \texttt{say}\ t\ \overline{a}\ g)$) denotes a graph (i.e., a set of statements), and is passed as a context to the dependent clause (**stat** X `author` B). This generalizes the GRAPH construct in SPARQL, where a graph would be restricted to the form $\lambda t.(\textbf{stat}\ t\ \texttt{graph}\ G)$, where the triple ($t$ `graph` G) would mean that the statement t is a member of ("has graph") the named graph G (a URI or a variable).

The other rules define prepositions (*Prep*) and prepositional phrases (*PP*), and their inclusion in sentences. A *PP* is introduced by **at** or **in**, and is followed

by a *Prep* and a *NP*. It modifies a sentence by adding an argument to its context (constructor **arg**), or by encapsulating it in a graph (constructor **graph**). Each argument is a pair $(uri, term)$, where *uri* defines the role of the argument w.r.t. the statement, much like "rdf:subject", "rdf:predicate", and "rdf:object"; and *term* is the argument itself. Those arguments provide means for expressing n-ary relationships. For example, "A is an author of X at rank 1" means that A is the first author of X; and "A sell-s X in year 2009 at amount 10" means that A sold 10 units of X in 2009.

The remaining rules say that prepositional phrases can occur anywhere in a sentence, and the syntagms *OP* (object phrase) and *CP* (complement phrase) are introduced to retain the leftmost-outermost scope rule for quantifiers. Therefore, the rigidity of this scope rule is balanced by the free ordering of *PP*s, the possible inversion between subject and object (**of**), and the global quantifiers introduced by **for**. An example of query that uses a number of those constructs is: "at which venue every professor that work-s at place X speak-s at some time".

$$
\begin{array}{lll}
NP & \rightarrow \textbf{that}\ S\ \{\ \lambda d.\lambda\overline{a}.\lambda g.(s\ ()\ \lambda t.(d\ t\ \overline{a}\ g))\ \} & \\
 & & \text{"that } [_S \text{A know-s B}]\text{"} \\
PP & \rightarrow \textbf{at/in}\ Prep\ NP\ \{\ \lambda s.(np\ \lambda z.(prep\ z\ s))\ \} & \\
 & & \text{"at place } [_{NP} \text{the city Rennes}]\text{"} \\
 & \mid\ \textbf{at/in}\ Det\ Prep\ AR \equiv \textbf{at}\ Prep\ Det\ \textbf{thing}\ AR & \\
 & & \text{"at } [_{Det} \text{some}] \text{ venue } [_{AR} \text{whose place is Rennes}]\text{"} \\
S & \rightarrow PP\ S\ \{\ pp\ s\ \} & \\
VP & \rightarrow PP\ VP\ \{\ \lambda x.(pp\ (vp\ x))\ \} & \\
 & \mid\ P1\ CP\ \{\ \lambda x.(cp\ (p1\ x))\ \} & \\
 & \mid\ P2\ OP\ \{\ \lambda x.(op\ (p2\ x))\ \} & \\
 & \mid\ \textbf{is/are}\ AP\ CP\ \{\ \lambda x.(cp\ (ap\ x))\ \} & \\
 & \mid\ \textbf{has/have}\ Det\ P2\ AR\ CP\ \{\ \lambda x.(det\ \lambda y.(cp\ (p2\ x\ y))\ ar)\ \} & \\
OP & \rightarrow PP\ OP\ \{\ \lambda d.(pp\ (op\ d))\ \} & \\
 & \mid\ NP\ CP\ \{\ \lambda d.(np\ \lambda y.(cp\ (d\ y)))\ \} & \\
CP & \rightarrow PP\ CP\ \{\ \lambda s.(pp\ (cp\ s))\ \}\ \mid\ \epsilon\ \{\ \lambda s.s\ \} & \\
Rel & \rightarrow \textbf{at/in which}\ Prep\ AR\ S\ \{\ \textbf{init}\ \lambda x.(\textbf{and}\ (ar\ x)\ (prep\ x\ s))\ \} & \\
 & & \text{"in which graph } [_S \text{A work-s}]\text{"} \\
Prep & \rightarrow \textbf{graph}\ \{\ \textbf{graph}\ \}\ \mid\ URI\ \{\ \textbf{arg}\ uri\ \} &
\end{array}
$$

It is now possible to clarify and define the constructors **init** and **arg** that have already been used. The constructor **init** reinitializes the list of arguments in some construct: $\textbf{init} = \lambda d.\lambda x.\lambda\overline{a}.(d\ x\ ())$. This is useful to restrict the passing of arguments to the predicate of a sentence, and to avoid its propagation to noun groups and relatives for instance. The constructor **arg** adds an argument to the current list of arguments, waiting to be passed to the main predicate of the sentence: $\textbf{arg} = \lambda uri.\lambda z.\lambda s.\lambda\overline{a}.(s\ ((uri, z), \overline{a}))$. **graph** is a special preposition that represents the membership of statements to SPARQL named graphs.

In order to pass the context down to arguments of algebraic operators, quantifiers, and question constructors, it is necessary to define rewriting rules like

the following: **true** $\overline{a}$ $g \rightsquigarrow$ **true**, **and** s_1 s_2 $\overline{a}$ $g \rightsquigarrow$ **and** $(s_1$ $\overline{a}$ $g)$ $(s_2$ $\overline{a}$ $g)$, **exists** d $\overline{a}$ $g \rightsquigarrow$ **exists** $\lambda x.(d\ x\ \overline{a}\ g)$.

3.8 Built-In Predicates and Aggregations

Built-in predicates are used in SPARQL constraints (e.g., comparison, arithmetic, string matching), and expect a variable number of arguments. They can be used as nouns or intransitive verbs (*Pred1URI*), when one argument plays the role of a subject, and as relation noun or transitive verbs (*Pred2URI*), when two arguments play the role of subject and object. Additional arguments are represented through prepositional phrases, assigning a different preposition URI to each. For example, assuming the binary predicate "match" for regular expression matching, and the unary predicate "Monday" denoting the set of Monday dates, the following query can be expressed: "which woman has a lastname that match-es "Smi.*" and a birth whose date is a Monday".

The head of a noun phrase (*NG1*) can be an aggregator followed by the set of what should be aggregated, and optionally followed by a list of grouping dimensions introduced by **per**. Each dimension is an *AP* that specifies the set of possible values for this dimension: e.g., "what is the count of the publication-s ?P per the year of ?P, and the affiliation of an author of ?P". This construction produces bindings for the aggregated value and each dimension.

$P1 \rightarrow Pred1URI\ \{\ \lambda x.(\textbf{pred1}\ uri\ x)\ \}$ "Monday"
$P2 \rightarrow Pred2URI\ \{\ \lambda x.\lambda y.(\textbf{pred2}\ uri\ x\ y)\ \}$ "match"
$NG1 \rightarrow AggregURI\ \textbf{of}\ AP\ (\textbf{per}\ AP_i^+)?\ \{\ \lambda x.(\textbf{aggreg}\ uri\ x\ ap\ (ap_i)_i)\ \}$
"count of [$_{AP}$the publication ?P] per [$_{AP}$the year of ?P]"

3.9 Handling of Ambiguity

The price for the natural and flexible syntax of SQUALL is ambiguity, i.e., the fact that some sentences can be parsed in different ways possibly leading to different semantics. In SQUALL, ambiguities are resolved by the following rules:

1. when forming a construct Δ from one or two constructs of same syntagm Δ (e.g., coordinating 2 *NP*s, modifying a sentence with a *PP*), algebraic operators have priority (in decreasing priority order: **not**, **maybe**, **and**, **or**, **where**) over sentence modifiers (*PP* as a prefix, and global quantifiers **for** *NP*). Punctuation has lowest priority, and right-associativity is used for binary coordinations;
2. smaller syntagms have priority over larger syntagms, i.e., in decreasing priority order: *P1*, *P2*, *Det*, *Rel*, *NG1*, *NG2*, *AP*, *NP*, *PP*, *CP*, *OP*, *VP*, *S*;
3. in case of ambiguity between forming two constructs of same syntagm, the shorter construct is chosen.

Round brackets can be used for every syntagms to escape those rules. Rule 2 implies that "a man or woman" is interpreted as "a (man or woman)" rather than "(a man) or woman", as *NG1* has priority over *NP*. Rule 3 implies that in "*A* know-s a researcher that *X* cite-s at venue *V*", the PP "at venue *V*" binds to the shorter VP "cite-s ..." rather than to the longer VP "know-s ...".

4 Translation to SPARQL

An operational semantics can be given to SQUALL by translating its intermediate language to SPARQL. Given a SQUALL sentence S with semantics s, the translation of S in the intermediate language is the formula $f = s\ ()\ g_0$, i.e., the sentence initialized with an empty list of arguments, and some default graph g_0 (e.g., $\lambda t.\mathbf{true}$). In order to simplify this formula, we remove some of the constructors of the intermediate language by giving them a definition. For example, the triple $(x\ p\ y)$ passed to the constructor **stat** is reified by stating the existence of a statement resource t, which is connected to its subject, predicate, object, and arguments through roles. Each connection is represented using the new constructor **triple** that represents a non-reifiable triple. Those connections are completed by stating that the statement t belongs to the given graph g. This can be simplified into **triple** $x\ p\ y$ when there are no argument, and the graph is $\lambda t.\mathbf{true}$. The constructor **eq** is defined by the built-in predicate $=$. The constructors **atleast** and **count** are defined in terms of the aggregator `COUNT`, and the built-in predicate $\geq$. The constructor **fold** is the classical iterator on lists. A term $(\mathbf{fold}\ f\ e\ (x, \overline{x}))$ reduces to $(\mathbf{fold}\ f\ (f\ e\ x)\ \overline{x})$, and a term $(\mathbf{fold}\ f\ e\ ())$ reduces to e.

$\mathbf{stat} = \lambda x.\lambda p.\lambda y.\lambda \overline{a}.\lambda g.(\mathbf{exists}\ \lambda t.(\mathbf{fold}$
$\quad \lambda q.\lambda(uri, z).(\mathbf{and}\ q\ (\mathbf{triple}\ t\ uri\ z))\ (g\ t)$
$\quad ((\text{rdf:subject}, x), (\text{rdf:predicate}, p), (\text{rdf:object}, y), \overline{a})))$
$\mathbf{eq} = \lambda x.\lambda y.(\mathbf{pred2}\ (=)\ x\ y)$
$\mathbf{atleast} = \lambda i.\lambda d.(\mathbf{exists}\ \lambda x.(\mathbf{and}\ (\mathbf{aggreg}\ \texttt{COUNT}\ x\ d\ ())\ (\mathbf{pred2}\ \geq\ x\ i)))$
$\mathbf{count} = \lambda d.(\mathbf{select}\ \lambda x.(\mathbf{aggreg}\ \texttt{COUNT}\ x\ d\ ()))$

After applying those definitions, the only remaining constructors are: **triple**, **bind**, **pred1**, **pred2**, **aggreg**, **true**, **not**, **and**, **or**, **option**, **where**, **exists**, **forall**, **the**, **ask**, **select**. We define in the following their translation to the queries and updates of SPARQL. Those translations are chosen to be concise, and not to be optimal in any way. The SPARQL translation of a formula f is denoted by $[f]$, and $[X \mid f]_Q$ is an auxialiary translation for multi-dimensional queries. The two other auxiliary translations are $[f]_G$ for producing graph patterns (that generate bindings), and $[f]_U$ for producing updates (that insert and delete triples). An update is a triple (I, D, G), where I is a graph to be inserted, D is a graph to be deleted, and G is a graph pattern.

$[\textbf{ask}\ f] = \texttt{ASK}\ \{\ [f]_G\ \}$
$[\textbf{select}\ d] = [?x \mid d\ ?x]_Q$
$[f] = \texttt{INSERT}\ \{I\}\ \texttt{DELETE}\ \{D\}\ \texttt{WHERE}\ \{\ G\ \} \quad \text{where } (I, D, G) = [f]_U$
$[X \mid \textbf{select}\ d]_Q = [X\ ?x \mid d\ ?x]_Q$
$[X \mid f]_Q = \texttt{SELECT}\ X\ \texttt{WHERE}\ \{\ [f]_G\ \}$

Formulas with constructors **ask** and **select** translate to the corresponding SPARQL queries, while other formulas translate to SPARQL updates. Every occurence of a SPARQL variable $?x$ assumes the generation of a fresh variable name. Those variables are used to instantiate description parameters of question, quantifier, and aggregation constructors.

$[\textbf{triple}\ s\ p\ o]_G = s\ p\ o\ .$
$[\textbf{bind}\ x\ y\ \overline{a}\ g]_G = \texttt{BIND}\ (\ x\ \texttt{AS}\ y\ )$
$[\textbf{pred1}\ pred\ x\ \overline{a}\ g]_G = \texttt{FILTER}\ pred(x, \overline{a})$
$[\textbf{pred2}\ pred\ x\ y\ \overline{a}\ g]_G = \texttt{FILTER}\ pred(x, y, \overline{a})$
$[\textbf{graph}\ x\ s\ \overline{a}\ g]_G = \texttt{GRAPH}\ x\ \{\ [s\ ()\ \lambda t.\textbf{true}]_G\ \}$
$[\textbf{aggreg}\ agg\ x\ d\ (d_i)_i\ \overline{a}\ g]_G =$
$\{\ \texttt{SELECT}\ (?z_i)_i\ (\ agg(?y)\ \text{AS}\ ?x\ )$
$\texttt{WHERE}\ \{\ [\textbf{fold}\ (\textbf{and})\ (d\ ?y)\ (d_i\ ?z_i)_i]_G\ \}$
$\texttt{GROUP BY}\ (?z_i)_i\ \}$
$[\textbf{true}]_G = \epsilon$
$[\textbf{and}\ f_1\ f_2]_G = [f_1]_G\ [f_2]_G$
$[\textbf{or}\ f_1\ f_2]_G = \{\ [f_1]_G\ \}\ \texttt{UNION}\ \{\ [f_2]_G\ \}$
$[\textbf{option}\ f]_G = \texttt{OPTIONAL}\ \{\ [f]_G\ \}$
$[\textbf{not}\ f]_G = \texttt{FILTER NOT EXISTS}\ \{\ [f]_G\ \}$
$[\textbf{where}\ f_1\ f_2]_G = [\textbf{and}\ f_1\ f_2]_G$
$[\textbf{exists}\ d]_G = [d\ ?x]_G$
$[\textbf{forall}\ d_1\ d_2]_G = [\textbf{not}\ (\textbf{exists}\ (\textbf{and}\ d_1\ (\textbf{not}\ d_2)))]_G$
$[\textbf{the}\ d_1\ d_2]_G = [\textbf{exists}\ (\textbf{and}\ d_1\ d_2)]_G$

Built-in predicates translate to SPARQL filters, and aggregations translate to SPARQL aggregative sub-queries. Arguments can be used for n-ary predicates, but there is no counterpart in SPARQL for aggregations. Algebraic constructors translate to their SPARQL counterpart, and quantifiers all translate to the implicit SPARQL existential quantifier and negation.

$[\textbf{triple}\ s\ p\ o]_U = (s\ p\ o\ .,\epsilon,\epsilon)$
$[\textbf{true}]_U = (\epsilon,\epsilon,\epsilon)$
$[\textbf{and}\ f_1\ f_2]_U = (I_1\ I_2, D_1\ D_2, G_1\ G_2)$
$[\textbf{not}\ f]_U = (D, I, G) \quad \text{where } (I, D, G) = [f]_U$
$[\textbf{where}\ f_1\ f_2]_U = (I_1, D_1, G_1\ [f_2]_G) \quad \text{where } (I_1, D_1, G_1) = [f_1]_U$
$[\textbf{exists}\ d]_U = [d\ ?x]_U$
$[\textbf{forall}\ d_1\ d_2]_U = [\textbf{where}\ (d_2\ ?x)\ (d_1\ ?x)]_U$
$[\textbf{the}\ d_1\ d_2]_U = [\textbf{where}\ (d_2\ ?x)\ (d_1\ ?x)]_U$

Compare the translation of **where** with graph patterns. In updates, it introduces a graph pattern, whereas in a graph pattern, it refines it like **and**. The same can be said for the quantifier **the**.

5 Implementation and Illustration

The contents of this paper has been implemented in less than 500 lines of OCaml[3], a functional language. The code is very close to the formalism used in this paper, which makes it easy to extend the theory and the code in parallel. The source code, and a SPARQL translator Web form are available from the SQUALL web page at `http://www.irisa.fr/LIS/softwares/squall`.

As an illustration of the translation to SPARQL, we consider the sentence given in the introduction: "for which researcher-s ?X, in graph DBLP every publication whose author is ?X and whose year $\geq$ 2000 has at least 2 author-s". Its syntactic analysis is

> "[$_S$for [$_{NP}$[$_{Det}$which] [$_{NG1}$[$_{P1}$researcher-s] [$_{AR}$[$_{App}$?X]]]], [$_S$[$_{PP}$in [$_{Prep}$graph] [$_{NP}$DBLP]] [$_S$[$_{NP}$[$_{Det}$every] [$_{NG1}$[$_{P1}$publication] [$_{AR}$[$_{Rel}$[$_{Rel}$whose [$_{NG2}$[$_{P2}$author]] [$_{VP}$is [$_{AP}$?X]]] and [$_{Rel}$whose [$_{NG2}$[$_{P2}$year]] [$_{VP}$[$_{P2}\geq$] [$_{NP}$2000]]]]]]] [$_{VP}$has [$_{Det}$at least 2] [$_{P2}$author-s]]]]]",

and its SPARQL translation is as follows (with some reformatting, and unreifying triples whose reification is not used):

```
SELECT ?r
WHERE {
    ?r rdf:type :researcher .
    BIND (?r AS ?X)
    GRAPH :DBLP {
        FILTER NOT EXISTS {
            ?p rdf:type :publication .
            ?p :author ?X .
            ?p :year ?y .
            FILTER (?y >= 2000)
            FILTER NOT EXISTS {
                { SELECT COUNT(?a) AS ?n
                  WHERE { ?p :author ?a . } }
                FILTER (?n >= 2) } } } }
```

6 Conclusion

SQUALL is a Semantic Query and Update High-Level Language that provides a controlled natural language on top of SPARQL 1.1, while preserving adequacy, interoperability, and expressiveness. Its syntax and semantics are formally defined as a Montague grammar. The semantics of SQUALL sentences are formulated in a logical intermediate language. We have sketched a translation from

[3] `http://caml.inria.fr/ocaml/`

this intermediate language to SPARQL, thus providing an operational semantics and a possible implementation for SQUALL. Possible future work include the addition of natural language constructs (e.g., arithmetic and string expressions, comparatives and superlatives, anaphoras other than variables), the full coverage of SPARQL 1.1 (e.g., expressions, property paths), and the use of ontologies and lexicons to improve the grammaticality of the language.

References

1. Antoniou, G., van Harmelen, F.: A Semantic Web Primer. MIT Press (2004)
2. Biber, D., Johansson, S., Leech, G., Conrad, S., Finegan, E.: Longman grammar of spoken and written English. Pearson Education Limited (1999)
3. Damljanovic, D., Agatonovic, M., Cunningham, H.: Identification of the question focus: Combining syntactic analysis and ontology-based lookup through the user interaction. In: Language Resources and Evaluation Conference (LREC). ELRA (2010)
4. Dowty, D.R., Wall, R.E., Peters, S.: Introduction to Montague Semantics. D. Reidel Publishing Company (1981)
5. Fuchs, N.E., Kaljurand, K., Schneider, G.: Attempto Controlled English meets the challenges of knowledge representation, reasoning, interoperability and user interfaces. In: Sutcliffe, G., Goebel, R. (eds.) FLAIRS Conference, pp. 664–669. AAAI Press (2006)
6. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 502–517. Springer, Heidelberg (2004)
7. Hitzler, P., Krãtzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2009)
8. Hsu, P.-Y., Parker Jr., D.S.: Improving SQL with generalized quantifiers. In: Yu, P.S., Chen, A.L.P. (eds.) Int. Conf. Data Engineering, pp. 298–305. IEEE Computer Society (1995)
9. Lopez, V., Uren, V., Motta, E., Pasin, M.: Aqualog: An ontology-driven question answering system for organizational semantic intranets. Journal of Web Semantics 5(2), 72–105 (2007)
10. Montague, R.: Universal grammar. Theoria 36, 373–398 (1970)
11. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 30–43. Springer, Heidelberg (2006)
12. Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A comparison of three controlled natural languages for OWL 1.1. In: Clark, K., Patel-Schneider, P.F. (eds.) Workshop on OWL: Experiences and Directions (OWLED), vol. 258. CEUR-WS (2008)
13. Smart, P.: Controlled natural languages and the semantic web. Tech. rep., School of Electronics and Computer Science University of Southampton (2008), `http://eprints.ecs.soton.ac.uk/15735/`

Answer Set Programming via Controlled Natural Language Processing

Rolf Schwitter

Centre for Language Technology
Macquarie University
Sydney 2109 NSW, Australia
Rolf.Schwitter@mq.edu.au

Abstract. Controlled natural languages are subsets of natural languages that can be used to describe a problem in a very precise way, furthermore they can often be translated automatically into a formal notation. We investigate in this paper how a controlled natural language can be used as a specification language for Answer Set Programming (ASP). ASP is a declarative approach to problem solving and has its roots in knowledge representation, logic programming, and constraint satisfaction. Solutions of ASP programs are stable models (= answer sets) that build the starting point for question answering. As a proof of concept, we translate a problem specification written in controlled natural language into an ASP program and compute a stable model that contains the answers to a number of questions.

Keywords: answer set programming, controlled natural language processing, model-based problem solving, knowledge representation.

1 Introduction

In this paper we present a case study and investigate how we can solve a problem stated in a controlled natural language (CNL) using the Answer Set Programming (ASP) paradigm [1, 14, 20]. ASP is a form of declarative programming oriented towards modeling and solving difficult search problems. ASP combines an expressive representation language with a model-based specification methodology and uses efficient solving tools [5, 20]. Our goal in this paper is to specify a problem in a CNL and then to translate the problem specification automatically into an ASP program that can be executed by an ASP solver. An ASP program is a logic program whose solutions correspond to zero, one or multiple stable models (= answer sets). Our proposal amounts to writing a declarative program in CNL instead of using the ASP notation; this process requires a program development methodology that is compatible with the ASP paradigm.

CNLs are well-defined subsets of full natural languages and have been used as specification languages for various application domains [7, 10, 16, 27]. They are characterised by a restricted grammar and a limited set of vocabulary with restricted meanings. These restrictions are carefully engineered to reduce or exclude ambiguity that is pervasive in full natural language and to make it easier for

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 26–43, 2012.

a machine to process a specification. Some of these CNLs have been specifically designed so that they can be translated unambiguously into a formal notation like first-order predicate logic [10, 27] or into a version of description logic [3, 25]. It has been shown that these CNLs can not only improve the processability of a specification but also its understandability compared to a specification written in a formal language [18].

The rest of this paper is structured as follows. In Section 2, we give a brief introduction to ASP since this is a rather new programming paradigm and serves as our formal target notation. In Section 3, we sketch the predominant ASP methodology that is used for writing ASP programs. In Section 4, we take a puzzle written in full natural language as a starting point, analyse it, and reconstruct it in a CNL using the suggested ASP methodology as a guiding principle. In Section 5, we present the CNL processor that we used to process the puzzle. In Section 6, we show and discuss the resulting translation of the puzzle in ASP notation. In Section 7, we evaluate our approach and show how ASP and question answering can be combined in an elegant way. In Section 8, we discuss a number of possible improvements and optimisations with regard to the expressiveness of the CNL and the encoding of the formal representation. In Section 9, we summaries the advantages of our approach and conclude.

2 Answer Set Programming (ASP)

ASP is a form of declarative problem solving that has its roots in logic-based knowledge representation, logic programming and constraint satisfaction [1, 5, 14, 20]. Instead of writing algorithms to solve a problem at hand, the programmer describes a problem in a declarative way using a formal language and an inference engine that processes the input and finds solutions to the problem.

ASP programs look similar to Prolog programs [8] and use an extended logic programming notation, but ASP programs rely on an entirely different computational mechanism. Instead of deriving a solution from a program specification via SLDNF resolution [21] like in Prolog, finding a solution in ASP corresponds to computing a family of stable models (= answer sets) that encode solutions to the problem described by the program [22].

ASP programs are typically processed in two steps: first an ASP grounder replaces the quantifier-free predicate logic program by an equivalent propositional program that does not contain any variables, and an ASP solver then computes the solutions in form of answer sets.

To make the grounding of an ASP program efficient, ASP grounders often exploit techniques such as partial evaluation, rewriting, and techniques from the field of databases, while ASP solvers rely heavily on methods developed in the field of satisfiability solving [5]. Modern ASP tools bundle the grounder and the solver in a single program [11–13].

2.1 The ASP Language

The building blocks of an ASP program are atomic formulas (= atoms), literals (= atoms or their negation) and rules. An ASP rule is an expression of the following form:

1. $A_1 \vee ... \vee A_m \leftarrow A_{m+1}, ..., A_n, not\ A_{n+1}, ..., not\ A_o.$

where A_i's are atoms. The connective *not* stands for negation as failure [6]; *not* A_i means that A_i is not known. The connective $\leftarrow$ stands for an implication. The expression on the left-hand side of the implication symbol is called the *head* of the rule that may consist of a disjunction ($\vee$) of atoms. The expression on the right-hand side is called the *body* of the rule. If the body of a rule is empty, then the rule is called a *fact*, and if the head of a rule is empty, then the rule is called a *constraint*. In the following discussion, we will further distinguish three types of rules: basic rules, disjunctive rules, and choice rules.

2.2 Basic Rules

We start our discussion with the simple ASP program in (2) that has the form of a normal logic program [9]. It consists in our case of a basic rule and two facts. The basic rule below has one atom in the head and three atoms in the body, one of them is negated by a negation as failure operator (`not`):

2.
```
p(X,Y) :- q(X), r(Y), not s(Y).
q(1). r(2).
```

This ASP program has exactly one answer set that consists of three atoms:

```
Answer 1:  q(1) r(2) p(1,2)
```

We are not always interested in the entire answer set but sometimes only in a particular atom of the answer set. In ASP, we can selectively display those atoms of an answer set we are interested in. We can do this by adding a `#hide` declaration in combination with a `#show` declaration to the ASP program in (2). A `#hide` declaration without arguments marks all atoms as hidden by default, while a `#show` declaration can be used to specify which atoms to include in the output, for example the declaration:

3.
```
#hide. #show p/2.
```

will only display the relevant atom(s) of the answer set, in our case:

```
Answer 1:  p(1,2)
```

This mechanism is in particular useful, if we are looking for a specific answer to a question, for example in a question answering context.

2.3 Disjunctive Rules

In contrast to Prolog programs, ASP programs allow for disjunction (|) in the head of rules. The following is an ASP program that consists of a disjunctive rule and two facts:

```
4. p(X,Y) | t(Y,X) :- q(X), r(Y), not s(Y).
   q(1). r(2).
```

Disjunctive rules result in logical uncertainty; that means we end up with two answer sets for (4) and not three because of the minimality criterion [5]:

```
Answer 1:  q(1) r(2) p(1,2)
Answer 2:  q(1) r(2) t(2,1)
```

It is interesting to see that the same result can be achieved in this case with unstratified negation [9]. We can move one of the atoms that occurs in the head of the rule to the body, extended with an extra negation as failure operator:

```
5. p(X,Y) :- q(X), r(Y), not s(Y), not t(Y,X).
   t(Y,X) :- q(X), r(Y), not s(Y), not p(X,Y).
   q(1). r(2).
```

This ASP program results in the same two answer sets as the program in (4). However, this method is considered less elegant than the one that uses true disjunction. Moreover, there exist certain reasoning problems that can only be expressed with true disjunction but not with unstratified negation. That means disjunctive rule heads are not syntactic sugar for unstratified negation because disjunctive logic programs under ASP semantics have higher complexity and expressive power then normal logic programs [5].

2.4 Choice Rules

A choice rule makes it possible to specify that any number of head literals in a rule may be included in an answer set. The following ASP program consists of a choice rule (marked by curly brackets in the head) and four facts:

```
6. { p(X,Y) } :- q(X), r(Y).
   q(1). q(2). r(1). r(2).
```

This choice rule says that if instances of atoms in the body of the rule are included in the answer set, then choose arbitrarily which of the corresponding instances of atoms in the head to include. Or in other words: the body of the rule gives the head atoms a reason to be in an answer set but does not enforce them to be there. Choice rules are often used to generate possible solutions. The ASP program (6) results in the following 16 different answer sets:

```
Answer 1:  q(1) q(2) r(1) r(2)
Answer 2:  q(1) q(2) r(1) r(2) p(1,1)
Answer 3:  q(1) q(2) r(1) r(2) p(1,2)
Answer 4:  q(1) q(2) r(1) r(2) p(1,2) p(1,1)
```

```
Answer 5:  q(1) q(2) r(1) r(2) p(2,1)
Answer 6:  q(1) q(2) r(1) r(2) p(2,1) p(1,1)
Answer 7:  q(1) q(2) r(1) r(2) p(2,1) p(1,2)
Answer 8:  q(1) q(2) r(1) r(2) p(2,1) p(1,2) p(1,1)
Answer 9:  q(1) q(2) r(1) r(2) p(2,2)
Answer 10: q(1) q(2) r(1) r(2) p(2,2) p(1,1)
Answer 11: q(1) q(2) r(1) r(2) p(2,2) p(1,2)
Answer 12: q(1) q(2) r(1) r(2) p(2,2) p(1,2) p(1,1)
Answer 13: q(1) q(2) r(1) r(2) p(2,2) p(2,1)
Answer 14: q(1) q(2) r(1) r(2) p(2,2) p(2,1) p(1,1)
Answer 15: q(1) q(2) r(1) r(2) p(2,2) p(2,1) p(1,2)
Answer 16: q(1) q(2) r(1) r(2) p(2,2) p(2,1) p(1,2) p(1,1)
```

We can add lower and upper **cardinality constraints** to a choice rule and specify, for example, that exactly one head atom should occur in the answer set for each solution found. If we replace the choice rule in (6) by the one in (7):

```
7. 1 { p(X,Y) } 1 :- q(X), r(Y).
```

we get exactly one answer set as solution:

```
Answer 1:  q(1) q(2) r(1) r(2) p(2,2) p(2,1) p(1,2) p(1,1)
```

We can use a choice rule together with a **conditional literal**, for example of the form `p(X,Y) : r(Y)` in the head of the rule. This allows us to specify – as (8) illustrates – that for all `X`'s such that `q(X)` holds, each answer set must contain exactly one atom of the form `p(X,Y)` given that `r(Y)` holds:

```
8. 1 { p(X,Y) : r(Y) } 1 :- q(X).
   q(1). q(2). r(1). r(2).
   #hide. #show p/2.
```

This results in the following four answer sets for the program in (8):

```
Answer 1:  p(2,2) p(1,2)
Answer 2:  p(2,2) p(1,1)
Answer 3:  p(2,1) p(1,2)
Answer 4:  p(2,1) p(1,1)
```

We can add another rule to the program in (8) and specify that for all `Y`'s such that `r(Y)` holds, each answer set must contain exactly one atom of the form `p(X,Y)` given that `q(X)` holds:

```
9. 1 { p(X,Y) : q(X) } 1 :- r(Y).
```

This further reduces the number of answer sets to two:

```
Answer: 1  p(2,2) p(1,1)
Answer: 2  p(2,1) p(1,2)
```

That means we have now only atoms for the relation `p(X,Y)` in the answer sets where not only the values for the `X`'s are different but also those for the `Y`'s. The first answer set represents the "reflexive" case and the second one the "symmetric" case for our program.

2.5 Constraints

Given the program in (8) plus the additional choice rule in (9), we can further restrict the number of answer sets using constraints. Adding, for example, the following constraint to the program:

10. `:- p(X,Y), X == Y.`

weeds out the answer set that contains reflexive relations, and we end up with the following answer set:

```
Answer 1:  p(2,1) p(1,2)
```

Adding the subsequent constraint (11) instead of (10) to the above program:

11. `:- p(X,Y), X != Y.`

makes sure that we keep only the answer set with the reflexive relations and eliminate the one with the symmetric relations:

```
Answer 1:  p(2,2) p(1,1)
```

Finally, note that we can get rid of the comparison predicate (`==`) in (10) and simply write: `:- p(X,X).` instead.

2.6 Negation

ASP distinguishes two kinds of negation: negation as failure (`not`) and strong negation (`-`). We can use the literal `not` *P* to express that *P* is not known to be true, and `-`*P* to express that *P* is false. We can combine these two forms of negations; for example, to express the closed world assumption [23] for a predicate:

12.
```
-p(X,Y) :- q(X), r(Y), not p(X,Y).
p(1,1). p(2,2). q(1). q(2). r(1). r(2).
```

The resulting answer set of this program includes the positive atoms as well as the complementary atoms for `p/2`:

```
Answer 1:  q(1) q(2) r(1) r(2) p(1,1) p(2,2) -p(2,1) -p(1,2)
```

It is important to note that strong negation is only a modelling convenience. Constraints can be used instead of strong negation to eliminate possible answer sets that contain complementary atoms.

3 The ASP Methodology

As the examples in the previous section illustrate, writing an ASP program often consists of combining choice rules with constraints. Choice rules generate multiple answer sets and constraints eliminate some of these answer sets. This suggests a predominant methodology for writing ASP programs; according to this methodology, simple ASP programs consist of three parts: a generate part, a test part, and a display part [19]. The writing of these ASP programs corresponds basically to the following three tasks:

a) Specifying rules that generate answer sets which represent potential solutions;
b) Specifying constraints that eliminate those answer sets which do not correspond to actual solutions;
c) Specifying which atoms of the answer sets should be included in the output.

More complex ASP programs consist of an additional definition part. In this definition part auxiliary rules are specified that are then used in the part (a) or (b). For example, the following program specifies a Hamiltonian cycle and distinguishes these four parts:

12.
```
% Generate
  1 { in(X,Y) : edge(X,Y) } 1 :- vertex(X).
  1 { in(X,Y) : edge(X,Y) } 1 :- vertex(Y).

% Define
  reachable(Y) :- in(1,Y).
  reachable(Y) :- in(X,Y), reachable(X).

% Test
  :- vertex(Y), not reachable(Y).

% Display
  #hide.
  #show in/2.
```

It is important to note that neither the order of rules nor the order of the literals within a rule does matter in an ASP program. More advanced ASP programs use sometimes an optimisation part to express optimisation statements that can be used to compute the quantitative cost of a solution. In the reminder of this paper we will only focus on the first four parts of this ASP methodology.

4 The Marathon Puzzle

The following text shows the English translation of a puzzle that was first published in a French book on linear programming [15]. The puzzle has been used on various occasions as a benchmark problem for ASP and constraint programming.

13. Dominique, Ignace, Naren, Olivier, Philippe, and Pascal have arrived as the first six at the Paris marathon.

    Reconstruct their arrival order from the following information:

    a) Olivier has not arrived last.
    b) Dominique, Pascal and Ignace have arrived before Naren and Olivier.
    c) Dominique who was third last year has improved this year.
    d) Philippe is among the first four.
    e) Ignace has arrived neither in second nor third position.
    f) Pascal has beaten Naren by three positions.

g) Neither Ignace nor Dominique are on the fourth position.

In order to solve this puzzle automatically, a specialist in ASP translates the puzzle by hand into a formal target notation and uses an ASP solver to find the solutions to the problem. Since the text is written in full natural language, the manual translation requires a number of normalisation steps and familiarity with the formal language of ASP. In the following, we first have a closer look at the puzzle from a linguistic perspective and then reconstruct the puzzle in a CNL, so that the puzzle is still easy to read and understand by a non-specialist but can be translated unambiguously into an ASP program.

4.1 Analysis of the Marathon Puzzle

The puzzle names six marathon runners that occupy the first six positions at a marathon, but we do not know their exact arrival order. The task is to establish this order with the help of seven constraints (*a*-*g*). What is striking from a linguistic point of view is that these constraints show two different grammatical aspects; some constraints (*a, b, c, e, f*) use a verb form in the present perfect to refer to events that occurred in the past, and some constraints (*d, g*) use a verb form in the present tense and refer to an actual position or to a possible range of positions. For a human reader some of the inferences that are required to solve this puzzle are obvious, since humans can apply their linguistic knowledge with ease and have a lot of commonsense knowledge about the world. For example, a human reader can easily infer that if Olivier has not arrived last, then Olivier has not arrived at the sixth position, or that if Dominique who was third last year and has improved this year, must have arrived at the first or second position this year. Being able to make these inferences is important for finding the solutions to the puzzle, but these inferences cannot be easily made by a machine without additional background knowledge or require a complete reconstruction of the puzzle. On the other hand, it is difficult for a human reader to do the bookkeeping mentally for these inferences and monitor the intermediate solutions; this is exactly what machines are good at. As we will see later in more detail, there exist a large number of possible answer sets for this puzzle before the constraints are applied, and each constraint will significantly reduce the number of possible solutions.

4.2 Reconstruction of the Marathon Puzzle in CNL

To enable the required inferences, we reconstruct the puzzle in the CNL PENG Light [27] and follow the same "generate-and-test" methodology as for ASP. Note that our approach is different to Baral & Dzifcak [2] who use preprocessed sample puzzles and their ASP representation to learn how to translate and solve new puzzles. Here is a reconstruction of the marathon puzzle in PENG Light:

1^c. Dominique, Ignace, Naren, Olivier, Philippe, and Pascal are runners.

2^c. There exist exactly six positions.

3^c. Every runner is allocated to exactly one position.

4^c. Reject that a runner R1 is allocated to a position and that another runner R2 is allocated to the same position and that R1 is not equal to R2.

5^c. Reject that Olivier is allocated to the sixth position.

6^c. If a runner R1 is allocated to a position P1 and another runner R2 is allocated to a position P2 and P1 is smaller than P2 then R1 is before R2.

7^c. Reject that Naren is before Dominique, Pascal, and Ignace.

8^c. Reject that Olivier is before Dominique, Pascal, and Ignace.

9^c. Reject that Dominique is allocated to a position that is greater than or equal to 3.

10^c. Reject that Philippe is allocated to a position that is greater than 4.

11^c. Reject that Ignace is allocated to the second position.

12^c. Reject that Ignace is allocated to the third position.

13^c. Reject that Pascal is allocated to a position P1 and that Naren is allocated to a position P2 and that P1 is not equal to P2 minus 3.

14^c. Reject that Ignace is allocated to the fourth position.

15^c. Reject that Dominique is allocated to the fourth position.

4.3 Discussion of the Reconstruction

In (1^c) we use a coordinated noun phrase in order to assign the six names in the puzzle to different runners, and in (2^c) we specify the existence of exactly six positions. These sentences belong to the definition part of the specification.

What is immediately noteworthy in the reconstruction of this puzzle is the introduction of the new relational expression *is allocated to* that occurs in most sentences. This expression allows us to abstract over other expressions used in the original version of the puzzle and to associate runners with positions in a systematic way. With the help of this expression, we can specify, for example, that every runner is allocated to exactly one position (3^c). This sentence belongs to the generation part of the specification. As we will see in Section 6, this sentence is translated into a choice rule that generates potential solutions.

The other important new element in the reconstructed puzzle is the command *reject that* that triggers constraints (4^c, 5^c, 7^c-15^c). Using (4^c), we exclude that two different runners end up at the same position. In (5^c) we specify the first explicit constraint that occurs in the puzzle and use the ordinal number *sixth* in the noun phrase *the sixth position* instead of the adjective *last*. We could use the noun phrase *the last position* instead, but then we would need an additional definitional rule that links the last position to the sixth position.

In (6^c) we define the meaning of *before* with the help of a conditional sentence that uses existing information about runners and their positions in the

antecedent, and a relation (*smaller than*) that is based on a comparison predicate. It is important to note that *R1* in (6^c) is a variable that stands for a name and that *P1* is a variable that stands for a number, since we compare numbers in the sentence using the *smaller than*-relation.

With the help of the two commands (7^c) and (8^c), we specify the second constraint of the puzzle that uses the *before*-relation defined in (6^c). In (9^c) and (10^c), we specify the third and fourth constraint of the puzzle; both commands include a comparison predicate. In (11^c) and (12^c), we specify the fifth constraint of the puzzle; these sentences use a noun phrase with an ordinal number to refer to an absolute position. In (13^c) we use a command that subordinates three simple sentences; the first two sentences speak about the two positions *P1* and *P2* and the third one computes a new value using the value of the second position, and then makes sure that this value is not equal to the value of the first position. Finally, the two commands in (14^c) and (15^c) are structurally similar to (11^c) and (12^c) and are used to further reduce the number of answer sets. However, a closer look at the puzzle reveals that (15^c) is redundant since the information expressed in this sentence is subsumed by sentence (9^c).

5 Processing the Marathon Puzzle

As already mentioned, the reconstructed puzzle follows the grammar rules of the CNL PENG Light [27]. PENG Light is a general-purpose CNL that has been designed for writing specification texts that can be translated into first-order logic. The writing process of a specification in PENG Light is supported by an authoring tool that constrains the input via look-ahead information while the text is written [24]. The language processor of PENG Light is implemented in Prolog and consists of a restricted grammar and a limited lexicon. The grammar is processed by a chart parser that resolves anaphoric references and builds up a discourse representation structure (DRS) [17] for the input text. The DRS uses a flat notation for atoms together with a small number of predefined predicates. Figure 1 illustrates this notation for the first four sentences of the reconstructed puzzle and shows that the command *reject that* has been translated into an operator (`NOTc`). This operator will trigger a constraint as we will see later when we translate the DRS into an ASP program. The only module that we had to build from scratch for our case study is the DRS-to-ASP translator.

For our application, we decided to use a simpler notation for the ASP program since we wanted to generate models that contain only information that is absolutely necessary. The DRS-to-ASP translator takes care of this transformation and produces the required output in ASP notation as we will see in the next section. The DRS-to-ASP translator is a Prolog program and looks similar to other programs that translate a DRS into another formal notation [4].

6 The Marathon Puzzle as an ASP Program

In this section we discuss the resulting ASP program that is generated by the CNL processor; but we focus in this discussion only on the translation of those

```
[A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T]
named(A,dominique)
object(B,runner)
predicate(C,isa,A,B)
named(D,ignace)
object(E,runner)
predicate(F,isa,D,E)
named(G,naren)
object(H,runner)
predicate(I,isa,G,H)
named(J,olivier)
object(K,runner)
predicate(L,isa,J,K)
named(M,pascal)
object(N,runner)
predicate(O,isa,M,N)
named(P,philippe)
object(Q,runner)
predicate(R,isa,U,Q)
cardinal(S,eq,6)
object(S,position)
predicate(T,exist,S)
```

```
[V]
object(V,runner)
==>
   [W,X,Y]
   cardinal(W,eq,1)
   object(W,position)
   predicate(X,be,V)
   property(X,allocated_to,W)
NOTc:
   [Z,A1,B1,C1,D1,E1,F1,G1]
   object(Z,runner)
   variable(Z,r1)
   object(A1,position)
   predicate(B1,be,Z)
   property(B1,allocated_to,A1)
   object(D1,runner)
   variable(D1,r2)
   predicate(E1,be,D1)
   property(E1,allocated_to,A1)
   predicate(G1,be,Z)
   operator(G1,\=,D1)
```

Fig. 1. DRS for the first four CNL Sentences (1^c-4^c) of the Marathon Puzzle

CNL sentences that introduce a new construction in ASP. The sentences are grouped into: concept assertions, existential sentences, universal sentences, commands, and conditional sentences. Those sentences that result in similar translations are only mentioned but not displayed. The entire ASP program can be found in the appendix to this paper.

Concept Assertions. A complex concept assertion such as:

1^c. Dominique, Ignace, Naren, Olivier, Philippe, and Pascal are runners.

is translated into a set of ASP facts of the following form:

1^a. `runner(dominique). runner(ignace). ...`

Existential Sentences. An existential sentence that uses a cardinal noun phrase in the nominal subject position such as:

2^c. There are exactly 6 positions.

is translated into an ASP fact that uses the range notation. The range notation is a shortcut that can be used to define numerical domains in a compact way:

2^a. `position(1..6).`

This representation is equivalent to six individual facts.

Universal Sentences. A universally quantified sentence with a cardinality constraint in object position such as:

3^c. Every runner is allocated to exactly one position.

is translated into a choice rule of the following form:

3^a. `1 { allocated_to(X,Y) : position(Y) } 1 :- runner(X).`

This rule will generate potential solutions.

Commands. A command is introduced by the predefined keyphrase *reject that* followed by a simple or complex sentence. This keyphrase is a syntactic element used to express a denial and triggers a constraint of the form: $false \leftarrow Q$.

A command with an ordinal number in the object noun phrase of the subordinate sentence such as:

5^c. Reject that Olivier is allocated to the sixth position.

is translated into the following constraint:

5^a. `:- allocated_to(olivier,6).`

Note that the meaning of the noun phrase *the sixth position* is equivalent to the meaning of the noun phrase *a position that is equal to 6* in PENG Light. Note also that the commands in (11^c, 12^c, 14^c, and 15^c) all result in similar constraints.

A command that uses an enumeration of names in the object position of the subordinate sentence such as:

7^c. Reject that Naren is before Dominique, Pascal, and Ignace.

is translated into three constraints of the following form:

7^a.
```
:- before(naren,dominique).
:- before(naren,pascal).
:- before(naren,ignace).
```

Note that the elements in the enumeration are automatically distributed over three constraints; of course, we could write (7^c) as three individual CNL commands. The translation of (8^c) results in a similar set of constraints as (7^c).

A command that subordinates a sentence that contains an object modifying relative clause such as:

9^c. Reject that Dominique is allocated to a position that is greater than or equal to 3.

is translated into a constraint with a built-in comparison predicate (`>=`):

9^a. `:- position(Y), Y >= 3, allocated_to(dominique,Y).`

A command that introduces a sequence of coordinated sentences such as in 4^c or in:

13^c. Reject that Pascal is allocated to a position P1 and that Naren is allocated to a position P2 and that P1 is not equal to P2 minus 3.

is translated into a single constraint. In our example this constraint contains an equality predicate (`==`) and an arithmetic function (`-`):

```
13a. :- position(Y1), allocated_to(pascal,Y1), position(Y2),
        allocated_to(naren,Y2), Y1 == Y2 - 3.
```

Conditional Sentences. A conditional sentence that defines the meaning of an expression such as:

6^c. If a runner R1 is allocated to a position P1 and a runner R2 is allocated to a position P2 and P1 is smaller than P2 then R1 is before R2.

is translated into a basic ASP rule with a built-in comparison predicate (`<`) in the body:

```
6a. before(X1,X2) :-
       runner(X1), position(Y1), allocated_to(X1,Y1),
       runner(X2), position(Y2), allocated_to(X2,Y2), Y1 < Y2.
```

The defined relation (`before/2`) is used in the constraint (7^a) above.

7 Evaluation

Once the specification has been translated into an ASP program, the program is first grounded by an ASP grounder and the resulting propositional program is then processed by an ASP solver. For our purpose, we use *clingo* [11–13], a state-of-the-art ASP tool, that combines the grounder and the solver in a single program. In our case, *clingo* generates one answer set for the puzzle as solution. If we add the corresponding display declaration (`#hide. #show allocated_to/2`) to the ASP program, then the following six atoms are displayed that inform about the allocation of names to positions:

```
Answer 1:  allocated_to(ignace,1) allocated_to(dominique,2)
           allocated_to(pascal,3) allocated_to(philippe,4)
           allocated_to(olivier,5) allocated_to(naren,6)
```

The following table (Table 1) shows what contribution each sentence of the specification makes to the solution of the puzzle. It is clearly visible how the choice rule derived from the sentence (3^c) generates the initial set of possible solutions (46656) and how the constraint derived from (4^c) dramatically reduces the number of these solutions (720). The subsequent constraints derived from (5^c, 7^c-14^c) further reduce the number of answer sets. As already explained,

Table 1. Number of Answer Sets after Processing a Sequence of Sentences

Sentence No.	Answer Sets
1^c	1
2^c	1
3^c	46656
4^c	720
5^c	600
6^c	600
7^c	150
8^c	42
9^c	24
10^c	12
11^c	10
12^c	6
13^c	3
14^c	1
15^c	1

sentence (6^c) is a definition and does not reduce the number of answer sets, and sentence (15^c) is redundant and does not contribute anything to the solution.

What is interesting in our context is that we can query an answer set using questions stated in PENG Light. Questions are translated into ASP rules that are then used to extract those atoms from the resulting answer set that we are interrogating. Below are a number of questions in PENG Light that one might want to answer over the marathon puzzle:

16^c. Which runner is allocated to what position?

17^c. Who is allocated to the first position?

18^c. Is Dominique allocated to the second position?

19^c. Who is before Naren?

These questions are translated into basic ASP rules and added to the original ASP program. For example, the translation of (16^c) results in (16^a) and the translation of (18^c) in (18^a):

16^a. `answer(X,Y) :- runner(X), position(Y), allocated_to(X,Y).`

18^a. `answer(yes) :- allocated_to(dominique,2).`

These rules use a special atom on the left hand side of the rule that is added to the answer set if the body of the rule is true. Answers can then be displayed by using a default display declaration for this type of atoms.

8 Improvements and Optimisations

While the presented implementation produces correct results, it can be improved and optimised in a number of ways. We focus here in particular on the

following points: the expressiveness of the CNL PENG Light, the representation of formulas in the ASP program, and on the question answering process.

8.1 Expressiveness

We have introduced all commands with the help of the same keyphrase (*reject that*) in the reconstructed puzzle. While this results in a uniform representation of constraints, one could go a step further and add the keyphrase *ensure that* to the CNL. This would allow us to express the meaning of 9^c alternatively as $9^{c'}$:

9^c. Reject that Dominique is allocated to a position that is greater than or equal to 3.

$9^{c'}$. Ensure that Dominique is allocated to a position that is smaller than 3.

The translation of $9^{c'}$ should finally result in a constraint that has the same effect as the one for 9^c.

8.2 Formal Representation

There are a number of ways how the ASP formulas in Section 6 can be optimised. It is important to note that even in a declarative programming environment the problem encoding matters. For example, it is a good idea to use as few variables as possible in an ASP program and to avoid built-in comparison predicates whenever possible. In particular, expressions such as `X < Y` and `X != Y` are costly because they enumerate cross products. Such cross products can be avoided by using negation as failure or linear chains. For example, one could further transform (13^a) into ($13^{a'}$):

```
13a. :- position(Y1), allocated_to(pascal,Y1), position(Y2),
        allocated_to(naren,Y2), Y1 == Y2 - 3.
13a'. :- allocated_to(naren,Y2), not allocated_to(pascal,Y2-3).
```

Note also that quadratic constructions such as in (4^a) do not scale very well in *clingo*; one could further transform this formula into ($4^{a'}$):

```
4a. :- runner(C), position(D), allocated_to(C,D), runner(E),
       allocated_to(E,D), C!=E.
4a'. :- position(D), 2 { allocated_to(_,D) }.
```

It is interesting to see that ($4^{a'}$) could be derived in a straightforward way from the following CNL sentence that uses a plural noun phrase:

$4^{c'}$. Reject that two runners are allocated to the same position.

Note that this sentence assumes a distributive reading of the plural noun phrase. This reading can be clarified with the help of a paraphrase in CNL.

8.3 Question Answering

In ASP, the answer to a question might depend on the found answer set. Let's assume that there are two valid solutions for the marathon puzzle: one where Dominique is at the first position and one where Dominique is at the second position. In the first case, the answer to question (18^c) is *no* and in the second case *yes*. The ASP tool *clingo* supports brave and cautious reasoning [12]. Under brave reasoning the answer needs to be found only in at least one answer set, but under cautious reasoning the answer needs to be found in every answer set. One could add the keyphrases *in a solution* or *in every solution* to the question and use this information to trigger the relevant reasoning mode.

9 Conclusion

In this case study, we showed that combinatorial puzzles can be reconstructed in a CNL and automatically translated into an ASP program. We noticed that this process is most successful if we use the same methodology that ASP employs. ASP is a form of declarative programming and is well-suited for processing programs that represent complex search problems. We observed that the centerpiece of the ASP program consists of a generate and a test part. The generate part uses choice rules that enumerate potential solutions and the test part uses constraints in order to eliminate unsuitable solutions. Definitional rules are often used to specify auxiliary predicates that support the generate and test part. Instead of writing the program in ASP notation, we use a CNL as a high-level specification language that makes it easier for a non-specialist to construct, verify and validate the ASP program. We showed that choice rules can be expressed naturally in CNL via universally quantified sentences with cardinality constraints. Constraints can be expressed as commands, and auxiliary predicates can be defined with the help of conditional sentences. An interesting aspect of this model-based approach is that questions about a specification can also be stated in CNL language. These questions are translated into ASP rules with distinct answer literals, and finally added to the ASP program. The entire ASP program is then processed by an ASP tool.

Acknowledgments. I would like to thank to three anonymous reviewers of CNL 2012 for their extensive and constructive comments on an earlier version of this paper and to Martin Gebser from the University of Potsdam for valuable hints on how to optimise answer set programs.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Baral, C., Dzifcak, J.: Solving Puzzles Described in English by Automated Translation to Answer Set Programming and Learning How to Do that Translation. In: Proceedings of KR 2012, pp. 573–577 (2012)
3. Bernardi, R., Calvanese, D., Thorne, C.: Lite Natural Language. In: Proceedings IWCS-7 (2007)

4. Blackburn, P., Bos, J.: Representation and Inference for Natural Language. A First Course in Computational Semantics. CSLI Publications (2005)
5. Brewka, G., Eiter, T., Truszczyński, M.: Answer Set Programming at a Glance. Communications of the ACM 54(12) (December 2011)
6. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press, New York (1978)
7. Clark, P., Harrison, P., Jenkins, T., Thompson, J., Wojcik, R.: Acquiring and Using World Knowledge using a Restricted Subset of English. In: The 18th International FLAIRS Conference (FLAIRS 2005), pp. 506–511 (2005)
8. Clocksin, W.F., Mellish, C.S.: Programming in Prolog: Using the ISO Standard, 5th edn. Springer, Heidelberg (2003)
9. Eiter, T., Ianni, G., Krennwallner, T.: Answer Set Programming: A Primer. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.-C., Schmidt, R.A. (eds.) Reasoning Web 2008. LNCS, vol. 5689, pp. 40–110. Springer, Heidelberg (2009)
10. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
11. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: Proceedings of IJCAI, pp. 386–392 (2007)
12. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T.: Potassco: The Potsdam Answer Set Solving Collection. AI Communications 24(2), 107–124 (2011)
13. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in *gringo* Series 3. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011. LNCS, vol. 6645, pp. 345–351. Springer, Heidelberg (2011)
14. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Proceedings of the 5th ICLP, pp. 1070–1080 (1988)
15. Gueret, C., Prins, C., Sevaux, M.: Programmation linéaire, 65 problèmes d'optimisation modélisés et résolus avec Visual Xpress, Eyrolles (October 2000)
16. Gunning, D., Chaudhri, V.K., Clark, P., Barker, K., Chaw, S.-Y., Greaves, M., Grosof, B., Leung, A., McDonald, D., Mishra, S., Pacheco, J., Porter, B., Spaulding, A., Tecuci, D., Tien, J.: Project Halo Update–Progress Toward Digital Aristotle. AI Magazine 31(3), 33–58 (2010)
17. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer, Dordrecht (1993)
18. Kuhn, T.: Controlled English for Knowledge Representation. Doctoral Thesis. Faculty of Economics, Business Administration and Information Technology of the University of Zurich (2010)
19. Lifschitz, V.: Answer Set Programming and Plan Generation. Artificial Intelligence 138, 39–54 (2002)
20. Lifschitz, V.: What Is Answer Set Programming? In: Proceedings of AAAI 2008, vol. 3, pp. 1594–1597 (2008)
21. Lloyd, J.: Foundations of Logic Programming. Springer (1987)
22. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Apt, K.R., Marek, V., Truszczyński, M., Warren, D.S. (eds.) The Logic Programming Paradigm: a 25-Year Perspective, pp. 169–181. Springer (1999)
23. Reiter, R.: On closed world data bases. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 119–140 (1978)
24. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE – A Look-ahead Editor for a Controlled Language. In: Proceedings of EAMT-CLAW 2003, pp. 141–150 (2003)

25. Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A Comparison of three Controlled Natural Languages for OWL 1.1. In: 4th International Workshop on OWL Experiences and Directions, Washington, USA, April 1-2 (2008)
26. Schwitter, R.: Controlled Natural Language for Knowledge Representation. In: Proceedings of COLING 2010, pp. 1113–1121 (2010)
27. White, C., Schwitter, R.: An Update on PENG Light. In: Pizzato, L., Schwitter, R. (eds.) Proceedings of ALTA 2009, pp. 80–88 (2009)

Appendix: The Marathon Puzzle in ASP Notation

```
runner(dominique).
runner(ignace).
runner(naren).
runner(olivier).
runner(pascal).
runner(philippe).
position(1..6).
1 { allocated_to(A,B) : position(B) } 1 :- runner(A).
:- runner(C), position(D), allocated_to(C,D), runner(E),
   allocated_to(E,D), C != E.
:- allocated_to(olivier,6).
before(F,G) :-
   runner(F), position(H), allocated_to(F,H),
   runner(G), position(I), allocated_to(G,I), H < I.
:- before(naren,dominique).
:- before(naren,pascal).
:- before(naren,ignace).
:- before(olivier,dominique).
:- before(olivier,pascal).
:- before(olivier,ignace).
:- position(J), J >= 3, allocated_to(dominique,J).
:- position(K), K > 4, allocated_to(philippe,K).
:- allocated_to(ignace,2).
:- allocated_to(ignace,3).
:- position(L), allocated_to(pascal,L), position(M),
   allocated_to(naren,M), L != M - 3.
:- allocated_to(ignace,4).
:- allocated_to(dominique,4).
answer(N,O) :-
   runner(N), position(O), allocated_to(N,O).
#hide. #show answer/2.
```

OWL Simplified English: A Finite-State Language for Ontology Editing

Richard Power

Department of Computing
Open University
Milton Keynes, UK
r.power@open.ac.uk

Abstract. We describe a controlled fragment of English for editing ontologies in OWL. Although this language substantially overlaps other CNLs that have been proposed for this purpose, it has a number of special features designed to simplify its learning and use. First, the language allows users to start typing in sentences with little or no preliminary effort in building a controlled vocabulary or lexicon. Second, it disallows sentences that *people* interpret as structurally ambiguous. Third, it employs a finite-state grammar, so facilitating fast and reliable implementation of an editing tool. These advantages are gained at the cost of severe restrictions in coverage, which mean that the majority of potential OWL axioms cannot be expressed. However, analysis of axiom patterns from several ontology repositories suggests that these constraints are almost invariably respected by ontology developers, so that in practice the loss of expressivity is rarely noticeable.

Keywords: semantic web, ontology editing, controlled natural language.

1 Introduction

A variety of controlled languages for the semantic web have been proposed, prominent examples being ACE (Attempto Controlled English) [7], SOS (Sydney OWL Syntax) [20], and Rabbit [6]. Most of these languages have been used both in systems that generate text from OWL code (*verbalisers*) and in systems that produce OWL code by interpreting text (*editors*). Examples of editing tools are AceWiki [8], ROO [4], RoundTrip Ontology Authoring [3], and FluentEditor [2]. These applications belong to a tradition in CNL research that predates the semantic web, and includes for example Computer Processable English [15] and PENG [18]; the general aim of this research is to design subsets of English that can be unambiguously interpreted in some system of formal logic, while allowing sufficient freedom for human authors to write fluent texts.

In comparing different CNLs for knowledge formation, it is useful to distinguish a number of design requirements and potential trade-offs among them. Schwitter [17] and Kuhn [9] have both distilled these requirements into four groups, which Kuhn calls clearness, naturalness, simplicity and expressivity.

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 44–60, 2012.

Clearness means that the syntax and semantics of the CNL should be well-defined, and that any well-formed sentence in the CNL should be mapped unambiguously to a formal semantic interpretation. *Naturalness* requires that sentences in the CNL are understandable by people, and includes issues like whether sentences are perceived as belonging to English or some other natural language, and whether people interpret them unambiguously as having the desired meaning. *Simplicity* requires that the CNL is easily described and processed: for people this depends on whether the language is easily learned and applied, for instance when checking that a sentence is correct; for programs it depends on whether a sentence typed in by the user can be efficiently parsed (so confirming that it belongs to the CNL) and interpreted. Finally, *expressivity* concerns how fully the CNL covers the desired problem domain—or in the case of a semantic web editor, its coverage of OWL. As will be obvious, these requirements may conflict, so that different trade-offs are possible between (for example) simplicity and expressivity.

We describe in this paper a CNL for editing ontologies which aims broadly to maximise *simplicity* and *naturalness* at the expense of expressivity; however, we also present empirical evidence that this loss of expressivity is more theoretical than practical, since the proposed language can cover almost all OWL patterns that are actually found in ontology corpora. Unlike ACE and some other languages mentioned above, this CNL is designed solely for convenience in verbalising and editing OWL ontologies, and has no other linguistic or logical pretensions whatever; for this reason we provisionally name it *OWL Simplified English*. In more detail, the design principles informing the CNL are as follows.

1. It should be possible to describe the language very briefly; as a rough guide, the basic rules should fit comfortably on a sheet of A4.
2. Any preliminary work on the lexicon should be minimised: ideally, a user should be able to type in sentences straight away, without having to list content words or entity names.
3. The grammar should disallow sentences that people perceive as structurally ambiguous.
4. The grammar should be finite-state, so that sentences can be parsed and interpreted efficiently by a finite-state transducer.
5. No effort should be made to guarantee that sentences are grammatical according to the conventions of normal English. Provided that the CNL makes it *possible* to write fluent English, adherence to conventional grammar can be left to the human author.

Empirically, OWL Simplified English is based on two findings derived from analysis of ontology corpora [13,14]. The first is that the names of individuals, classes and properties in an ontology have distinctive features which can be exploited in order to determine where they begin and end. The second is that in practice, complex OWL expressions are invariably right-branching, and so lend themselves to verbalisations that are structurally unambiguous and can be described by a finite-state grammar. These two ideas will be developed in detail in the following sections.

2 Recognising Entity Names

The following sentence verbalises a fairly complex OWL statement using strategies typical of the CNLs that have been proposed:

> London is a city that is capital of the United Kingdom and is divided into at least 30 boroughs.

To interpret this sentence (which is correctly formed in OWL Simplified English), the parser must recognise a number of *entity names*—that is, phrases denoting individuals and atomic classes or properties:[1] thus 'London' and 'the United Kingdom' name individuals, 'city' and 'boroughs' name classes, and 'is capital of' and 'is divided into' name object properties. The other words in the sentence provided a *scaffolding* signalling various axiom and class constructors; they comprise the copular 'is', the indefinite article 'a', the conjunction 'and', the relative pronoun 'that', and the quantifying phrase 'at least 30'. Abstracting from the entity names, we could represent the sentence pattern like this:

> [Individual] is a [class] that [has-property] [Individual] and [has-property] at least 30 [class].

In general, a CNL for verbalising OWL ontologies requires only a handful of function words to provide this scaffolding. As well as the words already illustrated, most CNLs use 'or' for union, 'not' for complement, 'exactly' for exact cardinality, 'at most' for maximum cardinality, and the determiners 'no' and 'every' in sentences expressing *DisjointClasses* and *SubClassOf*. By contrast, the potential vocabulary for entity names is vast, ranging from the commonplace to the highly technical, and will include many words that can belong to more than one part of speech (e.g., 'rank', which can be noun, adjective or verb). To achieve the goals of OWL Simplified English, we must find formation rules for entity names that allow the parser to determine where a name begins and ends, and whether it denotes an individual, a class or a property; moreover, the rules should allow users sufficient freedom to construct appropriate names, while not demanding a great deal of preliminary effort in specifying a controlled vocabulary. It is not at all obvious that this combination of desiderata can be met.

For evidence on how entity names are formed in practice, we can refer to studies on the structure of identifiers and labels in ontology corpora [10,13], which show that individuals, classes and properties are named by distinctive part-of-speech sequences. *Individual* identifiers are made up mostly of proper nouns, common nouns and numbers; where the opening word is not a proper noun, the definite article 'the' is often implicit. *Class* identifiers are composed mostly of common nouns and adjectives, although they may also contain numbers and proper nouns (e.g., '1912 Rolls Royce'). *Property* identifiers often open with a verb or auxiliary ('is', 'has') in the present tense; their other constituents are mostly common nouns, participles, and prepositions. In all three types of

[1] The qualification 'atomic' here distinguishes elementary classes/properties from ones that are constructed, for instance through a restriction or intersection functor.

identifier, the function words listed above as scaffolding are very rare—indeed, the only function words that are at all common are auxiliaries (especially 'has') and prepositions, both found mostly in property identifiers.

On the basis of these findings, suppose that we constrain entity names as follows:

1. Some listed function words including 'a', 'every', 'and', 'that', must not be used at all. These words can therefore serve as signals that an entity name has just ended, or is about to begin.
2. Individual names must begin either with a proper noun or the definite article 'the', and may not contain verbs or auxiliaries.
3. Property names must begin either with 'is' or 'has' (or their plurals), or with a verb in the present tense, and may not include proper nouns, numbers or strings (these would signal the onset of an individual name or a literal).
4. Property names opening with an auxiliary must contain at least one further word.
5. Class names may not contain verbs or auxiliaries (which would signal the onset of a property name).

Consider the application of these rules to the following sentences, in which listed function words are shown in italics:

Every capital city *is an* urban area.
Hyde Park *is* located in London.

Assume that we have no knowledge of the content words 'capital', 'city', etc., so that the first sentence might as well be 'Every xxxxxxx xxxx is an xxxxx xxxx'. We can tell immediately that 'capital' opens a class name, because it is preceded by 'every'. On reaching 'is' we know that the class name is over, but cannot yet tell whether 'is' serves as scaffolding or as the opening of a property name. The next word 'an' rules out a property name (which would require at least one further word after 'is'), and foreshadows another class name, which opens with 'urban' and continues up to the full stop.

The second sentence, which is effectively 'Xxxx Xxxx is xxxxxxx in Xxxxxx', provides little scaffolding, but we can still interpret it if we are allowed to assume that any unlisted word opening with a capital letter is a proper noun.[2] In this case 'Hyde' must open an individual name; this continues up to 'is', which as before has two possible interpretations, but this time the next word is unlisted, and so we may infer that 'is' opens a property name. After adding 'located' and 'in' to this property name we arrive at another proper noun ('London'), which in this context can only signal the opening of another individual name.[3]

[2] Note that this criterion would not work for languages such as German in which both common and proper nouns start with a capital letter. This raises the general issue of whether superficial methods, like those proposed here for English, could be found for other languages, so permitting for example an OWL Simplified German—a point that we have not yet investigated.

[3] Note that 'located', although a verb, cannot open a property name because it is not in the present tense.

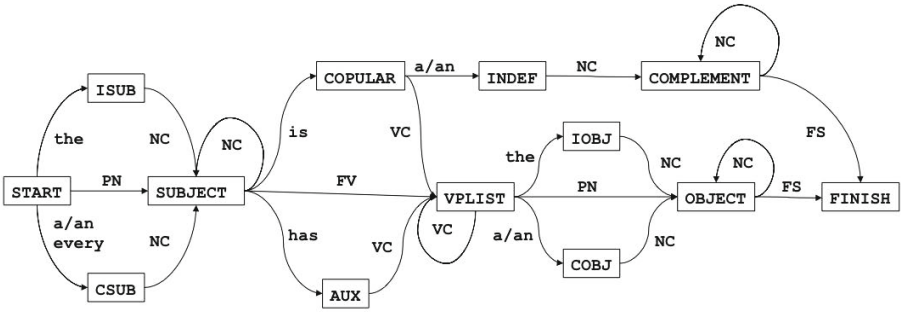


Fig. 1. Finite-state network for recognising basic clause patterns. PN = Proper Noun; NC = Noun-phrase Continuation; VC = Verb-phrase Continuation; FV = Finite Verb; FS = Full-Stop.

These examples are encouraging because they indicate how sentences might be interpreted *with minimal knowledge of content words*. Any words used as verbs in the present tense must be listed,[4] but otherwise the parser can rely on lists of common function words, supplemented by inferences based on typography through which content words can be classified as follows:

Category	**Description**
Verb	Non-auxiliary verb in the present tense, listed by the user
Number	sequence of digits, possibly including decimal point (10, 2.5)
String	sequence of characters inside double-quotes ("J. Doe")
Proper Noun	any other word beginning with a capital letter (London)
Noun/Other	any other word beginning with a lower-case letter

With this classification it is straightforward to define a finite-state transducer for basic clause patterns which can distinguish individual, class and property names without a lexicon of content words, and so construct an OWL expression from the input sentence. A simplified version of the grammar is shown in figure 1, where 'verb-phrase continuation' signifies only Noun/Other, while 'noun-phrase continuation' also includes Number, String, Proper Noun, and the definite article. The full grammar covers a wider range of sentences, and includes actions, defined on each arc, which build the OWL output. These are illustrated by the following table, which shows the state transitions in interpreting the sentence 'London is a capital city'.

[4] The reason verbs must be listed is that otherwise they could be construed as nouns continuing the preceding class name. Thus in the sentence 'Every building works manager reports to a regional director', both 'works' and 'reports' are potentially verbs opening a property name, and the program cannot correctly recognise 'building works manager' as a class name, and 'reports to' as a property name, unless 'reports' is a listed verb and 'works' is not.

State 1	Word	State 2	OWL expression
START	London	SUBJECT	`CA(?,I(London))`
SUBJECT	is	COPULAR	`CA(?,I(London))`
COPULAR	a	INDEF	`CA(C(?),I(London))`
INDEF	capital	COMPLEMENT	`CA(C(capital),I(London))`
COMPLEMENT	city	COMPLEMENT	`CA(C(capital_city),I(London))`
COMPLEMENT	.	FINISH	`CA(C(capital_city),I(London))`

Starting from the state START, the transducer reads the words of the sentence (including the final full-stop) one at a time, from left to right, traversing an arc every time a word is consumed. At each state, the current word will match the conditions on at most one arc, so that progress through the network requires no backtracking. If there are no outgoing arcs that match the current word, interpretation fails; if the current word is a full-stop and there is an arc leading to FINISH, interpretation succeeds.

OWL expressions are built progressively using actions defined on each arc. For instance, on consuming 'London' (classified as a proper name) at the initial state START, it can be inferred that the axiom will have the functor *ClassAssertion*, and that its second argument will be an individual with a name beginning 'London'.[5] This is not necessarily the complete individual name (perhaps the subject is 'London Bridge'), but on consuming 'is' we know that the name is complete, and await developments: perhaps 'is' opens a property name, and perhaps it merely denotes class membership. The next word 'a', which cannot occur in a property name, confirms the latter possibility, and prepares us to receive a class name. Two noun-phrase continuations follow, both of type Noun/Other; finally, the full-stop indicates that both the class name and the whole axiom are complete.

As will be obvious, the grammar in figure 1 will accept and interpret many sentences that are blatantly ungrammatical in English. To give just one example, the rule for constructing class names does not guarantee that anything remotely resembling a noun phrase will result: a nonsensical sequence of noun-phrase continuations such as 'of of the of' will be accepted, allowing such sentences as 'An of of the of is a the the'. Of course we could tighten the grammar to ensure, for example, that class names could not begin with a preposition or the definite article, but we see no advantage in doing so except for helping the user to correct accidental slips. This has to be set against several advantages of our permissive policy: (1) the transition network is simpler (and faster); (2) the rules of the language can be described more briefly; and (3) we allow slang or technical phrases that violate normal English (e.g., 'an in your face person', where the noun group opens with a preposition).

3 Basic Clause Patterns

We list in table 1 the linguistic patterns for expressing common axiom and class constructors in OWL Simplified English. For the most part these conform to

[5] We abbreviate `ClassAssertion` to `CA` and `Individual(#London))` to `I(London)` (etc.) so that the table fits on the page.

Table 1. OWL functors covered by the CNL. C = Class; I = Individual; OP = Object property; DP = Data property; L = Literal; N = Cardinality.

OWL Functor	CNL Pattern	Example
`SubClassOf(C1 C2)`	A [C1] is a [C2]	A city is a place
`ClassAssertion(C I)`	[I] is a [C]	London is a city
`ObjectPropertyAssertion(OP I1 I2)`	[I1] [OP] [I2]	London contains Hyde Park
`EquivalentClasses(C1 C2)`	A [C1] is any [C2]	A human is any person
`DisjointClasses(C1 C2)`	No [C1] is a [C2]	No village is a city
`ObjectSomeValuesFrom(OP C)`	[OP] a [C]	contains a park
`ObjectHasValue(OP I)`	[OP] [I]	contains Hyde Park
`DataPropertyAssertion(DP I L)`	[I] [DP] [L]	Hyde Park dates from 1536
`DataHasValue(DP L)`	[DP] [L]	dates from 1536
`ObjectAllValuesFrom(OP C)`	[OP] only [C]	contains only parks
`ObjectExactCardinality(N OP C)`	[OP] exactly [N] [C]	contains exactly 10 parks
`ObjectMinCardinality(N OP C)`	[OP] at least [N] [C]	contains at least 10 parks
`ObjectMaxCardinality(N OP C)`	[OP] at most [N] [C]	contains at most 10 parks

other CNLs [19], the main exception being the pattern for *EquivalentClasses*, where we suggest using 'any' in the predicate (e.g., 'A pet-owner is any person that owns a pet') as a means of articulating both directions of the equivalence.[6] Another exception is that we allow the indefinite article as well as the quantifier 'every' at the start of the pattern for *SubClassOf*; this derives from an an evaluation of the SWAT Tools verbaliser [21], in which ontology developers declared a preference for the indefinite article at least in some contexts.

To cover *DataPropertyAssertion* and *DataHasValue* the CNL needs names for data properties and literals. At present we assume that literals are named by single tokens belonging to the word categories Number and String (see last section), and that object and data properties are distinguished only by their context (i.e., whether they are followed by a literal). The only other complication comes from universal and numerical quantifiers in restrictions (e.g., 'only parks', 'exactly 10 parks'), which require a singular/plural distinction on class and property names—a potential problem for our approach in which the details of English syntax are disregarded. Our provisional solution is to pluralise by applying standard morphological rules (including common exceptions) to the *head* word of a class or property name, defined as the final word of a class name,[7] or the opening verb/auxiliary in a property name. It is left to the user to avoid names for which this simple rule yields odd results.

4 Coordination, Relative Clauses and Negation

We now turn to the class constructors for intersection and union, which are typically expressed through coordinated noun phrases or verb phrases, or through relative clauses introduced by 'that'. As we will see, the resulting sentences may

[6] We are planning an empirical study on which of the proposed formulations best expresses equivalence.

[7] Except for a name like 'student of art', in which case it will be the word preceding the preposition.

become complex and prone to structural ambiguity; to avoid this, OWL Simplified English strongly constrains the permitted sentence patterns.

The first constraint, a very sweeping one, is that constructed classes of any kind may be expressed *only in the predicate*:[8] we disallow sentences such as 'Every person that owns a pet is a pet-owner' where the subject is complex. In terms of the underlying logic, this constraint means that the left-hand side of an axiom constructed with '$\sqsubseteq$', '$\in$' or '$\equiv$' must be atomic: an axiom like $C_1 \sqcap \exists P.C_2 \sqsubseteq C_3$ is excluded.[9] We impose this constraint because it enormously simplifies the language at small practical cost, since axioms with a complex subject term occur very rarely in practice; in a corpus study of around half a million axioms, we found that 99.8% of them had atomic subject terms and only 0.2% had complex subject terms [14]. Perhaps this is not surprising, given that the purpose of description logic is to *describe* things—indeed, the graphical editor Protégé *assumes* that subject terms will be atomic and makes no provision for complex subjects at all [16].

4.1 Intersection

In agreement with all other CNLs that we know of, OWL Simplified English expresses *ObjectIntersectionOf* by the pattern '[X] and [Y]' when '[X]' and '[Y]' are both noun-phrases or both verb-phrases, and by the pattern '[X] that [Y]' when '[X]' is a noun-phrase and '[Y]' is a verb-phrase. This policy achieves the requirement (mentioned earlier) of simplicity, but runs the risk of violating *naturalness* since even fairly simple combinations of intersection and restriction may yield sentences that are structurally ambiguous:

OWL pattern	**English verbalisation**
$C \sqsubseteq \exists P.(C \sqcap C)$	Every dog lives in a house and a kennel
$C \sqsubseteq \exists P.(C \sqcap (\exists P.C \sqcap \exists P.C))$	Every dog lives in a kennel that is located in a garden and is painted a shade of pink

For the first of these sentences, the OWL pattern should mean 'Every dog lives in something that is both a house and a kennel'. For the second, ACE disambiguation rules would imply that it is the dogs, not the kennels, that are painted pink.

To eliminate these and other cases of structural ambiguity, OWL Simplified English allows only three strategies for constructing complex sentences: noun-phrase lists, verb-phrase lists, and verb-phrase chains. These are defined as follows:

[8] In most cases, the predicate of a sentence in OWL Simplified English corresponds to the second argument of the underlying OWL axiom. This holds for *SubClassOf* and *EquivalentClasses*, for example. Two exceptions are *ClassAssertion* and *ObjectPropertyAssertion*, for which the argument verbalised as the subject (in all proposed CNLs) is the second, and the other argument(s) are verbalised as the predicate.

[9] Note that a developer could overcome this limitation by introducing a new class $C_4 \equiv C_1 \sqcap \exists P.C_2$ and then asserting $C_4 \sqsubseteq C_3$.

1. A *noun-phrase list* has the form '[C] and [C] and [C] ...' where each '[C]' is a class name.
2. A *verb-phrase list* has the form '[P] a [C] and [P] a [C] and ...', or the equivalent using other quantifiers such as 'only' and 'exactly two' (or using individuals or literals instead of classes).
3. A *verb-phrase chain* has the form '[P] a [C] that [P] a [C] that ...', or the equivalent using other quantifiers or other property values (individuals or literals).[10]

Examples of the three strategies are as follows:

> (1) London is a city and a capital and a tourist attraction. (*Noun-phrase list*)
> (2) London is capital of the UK and has as population 15000000. (*Verb-phrase list*)
> (3) London is capital of a country that is governed by a man that lives in Downing Street. (*Verb-phrase chain*)

We believe that each of these constructions is free from structural ambiguity, and moreover that they can be *combined* in the same sentence provided that they come in the specified order: noun-phrase list precedes verb-phrase list, verb-phrase list precedes verb-phrase chain.

> London is a city that has as population 15000000 and is capital of a country that is governed by a man that lives in Downing Street.

Here and in the previous examples a parsing algorithm might find alternative analyses of a phrase like 'a city and a capital and a tourist attraction' depending on whether the bracketing was '[a city and a capital] and a tourist attraction' or 'a city and [a capital and a tourist attraction]', but these potential ambiguities are innocuous rather than 'nocuous' [24] owing to the associativity of intersection.[11]

[10] To be precise, the defining property of a verb-phrase chain is not the occurrence of the word 'that', but the embedding of (at least) one restriction within another retriction. Thus '[C] that [P] a [C] and [P] a [C]' is a one-term noun-phrase list followed by a verb-phrase list, despite the occurrence of 'that', and denotes the constructed class $C \sqcap \exists P.C \sqcap \exists P.C$, which does not contain an embedded restriction. Instead, '[C] that [P] a [C] that [P] a [C]' denotes a constructed class $C \sqcap \exists P.(C \sqcap \exists P.C)$ that contains an embedded restriction (verbalised as a relative clause within a relative clause), and thus ends in a chain.

[11] Since our aim is to avoid sentences that might be misinterpreted by human readers, we not interested here in whether a sentence is ambiguous relative to some specific grammar, but in whether people might assign it different meanings. Research in psycholinguistics has revealed two strategies called *minimal attachment* and *late closure* [1] which are relevant to some of the examples discussed here; however, for the most part we have relied on intuition. Ideally, we would like to test at least some doubtful patterns empirically, as has been done in the study of nocuous and innocuous ambiguity cited here.

4.2 Union

Using 'and' (or 'that') and 'or' in the same sentence almost always leads to ambiguity, as in these examples:[12]

John is a lawyer or an artist and a pet-owner.
John is a lawyer or an artist that owns a pet.

In one reading, John is definitely a pet-owner, as in 'John is (1) a lawyer or an artist, and (2) a pet-owner. In the other reading he might not be a pet-owner ('John is (1) a lawyer, or (2) an artist and a pet-owner'). As just shown, we can disambiguate these examples with careful punctuation, but this will become cumbersome or break down altogether for more complex sentences.

To avoid such ambiguities by simpler means, OWL Simplified English disallows usage of 'and'/'that' and 'or' in the same sentence. At present, just three patterns with 'or' are allowed: noun-phrase lists, verb-phrase lists, and restrictions over a noun-phrase list.[13] Here are examples of each:

$C \sqsubseteq C \sqcup C$	A married person is a husband or wife
$C \sqsubseteq \exists P.C \sqcup \exists P.C$	A student attends a school or attends a college
$C \sqsubseteq \exists P.(C \sqcup C)$	A student attends a school or college

4.3 Complement

For now we envisage just three cases in which negation will be allowed in a predicate: negating a simple class; negating a simple restriction,[14] and negating the second term of a simple intersection:

$C \sqsubseteq \neg C$	A whale is not a fish
$C \sqsubseteq \neg \exists P.C$	A child does not attend a university
$C \sqsubseteq C \sqcap \neg C$	A consonant is a letter that is not a vowel

As will probably be obvious, expressing the complement of an intersection or union almost always leads to structural ambiguity.[15]

5 Empirical Results on Coverage

We have outlined a CNL in which structural ambiguity (at least of the 'nocuous' kind) is avoided through severe constraints on the formation of complex predicates using 'and', 'or' and 'that'. In this section we summarise the theoretical

[12] Of course the underlying OWL axiom has a precise unambiguous argument structure; the ambiguity resides in the sentence verbalising this axiom, which could also be construed as the verbalisation of a different non-equivalent axiom.

[13] Chains are prohibited since 'or' cannot be used with 'that'.

[14] By a 'simple' restriction we mean one whose value is an atomic entity as opposed to a constructed class.

[15] Forming the complement of a class by prefixing 'non-' also leads to ambiguity for a multiword class name (e.g., 'non-blue whale').

and practical consequences of these constraints.[16] In a nutshell, we will argue for two claims:

1. The proposed CNL can express *only a tiny fraction* of the statements that could in theory be constructed using restriction, intersection and union.
2. In spite of this, the proposed CNL can express *almost all* the complex axioms that occur in practice.

To assess the first claim, we need to find some method of enumerating all non-equivalent classes that can be constructed using different combinations of restriction, intersection, and union, and then counting how many can be expressed linguistically under ther constraints defined above. Obviously the set of constructed classes is infinite, so the only practical approach is to count all patterns up to a given level of complexity.

To measure the complexity δ of a constructed class, we can count the number of class constructors: thus C (an atomic class) will have δ=0, $\exists P.C$ will have δ=1, and $C \sqcap C \sqcap C$ will have δ=2. The only complication here is that in OWL, *ObjectIntersectionOf* and *ObjectUnionOf* may have more than two arguments, so that $C \sqcap C \sqcap C$ would normally be encoded using a single functor with three arguments. However, since additional arguments imply additional complexity, we will assume that only two arguments are allowed, so that in such a case two functors would be needed. A class expression can then be represented as a binary tree in which the terminal nodes are atomic entities (classes, properties, etc.), and the non-terminal nodes are restriction, intersection or union functors; the complexity is then given by the number of non-terminal nodes. It is then straightforward to write a program that can generate all binary trees of a given complexity.

As an exercise, let us generate all trees of complexity δ=2, using only two functors, intersection ($\sqcap$) and existential restriction ($\exists$) — by far the most common combination in practice. Using P for any property and C for any atomic class, the full set of class expressions is shown in table 2. Of these patterns, we would argue that three can be disregarded. First, pattern 1 cannot be verbalised unless the inner restriction is recast into a form that can be expressed by a noun phrase;[17] to do this we have introduced 'something that', which is equivalent to expressing the more complex pattern $\exists P.(\top \sqcap \exists P.C)$ (which would be covered anyway in the list for complexity 3). Next, since intersection is commutative, patterns 3 and 4 are equivalent, and so are patterns 5 and 6; in such cases we need consider only the form in which the arguments are optimally ordered with simple preceding complex (so eliminating patterns 3 and 6). We are therefore left with three non-equivalent patterns for δ=2, of which two (patterns 4 and 5)

[16] These consequences will be explored more fully in a separate paper.

[17] We are assuming here that restrictions such as $\exists P.C$ are realised by verb phrases in which the object expresses the class C. This policy, which is followed in all the OWL CNLs that we are aware of, runs into difficulty when C is replaced by a restriction, yielding $\exists P.(\exists P.C)$, since for syntactic reasons a verb phrase cannot have another verb phrase as its object.

Table 2. All class patterns of complexity δ=2 that can be constructed using intersection and existential restriction. The proposed verbalisation patterns assume that atomic properties are expressed by verbs (V) and atomic classes by nouns (N). Patterns marked with an asterisk can be disregarded (see text).

	Logical pattern	**Verbalisation**
*1	$\exists P.(\exists P.C)$	Vs (something that) Vs an N
2	$\exists P.(C \sqcap C)$	Vs an N and an N
*3	$(\exists P.C) \sqcap C$	is something that Vs an N and an N
4	$C \sqcap (\exists P.C)$	is an N that Vs an N
5	$C \sqcap (C \sqcap C)$	is an N and an N and an N
*6	$(C \sqcap C) \sqcap C$	is an N and an N and an N

can be expressed according to the constraints of our CNL, and one (pattern 2) cannot. Thus if these three patterns were equally common in practice, one-third of the relevant OWL axioms could not be verbalised.

To ascertain the frequency of these patterns in practice, we have collected a corpus of around 550 ontologies from several repositories, containing some 500,000 axioms in all.[18] By far the majority of these axioms have simple predicates (complexities less than 2). Counting only axioms of complexity 2 using intersection and existential restriction, the corpus contains 897 predicates distributed as follows: 892 have the form $C \sqcap (\exists P.C)$, 5 have the form $\exists P.(C \sqcap C)$, and none at all have the form $C \sqcap (C \sqcap C)$. Thus instead of some 300 unverbalisable axioms, as we might have feared, we obtain only five.[19]

With increasing complexity, as one would expect, the proportion of unverbalised axioms rises sharply. Thus for δ=3 there are six non-equivalent patterns of which two are not verbalised (33%); for δ=4 the proportion rises to 60% (6/10); for δ=5, 65% (13/20); for δ=6, 81% (30/37); for δ=7, 86% (66/77); for δ=8, 93% (138/149); and so forth. Note that these figures apply only to classes constructed from restriction and intersection; if we add union and/or complement, the proportion of unverbalised axioms at a given complexity level naturally rises still further.

By restricting the CNL to non-ambiguous sentence patterns, we thus incur a potentially disastrous loss of coverage; our second claim is that in practice this loss is negligible. We cannot defend this claim in detail here, but as suggestive evidence table 3 lists the twenty most frequent predicate patterns in our corpus, including some with high complexity values (e.g., pattern 18 has δ=12): nevertheless, with one exception, *every single predicate* in the list can be verbalised by one of our permitted sentence patterns. Looking through the list, one finds

[18] Our corpus is taken from three sources: first, the University of Manchester TONES repository [22]; second, the Ontology Design Patterns corpus [11]; and finally a collection of about 160 ontologies downloaded from Swoogle [5]. It embraces a wide range of sizes, domains, and authoring styles.

[19] Here and elsewhere in this section, 'unverbalisable' means that the axiom in question cannot be verbalised *by our grammar*; obviously any axiom could be verbalised by *some* grammar.

Table 3. Frequencies of predicate patterns including intersection, union and complement. Restrictions are abstracted, so that $P.C$ covers existential, universal and numerical restrictions together with object and data values. Frequencies are based on a corpus of 3648 axioms of complexity $\delta \geq 2$.

	Predicate pattern	**Frequency**
1	$C \sqcap P.C$	1100
2	$P.(C \sqcap P.C)$	473
3	$P.(C \sqcup C)$	434
4	$C \sqcap (P.C \sqcap P.C)$	248
5	$P.(C \sqcap (P.(C \sqcap P.C)))$	164
6	$P.(C \sqcup (C \sqcup C))$	105
7	$C \sqcap (P.C \sqcap (P.C \sqcap P.C))$	78
8	$P.C \sqcap P.C$	59
9	$C \sqcap (P.C \sqcap (P.C \sqcap (P.C \sqcap P.C)))$	47
10	$P.(C \sqcup (C \sqcup (C \sqcup C)))$	43
11	$C \sqcap (P.(C \sqcap P.C))$	39
12	$P.(C \sqcup (C \sqcup (C \sqcup (C \sqcup C))))$	35
13	$P.(P.C)$	33
14	$P.\neg C$	28
15	$P.(C \sqcup (C \sqcup (C \sqcup (C \sqcup (C \sqcup C)))))$	26
16	$P.C \sqcup P.C$	25
17	$P.C \sqcap (P.C \sqcap P.C)$	20
18	$C \sqcap (P.C \sqcap (P.C \sqcap (P.C \sqcap (P.C \sqcap (P.C \sqcap P.C)))))$	20
19	$\neg P.C$	19
20	$P.(C \sqcup (C \sqcup (C \sqcup (C \sqcup (C \sqcup (C \sqcup C))))))$	17

that again and again ontology authors opt for the familiar forms of restriction lists and chains, while avoiding the combination of intersection and union in the same predicate, almost as if they were composing axioms with the intention of avoiding ambiguous verbalisations. The only exception is the pattern $P.(P.C)$ which, as already mentioned, cannot be verbalised for syntactic reasons unless it is reformulated as $P.(\top \sqcap P.C)$, which corresponds to pattern 2 in the table and can therefore be verbalised unambiguously.

6 Designing an Editing Tool

The sentence formation rules in OWL Simplified English ensure that complex sentences are *right-branching*, since the first constituent of a coordinated unit is always a simple noun-phrase or verb-phrase expressing an atomic class or simple restriction. As well as minimizing structural ambiguity, this constraint allows the grammar to be cast in the form of a finite-state transducer (as in figure 1) which reads sentences word by word from left to right, while progressively building an interpretation in OWL. Such a grammar brings an obvious advantage in efficiency—processing time is linear with sentence length; it also facilitates

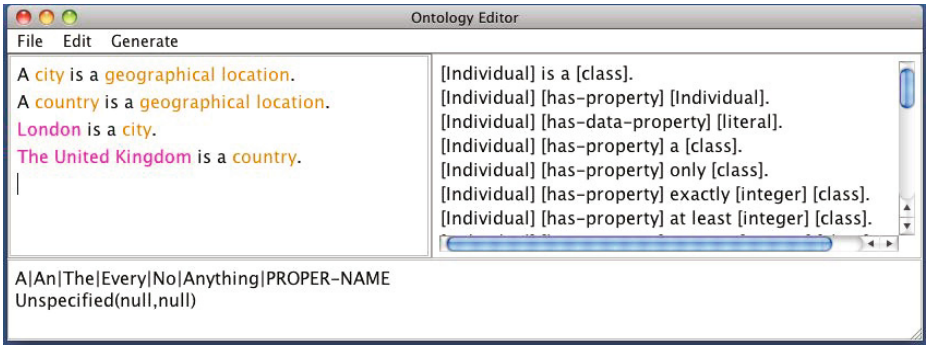


Fig. 2. Snapshot of editing tool

implementation of an editing tool that gives feedback and guidance as the user composes each sentence: handcrafted information can simply be associated with each state of the network.

We are now prototyping an editing tool that exploits these advantages in order to combine menu-guided sentence construction, as in WYSIWYM [12], with free text entry. The appearance of the interface is shown in figure 2. Four sentences have been composed in the editing pane at the top left; the cursor is now placed at the start of the fifth line, ready to start a new sentence. As can be seen, entity types are distinguished through a colour code based on the conventions of Protégé: violet for individuals, orange for classes, and blue for properties.[20] In the top right pane are shown all the minimal patterns for complete sentences expressing the various axiom types, with entity names represented through placeholders, as in WYSIWYM applications. The pane at the bottom indicates the words that may be typed in next, along with the interpretation of the current (empty) sentence, shown in OWL Functional Syntax.

To add a new sentence, the user can either select a sentence pattern from the options on the right, or simply start typing, perhaps entering the words 'The Tate'. Recognising a (possibly complete) individual name, the editor will colour this text violet, and the options will be modified to show possible *continuations*—for instance, they will include 'is a [class].'. The feedback in the lower pane will also change, including an OWL expression for which the functor is now known:

```
ClassAssertion(NamedIndividual(#The_Tate_),null)
```

After completing the name by typing 'Modern', the user might click on the option 'is a [class].', whereupon this string will be copied into the editing pane, so yielding the outline of a complete sentence:

The Tate Modern is a *[class]*.

[20] In versions of this paper formatted for printing in black and white, we will show individual names in **bold**, class names in *italics*, and property names underlined.

As in a WYSIWYM application, clicking on the place-holder will then yield a list of substitution options in the right pane, based in this case on the class names already used in the text:

The Tate Modern is a [class].	city country geographical location

Assuming that none of these options fits the bill, the user can simply type in a new class name, replacing the selected substring as in any text editor.

The Tate Modern is a *gallery*.

```
ClassAssertion(NamedIndividual(#The_Tate_Modern),Class(#gallery))
```

A minimal sentence is now complete. To extend the sentence, the user deletes the full stop and types a space: minimal continuations including 'and a [class].', 'or a [class].', and 'that [has-property] [Individual].' then appear in the options pane. If the user selects the last of these, overwrites the place-holder '[has-property]' with the new property name 'is located in', and clicks on '[Individual]', a more complex sentence takes shape:

The Tate Modern is a *gallery* that is located in [Individual].	London The Tate Modern The United Kingdom

Clicking on 'London' completes the sentence, interpreted as a class assertion with a constructed class as predicate.

The Tate Modern is a *gallery* that is located in **London**.

```
ClassAssertion(NamedIndividual(#The_Tate_Modern),
   ObjectIntersectionOf(Class(#gallery),
      ObjectHasValue(ObjectProperty(#is_located_in),
         NamedIndividual(#London))))
```

If desired, the sentence could be extended further by the same method, yielding perhaps 'The Tate Modern is a gallery that is located in London and contains at least three portraits that are painted by Strindberg.'. This continuation assumes that 'contains' has been listed as a verb; otherwise the finite-state transducer will fail at this point, with the substring after 'and' displayed in red. Verbs are listed through metadata statements marked by the initial character '#':

VERB contain contains

Apart from guiding the editing of single sentences, the prototype has most of the functionality that one would expect from an editing tool: ontologies can be saved, either as text files or in OWL/XML format; ontologies already in OWL/XML format can be imported and converted to text in OWL Simplified English;[21]

[21] Importing an existing ontology will often result in loss of information, since some axioms might lie outside the coverage of the CNL; also, entity names will have to be derived from identifiers or labels which might be unsuitable, for instance because they are not English-based.

texts can be regenerated from their OWL interpretations (this will help clear up any errors in morphology, such as 'a animal'); *alternative* versions of the text can also be generated, such as a glossary that groups together the statements about each individual or class, as in the SWAT verbaliser [23].

7 Conclusion

We have described work in progress on developing a CNL for specifying OWL ontologies, and an accompanying editing tool. The hypothesis underlying this work is that ontology editing can be supported using a simple finite-state grammar in which complex sentences are always right-branching. We have argued that such a language brings several potential advantages: effort in specifying a vocabulary or lexicon is minimized; structurally ambiguous sentences are avoided; and sentences can be parsed by a finite-state transducer which efficiently provides feedback on the interpretation assigned so far, the words that may be typed in next, and the linguistic patterns by which a sentence may be completed (or extended when potentially complete). These advantages are gained only by accepting severely reduced coverage of the OWL statements that are possible in principle; however, analysis of several hundred existing ontologies indicates that in practice, developers overwhelmingly favour right-branching expressions that can be verbalised by our grammar. As will be obvious, a crucial question not yet addressed is whether the proposed CNL and editing tool prove effective in user trials, especially for domain experts with limited knowledge of OWL.

References

1. Altmann, G.: Ambiguity in sentence processing. Trends in Cognitive Sciences 2(4), 146–152 (1998)
2. Cognitum. Fluent Editor for OWL (2012), `http://www.cognitum.eu/Products/FluentEditor/` (last accessed: March 5, 2012)
3. Davis, B., Iqbal, A.A., Funk, A., Tablan, V., Bontcheva, K., Cunningham, H., Handschuh, S.: RoundTrip Ontology Authoring. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 50–65. Springer, Heidelberg (2008)
4. Denaux, R., Holt, I., Corda, I., Dimitrova, V., Dolbear, C., Cohn, A.G.: ROO: A Tool to Assist Domain Experts with Ontology Construction. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) Proceedings of the 5th European Semantic Web Conference (2008)
5. Ding, L., Finin, T., Joshi, A., Pan, R., Scott Cost, R., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: a search and metadata engine for the semantic web. In: Proceedings of the 13th ACM International Conference on Information and Knowledge Management, pp. 652–659. ACM, New York (2004)
6. Hart, G., Johnson, M., Dolbear, C.: Rabbit: Developing a Control Natural Language for Authoring Ontologies. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 348–360. Springer, Heidelberg (2008)

7. Kaljurand, K., Fuchs, N.: Verbalizing OWL in Attempto Controlled English. In: Proceedings of OWL: Experiences and Directions, Innsbruck, Austria (2007)
8. Kuhn, T.: How Controlled English can Improve Semantic Wikis. In: Lange, C., Schaffert, S., Skaf-Molli, H., Völkel, M. (eds.) Proceedings of the Fourth Workshop on Semantic Wikis, European Semantic Web Conference 2009. CEUR Workshop Proceedings. CEUR-WS, vol. 464 (June 2009)
9. Kuhn, T.: Controlled English for Knowledge Representation. PhD thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich (2010)
10. Mellish, C., Sun, X.: The semantic web as a linguistic resource: Opportunities for natural language generation. Knowledge-Based Systems 19(5), 298–303 (2006)
11. Ontology Design Patterns. Repository of ontologies (NEON Project) (2010), `http://ontologydesignpatterns.org/wiki/` (last accessed: April 21, 2010)
12. Power, R., Scott, D.: Multilingual authoring using feedback texts. In: Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics, Montreal, Canada, pp. 1053–1059 (1998)
13. Power, R.: Complexity assumptions in ontology verbalisation. In: 48th Annual Meeting of the Association for Computational Linguistics (2010)
14. Power, R., Third, A.: Expressing OWL axioms by English sentences: dubious in theory, feasible in practice. In: Proceedings of the 23rd International Conference on Computational Linguistics (2010)
15. Pulman, S.: Controlled language for knowledge representation. In: Proceedings of the first International Workshop on Controlled Language Applications. Katholieke Universiteit Leuven, Belgium (1996)
16. Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., Wang, H., Wroe, C.: OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. In: 14th International Conference on Knowledge Engineering and Knowledge Management, pp. 63–81 (2004)
17. Schwitter, R.: Controlled natural languages for knowledge representation. In: Proceedings of the 23rd International Conference on Computational Linguistics, pp. 1113–1121 (2010)
18. Schwitter, R.: Processing Coordinated Structures in PENG Light. In: Australasian Conference on Artificial Intelligence, pp. 658–667 (2011)
19. Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A Comparison of three Controlled Natural Languages for OWL 1.1. In: 4th OWL Experiences and Directions Workshop (OWLED 2008), Washington, DC, April 1–2 (2008)
20. Schwitter, R., Meyer, T.: Sydney OWL Syntax - towards a Controlled Natural Language Syntax for OWL 1.1. In: OWLED: OWL Experiences and Directions (2007)
21. Stevens, R., Malone, J., Williams, S., Power, R., Third, A.: Automating generation of textual class definitions from OWL to English. Journal of Biomedical Semantics 2 (suppl. 2), S5 (2011)
22. TONES. Repository of ontologies at the University of Manchester (2010), `http://owl.cs.manchester.ac.uk/repository/browser` (last accessed: April 21, 2010)
23. Williams, S., Third, A., Power, R.: Levels of organisation in ontology verbalisation. In: Proceedings of the 13th European Workshop on Natural Language Generation, pp. 158–163 (2011)
24. Yang, H., De Roeck, A.N., Willis, A., Nuseibeh, B.: A methodology for automatic identification of nocuous ambiguity. In: Proceedings of the 23rd International Conference on Computational Linguistics, pp. 1218–1226 (2010)

Spatiotemporal Extensions to a Controlled Natural Language

William R. Murray and Tomas Singliar

Boeing Research and Technology, P.O. Box 3707, Seattle, WA 98124, USA
{william.r.murray,tomas.singliar}@boeing.com

Abstract. We discuss extensions to the controlled natural language CPL to explicitly define and reason about temporal and spatial concepts. New sentence templates are added to define culturally-specific temporal and spatial names. Two kinds of queries are supported: (1) proximity queries, and (2) relational (e.g., overlap or containment) queries. Proximity queries require concepts grounded in terms of absolute time (UTC times) or location (latitude and longitude). Relational queries can be answered using Allen's Interval Algebra, for time; or Region Connection Calculus (RCC-8), for regions. Our motivation for handling proximity queries is to allow easy feature definition for a machine learning algorithm that detects anomalous vehicle travel. Our motivation for addressing more general, relational queries is to improve CPL's modeling of the temporal and spatial relationships of events and objects that occur in sentences, in order to further improve its semantic reasoning capabilities in applications such as Q&A.

Keywords: controlled natural language, CPL, spatiotemporal reasoning, Allen's Interval Algebra, RCC-8, inverse reinforcement learning.

1 Introduction

We are currently extending CPL (Computer Processable Language) [1] to act as a controlled natural language (CNL) interface for a surveillance data exploitation system with a machine learning algorithm at its core. A CNL interface allows us to easily define new features, refine them, and find the best to model the domain. The task of the machine learning algorithm is to identify anomalous vehicle tracks, e.g., car tracks on the ground or ship tracks in the sea, that may indicate unusual behaviour, such as smuggling or insurgent activities. Each track consists of a large series of track points for a vehicle. The machine learning paradigm used is inverse reinforcement learning (IRL). IRL [2] infers a "utility function" to represent normal behaviour (e.g., generally preferring shortest route paths) and then flags tracks that are markedly different. Each track point has a time stamp, expressed as a UTC time, and a location, expressed as map coordinates. The intuition is that in the aggregate, most normal traffic is the result of rational behavior of agents optimizing a utility function, e.g., attempting to minimize travel time and distance. Vehicles whose behavior deviates from normal may be optimizing a quite different utility function, e.g., civilian combatants may have

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 61–78, 2012.

different goals, preferring overwatch of checkpoints, travel circuits close to convoy routes, or travel on remote roads at nighttime with stops in unobserved areas. The first goal of the system is to present the human analyst with the deviating tracks. The analyst decides whether the behavior is indeed interesting, and if it is not, provides discriminating features. For example, if we include gas station locations, then otherwise unexpected vehicle stops and longer paths may be explained, reducing the false alarm rate. The second goal of the system is to point out tracks that are unusual with regards to particular features (e.g., vehicles with a high implied cost of passing through a checkpoint are interesting). For more details, see [3].

The choice of the spatial and temporal features that best model these normal or abnormal pathways is similar to the selection of the features in standard concept learning. Once a set of possible features has been provided, the IRL algorithm learns the best linear combination of them from data. CNL facilitates rapid experimentation with different spatial and temporal concepts that may act as features for the IRL domain modeling. Well-selected features improve the mathematical modeling of the problem by becoming unit basis vectors of the reward (utility) function space. We will just refer to these as features, rather than unit basis vectors, to simplify discussion. Without the CNL, each change requires new code or programming modifications to introduce or refine the features. As many of these features are culturally and situationally dependent, it is not a priori obvious how best to model a situation. Iterative experimentation with different feature sets is the best method for the system to quickly acquire a compact set of informative features. The intention of the CNL interface is to accelerate the experimentation process and to simplify the modeling process.

1.1 Culturally Dependent Concepts

Ideas of time and space are relative to a culture [4-6]. If we wish to model traffic flow in Spain, compared to the United States or Iraq, we need to take into account cultural definitions of work times, meal times, and holidays, as we expect increased or decreased traffic at these times. We also need to model important places in a town or city that affect traffic, e.g., Army checkpoints, bridges, constructions, traffic barriers, markets, etc.

Most culturally defined spatiotemporal concepts are not sharply demarcated. For example, is 1:40 PM part of lunch time? Where does the city center start, for a city? Instead, concepts are represented as prototypes [7] (also called schemas, cognitive frames, and idealized cognitive models) characterized by family resemblances, typical examples (e.g., the typical airplane flight), ideal examples (e.g., ideal air travel), and salient examples (e.g., hijackings, flight attendant disruptions, and other memorable exceptions).

The reason this imprecision is important is that our IRL algorithm needs to know how "close" or "near" a point in time and space is to one of the concepts (e.g., "market", a place, and "market day", a time). The IRL algorithms require, for feature scaling reasons, that the value for each proposed feature is a result from 0.0 to 1.0 and there should be a soft fall-off as a track point moves away from satisfying a feature.

The IRL features and their values are akin to fuzzy set membership, but without the full set of logical operators that would define a fuzzy logic. Conjunction and disjunction are defined with operators such as **min** and **max**, but we have not yet added negation or a means of lexical expression for it (e.g., 'away from the city').

The spatiotemporal concepts are vaguely defined. For example, if one of our features is Friday prayer (*jumu'ah*), an Islamic prayer held shortly after noontime on a Friday, then we wish to consider vehicle tracks *around* this time. Clearly a vehicle track at 12:01 PM on Friday qualifies, but what about 12:16 PM, 12:47 PM, or 1:15 PM? With formal logic we would be constrained to a predicate, e.g., **jumuah**(*time*), which would be either false or true, requiring a sharp cut-off at some time. Our IRL application requires a smoothed fall-off, so 12:01 PM may have a "membership" of 0.999, the 12:16 PM time a membership of 0.80, 12:47 PM a membership of 0.5 (questionable), and 1:15 PM a membership of 0.1 (not a good fit). Humans, unlike machines, think in terms of prototypes that have characteristic examples [4, 5]. For example, we may consider December 25th to be the best example of a day in the "Holiday Season". Similarly, a city center is not necessarily the geographical center of a city, but more likely a cultural, business, or population center. We would like our extensions to CPL to facilitate an intuitive use of prototypes in defining times and places where possible.

1.2 Time and Space in Controlled Natural Languages

In general, controlled natural languages (e.g., ACE [8], Rabbit [9], and CELT [10]) have not made any special provision for time and space other than distinguishing between times and places in terms of what an appropriate answer to a *when* or *where* question would be.

Two exceptions are CPL [11] and PENG [12, 13]. CPL relies on the situation calculus for its formal semantics, and is implemented in the Knowledge Machine (a frame-like knowledge representation language, see [14]), with its context mechanism, and each context is called a situation [15]. PENG uses the event calculus and relies on first-order logic theorem provers such as Otter [16], or its successor, Prover9 [17] to implement its inference. In both cases, the focus is on representing possible worlds that result from actions, and making inferences in those worlds. Neither addresses the same kinds of spatiotemporal reasoning addressed in this paper—whether events are close to places and times of interest, and the interrelationship of possibly overlapping regions in time and space. Instead, they address a *different* kind of temporal reasoning, oriented towards planning and hypothetical reasoning. Spatial reasoning is not addressed.

In the other controlled natural languages, concepts related to temporal intervals or spatial regions may be placed into class categories and have their own instances, but there is no special provision for temporal or spatial reasoning, or built-in library of axioms for those kinds of reasoning. For example, Baghdad may be classified as an instance of city in the WordNet [18] ontology, and Ramadan classified as an Islamic calendar month (a hyponym of time_period#n1). In fact, CPL incorporates WordNet's ontology and thus captures these kinds of lexical relationships. However, CPL previously had no spatiotemporal reasoning capabilities, other than by representing and

traversing linear sequences of events, or by performing type inheritance over the kinds of lexical concepts illustrated above.

Why, then, should time and space benefit from any *further* special treatment in a controlled natural language (CNL)? First, human experience is embodied so our spatial and kinetic experience in the world is primary [4,7]. Second, as our thinking is metaphoric [19], much of our reasoning about time is built on reasoning about space (e.g., "the days ahead", "events behind me", etc). So, although our CNL could model these concepts using logical axioms alone, we can provide a more useful (easier-to-use for humans) CNL if our model of time and space aligns more closely to human thought, while still retaining a computationally tractable substrate.

For our particular IRL application, we further need a way to allow a smooth fall-off for track points that almost fall within a time period or that are close to significant places. Providing this kind of utility basis function requires that we go beyond first-order logic.

1.3 Two Kinds of Reasoning: Proximity and Relational Reasoning

We are interested in two kinds of reasoning:

1. *Proximity reasoning*—we would like to know whether a track point satisfies temporal and spatial concepts, or comes close to satisfying them. If it is "close", how close is it?
2. *Relational reasoning*—we also want to support more general Q & A with CPL, where intervals may be more loosely associated with events, and places with settings, rather than the precise times and locations afforded by track points. In this second kind of reasoning, we want to be able to infer relations between temporal concepts (e.g., given x before y and y during z, what relationships can x and z have?), or between spatial concepts (e.g., given x outside y and z inside y, what relationships can x and z have?), given descriptions of how events and places are interrelated.

Proximity Reasoning

For proximity reasoning, we need to be able to peg named times to the timeline and named places to the map. We use UTC [20] as a standard format for times. We use latitude and longitude as a standard format for map location. For times, we ultimately need to compare UTC times, such as 2012-08-29T13:00:00, for 1 PM, August 29th, 2012, with temporal intervals that may consist of multiple temporal concepts. For example, we can ask, is the time 2012-08-29T13:00:00 a "Monday lunch", a "summer afternoon", an "August weekend", a "day in Ramadan", or close to one of these intervals? To make the comparisons, we ultimately have to translate terms such as "weekend" and "lunch" into UTC times that we can compare against our target time. When there are multiple temporal constraints (e.g., "August" and "weekend" in "August weekend") to be satisfied, our reasoning approach evaluates the closeness of fit for each temporal concept alone, and then combines (e.g., using **min**) the results to obtain a single overall membership value between 0.0 and 1.0.

Places are similar: They need to be pegged to the map for proximity reasoning. We can take a track point, e.g., 47° 22′ 0″ N, 8° 33′ 0″ E at the same time, 2012-08-29T13:00:00, and ask similar questions, such as, is this near or in "a town", "Europe", "Switzerland", or the "University of Zurich"? The concept of “near” clearly needs to be defined on an appropriate scale that depends on the viewpoint (Is Zurich near to or far from Lake Constance?). In general, resolution of this ambiguity is a context-dependent AI task.

Relational Reasoning
Relational reasoning does not need to peg named times to UTC times or named places to latitude and longitude. Named times and places can still be placed in relationship to each other: We can assert that one time interval is before, after, or during another; we can assert that one place does not overlap a second place, but is inside a third place. Then, using constraint propagation techniques in Allen's Interval Algebra [21] and RCC-8 [22], new relationships may be inferred and existing relationships refined. Essentially, a partial ordering of named times and intervals is derived, with the detail depending on how much is known about concept interrelationships.

Finally, if named temporal or spatial regions are ultimately pegged to the timeline and map, then we can also perform proximity reasoning on them and use the numeric coordinates on the timeline or map as additional constraints to further refine the temporal and spatial constraint set.

The rest of this paper explores CPL's extensions in progress to support these two kinds of reasoning. How are they realized in a controlled natural language? What is their semantics? What kind of queries can we ask for knowledge entered with these extensions?

First, we consider proximity reasoning that supports our inverse reinforcement learning application for detecting instances of anomalous vehicle travel.

2 Defining New Concepts for Proximity Reasoning

We consider how to define new concepts for proximity reasoning. A *sentence template* in CPL is a prescribed sentence format that is interpreted in such a way as to change the CPL lexicon itself at run-time. We use sentence templates to define new temporal concepts (e.g., afternoon tea) and new spatial concepts (e.g., FOB, or forward operating base). We consider temporal concepts first.

2.1 Defining New Temporal Concepts for Proximity Reasoning

To define a new temporal concept, CPL needs to know its time and to have some way of determining what is the most prototypical time exemplifying that concept, along with boundaries defining when that concept clearly holds. The size of the boundaries will be used to determine the rate of the fall-off. The midpoint μ between the upper and lower boundaries is interpreted as the most prototypical time, and the distance from the midpoint to a boundary is used as the standard deviation σ for a bell curve,

where μ is defined to be the most prototypical time. Later, we may wish to allow for asymmetric models (e.g., 11:30 AM as the lowermost boundary of lunch, noontime as the most prototypical example of the concept, and an upper boundary of 1:30 PM), but our first model is symmetric to simplify definition.

We use UTC [20] as a canonical way of expressing and comparing times. We extend our parser to allow multiple variants for expressing dates and times and converting them to UTC.

Defining a Named Time in Terms of a Specific Time or Date

There are two kinds of sentence templates that CPL supports for defining new temporal concepts:

1. *Definition in terms of an expected time*: A time concept can be defined in terms of an expected *time-period* of a given granularity (month, day, hour, minute, or second):

S ::= *time-concept* occurs [every / this] *time-period1*
[at / on] [the *time-period2* of] *time.*

Examples:

a. Lunch occurs every day at the hour of 12 o'clock.
b. Midnight occurs every day at 24:00.
c. Christmas occurs every year on the day of December 25th.
d. The fire drill occurs this day at the time of 11:15 AM.

Notes: When a temporal concept is defined in this way, then we are associating the name of the concept with its ideal, or most prototypical, value. If the time is a recurrent time period, then it repeats according to the first time period *time-period1* (e.g., "every year"). The second time period, *time-period2*, optionally indicates the size of the time interval being defined. The default is one minute. It determines how "near" a time point is when it falls outside of the interval. The size of the interval surrounding the ideal time is plus or minus one-half the size of the second time period given. So, in (a), an hour is given, so lunch is defined as having membership 1.0 from 11:30 AM to 12:30 PM and then falling off, with $\sigma = 30$ minutes for values outside of that range. In (b) and (d) the interval is a minute, and $\sigma = 30$ seconds. In (c) it is a day, and $\sigma =$ 12 hours.

If we want the interval to be different than ½ the granularity of a month, day, hour, minute, or second, then we need to use the next sentence template.

Defining a Named Time in Terms of Boundary Times

2. *Definition in terms of boundary times:* A time concept can be defined in terms of a start and stop time, where the start and end times can be any UTC times:

S ::= *time-concept* [starts / is from] *start-time*
[and ends / until] *end-time*.

Examples:

a. Ramadan starts 20 July 2012 and ends 18 August 2012.
b. Lunch is from 11:00 until 13:00.
c. Vacation holiday is from Christmas until New Year's.

Notes: When a temporal concept is defined in this way, then we are associating the name of the concept with an interval where we are given the start and stop dates and times.[1] The midpoint of this interval is taken as the ideal, or most prototypical, value. The size of the interval is given directly by the start and end times. So, in (a), Ramadan is defined as having an interval of 30 days (from 2012-07-20 to 2012-08-18 inclusive of both days), so $\sigma = 15$ days and the midpoint μ is August 3rd, 2012. Similarly, in 2 (b), we have defined lunch as having an interval of 2 hours, rather than the default 1 hour in 1 (a).

In both cases, the temporal concepts are represented as partially specified UTC times, where a lower limit, upper limit, and midpoint are provided. The unspecified parts of the UTC times are filled in from a particular *query time,* where any query time is fully instantiated with years, months, days, hours, minutes, and seconds. The query times are taken from track data that associate vehicle movements at particular times with geographic coordinates.

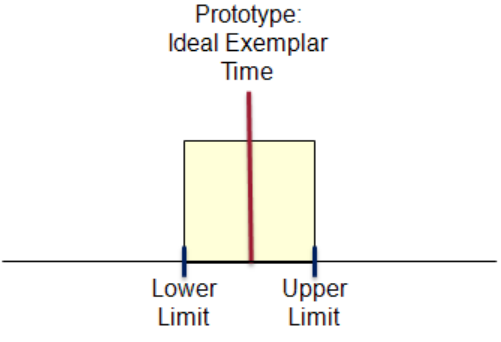


Fig. 1. A binary predicate representation of a temporal interval has sharp cut-offs. We model a culturally defined time such as "lunch" with a prototype and time intervals surrounding it.

What we have done is taken a hard predicate for a time interval, as shown in Fig. 1, and softened it by overlaying a bell curve on it and then lopping off the top,

[1] Where low order date-time information is omitted, we assume the minimum values for the start time and maximum values for the end time of the interval (e.g., from July 3rd to July 7th we would assume as meaning July 3rd at 0:00:00 to July 7th at 23:59:59.) Where high order date-time information is missing, we obtain any missing high-order date-time information from the query date time in performing a test (the process is similar to merging file pathnames). For example, if we are testing July 5, 2013 at 3:30 PM (UTC 2013-07-05T15:30:00) against the partially specified intervals July 3rd to July 7th, we test to see if UTC 2013-07-05T15:30:00 is in the interval UTC 2013-07-03T00:00:00 to UTC 2013-07-07T23:59:59 and find that it is.

essentially clamping the value to 1.0 between μ-σ and μ+σ. Normal bell curve fall-off happens outside of that range, allowing our predicate to be "softened," as shown in Fig. 2 (IRL has no requirement that the area under the curve is unity. The value of the membership function is not a probability. However, having a maximum of 1 is convenient for later comparison and interpretation of the learned linear combinations).

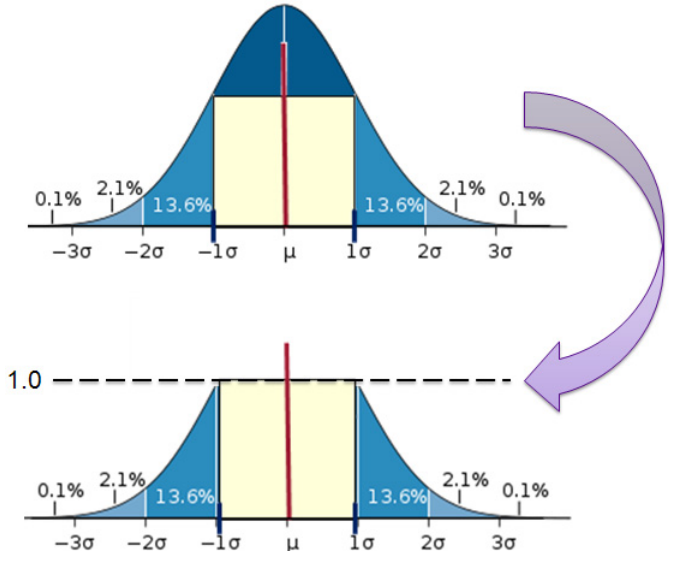


Fig. 2. Our membership function "softens" the binary predicate to provide values from 0.0 to 1.0 falling off as we go outside the bounds of the temporal concept

So, if we define "lunch" as above, from 11 AM to 1 PM, we provide upper and lower limits as shown in Fig. 3. Fig 3 shows an initial binary predicate. Smoothing with a bell curve provides the results in Fig. 4. For proximity reasoning, we use the smoothed membership function, while for relational reasoning (e.g., "Did Jack visit after lunch?") we will use the predicate definition. If we want to know the membership $\boldsymbol{u}$ of a time point, say 11:30 PM, its value is 1.0 for $\boldsymbol{\mu-\sigma \leq x \leq \mu+\sigma}$ and otherwise the membership $\boldsymbol{u}$ is given by:

$$u = e^{\left(\frac{1-\left(\frac{x-\mu}{\sigma}\right)^2}{2}\right)}$$

The latter formula is the same as that for a normal distribution curve, except it has been multiplied by a constant so that the value $\boldsymbol{u}$ = 1 for $\boldsymbol{x = \mu \pm \sigma}$, and then it falls off from there. For 1:30 PM, x is μ+1.5σ, so f(x) = $e^{-0.625}$, or 0.535261. Essentially, our equation multiplies the normal curve equation by a constant, so that the value at μ+σ is 1.0, and at μ+2σ it is $e^{-1.5}$, and so on. Thus, 2 PM has membership 0.223130.

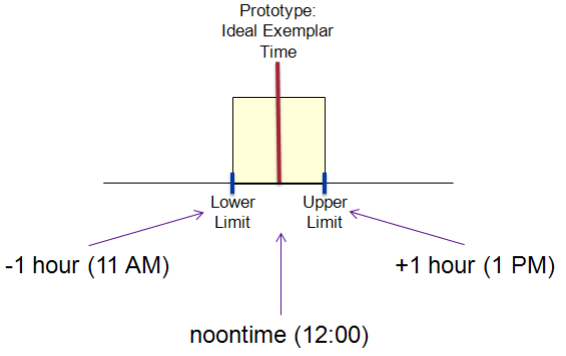


Fig. 3. The definition of 'lunch' from 2(b), where lunch is defined as happening between 11:00 and 13:00, provides a binary predicate **lunch**(t) that is true when $11{:}00 \leq t \leq 13{:}00$.

2.2 Defining New Spatial Concepts for Proximity Reasoning

We define spatial concepts similarly: First, we construct a binary predicate suitable for defining whether a point is within a spatial region, and then we "soften" it. We define spatial concepts with proper names (e.g., "The Green Zone" or "Baghdad") and then peg them to a map so we can decide whether a track point comes "near" them or not.

To simplify our comparisons and proximity reasoning, we model places (spatial regions) as having a central point (the most prototypical point, e.g., this could be 'downtown' for a city, or a town square off the high street for a village) with a circle around it. This modeling is very poor for large, irregularly shaped regions (e.g., cities such as Seattle), but should be adequate for modeling proximity to the urban-scale structures such as mosques or gas stations that we envision as most typical of our traffic modeling application. A more precise definition of spatial concepts would rely on geographic data sources, especially geographic information systems (GIS). A more sophisticated reasoning approach would provide more precise spatial reasoning, e.g., supporting polygons and boundary boxes for regions, and would be capable of determining if one region, or its bounding box, overlapped another. Still, a default definition independent of such data sources, and the use of simpler reasoning methods, provides graceful performance degradation and convenience for conceptual development and experimentation.

In the current approach, the radius of a circle modeling a spatial region models the size of the physical location, encompassing the bulk of its area, and intuitively how "close" or "near" a track point may be to the place. Thus, a car may be "close" to a major bridge if it is 1 km from it, but not "close" to a café until it is within the same block. This fuzziness is similar to calling a galaxy small if it is smaller than our own ('small' here has a scale of light-years), and a chameleon small if it is one that can fit on one's fingertip ('small' here has a scale of millimeters). In both examples, the spatial descriptor is relative to the size of the object being discussed. We use a similar intuition here, and allow categories of places (e.g., churches) to have a default radius that we override for exceptionally large or small instances.

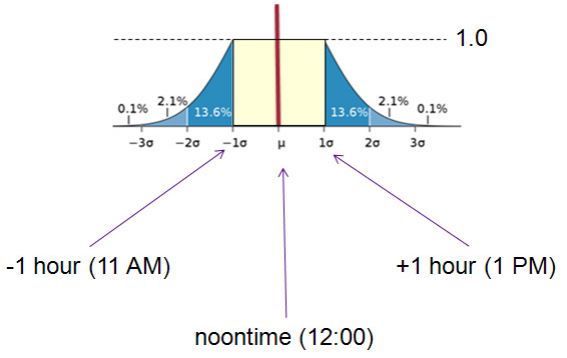


Fig. 4. Membership function for "lunch" when σ=1 and μ=12. There is a rapid, but smooth fall-off for values of time outside of 11 AM and 1 PM.

Each track point has a location, expressed as a latitude and longitude, along with a timestamp. We use map coordinates as the basis for our definitions of spatial regions, too:

1. *Definition in terms of a most central point*: A spatial concept is defined in terms of some point in space that is most prototypical of that place. For example, this might be a town square in a village, or a salient point, such as the White House in Washington D.C., or simply the rough center of a city. In Wikipedia, each city has a latitude and longitude given; for Seattle, that point is a downtown location.

S ::= *place-concept* is a *kind-of-place* [with radius *r*] at *coordinates*

We require that *kind-of-place* either be "place", or a WordNet word with some sense that inherits from the single WordNet sense of "geographic area", such as "public square", or the first sense of "building", such as "church". The reason we would like to be able to categorize a kind of place is that then we can estimate a default size for a particular place of that kind (e.g., a typical village church), so the radius can be omitted if a place instance is of a 'typical' size.

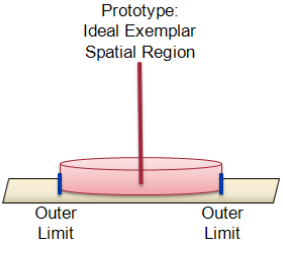


Fig. 5. Our first-step binary predicate representation of a spatial region is similar to Fig. 1, extrapolated into two dimensions and still having sharp cut-offs. We model a culturally defined place, such as a mosque, with a prototypical center with a radius surrounding it such that the main mass of the spatial region is encompassed.

Examples:

a. Seattle is a city at 47°36′35″N 122°19′59″W.
b. Stow-on-the-Wold is a village at 51.928°N 1.718°W.
c. Alamo Square is a park at 37.776384°N 122.434709°W with radius 0.6 km.
d. The Space Needle is a tower at 47.6204, -122.3491.

Notes: When a place name is defined in this way, then we associate the name with the single most salient map location considered to best characterize it. The particular place that is considered the most salient example of a location, and the boundaries of a place, will often be culturally defined. For example, the "Professorville" location of Palo Alto is "The historic district is bounded by Kingsley and Addison avenues and the cross streets of Ramona and Waverly.", but "The community considers the district to be larger and bounded by Addison and Cowper St. to the north west and north east and Emerson St. and Embarcadero Rd. to the south west and south east." [23].

We can define a default radius for a new kind of geographic region or building by relating the area of the spatial region to its radius. We use this sentence template:

A *kind-of-place* has a default area of *area*.

Examples:

a. A house has a default area of 2000 square feet.
b. A city has a default area of 150 square miles.
c. A village has a default area of 2.5 square miles.

Notes: Once we are given a default area, we infer a default radius for the area by assuming the area is circular.

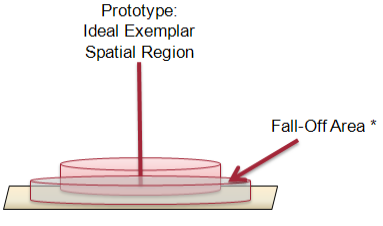


Fig. 6. Next, we "soften" our binary predicate for spatial locations to have a fall-off area, similar to our handling of temporal concepts. Now, rather than having a predicate such as mosque(p) that returns True or False, we have a membership function that returns 0.0 (far from the mosque) to 1.0 (inside the mosque, at its center), with a fall-off (e.g., 0.8 could be outside the mosque but close to it).

For our application, most of the regions of interest will be particular kinds of buildings or locations, such as checkpoints, mosques, and markets that can be sufficiently approximated with circular regions when looked at from a city-map. For applications other than our traffic anomaly detection application, where irregular city and structure shapes need to be modeled, we would, of course, need more sophisticated modeling techniques.

3 Relational Reasoning

We now consider reasoning to support potential question and answer applications outside of the IRL task; we would like our extensions to support these applications, too. Unlike the proximity reasoning, these extensions are planned, but not yet implemented. We can use existing CPL sentence templates to introduce names for temporal and spatial concepts that do not require the level of detail required for proximity reasoning:

- The U.S. Embassy is an instance of a building.
- An obstacle belt is a kind of location.
- High tide is a kind of time.

Standard (non-template) CPL sentences will also introduce events:

- Fred drives home from work before he eats dinner.
- Fred reads the paper before going to bed.

In the first set of examples, we have only defined names for new times and places, but not the specifics: We do not say *where* the U.S. Embassy is located, or *when* high tide occurs, just that one is a place and one is a time. Instead times and places will be defined relative to other places and time intervals: "The Embassy is next to the checkpoint." or "High tide occurs during the beginning of next month." Similarly, for the events in the second set of examples, we can relate one to another, e.g., "Fred left work after Harry left."

In CPL, built-in WordNet concepts such as park#n1, public_square#n1, church#n1 provide predefined spatial categories. These predefined categories can be identified as hyponyms of word senses for spatial regions, such as location#n1, region#n1, construction#n1, and so on. Similarly, predefined WordNet temporal categories, such as Monday#n1, Ramadan#n1, and lunchtime#n1, are hyponyms of word senses for temporal intervals or points in time such as time#n1, periodic_event#n1, and time_period#n1. Note that in the examples of proximity reasoning, we provided additional information so our algorithm could determine how "close" a track point was to a temporal concept such as "lunch" or "Christmas". WordNet does not provide enough detail to support that kind of reasoning.

In relational reasoning, our main focus is no longer the track data points used in the inverse reinforcement reasoning example. Instead, we are concerned mainly with the temporal ordering of events and the spatial relationships of the actors and props in the sentences.

3.1 Temporal Relational Reasoning

We use Allen's Interval Algebra [21] to provide a formal grounding to handle temporal prepositions. For example, CPL currently represents event sequences where the ordering is explicitly provided, but it does not represent the more general temporal relationships in Fig. 7.

Relation	Illustration	Interpretation
$X<Y$ $Y>X$	X Y	X takes place before Y
$X\mathrm{m}Y$ $Y\mathrm{mi}X$	X Y	X meets Y (*i* stands for *inverse*)
$X\mathrm{o}Y$ $Y\mathrm{oi}X$	X Y	X overlaps with Y
$X\mathrm{s}Y$ $Y\mathrm{si}X$	X Y	X starts Y
$X\mathrm{d}Y$ $Y\mathrm{di}X$	X Y	X during Y
$X\mathrm{f}Y$ $Y\mathrm{fi}X$	X Y	X finishes Y
$X=Y$	X Y	X is equal to Y

Fig. 7. We use Allen's Interval Algebra for handling prepositions such as "before", "during", "after", "while", "until", "starts at", and "ends at"

We will need to extend CPL semantic role classification to map prepositions such as those listed in Fig. 7 to the 13 relations possible in Allen's interval algebra. When a sentence that asserts a temporal relation is interpreted by CPL, we will use constraint propagation to update the known temporal relations between events.

Sentence triples are currently stored in a PROLOG knowledge base, allowing retrieval using PROLOG queries. Additional triples will be added to express inferred temporal relations. The intention is that we will be able to answer questions such as, "When did John call?" with reference to other event times, e.g., "John called when Sue was eating cake."

3.2 Spatial Relational Reasoning

Our formal reasoning approach for spatial reasoning is RCC-8 [22]. It treats places as "blobs" that can touch, overlap, be inside, or be totally disconnected from other regions. The shape, size, and geographic location of the regions are not considered. We will need to extend the semantic role classification to map prepositions such as those listed in Fig. 8 to the 8 relations possible in RCC-8. When a sentence asserts a new spatial relationship that we can model, then we will add the new constraints to the existing set of spatial constraints, perform constraint propagation, and update the PROLOG triple store.

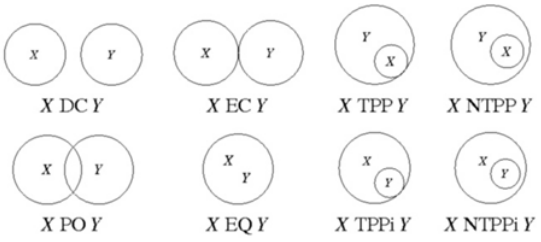


Fig. 8. We use RCC-8 to handle spatial prepositions such as "inside", "outside", "overlapping", "adjoining", "connected to", and "separate from"

3.3 Extending the Parser, Semantic Roles, and Adding Stative Verbs

We plan to extend the CPL parser, SAPIR [24], to take into account whether prepositions are more likely to modify events or nouns, based on whether prepositional objects are known temporal or spatial concepts.

Since CPL is a controlled language, we can choose to interpret prepositional objects that are temporal concepts as modifying the main verb of a sentence. So, in "John saw the man in the park with the telescope on the weekend." the preposition "on the weekend" will be taken to modify the verb "saw", as "weekend" is a temporal concept. Thus, "the weekend" describes when the event "saw" occurred.

Prepositional objects that are spatial concepts will be taken to modify a preceding noun. In the previous sentence, "in the park" will be taken as providing the location of "the man", rather the event "saw", since "park" is a spatial concept (a hyponym of "geographic region" in WordNet).

Where neither of these new rules apply, the current parser heuristics will apply as before. Some of these heuristics are domain-specific and were originally tuned for the scientific domains of physics, chemistry, and biology in the project HALO [25], which aims to achieve AP-level proficiency in these subjects and to act as a knowledge resource for students, in addition to pushing state-of-the-art research in high-performance knowledge bases.

In addition to the prepositions mentioned in Fig. 7 and Fig. 8, we can add similar stative verbs, such as "overlaps", "contains", and "touches" for spatial prepositions. For temporal prepositions, we can add "precedes", and "follows". Some prepositions can apply to either spatial or temporal concepts, such as "in", "from", and "to". When applied to spatial concepts "in" means one object is "inside" another. When applied to temporal concepts, the word "in" means "during". Our rules for semantic role disambiguation can take into account whether concepts are known to be spatial or temporal in making the distinction.

4 Summary

We have defined extensions to a controlled natural language, CPL, to handle concepts of time and space. We have presented current work on extensions to CPL for proximity reasoning that allow user-defined concepts for times and places. By tying temporal intervals to UTC times, and spatial regions to latitude and longitude, we can determine whether an event occurs during an interval, or inside a circle around the point coordinates. More important for our IRL application, we can provide "smoothing": events that fall on the boundaries, or close to temporal intervals and spatial locations, are given values that fall off more rapidly according to the distance from the boundaries.

The immediate application of our work is to improve domain modeling for an inverse reinforcement learning algorithm that detects anomalous vehicle behavior. As such, our extensions must allow an easy way to define time intervals and spatial regions that can act as features in classifying track points. Features that affect normal (e.g., civilian) and anomalous (e.g., combatant) traffic flow are important to model and an iterative process will normally be required to handle local conditions (culture,

time, geography, changing road and urban conditions, temporary events such as road construction, etc). The controlled natural language extensions are intended to facilitate rapid experimentation.

The temporal concepts for time intervals act as a template for each temporal event occurrence (e.g., each Friday from 1 pm – 3 pm may be labeled "Friday Prayer"). Spatial concepts may also be categorized (park, city, town, public square, etc). Each named instance can be given a default size. Once defined, these spatial and temporal concepts may apply to thousands of time intervals or place instances that may need to be considered when evaluating track points for a city.

Temporal intervals and spatial regions need not be pegged to specific times or coordinates initially, when reasoning is not in service of the IRL algorithms. Instead, we may assert spatial relationships ("inside", "outside", "touching", "overlapping") between spatial regions and draw inferences with RCC-8. These inferences are somewhat limited, as the regions are represented as blobs. Similarly, we can assert temporal relationships between intervals (including events, as all have associated intervals), such as "before", "after", “during", "starts at", "ends at", "until", and "from___to___". If either spatial or temporal concepts are pegged to map coordinates or UTC times, then the concepts are further constrained and we can then also support proximity reasoning (e.g., to answer a question such as, "Was John near a gun when he murdered Sally?").

Fig. 9. Screenshot from IRL interface showing higher membership function values for track points on paths closer to a farm, corresponding to the spatial phrase "near Khalil's Farm"

A particularly interesting aspect of the definitions of both temporal and spatial concepts is the use of culturally-defined prototypes to help define vague concepts in a way that allows formal reasoning. The spatial and temporal terms, such as 'morning', or 'marketplace', are vaguely defined, along with related concepts such as 'in the morning' or 'in the marketplace', which can be true to varying degrees. In contrast, most logic-based controlled natural languages, with ACE being the prototypical example of such a CNL, emphasize precise context-free meanings. Terms mapping to first-order logic predicates are either wholly true or false for any domain object. CPL allows some measure of context-dependence [11] to handle limited *ambiguity* in resolving word meanings, but ultimately it, too, maps to logic. The WordNet terms can stand in for word meanings that are *vague* but there is no mechanism to formally reason with that vagueness, except as described in this paper. Instead, for vague words such as "peaceful", "beautiful", and "fair" the terms are treated as predicates that are wholly true or not of objects. The spatiotemporal reasoning presented in this paper only handles some of the aspects of vague concepts, and only related to times and places, yet it is a start. The use of a central-most point to stand in for the prototype of a time or place is admittedly a crude approximation of human cognitive prototypes, but it does have the advantage of supporting proximity measurements directly. Humans naturally think in terms of prototypes, so we have provided a *start* to more cognitively appropriate modeling within the confines of controlled natural languages. Human cognitive prototypes are, of course, *much richer*, e.g., they may include examples of good and bad Christmases, characteristics of an 'ideal' Christmas, and other cognitive frames, commercial and religious, that may be triggered when thinking of Christmas [4, 7].

So where are we now? We have currently implemented the proximity reasoning extensions for both spatial and temporal concepts and have provided an early integration with the IRL algorithm. The Google Earth screenshot in Fig. 9 shows an example of a membership function corresponding to the phrase "near Khalil's Farm". What appear to be large walls graphically depict the larger values of the membership function for track points that traverse those edges. Track points are shown as very small arrow-heads. Raised circles indicate vertices in the path network. Work is in progress to add relational reasoning capabilities for time and space, but not completed.

Machine learning is a promising application area for controlled natural language. The overwhelming amount of data increasingly provided to intelligence analysts requires highly automated means to process. Machine learning, data mining, link analysis, and social network algorithms can all play a role. However, the intelligence analyst must remain master of these tools, understand their results, and above all find a way to impart knowledge in a collaborative effort to leverage the tools' capabilities. In the IRL example, the analysts provide knowledge of what temporal and spatial concepts best explain normal vehicle traffic. They also provide culturally-specific knowledge, such as customary times and places for meals and religious observances. Controlled natural language plays the role of simplifying this man-machine interface. We hope our research will illustrate the potential synergy between machine learning and controlled natural language, and encourage similar fruitful interactions between the two fields.

Acknowledgements. We would like to thank the reviewers for their advice, Pete Clark for detail on CPL's handling of situations, and Dragos Margineantu for supporting this research.

References

1. Clark, P., Harrison, P., Jenkins, T., Thompson, J., Wojcik, R.: Acquiring and Using World Knowledge using a Restricted Subset of English. In: The 18th International FLAIRS Conference, FLAIRS 2005 (2005)
2. Ng, A., Russell, S.: Algorithms for inverse reinforcement learning. In: Proceedings of ICML (2000)
3. Singliar, T., Marginenantu, D.: Scaling up Inverse Reinforcement Learning through Instructed Feature Construction. Snowbird Learning Workshop (2011), http://snowbird.djvuzone.org/2011/abstracts/132.pdf
4. Kövecses, Z.: Metaphor: A Practical Introduction. Oxford University Press, USA (2002)
5. Kövecses, Z.: Metaphor in Culture: Universality and Variation. Cambridge University Press (2005)
6. Kövecses, Z.: Language, Mind, and Culture: A Practical Introduction. Oxford University Press, USA (2006)
7. Lakoff, G.: Women, Fire, and Dangerous Things. University of Chicago Press (1990)
8. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English – Not Just Another Logic Specification Language. In: Flener, P. (ed.) LOPSTR 1998. LNCS, vol. 1559, pp. 1–20. Springer, Heidelberg (1999)
9. Engelbrecht, P., Hart, G., Dolbear, C.: Talking Rabbit: A User Evaluation of Sentence Production. In: Fuchs, N.E. (ed.) CNL 2009. LNCS (LNAI), vol. 5972, pp. 56–64. Springer, Heidelberg (2010)
10. Pease, A., Murray, W.R.: An English To Logic Translator For Ontology-Based Knowledge Representation Languages. In: Proceedings of 2003 IEEE International Conference on Natural Language Processing and Knowledge Engineering, NLPKE 2003 (2003)
11. Clark, P., Murray, W.R., Harrison, P., Thompson, J.: Naturalness vs. Predictability: A Key Debate in Controlled Languages. In: Fuchs, N.E. (ed.) CNL 2009. LNCS (LNAI), vol. 5972, pp. 65–81. Springer, Heidelberg (2010)
12. Schwitter, R., Tilbrook, M.: PENG: Processable English. Technical report, Macquarie University, Australia (2004)
13. Schwitter, R.: Controlled Natural Languages for Knowledge Representation. In: COLING 2010, pp. 1113–1121 (2010)
14. Clark, P., Porter, B.: KM - The Knowledge Machine 2.0: Users Manual, http://www.cs.utexas.edu/~mfkb/km.html
15. Clark, P., Porter, B.: Situations Manual. Note: The Situations Manual is an adjunct manual for the KM Manual. Both can be obtained from, http://www.cs.utexas.edu/users/mfkb/km.html
16. Otter. William McCune, http://www.cs.unm.edu/~mccune/otter/
17. Prover9. William McCune. For Prover9, http://www.cs.unm.edu/~mccune/mace4/, http://www.cs.unm.edu/~mccune/otter/
18. WordNet: An Electronic Lexical Database (Language, Speech, and Communication) by Christiane Fellbaum (Hardcover - May 15, 1998)

19. Lakoff, G., Johnson, M.: Metaphors We Live By, 2nd edn. University of Chicago Press (1980)
20. Wikipedia. Coordinated Universal Time, http://en.wikipedia.org/wiki/Coordinated_Universal_Time (retrieved April 11, 2012)
21. Wikipedia. Allen's Interval Algebra, http://en.wikipedia.org/wiki/Allen%27s_interval_algebra (retrieved April 10, 2012)
22. Wikipedia. Region Connection Calculus, http://en.wikipedia.org/wiki/Region_connection_calculus (retrieved April 10, 2012)
23. Wikipedia. Professorville, http://en.wikipedia.org/wiki/Professorville (retrieved April 11, 2012)
24. Harrison, P.: A New Algorithm for Parsing Generalized Phrase Structure Grammar. Ph.D. dissertation, University of Washington, Seattle (1988)
25. Barker, K., Porter, B., Clark, P.: A Library of Generic Concepts for Composing Knowledge Bases. In: Proc. 1st Int. Conf. on Knowledge Capture, K-Cap 2001 (2001)

Controlled Natural Language in Speech Recognition Based User Interfaces

Kaarel Kaljurand[1] and Tanel Alumäe[2]

[1] Institute of Computational Linguistics, University of Zurich, Switzerland
kaljurand@gmail.com
[2] Institute of Cybernetics, Tallinn University of Technology, Estonia
tanel.alumae@phon.ioc.ee

Abstract. In this paper we discuss how controlled natural language can be used in speech recognition based user interfaces. We have implemented a set of Estonian speech recognition grammars, a speech recognition server with support for grammar-based speech recognition, an Android app that mediates the communication between end-user Android apps and the speech recognition server, and an end-user Android app that lets the user execute various commands and queries via Estonian speech. The overall architecture is open and modular, offers high precision speech recognition, and greatly simplifies the building of mobile apps with a speech-based user interface. Although our system and resources were developed with the Estonian speaker in mind and currently target a small number of domains, our results are largely language and domain independent.

Keywords: grammar-based speech recognition, user interfaces, controlled natural language, Grammatical Framework, Estonian.

1 Introduction

Speech recognition based user interfaces can be effective in many environments. The only requirements are the lack of major background noise and privacy/security concerns that one might have when audibly communicating with the machine. In many domains such user interfaces provide the most efficient form of human-machine communication because alternative interfaces (e.g. keyboard, visual menu system, etc.) are not available or are cumbersome to use. For example, when driving a car, one's eyes and hands are occupied with driving while the speech faculty is freely available. Recently a general application platform for speech based user interfaces has emerged in the form of mobile devices (smartphones and tablets). Such devices feature a small display and a small (and often only virtual) keyboard. Many types of otherwise simple tasks (looking up a phone number, launching an application, setting an alarm, etc.) can become time consuming and cumbersome if performed on a mobile device. Performing such tasks via speech can solve this problem, and this has already been demonstrated by applications like Apple's Siri and Google's Voice Actions, which have become quite popular among users [6].

A controlled natural language (CNL) for speech recognition is a CNL like any other — it has a precisely defined syntax, its sentences have a formal (executable) meaning, and it comes with an end-user documentation describing its syntax, semantics and usage

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 79–94, 2012.

patterns. A CNL for spoken input differs from a CNL for written input because its design has to take into consideration the properties of speech: there are words which sound the same, or can sound the same if distorted by background noise, some character sequences cannot be (directly/reliably) pronounced (numbers, punctuation), utterances are syntactically simpler, etc.

Many applications naturally feature grammatically and vocabulary-wise reduced "language", examples include a calculator, a measurement unit converter, a car navigation system, an address book browser, an alarm clock, a flight booking website's form filling interface. Basing the interaction with such applications on a CNL offers two benefits — (1) the recognition of user input works in a precise way, (2) it becomes much easier for the application developer to map the user input to the application's executable code (e.g. the formal expression that actually sets the time on the alarm clock).

In this paper we describe a set of Estonian speech recognition grammars that cover the language of some important tasks that one normally performs on a mobile device. The grammars have been implemented in Grammatical Framework (GF) [9] making them easily extendable and portable to other languages and application domains. We also describe the overall architecture of our speech recognition system which combines an online speech recognition server, two Android-based apps and a repository of grammars. This modular architecture simplifies the building of natural and easy to use interfaces to applications, potentially ranging from a simple alarm clock to in-car navigation systems and control panels for the "smart house".

In section 2 we review related work; in section 3 we discuss the special features demanded by speech recognition oriented CNLs; in section 4 we briefly introduce GF; in section 5 we provide an overview of the grammars that we have developed; in section 6 we describe the architecture of the overall speech recognition system; in section 7 we summarize our main results; and in section 8 we mention some loose ends and general future work.

2 Related Work

Speech applications are a well-studied field which also includes grammar-based and even GF-based applications. The use of GF in speech applications has been studied in [4] and resulted in multi-modal systems which combine speech input with input from the touch screen. These systems can also hold a dialog where the eventual result is obtained after a sequence of conversation turns. In addition these systems are multilingual allowing input/output in several natural languages. The concrete applications include ordering a pizza (with various toppings and drinks) and asking for directions in a city public transportation system. GF is used as a framework for describing the semantic model of the system and its various expressions in natural and formal languages (e.g. image of the pizza with the specified toppings). Compared to this work, our current system is simpler in several dimensions (single language, single modality, no dialog), we have rather focused on developing a general platform on which more sophisticated applications can be easily built.

An architecture similar to ours is developed in [13]. It consists of a mobile application for calendar events management by English speech, a Nuance[1] speech recognition server, and a Regulus grammar [11]. In addition to grammar-based speech recognition, the system also uses a statistical recognizer that is applied to out-of-grammar input with an eventual goal of guiding the user towards supported phrases. The built-in help system makes the architecture more sophisticated than ours, but the overall system is much harder to deploy as it relies on commercial closed-source software (Nuance, SICStus Prolog). We did not explore if the system would be portable to Estonian and to a free/open platform.

In general, the possibility of speech input is available on all the major smartphone platforms (Android, iOS, Windows Phone) via end-user applications such as Google Voice Actions and Siri. However these applications do not support user-specified speech recognition grammars and offer only a limited API and configurability. They are also closed source and thus not portable to other domains and languages by third parties.

Computational approaches to the Estonian language have so far focused on large coverage shallow parsing, detailed description of morphology and word senses (WordNet), statistical methods for machine translation and speech recognition, etc. [7]. For the most part, the developed resources are not directly reusable for building controlled Estonian grammars. The existing tools for morphological synthesis could be used in principle to automatically generate wordforms for the grammars, but for the current system we decided not to integrate these tools and generate the required wordforms with *ad hoc* rules formulated using GF's regular expressions.

3 Requirements

Spoken language differs from written language in various ways which must be reflected in the design of speech recognition oriented CNLs. Certain written forms and orthographic conventions are not reflected in speech. For example:

- homophones (e.g. 'cite', 'site', 'sight') cannot be distinguished with only acoustic cues (in Estonian, homophones occur rarely, e.g. in the case of word-initial plosives, e.g. 'baar' (*bar*) and 'paar' (*pair*) are pronounced in the same way);
- there is no necessary break between words as there is in written speech, creating confusion pairs ("oronyms"), such as 'ice cream' vs. 'I scream', 'depend' vs. 'deep end';
- numbers, abbreviations, URLs are expanded to words (e.g. '3G' $\mapsto$ 'three gee'); often, the expansion is non-deterministic (e.g. '1990' can be verbalized as 'nineteen ninety', 'one thousand nine hundred and ninety' or 'nineteen hundred and ninety');
- punctuation symbols are typically not verbalized.

CNLs in speech recognition applications should avoid short or unpronounceable words, and avoid similarly sounding words that occur in the same syntactic position. These CNLs cannot rely on technologies like look-ahead editing [12] to guide the user

[1] `http://nuance.com/`

towards the completion of syntactically correct sentences. The sentences must therefore be shorter and syntactically simpler. Also, because many existing speech recognition engines rely on finite-state automata technology, the CNL might have to be expressible by a regular grammar.

When choosing the formalism to implement our grammars we found the following requirements important:

- support for a declarative description of how the raw speech transcription (e.g. 'half past six in the evening') maps to the formal application language ('18:30');
- built-in handling of the complexities of natural language, e.g. the Estonian morphology;
- user-friendly syntax and semantics and/or editing environments which allow also novice grammar engineers to write well-functioning grammars;
- compatibility with open source speech recognition toolkits;
- support for standard software engineering practices (reusable modules, unit and regression testing, etc.).

Existing standard speech recognition grammar formalisms (such as SRGS[2] and JSGF[3]) do not cover all these requirements. They are usually simple BNF languages without any special support for natural languages nor the capability to assign a formal meaning to the language described by the grammar.

4 Grammatical Framework

In order to implement our CNLs we chose Grammatical Framework (GF) [9] because it covers our requirements and has been successfully used before to build speech applications [4]. GF is a functional programming language for grammar engineering. The grammar author implements an *abstract syntax* and its corresponding *concrete syntax* and by doing that describes a mapping between language strings and their corresponding abstract syntax trees. As this mapping is bidirectional — strings can be *parsed* to trees and trees *linearized* to strings — this architecture supports multilinguality. There can exist multiple concrete syntaxes corresponding to a single abstract syntax, and the respective languages can be automatically *translated* from one to the other.

GF is optimized to handle natural language features like morphological variation, agreement, long-distance dependencies, etc. In terms of the expressivity of the grammar formalism GF covers context free languages and even goes beyond. GF grammars can be implemented in a modular fashion and tested by random or exhaustive generation of abstract trees and their linearizations. GF comes with various tools that cover grammar authoring, compatibility with many popular programming languages, conversion into other grammar formats (incl. several speech recognition formats), and a reusable grammar library for $\sim$25 natural languages (unfortunately excluding Estonian at the time of writing).

[2] `http://www.w3.org/TR/speech-grammar/`
[3] `http://www.w3.org/TR/jsgf/`

5 Grammars

We have developed several grammars targeting the general topics of time, location and numerical calculation. A GF grammar is a set of concrete language definitions with their corresponding single abstract language definition. In our more specialized sense each grammar contains at least two types of concrete syntaxes, which we refer to as *parsing syntax* and *linearization syntax*. The parsing syntax

- is used for parsing;
- directly corresponds to natural language speech, e.g. it uses words ('two', 'plus'), not symbols ('2', '+');
- allows ambiguous input (i.e. produces multiple parse trees);
- allows variants (synonyms).

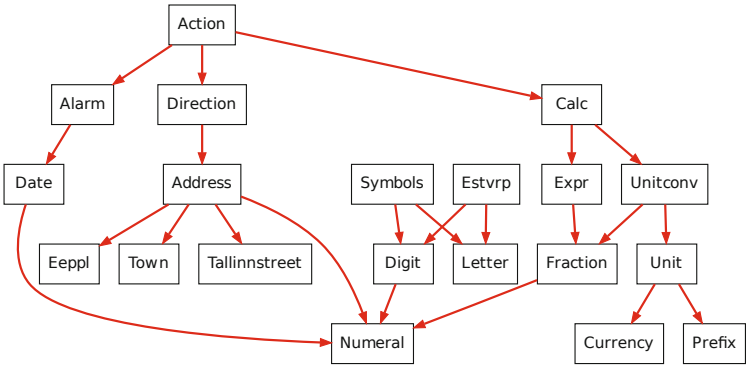


Fig. 1. The hierarchy of abstract grammar modules. The largest grammar is *Action*, which covers arithmetical, unit conversion, alarm setting and address query expressions. The most imported grammar is *Numeral* which is a building block in all the mentioned expressions.

The linearization syntax on the other hand

- is used for generation;
- typically corresponds to a machine language (where symbols are not necessarily pronounceable);
- does not require the presence of syntactic sugar.

The meaning of the speech input is assigned by parsing it with the parsing syntax and then linearizing the obtained abstract tree with the linearization syntax. A grammar can be easily extended by adding a new concrete language that provides the linearization of all the functions already described in the abstract syntax. The new concrete language can either stand for a parsing syntax (i.e. support for a new natural language) or the linearization syntax (i.e. support for a new machine format).

In our case, the parsing syntax of our grammars corresponds to Estonian speech and the linerization syntax targets the input language of an existing application such as

standard calculator, Google Maps, or WolframAlpha. Note however that in most cases the actual language supported by these tools is not publicly and precisely documented, nor can it be considered a stable API. We next provide a more detailed look into the main grammars.

GF offers various support for modularity, including an extension relation between grammars, which we use to share the implementation of numbers and to build union grammars (see figure 1).

5.1 Direction

The *Direction* grammar describes Estonian places by containing a list of settlement names, a list of street names, positive integers to stand for house numbers and a phrase for combining two place names in a directions query. As there are many place names this grammar contains a large number of terminals, but is syntactically very simple, covering the patterns

```
Placename = Streetname Housenumber (Town) (Country)
Placename = Town (Country)
Direction = From Placename To Placename
```

where the optional `Town` and `Country` can be mapped to some reasonable defaults (e.g. 'Tallinn' and 'Estonia') if they are missing. All the place names are in the nominative case.

The linearization syntax targets the language understood by Google Maps. For the most part it is identical to the parsing syntax, i.e. it contains the same terminals. The only difference are the *from* and *to* phrases. The following examples list the input utterance in Estonian, its corresponding machine format, and its English translation (for the purposes of this paper).

Example 1.
algus akadeemia tee kaks kümmend üks lõpp räpina
`FROM Akadeemia tee 21, Tallinn TO Räpina, Estonia`
begin Akadeemia street 21 end Räpina

The *Direction* grammar is compiled on the basis of two freely available lists of Estonian place names:

- settlements in Estonia (towns, villages, etc.), 4300 names from GeoNames[4];
- names of streets in Tallinn, 1500 names from the place names' resource of the Institute of the Estonian Language[5].

At the moment the grammar does not model naming variation and ambiguity, i.e. bijection holds between the set of places (abstract functions) and the set of place names (their linearizations). This has the consequence that disambiguation must be performed by an external application (e.g. Google Maps).

[4] `http://www.geonames.org/`
[5] `http://www.eki.ee/knab/`

The grammar currently lacks the street names of Estonian towns other than Tallinn, Estonian names of foreign places (e.g. 'Venemaa', 'Riia'), and names of places which are not towns nor buildings with a street number (e.g. parks, landmarks). It should be noted that the grammar does not include a statistical component that would e.g. assign a lower prior probability to smaller places which are rarely visited (e.g. a short street with just one house). Also it allows house numbers up to 999 with any street name, i.e. this grammar cannot be used in applications that use exhaustive generation or look-ahead editing because it can generate addresses which do not exist.

5.2 Expr

The arithmetical expression grammar *Expr* describes positive integers up to 10^{12} (following the abstract syntax of the GF numerals grammar [5]), their negative counterparts and their combinations with a dot (to cover some of the rational numbers). The numbers can be combined with the standard arithmetical operators of addition, subtraction, multiplication, division, and exponentiation in order to form (possibly infinitely long) sentences. Our implementation follows the example provided in [9], but only the left-associative interpretation is supported and all operators are considered to have equal precedence.

Example 2.
kaks pluss kolm miinus miinus neli korda viis jagatud kuus astmel seitse koma sada
`(((((2 + 3) - (-4)) * 5) / 6) ^ 7.100)`
two plus three minus minus four times five divided-by six to-the-power-of seven point hundred.

The strings that correspond to the formal expressions can be directly evaluated with any standard calculator, e.g. the one included in Google Search.

5.3 Unitconv

The unit conversion grammar *Unitconv* includes the same numbers as the *Expr* grammar, ∼50 base measurement units (of ∼10 physical quantities), and ∼15 world currencies. The grammar describes how the base units form more complex units via

- SI prefixing: 'meeter' (`m`) → 'kilo meeter' (`km`),
- exponentiation: 'senti meeter' (`cm`) → 'kuup senti meeter' (`cm^3`), and
- fractions: 'kilo meeter' (`km`) and 'tund' (`h`) → 'kilo meetrit tunnis' (`km/h`).

All these constructors take into account the types of the input components (e.g. length) and can thus generate the correct complex unit (e.g. volume). A number and two units can be combined into a unit conversion expression as shown in the examples.

Example 3.
viis koma kaks meetrit jalgades
`convert 5.2 m to ft`
five point two meters in feet

Example 4.
viis miili ruut tunnis meetrites ruut sekundis
`convert 5 mi*h^-2 to m*s^-2`
five miles per square hour in meters per square second

Example 5.
viis ameerika dollarit rootsi rahas
`convert 5 USD to SEK`
five American dollars in Swedish money

The grammar supports some variation and ambiguity, e.g. 'kroon' (*crown*) refers to multiple currencies (SEK, NOK, DKK, ...), unless disambiguated by e.g. 'rootsi kroon' (*Swedish crown*) or (if the user does not know the name of the currency) 'rootsi raha' (*Swedish money*).

The nouns corresponding to units can potentially have 3 different inflectional endings depending on their role in the sentence. The morphological forms are calculated automatically from the partitive base form, making the extending of the grammar with new units almost as simple as extending the *Direction* grammar with new place names.

The linearization syntax targets the unit conversion syntax supported by Google Search and WolframAlpha, i.e. it consists of English phrases ('convert') and ASCII conventions for writing formulas (`m*s^-2`).

5.4 Alarm

The *Alarm* grammar targets one of the most widely performed tasks on mobile devices — setting alarms. The grammar supports pointing to an exact minute in a 24h clock (when the alarm should go off) and specifying a duration by a positive integer number of minutes (after which the alarm should go off).

Example 6.
ärata mind kell seitse null üks
`alarm 07:01`
wake me at seven oh one

Example 7.
ärata mind kaheksateist minutit hiljem
`alarm in 18 minutes`
wake me eighteen minutes later

The linearization grammar targets the language understood by some of the "intelligent assistant" apps found on Google Play[6], notably Speaktoit Assistant[7].

5.5 Estvrp and Symbols

The *Estvrp* (Estonian vehicle registration plate) grammar lets one spell the Estonian car registration numbers, which typically consist of three letters followed by three digits

[6] `http://play.google.com/store`
[7] `http://www.speaktoit.com/`

(e.g. ‘ABC123’). As the speech recognizer discriminates short sounds (e.g. /oo/ vs. /uu/) very unreliably, the grammar denotes letters by a set of selected longer proper names (e.g. ‘Artur’ for ‘A’).

There exists also a *Symbols* grammar that describes an arbitrary length sequence composed of letters (*Letter*) and digits (*Digit*).

5.6 Ambiguity

In general, the output of our grammar-based speech recognizer is not a single string but a set of strings. This is because our grammars not only accept/reject the given speech input, but also translate it to another concrete format. For example, the recognized speech might result in a string ‘pii’ which is further mapped to (= is ambiguous between) three strings ‘π’, ‘Pii village, Estonia’, ‘Pii street, Tallinn’. One way to deal with such ambiguity is to make sure that the grammar allows for alternative (synonymous) forms which lack the ambiguity, e.g. ‘pii küla’ (*Pii village*), ‘pii tänav’ (*Pii street*), ‘arv pii’ (*the number* π). Support for variation is also generally good because usually there exist many equally probable ways for saying a command or query, even in the case of simple applications like calculators.

Ambiguity can also be exploited in various interesting ways. For example, a grammar that describes currency conversion can decide to include the string ‘european currency’ and implement it as ambiguous between ‘EUR’, ‘CHF’, ‘SEK’, etc. resulting in multiple translations for the the expression ‘convert one dollar to european currency’. The end-user application can either ask the user to clarify which exact currency was meant or alternatively (and more interestingly) visualize the result set as a comparison table or plot.

6 Overall Architecture

We next describe the implementation of the client-server architecture of our grammar-based real-time mobile-oriented speech recognition system (see figure 2).

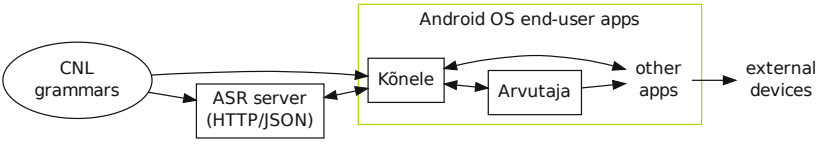


Fig. 2. The overall system consists of an online repository of CNL grammars, an automatic speech recognition (ASR) server that is accessible over HTTP, an Android service *Kõnele* that mediates the communication of end-user apps with the recognition server, and various end-user apps that offer a speech-based UI. Some of such apps (e.g. *Arvutaja*) act as hubs that dispatch incoming speech commands/queries to other apps. Finally, the commands can be carried out on something external to the mobile device (e.g. when opening a garage door by a voice command).

6.1 Speech Recognition Server

Our speech recognition server [2] offers real-time transcription of short (up to 20 seconds) Estonian speech signals. The server is designed to respond with at most a couple of seconds delay after input speech has stopped independent of the length of the input. The server is thus usable in applications where the transcription must be obtained immediately such as web search or dictation of an SMS message.

The speech recognition server is based on various open-source technologies. The core of the system is the Pocketsphinx decoder of the CMU Sphinx speech recognition toolkit[8]. We selected this decoder as opposed to other freely available recognition engines because we have found it to be accurate, fast and have a relatively small memory footprint. The decoder is integrated with other parts of our server via its GStreamer interface. The main request handling and management code is written in the Ruby programming language. The source code of the server is available under the BSD-license[9].

The Estonian acoustic models[10] for the speech recognition service were trained on various wideband Estonian speech corpora: the BABEL speech database, a corpus of Estonian broadcast news, a corpus of broadcast conversations, a corpus of lecture and conference recordings, and a corpus of spontaneous dialogs, totaling in around 90 hours. The models are triphone HMMs, using MLLT/LDA-transformed MFCC features, with 3000 tied states, each modeled with 32 Gaussians. The model inventory consists of 25 phonemes and 7 silence/filler models. Cepstral mean normalization was applied.

Speech recognition engines usually rely on statistical (e.g. trigram) language models. Such models are appropriate for many applications of speech technology (dictation of letters, transcription of meetings). By default, our server uses a pre-built trigram model for decoding incoming requests. However, grammar-based decoding can be invoked by specifying the name of the grammar in the request parameters.

Our server allows uploading of grammars in GF's portable runtime format (PGF)[3]. When a new grammar is uploaded, several steps are automatically invoked on the server. First, the concrete syntax of the input language (Estonian) is extracted from the PGF and converted to the JSGF format[11]. The JSGF grammar is further converted to a finite-state automaton that can be used by the speech recognition engine. One of the shortcomings of the current implementation is that the resulting finite-state automaton is a regular approximation of the GF concrete syntax and allows certain inputs that the original GF grammar does not actually allow.

The concrete syntax of the input language is assumed to contain orthographically correct words. This allows the server to automatically build a pronunciation dictionary (mapping of words to phoneme sequences) for all the words in the input language. Since Estonian is almost a phonetic language, we can generate pronunciations using a simple transducer, with a list of exceptions for common foreign names [1]. A more flexible architecture would allow the user to provide her own pronunciation dictionary as well as the acoustic models that define the phonemes.

[8] http://cmusphinx.org
[9] http://github.com/alumae/ruby-pocketsphinx-server
[10] http://github.com/alumae/et-pocketsphinx-tutorial
[11] http://www.w3.org/TR/jsgf/

The server has an HTTP interface designed to be similar to the (publicly undocumented) Google's speech recognition server which is used by the Chrome browser to implement the W3C Speech Input API. When a recognition request is made to the server, the name of the PGF grammar can be specified in the request parameters to activate grammar-based recognition. The server uses the regular approximation of the input language concrete syntax of the PGF to decode the audio signal. The resulting recognition hypotheses can be automatically translated to the output language(s) specified in the request. The server is able to return an N-best list of recognition hypotheses for each request. The size of the N-best list can be specified with a request parameter.

The results are returned in the JSON format. The example below shows a transcription of the utterance "mine neli meetrit edasi" ("go four meters forward") and its translation into three concrete languages *Eng*, *Est*, *App*. In this case the hypotheses set contains just a single element.

```
{ "status": 0,
  "hypotheses": [
    { "linearizations": [
        { "lang": "App", "output": "4 m >" },
        { "lang": "Eng", "output": "go four meters forward" },
        { "lang": "Est", "output": "mine neli meetrit edasi" }
      ],
      "utterance": "mine neli meetrit edasi"
    }
  ],
  "id": "d9abdbc2a7669752059ad544d3ba14f7"
}
```

The request fails if the server fails to match the audio signal to the grammar (e.g. because the signal is too noisy or quiet), or if the server recognizes it due to the regular approximation as something which is not covered by the original grammar. Both types of failures are exposed to the user in the same way, and can usually be overcome by repeating the input phrase in a clearer voice.

To our knowledge, the described recognition server is the first free, open source and publicly available web service in the world that supports speech recognition with user-defined grammars.

6.2 Android Service *Kõnele*

A wide variety of Android apps support input by speech, typically having a small microphone button as part of their user interface, e.g. keyboard apps, apps with a search bar, apps for in-car use, Siri-like intelligent assistant apps. The Android operating system offers an API[12] (the *RecognizerIntent* and *RecognitionService* classes) that lets such end-user apps call a central service that performs the speech recognition and returns the transcription. In this way the multitude of speech-enabled apps can share the speech recognition provider and do not have to implement this functionality themselves.

We developed an Android app *Kõnele*[13] ('kõnele' is the imperative form of 'to speak' in Estonian) which offers such a speech recognition service to other apps on the device.

[12] `http://developer.android.com/reference/android/speech/package-summary.html`

[13] `http://recognizer-intent.googlecode.com`

In addition to being a background service, *Kõnele* also offers a configuration panel which allows the end-users to assign different grammars to different apps by specifying the URL of the grammar and the Java package name of the app. If the *Kõnele* service is called in the context of an app for which there exists a grammar assignment then *Kõnele* sends the grammar URL to the server along with the audio data, which triggers grammar-based speech recognition and translation into the format of the app.

6.3 Android App *Arvutaja*

Arvutaja[14] (Estonian for 'the one who calculates') is a tool that helps the user to evaluate arithmetical and unit conversion expressions, query for directions between Estonian addresses, and set the alarm clock or timer. Commands to *Arvutaja* are given in Estonian speech which is transcribed via *Kõnele*, guided by the union of *Expr*, *Unitconv*, *Direction* and *Alarm* grammars. The transcription is evaluated/executed and displayed to the user by a built-in library (for unit conversion and arithmetical expressions) and/or by an external app (for displaying a route on the map between the given locations or setting the alarm to ring at the given time). In case of ambiguous queries, the user is presented with all the interpretations. Figure 3 shows a screenshot of *Arvutaja*.

The Android framework supports a messaging mechanism called *intents*[15] that lets apps call each other with the purpose of "outsourcing" operations (performing a calculation, displaying a map, setting an alarm). *Arvutaja* uses the intent mechanism to call Google Maps, WolframAlpha, and an alarm clock app for certain tasks. Such an architecture where the user input is directly mapped to an external application to handle this input makes *Arvutaja* defined almost entirely by the externally developed grammars and apps, and thus easily extendable to completely new languages and domains.

6.4 Grammars

An important position in this overall architecture is held by the repository of CNL grammars. This repository makes available the grammars so that they can be accessed both by the speech recognition server and by *Kõnele*. It also facilitates learning to use the grammars and (collaboratively) building new grammars.

Our GF grammars are compiled into the PGF platform independent format and made available via public URLs on GitHub[16], see `http://kaljurand.github.com/Grammars/` As all the grammars are available in the source format, GitHub can also be used as a collaborative editing environment where the users can easily fork a grammar, extend it and contribute back the modifications. In order to familiarize themselves with the language that the grammars support, the users can look at help documentation and lists of (automatically generated) example sentences. A more sophisticated infrastructure (which is currently not implemented) would also include existing GF tools [10], e.g. Minibar, Translation Quiz, and the online grammar editor[17].

[14] `http://kaljurand.github.com/Arvutaja/`
[15] `http://developer.android.com/guide/topics/intents/intents-filters.html`
[16] `http://github.com/`
[17] `http://cloud.grammaticalframework.org/`

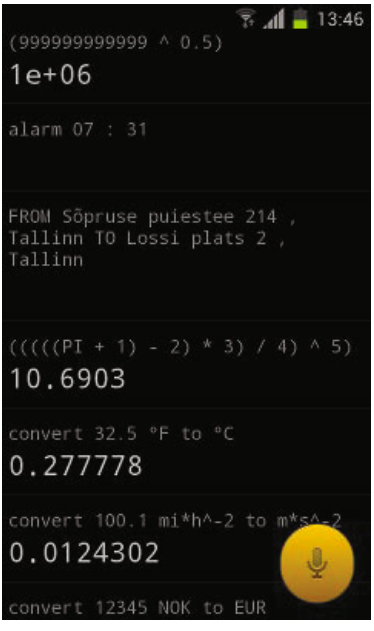


Fig. 3. Screenshot of *Arvutaja*. The user interface prominently includes a microphone button, tapping on which triggers the recording and transcribing of the spoken query. The results are presented in a history list. In some cases the transcription is accompanied by its evaluation. In some cases (e.g. address query), the user needs to tap on the transcription to execute it and view the result (a map) via an external application (Google Maps). Such a user interface is at least on a visual level much simpler than e.g. a typical unit converter interface which usually features a complex menu system for selecting the involved units.

In general, such a repository should make grammar creation easy for everybody. We envision that there can be many different grammar usage scenarios: private grammars used internally in a company, different subset grammars (e.g. address grammars optimized for one country or one town), new grammars with tiny variations with respect to existing grammars (e.g. to account for the variation in an intelligent assistant language skills). It is thus important to make existing grammars reusable and make the creation of new grammars simple.

7 Results

The main contribution of our work is an implementation of an open and extendable speech application architecture which supports controlled natural language based speech interfaces, as well as a set of grammars covering a diverse set of domains that can serve as building blocks and examples for future grammars. We have applied the developed stack and grammars to Estonian grammar-based speech recognition and believe that it enables engineers with limited knowledge of speech processing and grammar engineering to quickly build speech-enabled user interfaces.

Flexible Solution for Application Builders. The described architecture makes it easy to port existing speech recognition based UIs to new languages. For example, the Android app Speaktoit Assistant has been developed with only English speakers in mind. With no change to the original app, it can be made to accept another language. One only needs to implement a grammar that maps the commands in the new language to the format required by the application, and add a new acoustic model to the speech recognition server (if it does not already exist). Using the Android's intent mechanism, it is also possible to easily add a speech-based UI to an app that did not have it before.

Improved Accuracy over Statistical Language Models. We compared the accuracy of the grammar-based recognizer with the accuracy of a wide coverage recognizer whose statistical language model is trained over mainly news texts and uses a lexicon of 200,000 non-compound wordforms. The error rate of using Google Maps with the *Direction* grammar was much lower than when using it with free-form input. The experiment was based on 100 audio recordings made by two speakers with a smartphone in a room environment. Each recording contained a second-long utterance of a Tallinn address in the form of *street-name house-number*. These addresses were selected randomly from a large set of Tallinn street addresses scraped from the web. The grammar-based recognizer transcribed 90% of the recordings correctly, while the free-form recognizer achieved only 60% correct results. It should be noted that the free-form recognizer used a general language model and a much larger vocabulary (which might give preference to words which are frequent in Estonian, but not necessarily in Estonian street addresses).

Scalability to a Large Number of Terminals. The *Direction* grammar also demonstrates that our speech recognition architecture can handle a large number of terminals without any slowdown in processing nor major negative effect on precision. The $\sim$6000 place names in the grammar cover most of the geographical locations in Estonia, i.e. such a grammar can be used as a component in a car navigation system developed for speakers of small languages.

Uptake of Grammar-Based Solutions. After six months on Google Play, *Arvutaja* has proved to be a relatively popular Android app in Estonia with an install base of $\sim$1300 users and $\sim$19,000 queries since the initial launch. This makes it also the most frequent client application of *Kõnele* whose install base is 3700.

8 Future Work

There is a number of extensions to the described work which we want to explore in the future. It would be useful to combine controlled with free-form input, e.g. for utterances like "text Bob I'm running late", where only the first two words are controlled and mapped to a structured format, while the last part could be recognized using a statistical model and returned as an uninterpreted string.

In order to make grammar creation accessible to casual users, it must be made really simple. In some cases, it can also be completely automated, e.g. turning one's contacts list (which is usually available on the mobile device) into a grammar and making

it available in speech applications via commands and queries like “call mom’s work number”, “driving directions to Mati’s home”.

Our current grammars are limited to being regular in expressivity because our speech recognition engine processes them with finite-state technology. We plan to investigate what are the most compelling reasons for using full GF expressivity in the context of speech recognition and intend to integrate GF better into our speech recognition engine. In order to obtain higher transcription precision with larger grammars we want to look into probabilistic GF grammars.

Providing a full set of “smart paradigms” for Estonian (see [8] for Finnish) would simplify the work of grammar developers, who then would not need to know the details of Estonian morphology. Also it would simplify the introduction of more variation into the grammars, e.g. ‘viis jagatud kuus’ and ‘viis jagatud kuuega’ are equally likely expressions of $5/6$, but the latter is currently not supported because it contains a non-nominative case ending.

Sometimes the application response is structurally more complex than a simple “setting of an alarm”, e.g. an address query can result in a set of driving directions. In this case it can be useful to present the results in natural language and even in the form of speech. GF could again help to translate the application format into a format suitable to the speech synthesizer. The speech synthesizer input, in this case, is not necessarily an orthographic text but could contain the information that is available in speech, e.g. palatalization and vowel/consonant quantity degree in case of Estonian.

The recognition server query logs provide valuable data about the current usage of our Android apps. Analyzing *Arvutaja* queries for the most common causes of out-of-grammar failure would let us fine-tune the grammars.

Acknowledgments. This research was supported by the Estonian Ministry of Education and Research target-financed research theme no. 0140007s12.

References

1. Alumäe, T.: Methods for Estonian large vocabulary speech recognition. PhD thesis, Tallinn University of Technology (2006)
2. Alumäe, T., Kaljurand, K.: Open and extendable speech recognition application architecture for mobile environments. In: The Third International Workshop on Spoken Languages Technologies for Under-resourced Languages (SLTU 2012), Cape Town, South Africa (2012)
3. Angelov, K., Bringert, B., Ranta, A.: PGF: A Portable Run-time Format for Type-theoretical Grammars. Journal of Logic, Language and Information 19(2), 201–228 (2010), doi:10.1007/s10849-009-9112-y
4. Bringert, B.: Speech Recognition Grammar Compilation in Grammatical Framework. In: SPEECHGRAM 2007: ACL Workshop on Grammar-Based Approaches to Spoken Language Processing, Prague, June 29 (2007)
5. Hammarström, H., Ranta, A.: Cardinal Numerals Revisited in GF. In: Workshop on Numerals in the World’s Languages, Dept. of Linguistics, Max Planck Institute for Evolutionary Anthropology, Leipzig (2004)
6. Meisel, W.: Life on-the-Go": The Role of Speech Technology in Mobile Applications. In: Neustein, A. (ed.) Advances in Speech Recognition, pp. 3–18. Springer, US (2010)

7. Meister, E., Vilo, J.: Strengthening the Estonian Language Technology. In: Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008), Marrakech, Morocco (2008)
8. Ranta, A.: How predictable is Finnish morphology? An experiment on lexicon construction. In: Nivre, J., Dahllöf, M., Megyesi, B. (eds.) Resourceful Language Technology: Festschrift in Honor of Anna Såagvall Hein, pp. 130–148. University of Uppsala (2008)
9. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011) ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth)
10. Ranta, A., Angelov, K., Hallgren, T.: Tools for multilingual grammar-based translation on the web. In: Proceedings of the ACL 2010 System Demonstrations, Uppsala, Sweden, pp. 66–71. Association for Computational Linguistics (July 2010)
11. Rayner, M., Hockey, B.A., Bouillon, P.: Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler. CSLI Publications, Stanford (2006)
12. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE—A Look-ahead Editor for a Controlled Language. In: Proceedings of EAMT-CLAW 2003, Joint Conference combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop on Controlled Translation, May 15–17, pp. 141–150. Dublin City University, Ireland (2003)
13. Tsourakis, N., Georgescul, M., Bouillon, P., Rayner, M.: Building Mobile Spoken Dialogue Applications Using Regulus. In: Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008), Marrakech, Morocco (2008)

An Adaptation Technique for GF-Based Dialogue Systems

Faegheh Hasibi

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
faegheh@student.chalmers.se

Abstract. This paper introduces a technique for adapting GF (Grammatical Framework)-based dialogue systems. This technique can be used to adapt dialogue systems in two aspects: user adaptation and self-adaptation. By user adaptation, users can customize the system to their own needs and define alternatives for a series of information to use in later utterances. By self-adaptation, the system can update GF grammar to keep the system adequate when new situations occur. This technique is demonstrated by a multi-lingual transport query system, which allows users to find up-to-date travel plans. Adapting GF-based dialogue systems improves the functionality of speech recognizers by defining alternatives for specific phrases and also keeps the dialogue system always updated.

Keywords: Adaptation, user adaptive, self-adaptive, transport dialogue system, Grammatical Framework, Controlled Natural Language, travel planning, dialogue system, speech recognition.

1 Introduction

The notation of adaptivity is an important area in the spoken dialogue systems and concerns the natural manner of humans while interacting with a dialogue system [1]. These manners consist of different interaction styles, behavior, vocabulary and preferences [1]. For instance users may need to adapt the dialogue system to their own needs and communicate with the system by specific utterances. This paper presents a novel technique for adapting information-seeking dialogue systems in which the dialogue system integrates a huge database, a lexicon and a set of dialogue plans. This technique can extend an information-seeking dialogue system to either a self-adaptive or user-adaptive system and is applicable to various domains of controlled natural languages (CNLs).

For instance, the SAMMIE [2] is an in-car dialogue system for a music player application that allows users to control the currently playing song, construct an edit playlists [2, 3]. In this system, the users interact with the system using lots of foreign words to look for songs, artists, albums and etc. Applying user adaptation to this dialogue system, allows the users to define alternatives for a set of foreign words or to shorten the length of utterances. Consequently, these laconic conversions will increase the driver's attention to the primary driving task.

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 95–109, 2012.

Another example is price comparison services, such as PriceRunner[1], where users define features of a specific product to compare between different retailers. A dialogue system in this domain needs to record the features of a product as a special name which can be used in later utterances. For instance, when the user wants to check the price of a particular camera regularly, he can define a synonym (e.g. my camera) for a special camera rather than repeating the camera model every time.

Similarly, the user adaptation can be implemented in transport dialogue systems, where the users ask for a travel plan. For instance, the Gothenburg Tram Information System (GOTTIS) [4] is a multilingual and multimodal transport dialogue system, which uses a weighted directed graph for finding the shortest path through a subset of the Gothenburg public transportation network [5, 6]. In such systems, the user can ask a travel plan by using an alternative instead of a specific bus/tram stop, date and time to have more concise dialogues.

The idea of extending dialogue systems by allowing users to reconfigure the system to their interest is represented by voice programming [7]. This kind of adaptation is simply done when users define their own commands in speech dialogues [7]. The demonstration technique allows user to adapt a dialogue system by using the idea of voice programming. In other words, the users can reconfigure the system by defining synonyms for expressions that will be used frequently in later conversations. This kind of adaptation will improve the speech recognition of dialogue system due to the elimination of foreign words and use of shorter dialogues. Moreover, it will result in concise and brief dialogues that will increase the satisfaction of users.

In addition to user adaptation, our technique can be used for self-adaptation, where the system can modify the lexicon of a dialogue system in response to changes in the system environment. Hence, self-adaptation is a solution to the problem of context change in online dialogue systems as it keeps the lexicon always updated. For instance, when new bus/tram lines are added to a transport network, the transport dialogue system should adapt itself by adding the new bus/tram lines to the lexicon.

In this paper we explain an adaptation technique for dialogue systems that are based on Grammatical Framework (GF) [8]. GF is a grammar formalism designed for supporting grammars of multi-lingual controlled languages. The key feature of GF is the distinction between two components of grammars: abstract syntax and concrete syntax. Every GF grammar has one abstract syntax and one or more concrete syntaxes. The abstract syntax defines which expressions can be built by the grammar and the concrete syntax describes how these expressions are linearized in a particular language.

The baseline system for demonstrating the adaptation technique is a GF-based query system for planning journeys, which is described in section 2. The method of dialogue system adaptation is described in section 3, and the usages of this technique are illustrated by some examples in section 4. These examples show how our technique can be used in dialogue systems to provide both user adaptation and self adaptation. Finally we evaluate our results and summarize our experiments in section 5 and 6, respectively.

[1] http://www.pricerunner.co.uk/

2 The Baseline System

The baseline system is a multi-lingual GF-based query system which presents up-to-date travel plans. This system uses PGF [9] (Portable Grammar Format) grammars which are used as a target of compiling grammars written in GF. In order to work with PGF, a PGF interpreter is needed, which can perform a subset of GF system functionality, such as parsing and linearization. As shown in figure 1, the embedded PGF interpreter translates a user input to a HTTP request, which can be sent to the transport web service. After that, travel information is retrieved from the web service response and the corresponding abstract syntax tree is generated and linearized to a natural language answer.

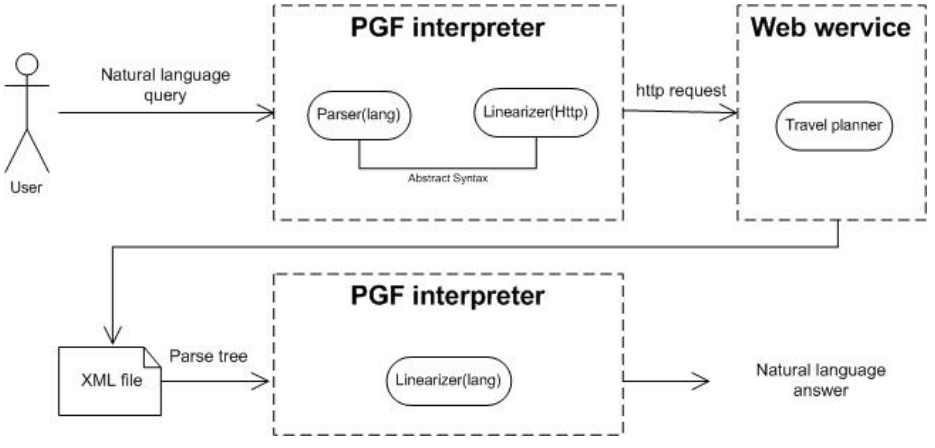


Fig. 1. Architecture overview of multilingual travel planning query system

As a multi-lingual system, the system responses should be presented in different languages. Accordingly, system answers are constructed by linearizing parse trees to target languages. So, to port the system to a new language, all that is needed is defining a new concrete syntax and since the HTTP language and natural languages have the same abstract syntax, both parsing and linearization can be done for phrases in a new language.

The described system uses the Gothenburg transport web service[2] and supports queries in both English and Swedish. The following is an ordinary interaction between user and system:

- U: I want to go from Chalmers to Valand today at 11:30
- S: Take tram number 7 from Chalmers track A to Valand track A at 11:31

In order to use this system for a new transport network, the bus/tram stops must be changed to new ones. To address this issue, the GF modules that hold stops, are generated automatically by the GF writer application, which will be introduced in the next subsection.

[2] `http://www.vasttrafik.se/`

2.1 The GF Writer Application

The embeddable GF Writer[3] is designed to dynamically construct and edit GF modules in a Java program. In other words, it can produce or update GF grammars during execution of a program. In order to generate a new module, essential part of the grammar, such as module header and body, flags, categories and function declarations must be defined. Similarly, for updating a GF grammar, a module name and a new function definition are needed. After generating or modifying a GF grammar, a new PGF file will be generated and replaced with the previous one.

The GF Writer offers three main classes for creation and modification of the GF modules: *Abstract*, *Concrete* and *Customizer* class. The *Abstract* and *Concrete* classes are designed for creation of abstract and concrete modules. In contrast, the *Customizer* class is aimed to update both abstract and concrete modules. Using methods of *Customizer* class, grammars will be updated according to user's requests.

The GF Writer can be used effectively in programs that use GF grammars. For instance, it can be used to apply changes to the GF grammar of spoken language translator, dialogue systems and localization purpose applications. Moreover, this application makes a major contribution in the implementation of the adaptation technique.

2.2 System Grammar

The transport system contains a set of modules for query and answer utterances. Figure 2 depicts an overview of grammar module, produced with the module dependency visualization feature in GF. According to this figure, both *Query* and *Answer* grammars use the same modules for travel time and stop names and on top of these grammars, the*Travel* grammar extends both *Query* and *Answer* grammars to put all grammar rules together.

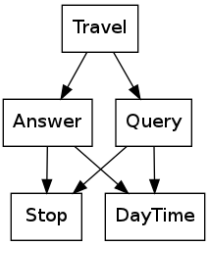


Fig. 2. Grammar design pattern for a transport query system

Stop Grammar. This grammar represents the bus/tram stops of the transport network. The straightforward approach for supporting stop names in the system grammar is to use string literals, but this *Stop* grammar is useful to suggest the stop names to the user by using the word prediction feature of the PGF parser, while parsing the user's queries.

[3] `https://github.com/hasibi/DynamicTravel`

The *Stop* modules are automatically generated by the GF writer application to make the system portable to other transport systems, which have the same format in HTTP requests and XML file. In order to generate the *Stop* grammar, the following tasks are performed in the transport query system:

1. Sending a query to transport web service to get a list of all bus/tram stops
2. Parsing XML file and retrieving the information of stops
3. Generating transport network modules for both natural languages and HTTP request

The generated abstract syntax offers a unique numerical function name for each stop. These numerical function names prevent the ambiguity of stops with the same names, which are different in other details (e.g. the same street in two different regions).

```
fun
  St_0 : Stop;
```

The English concrete module linearizes *Stop* terms in records with the name, region and track of each stop. Using this structure, the query functions are free to use the stop details (e.g. region, track number) or not.

```
concrete StopEng of Stop = {
lincat
  Stop = {s: Str; r: Str; t: Str};
lin
  St_0 = {s = "Valand"; r = "Göteborg"; t = "track A"};
  St_1 = {s = "Abacken"; r = "Skövde"; t = "track B"};
  . . .
}
```

The *StopHttp* concrete syntax contains the stops' identifiers. Due to the same abstract syntax for English and HTTP concrete syntaxes, each stop is simply mapped to its identifier.

```
concrete StopHttp of Stop = {
lincat
  Stop = {s : Str};
lin
  St_0 = {s = "9022014004420003"};
  . . .
}
```

DateTime Grammar. To get a precise travel plan, the user should mention both the day and the time of the travel. Commonly, the user mentions week days or some adverbs, such as today or tomorrow, to refer to the date of the travel in a dialogue system. Regarding this fact, we encode each day to a number in *DayTimeHttp* module and replace it in HTTP request after calculating the corresponding date in our Java program. The following code is a part of the *DayTime* module related to the day of travel.

```
fun
  Today, Tomorrow, Monday, ... : Day;
```

In the English concrete syntax of the *DayTime* module, the indexical expressions are defined in the `Today` function.

```
lin
  Today = {s = "today"} | {s = ""};
```

This function means that the queries with no reference to a day are interpreted as current day. Accordingly, this feature adds both indexicality and context-awareness to the query system.

Query Grammar. The HTTP concrete module of the *Query* abstract module generates HTTP requests in collaboration with other concrete modules. The developed dialogue system uses Göteborg transport service as a travel finder. Since this web service supports the HTTP GET method, the `GoFromTo` function produces an HTTP request with respect to the user's query.

```
fun
  GoFromTo : Stop -> Stop -> Day -> Time -> Query ;

lin
  GoFromTo from to day time =
    {s = "date=" ++ day.s ++ "&time=" ++ time.s ++
         "&originId=" ++ from.s ++ "&destId=" ++ to.s};
```

Answer Grammar. The system response grammar is described in the *Answer* grammar, which consists of vehicle, departure stop and time of travel. Since the system responses are generated by linearizing a parse tree, the HTTP concrete can have no rules. In this system we used a simple grammar for natural language utterances; however for the mature system the Resource Grammar Libraries (RGLs) [10] can be used.

```
fun
  Routing : Vehicle -> Stop -> Stop -> Time -> Answer;
  Vhc : VehicleType -> Label -> Vehicle ;
  Lbl : Number -> Label ;
  Buss, Tram : VehicleType;

lin
  Routing vehicle from to time =
    {s = "Take" ++ vehicle.s ++
         "from" ++ from.s  ++ from.t ++
         "to" ++ to.s ++ to.t ++
         "at" ++ time.s};
```

Travel Grammar. The travel module extends both *Query* and *Answer* modules. The main feature of this module is putting *Answer* and *Query* category in one category,

which is the start category for parsing and dialogue generation. Furthermore, putting all grammars in one module allows the GF grammar to be adapted, which will be described in the next section.

```
abstract Travel = Query, Answer ** {
flags
  startcat = Stmt;
cat
  Stmt ;
fun
  Ask : Query -> Stmt;
  Reply : Answer -> Stmt;
}
```

3 The Adaptation Technique

GF can be used as a component in controlled natural language systems, such as dialogue systems. To make these systems adaptive, the GF grammars need to be updated when it is required and to reach this goal, GF grammars should be modified efficiently during execution of the system.

As far as time is concerned, GF grammar adaptation can be costly while performing these two tasks: Modifying GF modules and reproducing the PGF file. Firstly, the GF modules must be modified since the GF system can only parse utterances that match grammar rules. Module modification needs a sequence of time consuming tasks, such as opening a GF file, searching through rules and writing new rules. Moreover, user adaptations may cause changes not only in one module, but also in different modules. Accordingly, the modification process can be inefficient when changes need to be applied for several modules and the situation can be even worse when the GF module is huge.

Secondly, compiling GF grammars and producing new PGF files during execution of a program takes some time and can be annoying for users. Since a PGF file is the linkage of GF object files (.gfo files), more modified files causes more new GF object files and consequently the creation of a new PGF file would be more costly. All in all, adapting GF grammar by changing the grammar modules is time consuming and will be done in an unacceptable time for users.

Regarding these problems, our approach is targeted toward applying changes to an extension grammar rather than changing the main grammar itself. The extension grammar is a GF module that extends all other grammars and thus it contains all categories and functions of the main grammar. According to the separate compilation feature of the GF software, the only grammar module that is needed to compile after every adaptation will be the extension module. So the execution time of the GF grammar adaptation becomes acceptable for the users. Figure 3 describes the relation of the extension and main grammar.

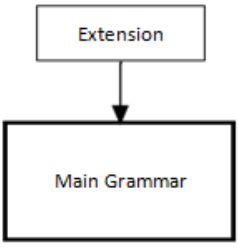


Fig. 3. GF grammar adaptation pattern design

For instance, to extend a GF grammar to an adaptive one, two grammars are needed: *Ext* and *Def*. The *Ext* module is initially empty and grammar rules will be added gradually for every new definition. The *Def* module contains grammar rules for user definitions. These two modules are shown in figure 4.

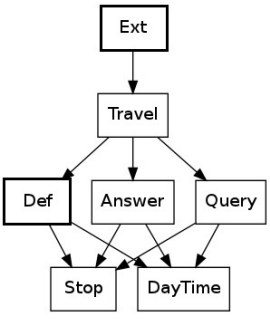


Fig. 4. Grammar design pattern for an adaptive transport dialogue system

In order to change our dialogue system to an adaptable one, the GF grammars must be changed in some aspects. These changes are divided into two groups: dialogue manager and GF programmer changes. Dialogue manager changes means modifying *Ext* abstract syntax and its concrete modules. These modifications are done dynamically in the dialogue manager and when a user defines new meanings. On the other hand, there are some changes that the GF programmer must consider in GF modules while writing the GF grammars. Adding type definitions, functions and operations are some of the examples. We explain these changes in the following subsections in more details.

This adaptation technique can be used for both self adaptation and user adaptation. These two types of adaptation are explained by some examples in the following subsections.

3.1 User Adaptation

The user adaptation is based on the idea of voice programming, where users can explicitly adapt some aspects of a dialogue system to their own needs [7]. Regarding voice programming, the user adaptation is described for two aspects of the transport system.

Definition Grammar. The *Def* grammar specifies how a user can communicate to the dialogue system for the purpose of customization. A simple syntax of the *Def* grammar is shown below, which offers some synonyms to the user for defining a new meaning for a series of information.

```
fun
  DefPlace : New -> Stop -> Def;
  DefPlaceDayTime : New -> Stop -> Day -> Time -> Def;
  Home, Work, Gym : New;

lin
  DefPlace new stop = {s = new.s ++ "means" ++ stop.s};
  DefPlaceDayTime new stop day time =
   {s = new.s ++ "means" ++ stop.s ++ day.s ++ time.s};
```

Stop Customization. In this subsection, an example of customizing the name of a stop is described. Firstly, the user defines a new command such as below:

— U: home means Valand

Then the parser produces corresponding abstract syntax tree.

(Customize
((DefPlace Home) St_1639))

Having this parse tree, the required information is extracted; *Home* and *St_1639*. In the next step, a new rule is added to the concrete modules of *Ext*. Parallel to the English concrete syntax of *Ext*, a corresponding rule is added to the concrete syntax of other languages.

```
concrete ExtEng of Ext = TravelEng-[ St_1639 ] ** {
lin
  St_1639 = toStop TravelEng.Home TravelEng.St_1639;
}
```

After adding the new linearization rule to the extension concrete module, the PGF file is reproduced to support recent changes. Both the module modification and the PGF file production are done using GF Writer methods.

Travel grammar changes. There are some changes that must be applied to the grammar by GF programmer before releasing the dialogue system. In the *ExtEng* module shown above, we used restricted inheritance that excludes `St_1639` from *TravelEng*. This design causes non-ambiguity and simultaneously keeps the previous type definition of this function by `toStop` operation. The purpose of this operation is to change the type of user's alternative to the desired category, e.g. String (home) to Stop (Valand). The `toStop` operation is a part of *TravelEng* module which is written by the GF programmer.

```
oper
  toStop : {s : Str} -> Stop -> Stop = \new, stop ->
    {s = stop.s; r = stop.r; t = stop.t;
     alt = stop.alt | new.s};
```

As it is shown in the `toStop` operation, the type of `Stop` has changed in comparison with section 2, by adding the `alt` object. This object contains a linearization of the stop name and its alternatives in the form of variants.

```
mkStop : Str -> Str -> Str -> TStop =
  \stop, region, track ->
  {s = stop; r = region; t = track; alt = stop ++ track};
```

Adding the `alt` object in the type definition allows GF programmers to use these alternatives whenever they need. For instance, we use the `alt` object for linearizing user queries but not the system response.

```
GoFromTo from to day time =
  {s = "I want to go from" ++ from.alt ++ "to" ++ to.alt
       ++ day.alt ++ "at" ++ time.s };

Routing vehicle from to time =
  {s = "Take" ++ vehicle.s ++ "from" ++ from.s  ++ from.t
       ++ "to" ++ to.s ++ to.t ++ "at" ++ time.s};
```

Multiple definitions for a stop. The user may define several names for a special place. For instance, he defines Valand as gym in addition to home.

— U: gym means Valand

Since the ExtEng module has already a linearization for this stop, this alternative will be added as a variant to the linearization rule.

```
St_1639 = toStop TravelEng.Gym (TravelEng.Home
                                TravelEng.St_1639);
```

Stop, Day and Time Customization. In the previous subsection we mentioned how an existing function in the GF grammar can be modified to hold user adaptations. In addition to this type of adaptation, the user may need to define an alternative for complicated phrases. In our transport dialogue system, this definition can be any combinations of stops, day, and time. To handle these definitions we need to introduce new types and consequently some operations. We describe our solution for this type of adaptation in the following example, where a user defines a word to mean a special place, day and time. A user utterance and its parse tree are like this:

— U: work means Chalmers on Monday at 7:30

(Customize
((((DefPlaceDayTime Work) St_1592) Monday)
((HourMin (Num N7)) ((Nums N3) (Num N0)))))

As the next step, the `WorkStopDayTime` function and its linearization are added to the *Ext* abstract and concrete modules.

```
fun   WorkStopDayTime : StopDayTime;

lin  WorkStopDayTime = toStopDayTime TravelEng.Work
                                     TravelEng.St_1592
                                     TravelEng.Monday
                                     "7:30";
```

Travel grammar changes. Since `Stop`, `Day` and `Time` are already declared in the grammar categories, the `StopDayTime` category is introduced to hold a stop, day and time together with a string as an alternative. As it is shown below, type of `time` in the `StopDayTime` record is String and not `Time`. This is due to the fact that time is similar in all languages and is not translated between languages.

```
StopDay = {stop: Stop; day: Day; Time: Str; alt: Str};
```

Similar to the previous subsection, we need an operation to create `StopDay` type from given string, stop and day:

```
toStopDayTime: {s:Str} -> Stop -> Day -> Str -> StopDay =
  \new, st, d, t ->
  {stop = st; day = d; time = t; alt = new.s};
```

In addition to the English grammar, the HTTP grammar should also have the `toStopDay` operarion and type definition for `DayTime` category. Since user definitions are not used in HTTP queries, the `alt` abject is omitted.

```
Lincat  StopDayTime = {stop : R.TStop; day : TDay};

oper toStopDayTime : Stop -> Day -> Str -> StopDaTime =
     \st, d, t -> { stop = st; day = d; time = t };
```

According to *Ext* module, the `WorkStopDayTime` rule means having a certain stop name, day and phrase, a new instance of *StopDayTime* type is produced. But it does not mean that the user can ask a query such as below.

— U: I want to go from Valand to work at 9:30

To address this issue, a new kind of `GoFromTo` function is introduced that accepts an instance of `StopDayTime` category rather than separate `Stop, Day` and `Time` instances.

```
GoFromToStopDayTime from stopDayTime =
  {s = "I want to go from" ++ from.alt ++
       "to" ++ stopDayTime.alt };
```

The corresponding function in TravelHttp module is shown below. Since the HTTP requests need exact information for the user's query, all fields of `WorkStopDay-Time` are used in linearization.

```
GoFromToStopDayTime from stopDayTime =
  {s = "date=" ++ stopDayTime.day.s ++
       "&time=" ++ stopDayTime.time.s ++ "&originId=" ++
       from.s ++ "&destId=" ++ stopDayTime.stop.s};
```

Multiple definitions for stop, day and time. As it is shown in the *Ext* module, a new function will be declared for each customization of stop and day. Due to the GF grammar syntax, each function name must be unique in the abstract syntax. Accordingly, we formulate the function name generation by combining the alternative (e.g. Home) and *StopDayTime*. As a consequence, when a user gives a new definition for an existing function, the linearization will be changed to the new one. For instance, the `WorkStopDayTime` function will be the following when the user defines a new meaning for work.

— U: work means Chalmers Tvärgata on Monday at 8:00

```
WorkStopDayTime = toStopDayTime TravelEng.Work
    TravelEng.St_1590 TravelEng.Monday "8:00"
```

3.2 Self-adaptation

Vehicle labels in our dialogue system are represented by type of the vehicle (e.g. bus, tram) and a number, like bus number 10. Assuming the following query, the web service offers Grön express bus.

— U: I want to go from Delsjömotet to Berzeliigatan today at 1 1 : 3 0

Since the "Grön express" vehicle label is not supported in the system grammar, the dialogue manager will fail to produce parse tree and the linearizer cannot generate system response. To solve this problem, the desired vehicle label must be added to the GF grammar.

When travel planner offers a vehicle with specific name, the dialogue manger checks whether this label is already added to the grammar or not. So the parser tries to parse the vehicle label and if it succeeds, the function name will be used in the answer parse tree. Otherwise, a new function will be added to the extension abstract and concrete modules.

```
fun
  Lbl_1 : Label;

lin
  Lbl_1 = toLabel "GRÖN EXPRESS";
```

By adding new labels to the grammar, the PGF file is updated and this answer will be shown to the user:

— S: Take bus Grön express from Delsjömotet to Berzeliigatan at 1 1 : 4 1

4 Example: An Adaptable Transport Query System

We have implemented an adaptable transport query system by applying the adaptation technique to the base line system.

The following conversation shows a normal interaction with the system before applying user adaptation.

- U: I want to go from Chalmers to Valand today at 11:30
- S: Take tram number 7 from Chalmers track A to Valand track A at 11:31

However, these examples show how a user can record some commands and use them in later queries to have a short conversation. Meanwhile, the Swedish utterances depict multilingual aspect of the designed system.

- U: work means Chalmers on Monday at 7:30
- U: home means Valand
- U: Jag vill åka från hem till jobbet
  (I want to go from home to work)
- S: Ta spårvagn nummer 10 från Valand läge B till Chalmers kl 07:33
  (Take tram number 10 from Valand track B to Chalmers at 07:33)

Some of the supported definitions in the adaptive query system are stated below:

- Work means Chalmers.
  - (`VALUE` **means** `STOP-NAME`)
- Work means Chalmers on Monday.
  - (`VALUE` **means** `STOP-NAME DAY`)
- Work means Chalmers on Monday at 11:30.
  - (`VALUE` **means** `STOP-NAME DAY TIME`)
- Birthday means Saturday
  - (`VALUE` **means** `DAY`)

This text-based query system can be extended to a multimodal dialogue system [5]. In such system, the users can define a synonym for a stop name, which cannot be recognized by speech recognizer. After that, the user can refer to that place by using a usual word in later dialogues. Moreover, the user adaptations can be saved in a log file, which will be retrieved when the stop grammar changes.

5 Evaluation

Our criterion of evaluation was to assess the effects of user adaption on speech recognition. To do this task, we tested 120 random generated input queries of our transport query system by an ordinary speech recognizer. These utterances were equally divided into two groups of adapted and non-adapted queries and were fed to the speech recognizer, which was Google speech recognizer[4]. After collecting the outputs of the

[4] `http://www.google.com/insidesearch/features/voicesearch/index-chrome.html`

Google speech recognizer we noticed that all of the non-adapted queries failed, whereas most of the adapted queries were passed.

The behavior of the speech recognizer while encountering foreign words is shown in this typical example:

- Input: I want to go from **Åketorpsgatan to Billdal** on Monday at 11:31
- Output: I want to go from **pocket doors car tom to build on that** on Monday at 11:31

According to this example, the speech recognizer's trend is to find known words instead of analyzing foreign words. In other words, it extracts a sequence of common words rather than guessing the given place name; so it cannot translate even a plain name, such as Billdal. However, having look to the failed adapted queries demonstrates that the recognized text is very similar to the speech and the error rates are low. For instance, in the following query the word "pub" is translated to "park", which is due to the difficulty of discriminating between special alphabets.

- Input: I want to go from the **pub** to park on Friday at 15:35
- Output: I want to go from the **park** to park on Friday at 15:35

In contrary, the following examples show some adapted passed queries:

- I want to go from hospital to restaurant
- I want to go from bank to cinema at 6:17
- I want to go from university to university on Monday at 4:20

In order to evaluate and compare the word error rate of speech recognizer for both adapted and non-adapted queries, the similarity of each query to the recognized one was numerated word by word and in a sequential order. We chose this type of similarity according to the GF parser, which parses a given sentence token by token and using a top-down algorithm [11]. The following table shows the rates of sentence error and word error for both adapted and non-adapted queries:

Table 1. Error rate of speech recognizer for adapted and non-adapted queries

	Word error rate	Sentence error rate
Non-adapted queries	58	100
Adapted queries	26	53

To sum up, user adaptation affects the speech recognition process and results in a more reliable system. This is due to the elimination of foreign words and shorter dialogues.

6 Conclusion

In this work, we presented an adaptation technique for the GF-based dialogue systems, which can be used for both user adaptation and self-adaptation. The user adaptation is based on the idea of voice programming that allows users to reconfigure dialogues systems by using natural language commands. These commands are

interpreted by the query system and then stored as a rule in the system. User adaptation results in concise dialogues and accurate speech recognition, which will increase the users' satisfaction.

The baseline system that was used for implementation of the adaptation technique is a transport query system that communicates with a transport web service and presents accurate travel plans to users. In this system, a GF writer application generates a set of grammar modules for stops dynamically, which makes the system portable to other transport networks. Moreover, we applied the system's self-adaptation by adding new bus/tram lines to keep the system always updated.

Acknowledgment. The author would like to thank Aarne Ranta for his enthusiastic guidance and suggestions through this research and for his valuable comments about this manuscript. The special thanks also go to Grégoire Détrez, Krasimir Angelov, Ramona Enache and John Camilleri for their undeniable help and productive discussions during this work.

References

1. Lemon, O., Pietquin, O.: Introduction to Special Issue on Machine Learning for Adaptivity in Spoken Dialogue Systems. ACM Transactions on Speech and Language Processing 7(3) (2011)
2. Becker, T., Poller, P., Schehl, J., Blaylock, N., Gerstenberger, C., Kruijff-Korbayova, I.: The SAMMIE system: Multimodal in-car dialogue. In: Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions, Sydney, Australia, pp. 57–60 (2006)
3. Perera, N., Ranta, A.: Dialogue System Localization with the GF Resource Grammar Library. In: SPEECHGRAM 2007: ACL Workshop on Grammar-Based Approaches to Spoken Language Processing, Prague, June 29 (2007)
4. Bringert, B.: Speech recognition Grammar Compilation in Grammatical Framework. In: Proceedings of the ACL 2007 Workshop on Grammar-Based Approaches to Spoken Language Processing, Prague, Czech Republic, June 29, pp. 1–8. The Association for Computational Linguistics (2007)
5. Bringert, B., Cooper, R., Ljunglöf, P., Ranta, A.: Multimodal Dialouge System Grammars. In: Proceedings of DIALOR 2005, Ninth Workshop on the Semantics and Pragmatics of Dialogue, Nancy, France, pp. 53–60 (June 2005)
6. Bringert, B.: Embedded grammars. MSc Thesis, Department of Computing Science, Chalmers University of Technology (2004)
7. Georgila, K., Lemon, O.: Programming by Voice: enhancing adaptivity and robustness of spoken dialogue systems. In: BRANDIAL 2006, Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue, pp. 199–200 (2006)
8. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011) ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth)
9. Angelov, K., Bringert, B., Ranta, A.: Pgf: A portable runtime format for type-theoretical grammars. Journal of Logic, Language and Information 19, 201–228 (2010)
10. Ranta, A.: The GF Resource Grammar Library. In: Linguistic Issues in Language Technology, LiLT, vol. 2(2) (December 2009)
11. Angelov, K.: Incremental Parsing with Parallel Multiple Context-Free Grammars. In: European Chapter of the Association for Computational Linguistics (2009)

General Architecture of a Controlled Natural Language Based Multilingual Semantic Wiki

Kaarel Kaljurand

Institute of Computational Linguistics, University of Zurich, Switzerland
kaljurand@gmail.com

Abstract. In this paper we propose the components, the general architecture and application areas for a controlled natural language based multilingual semantic wiki. Such a wiki is a collaborative knowledge engineering environment that makes its content available via multiple languages, both natural and formal, all of them synchronized via their abstract form that is assigned by a shared grammar. We also describe a preliminary implementation of such a system based on the existing technologies of Grammatical Framework, Attempto Controlled English, and AceWiki.

Keywords: semantic wiki, multilinguality, controlled natural language, AceWiki, Attempto Controlled English, Grammatical Framework.

1 Introduction

A wiki is a user-friendly collaborative environment for creating articles in natural language. The most well-known example of a wiki is Wikipedia, an encyclopedia that is being built by millions of people in hundreds of different languages. Numerous wikis have also been built for smaller domains. Even though wiki engines generally allow for the wiki articles to be written in multiple languages, these different language versions exist independently of each other and only article-level granularity is offered by the system for interlinking the multilingual content.

Semantic wikis [23] combine the main properties of wikis (ease of use, collaboration, linking) with knowledge engineering technology (structured content, knowledge models in the form of ontologies, automatic reasoning). Wiki editors simultaneously work with the natural language content and its underlying formal semantics representation. The resulting wiki offers more powerful content management functions, e.g. dynamically created pages and detection of semantic errors, but has to somehow meet the challenge of keeping the editing interface user-friendly. In the most well-known existing systems (e.g. Semantic Mediawiki[1], Freebase[2]) the semantics is represented using RDF-like subject-predicate-object triples, e.g. typed wikilinks (predicates) between two articles (the subject and the object).

A controlled natural language (CNL) is a subset of a natural language, where the subset has a precisely defined syntax, its sentences have a formal (executable) meaning, and it is accompanied by end-user documentation describing its syntax, semantics

[1] http://semantic-mediawiki.org/
[2] http://www.freebase.com/

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 110–120, 2012.

and usage patterns. CNLs and their editing tools facilitate the creation of texts that are natural yet semantically precise, and can thus function well in human-machine communication. Many controlled natural languages (both for general purpose and for fixed domains) have been developed based on many different natural languages [18]. However, there has not been an effort to bring them under the same semantic model or synchronize their development in a community-driven manner [16].

In this paper we define the notion of a multilingual CNL-based semantic wiki, describe the general architecture of such a system, and propose existing components for its implementation. A CNL-based semantic wiki bases the content of the wiki on a precisely defined CNL, thus supporting the user in entering semantically rich content while keeping the user-interaction with the wiki still simple and intuitive. A multilingual CNL-based semantic wiki adds the dimension of multilinguality, i.e. that several natural language fragments can be used in the wiki, both for the content, and for the meta aspects of the wiki. In terms of multilinguality our envisioned wiki system has some similarities with the OWL ontology editor described in [2] which allows the user to view the ontology in three CNLs, two based on English and one on Chinese. As the main difference, our system uses the CNLs as the only user-interface for both editing and viewing. Also, our design abstracts away from the concrete underlying reasoning language.

A multilingual CNL-based semantic wiki could have several useful applications. It could be used to collaboratively build a multilingual domain-specific resource, e.g. tourist phrase book, a museum catalog, a technical manual, or a collection of mathematics exercises, where a precise translation linking the content on a sentence level is desired. It could also be used to build and query ontology and rule based systems were the content has a meaning defined in formal logic and made available by an automatic reasoning tool. User activities in such wikis can focus on developing the content, developing a language for expressing this content, developing a query language for accessing this content, or simply viewing and querying the content.

In section 2 we discuss the existing technologies that our work builds on, namely Attempto Controlled English, AceWiki, and Grammatical Framework; in section 3 we describe the overall architecture of the system; in section 4 we discuss its preliminary implementation; and in section 5 we describe the future work.

2 Background Technologies

2.1 Attempto Controlled English

Attempto Controlled English (ACE) [6] is a general purpose first-order language (FOL) with English syntax, i.e. ACE can be viewed as both a natural language understandable to every English speaker, as well as a formal language with a precisely defined syntax and semantics understandable to automatic theorem proving software. ACE texts are deterministically interpreted via Discourse Representation Structures (DRS) [13]. The syntactically legal sentence structures and their unambiguous interpretation is explained in the end-user documentation of *construction* and *interpretation* rules. The ACE toolchain includes a parser that maps ACE sentences into a concrete DRS form [7]

and further into formats supported by existing automatic reasoners (e.g. OWL, SWRL, TPTP).

End-users working with ACE can specify a lexicon that maps English wordforms (nouns, verbs, adjectives, ...) into logical atoms, which the users can interpret as they wish, but otherwise the ACE grammar or its mapping to DRS cannot be changed.

2.2 AceWiki

AceWiki[3] [14] is a semantic wiki system that uses OWL [8] as its underlying semantic framework integrating its main reasoning tasks (consistency checking, classification and query answering). Because the standard OWL syntaxes are largely impenetrable to a general wiki user, AceWiki offers a CNL front-end in the form of ACE. The wiki articles are edited entirely in ACE and automatically mapped to OWL in the background without the user ever having to rely on explicit knowledge of OWL. This is an important difference with respect to other semantic wikis where the user must work in two different languages: the free-form natural language and the semantics modeling language.

The content of an AceWiki wiki is written in a subset of ACE formally defined by a Codeco grammar [15]. The grammar targets an OWL-compatible fragment of ACE, i.e. ACE sentences that are semantically outside of the OWL expressivity cannot be expressed in the AceWiki fragment. Also, this grammar is used to drive a look-ahead editor which explicitly restricts the user from entering more expressive or completely free-form sentences.

The content is structured into a set of articles, each article containing a sequence of entries where the order is not semantically significant. Each entry is either a declarative sentence, interrogative sentence (question), or an informal comment. The declarative sentences assert new information into the knowledge base and therefore influence the consistency and entailments of the knowledge base, the questions provide an access point to the entailed knowledge. The comments are not restricted by the look-ahead editor nor checked by the semantic reasoner. They are structurally a plain wiki text with possible wikilinks to other articles.

The content words (proper names, nouns, transitive verbs, relational nouns and transitive adjectives) in the sentences map one-to-one to wiki articles, i.e. each content word comes with its own article, although the content of the article is in no way determined by the word. Semantically, content words correspond to OWL entities: proper names to OWL individuals, nouns to OWL classes, and the relational words to OWL properties. Declarative sentences map to OWL axioms and interrogative sentences to OWL class expressions [10]. Upon every change in the wiki an OWL reasoner determines its effect and flags inconsistencies or updates the dynamically generated parts of the wiki (e.g. answers to questions).

From the viewpoint of a general CNL-based semantic wiki, the AceWiki user is restricted to using a single CNL with an unchangeable grammar and a predetermined reasoning mechanism.

[3] http://attempto.ifi.uzh.ch/acewiki/

2.3 Grammatical Framework

Grammatical Framework (GF) [20] is a functional programming language for the engineering of multilingual grammars. The grammar author implements an *abstract syntax* (a set of functions and their categories) and a set of its corresponding *concrete syntaxes* which provide the implementation of the abstract functions and categories. The resulting grammar describes a mapping between concrete language strings and their corresponding abstract syntax trees. This mapping is bidirectional — strings can be *parsed* to trees, and trees *linearized* to strings. As an abstract syntax can have multiple corresponding concrete syntaxes, the respective languages can be automatically *translated* from one to the other by first parsing a string into a tree and then linearizing the obtained tree into a new string.

GF is optimized to deal with natural language features like morphological variation, agreement, long-distance dependencies. The GF infrastructure provides a *resource grammar library*, a reusable grammar library for about 25 natural languages (English, Finnish, Urdu, ...) that covers their main syntactic structures and morphological paradigms and makes these accessible via a language-independent API [19].

GF has been used to implement multilingual domain-specific CNLs such as the MOLTO Phrasebook language [22] or a language for mathematical proofs [9]. GF has also been used to turn the existing controlled natural language ACE multilingual [21,4]. The development of GF has focused on parsing tools, grammar editors, and extending the grammar library to new languages. So far the GF infrastructure has not focused on a wiki-like tool for manipulating sets of sentences, querying abstract trees, etc. One can mostly work with single sentences (e.g. using the Minibar application[4]), although using GF in a multilingual wiki context has been investigated [17].

3 General Architecture

A multilingual semantic wiki supports the users in collaboratively creating and viewing semantically backed content in multiple natural languages. In the following we focus on the unique aspects that such a wiki system must manage, namely multilinguality, formal grammars, and formal reasoning, and we do not focus on the standard wiki features that make the collaborative editing possible such as "talk pages", community rules, revision history, user identification and access rights, reward system, etc. We consider such features general enough that they apply without change also to our envisioned wiki system.

3.1 Multilinguality

The languages in the wiki serve different purposes, namely

- expressing the wiki content, both user-edited and dynamically created;
- localizing the user interface (button labels etc.);
- expressing meta queries (e.g. about authors and edits in the wiki).

[4] `http://www.grammaticalframework.org/demos/minibar/minibar.html`

The languages fall into two types — natural (e.g. English, German, ...) and formal (e.g. first-order logic, algebraic expressions). While the natural languages appear in the user interface via which users edit and view the content, the formal languages are used as input to formal reasoners to reason about the content or automatically generate new content such as entailments. Languages can also be classified into general purpose and domain-specific (closed vocabulary) languages. For example, ACE in this system can be viewed as a general purpose content language usable as both a natural language and a formal language.

The texts in the wiki (i.e. content articles, UI labels) have multilingual representations which are all updated once one of the representations is changed so that the same state of the wiki is always available in all supported languages. Also, the wiki provides sentence-level (or even word-level) interlinking between copies of the same text in multiple languages.

GF provides a good platform for implementing such a multilingual architecture as the core idea of GF is a grammar that relates multiple concrete languages via a language-neutral abstract syntax. Furthermore, GF comes with a useful library of natural language syntactic and morphological forms for many relevant languages.

3.2 Multiple Dynamic Grammars

The controlled languages of the wiki are governed by one or more formal grammars. In the general case, these grammars are collaboratively modifiable by the wiki users. Examples of grammars are general purpose CNL grammars like ACE (ported into languages other than English), grammars for fixed domain languages like the MOLTO Phrasebook, grammars for the natural language labels in the user interface, and grammars of the supported query languages.

The general architecture would allow the wiki users to enter content under multiple grammars and collaboratively change the grammars. This raises several questions that the implementation should answer:

- what is the granularity of mixing the grammars, e.g. can a wiki article contain sentences parsed by different grammars;
- how does the wiki handle the possible relations (importing and resource sharing) between grammars; and
- what type of user changes to grammars are allowed.

The simplest change — adding syntactic sugar (a GF variant) to a concrete language does not affect the other languages. It is also relatively simple to support the addition of new words into existing GF categories as this would not invalidate the existing wiki content. However, changing a syntax rule would potentially make some of the content syntactically incorrect. In the simple case, the user interface must integrate an editor for modifying words and their forms (in multiple languages), but in the general case it must include a full GF grammar editor.

3.3 Reasoning and Query

The described architecture of multilinguality already provides a large degree of "semantics" to the wiki which is not available in traditional wikis. For example, the

language-neutral common form of wiki sentences makes it possible to edit a sentence in different language articles simultaneously. Also, the query system of the wiki can operate not just over words but also on the syntactic templates of the sentences. Additional and more powerful semantics can be provided by a set of tools which help the user to automatically monitor the wiki content and detect semantic consistency problems. We thus understand reasoning and query in a very general sense — it can be provided via the language-neutral abstract form or via a semantic (e.g. first-order) underlying model attached to one of the concrete languages.

The abstract tree based query support goes much beyond the keyword search and interlinking offered by traditional wikis, and the aspect of multilinguality makes the querying also richer than available in semantic wikis. Examples of such queries are

- what does the given sentence/phrase look like in other languages?
- which sentences contain the given word (in any of its forms)?
- which sentences contain the given syntactic structure, i.e. the same tree shape with possibly different leaf nodes?

For example, to get a list of different greetings described in a multilingual phrasebook wiki one could pose a meta query `(Greeting ?)` where the answers will be bindings for the wildcard, linearized into one of more natural language phrases. Also, such syntactic queries could be posed in a natural language, by providing a partial sentence, e.g. `Where is the closest ...?`.

Examples of tools that provide the second type of reasoning and querying are general purpose reasoners like FOL or OWL reasoners, or reasoners specialized for a certain domain like linear algebra. This type of reasoning operates with notions of consistency, entailment, automatic question answering, e.g.

- does the given section of the wiki contain a contradiction?
- what sentences are entailed from the given sentence?
- which sentences answer the given question?
- how can the given sentence be paraphrased?

In order to accommodate different types semantic reasoners (ontology reasoners, mathematical reasoners and custom domain-specific reasoners) the wiki should offer a general API with functions like "what are the answers to this question given this section of the wiki". Some of the existing reasoners can also be accessed via ACE, and an ACE-based wiki would also offer paraphrasing support [12].

An important part of such semantic reasoning support is also the explanation of the reasoning results. For the explanation facilities available in ACE-based systems see e.g. the ACE reasoner RACE [5] and the ACE View ontology editor [11].

3.4 Article Structure

The wiki is a structure of interlinked articles where each article is composed of sentences governed by a grammar. As we assume that the underlying formal grammar of the wiki is always changing, the component sentences in the wiki are at a given time

either parsable or unparsable. Support for unparsable sentences offers also the possibility of test-driven development of grammars where the users intentionally create ungrammatical sentences and then proceed by editing the grammar to make the sentences parseable. Support for partial parsing would additionally help to measure the progress in this process.

Sentence editing is assisted by a syntax-aware editor that knows the underlying grammar and vocabulary, and can therefore direct the user towards the completion of a syntactically correct sentence. Existing methods of such support include look-ahead editing, template-based editing, example-based editing, random generation (see e.g. [24,22,17]).

Wiki articles should also be allowed to include free-form components such as unrestricted text, hyperlinks, tables and images which are not intended to be parsed even if the grammar evolves. Regarding the order of all the components, the wiki authors should be able to decide and formally declare if the order has any significance or if the content can be sorted as the viewer wishes.

As in traditional wikis, our wiki consists of interlinked articles. There are two types of articles: user-created articles and dynamically created views based on a query. In the dynamic view, the sentence order plays no semantic role and if a sentence is edited then it affects all the actual occurrences of this sentence in the wiki. Such a dynamic view can be generated by any query which happens over the set of abstract trees, e.g. trees which contain a given function, trees which unify with a partially instantiated tree, etc. Such queries can be saved as links as part of the wiki content.

3.5 Ambiguity

CNLs like ACE are structurally unambiguous, i.e. they can be deterministically parsed into their native logical form (e.g. DRS). For a general architecture we must also consider a more general handling of ambiguity and its possible exploitation in the system.

In a GF-based system ambiguity arises from the fact that the parser can potentially assign more than one abstract tree to a string. The ambiguity can thus be analyzed by observing the set of trees. In the context of the wiki with goals of user-friendliness, the trees should be hidden and ambiguity presented by other means, e.g. via the other languages in the system or using a dedicated disambiguation grammar developed for every language [22].

In order to automatically resolve ambiguity the wiki system can employ reasoning and the assumption that sentences which make the underlying knowledge base inconsistent (or which are already entailed by the system) are unwanted, i.e. their corresponding abstract tree can be discarded. Also, the collaborative nature of the wiki offers the feature of collaborative ambiguity resolution: an ambiguous sentence created by a speaker of one language can be viewed by a speaker of another language as a set of different unambiguous sentences, some of which must be deleted, resulting in an unambiguous underlying form. This requires the storage unit via which the sentence representations in multiple languages are generated to be a set of trees rather than a single tree.

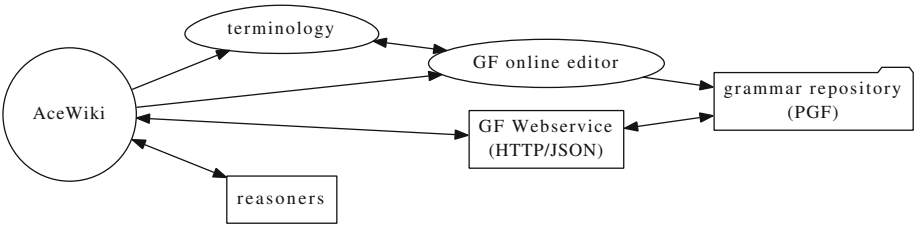


Fig. 1. Our implementation of the multilingual CNL-based semantic wiki system is a set of loosely coupled systems which can all be used independently as well. *Terminology* (currently not implemented) stands for a multilingual repository of terms which comes with its own (collaborative) editor. *GF online editor* provides a UI for editing the source of GF grammars, compiling them as PGF files, and saving them into a *grammar repository*. *AceWiki* manages the set of GF abstract trees, structuring them into wiki articles and making them available via their natural language representations. It interacts programmatically with the *GF Webservice* for parsing and linearizing, and with (the OWL) *reasoners* for semantic feedback. It directs users to the terminology and grammar editors if changes to the grammars are desired.

4 Implementation

We are currently in the process of implementing the described architecture as part of the EU research project MOLTO[5]. The implementation integrates various existing tools (figure 1):

- AceWiki provides the user interface and the storage of the wiki content, as well as an API via ACE to several OWL reasoners;
- GF Webservice [3] provides GF parsing and linearization (and the related look-ahead and translation) services for grammars in the portable PGF format[1];
- GF online grammar editor[6] provides GF grammar editing in the source format and compilation into PGF.

All these components have a web interface (either end-user oriented or a programmatic API) so that the complete system can be easily put together using loose coupling of otherwise independently developed tools. A tighter integration would still be needed e.g. to track the user activities in the separate components. The main features of the current implementation are:

- the standard AceWiki UI has been largely preserved and only extended with e.g. the language switching menu;
- the user can edit content under multiple grammars (although each wiki can have only a single underlying grammar);
- the wiki content can be viewed and edited in each language that is supported by the chosen grammar;

[5] `http://www.molto-project.eu`
[6] `http://cloud.grammaticalframework.org/gfse/`

- we have created a multilingual GF grammar of the AceWiki OWL-oriented subset of ACE [4], essentially allowing the users to formulate ACE (and OWL) statements in 10 different natural languages;
- the persistent storage is based on GF abstract trees;
- the sentence entry is supported by the standard AceWiki look-ahead editor (although the predicted words are not classified by word class as in standard AceWiki);
- advanced users can look at the GF parser output which provides for a sentence its syntax trees, translations, and word alignment diagrams;
- grammar editing is possible using the external GF online grammar editor, and the changes to the grammar are immediately effective in the wiki.

The current system still lacks a multilingual terminology editor, as AceWiki's own lexical editor is specific to English and cannot be easily reused. Also, even though AceWiki includes a general API for ontology reasoners, it currently lacks an interface to other types of concrete reasoning tools.

The implementation extends the open-source AceWiki system and takes place on GitHub[7].

5 Conclusions and Future Work

We described a multilingual collaborative semantic system which could be useful in several application areas where current (semantic) wiki systems do not perform well because they lack support for precise multilinguality and/or rich semantic processing. Our own implementation of this system is in the early stages, currently demonstrating the basic structure but not yet targeting some important and interesting aspects e.g. ambiguity management, collaborative editing of multilingual lexicons and grammars, and a general reasoner interface. Additionally, there is a multitude of requirements demonstrated by real-world collaborative systems (see the list in the introduction of section 3) that our system currently does not meet.

Acknowledgments. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement FP7-ICT-247914.

References

1. Angelov, K., Bringert, B., Ranta, A.: PGF: A Portable Run-time Format for Type-theoretical Grammars. Journal of Logic, Language and Information 19(2), 201–228 (2010), doi:10.1007/s10849-009-9112-y
2. Bao, J., Smart, P.R., Shadbolt, N., Braines, D., Jones, G.: A Controlled Natural Language Interface for Semantic Media Wiki. In: 3rd Annual Conference of the International Technology Alliance, ACITA 2009 (September 2009)
3. Bringert, B., Angelov, K., Ranta, A.: Grammatical Framework Web Service. In: Proceedings of EACL 2009 (2009)

[7] `http://github.com/AceWiki/AceWiki`

4. Camilleri, J.J., Fuchs, N.E., Kaljurand, K.: Deliverable D11.1. ACE Grammar Library. Technical report, MOLTO project (June 2012), http://www.molto-project.eu/view/biblio/deliverables
5. Fuchs, N.E.: First-Order Reasoning for Attempto Controlled English. In: Rosner, M., Fuchs, N.E. (eds.) CNL 2010. LNCS, vol. 7175, pp. 73–94. Springer, Heidelberg (2012)
6. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
7. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Discourse Representation Structures for ACE 6.6. Technical Report ifi-2010.0010, Department of Informatics, University of Zurich, Zurich, Switzerland (2010)
8. W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. W3C Recommendation October 27 2009. Technical report, W3C (2009), http://www.w3.org/TR/owl2-overview/.
9. Humayoun, M., Raffalli, C.: MathNat—Mathematical Text in a Controlled Natural Language. Journal on Research in Computing Science 46 (2010)
10. Kaljurand, K.: Attempto Controlled English as a Semantic Web Language. PhD thesis, Faculty of Mathematics and Computer Science, University of Tartu (2007)
11. Kaljurand, K.: ACE View — an ontology and rule editor based on Attempto Controlled English. In: 5th OWL Experiences and Directions Workshop (OWLED 2008), Karlsruhe, Germany, October 26–27, 12 pages (2008)
12. Kaljurand, K.: Paraphrasing controlled English texts. In: Fuchs, N.E. (ed.) Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009). CEUR Workshop Proceedings, vol. 448 (2009)
13. Kamp, H., Reyle, U.: From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory. Kluwer Academic Publishers, Dordrecht (1993)
14. Kuhn, T.: Controlled English for Knowledge Representation. PhD thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich (2010)
15. Kuhn, T.: Codeco: A Practical Notation for Controlled English Grammars in Predictive Editors. In: Rosner, M., Fuchs, N.E. (eds.) CNL 2010. LNCS, vol. 7175, pp. 95–114. Springer, Heidelberg (2012)
16. Luts, M., Tikkerbär, D., Saarmann, M., Kutateladze, M.: Towards a Community-Driven Controlled Natural Languages Evolution. In: Rosner, M., Fuchs, N.E. (eds.) Pre-Proceedings of the Second Workshop on Controlled Natural Languages (CNL 2010). CEUR Workshop Proceedings. CEUR-WS, vol. 622 (2010)
17. Meza Moreno, M.S., Bringert, B.: Interactive Multilingual Web Applications with Grammatical Framework. In: Nordström, B., Ranta, A. (eds.) GoTAL 2008. LNCS (LNAI), vol. 5221, pp. 336–347. Springer, Heidelberg (2008)
18. Pool, J.: Can Controlled Languages Scale to the Web? In: 5th International Workshop on Controlled Language Applications (2006)
19. Ranta, A.: The GF Resource Grammar Library. Linguistic Issues in Language Technology 2(2) (2009)
20. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011) ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth)
21. Angelov, K., Ranta, A.: Implementing Controlled Languages in GF. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 82–101. Springer, Heidelberg (2010)

22. Ranta, A., Enache, R., Détrez, G.: Controlled Language for Everyday Use: The MOLTO Phrasebook. In: Rosner, M., Fuchs, N.E. (eds.) CNL 2010. LNCS, vol. 7175, pp. 115–136. Springer, Heidelberg (2012)
23. Schaffert, S., Bry, F., Baumeister, J., Kiesel, M.: Semantic wikis. IEEE Software 25(4), 8–11 (2008)
24. Schwitter, R., Ljungberg, A., Hood, D.: ECOLE — A Look-ahead Editor for a Controlled Language. In: Proceedings of EAMT-CLAW 2003, Joint Conference Combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop on Controlled Translation, May 15–17, pp. 141–150. Dublin City University, Ireland (2003)

FrameNet Resource Grammar Library for GF

Normunds Gruzitis, Peteris Paikens, and Guntis Barzdins

Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, Riga, LV-1459, Latvia
{normunds.gruzitis,peteris.paikens,guntis.barzdins}@lumii.lv

Abstract. In this paper we present an ongoing research investigating the possibility and potential of integrating frame semantics, particularly FrameNet, in the Grammatical Framework (GF) application grammar development. An important component of GF is its Resource Grammar Library (RGL) that encapsulates the low-level linguistic knowledge about morphology and syntax of currently more than 20 languages facilitating rapid development of multilingual applications. In the ideal case, porting a GF application grammar to a new language would only require introducing the domain lexicon – translation equivalents that are interlinked via common abstract terms. While it is possible for a highly restricted CNL, developing and porting a less restricted CNL requires above average linguistic knowledge about the particular language, and above average GF experience. Specifying a lexicon is mostly straightforward in the case of nouns (incl. multi-word units), however, verbs are the most complex category (in terms of both inflectional paradigms and argument structure), and adding them to a GF application grammar is not a straightforward task. In this paper we are focusing on verbs, investigating the possibility of creating a multilingual FrameNet-based GF library. We propose an extension to the current RGL, allowing GF application developers to define clauses on the semantic level, thus leaving the language-specific syntactic mapping to this extension. We demonstrate our approach by reengineering the MOLTO Phrasebook application grammar.

Keywords: controlled natural language, frame semantics, FrameNet, multilinguality, Grammatical Framework.

1 Introduction

Controlled natural languages (CNL) can be divided into two general types according to the formalist or the naturalist approach [1]. The formalist approach supports a deterministic, bidirectional mapping of CNL to a formal language like first-order logic (FOL) or, more commonly, to description logic, namely OWL (Web Ontology Language) [2], allowing the integration with existing tools for reasoning, consistency checking and model building. Although logic-based CNL provides a seemingly informal high-level means for knowledge representation, essentially it is still a formal language that is just as expressive as the corresponding formalism, and whose interpretation is deterministic (predictable). In contrast, in the naturalist approach possible

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 121–137, 2012.

ambiguities are decreased but not excluded, thus allowing for a wider coverage of NL and more informal applications, such as semantically precise machine translation within a CNL.

In other words, there are CNLs that have an underlying logic-based formalism defining the semantics of a text (e.g. Attempto Controlled English [3]), and there are CNLs that do not have an underlying logic-based formalism (e.g. MOLTO Phrasebook [4] for multilingual translation of touristic phrases). In the first case, the semantics of CNL statements is interpreted by both a human and a formal-logic reasoning machine. In the second case, the interpreter is primarily a human and possibly a domain-specific application that uses CNL, for example, for information retrieval from a predefined domain-specific database.

Grammatical Framework (GF) [5] is a categorial grammar formalism and a toolkit for programming multilingual grammar applications. It is similar to definite clause grammars (DCG) in Prolog in that both support parsing and synthesis using the same (categorical) grammar definition. Besides the grammar formalism itself, an important part of GF is its Resource Grammar Library (RGL) [6] that encapsulates the low-level linguistic knowledge about morphology and syntax of currently more than 20 languages (the number is constantly growing). RGL facilitates rapid development and porting of application grammars in many parallel languages: all GF resource grammars implement the same syntactic interlingua (API) enabling automatic translation among languages via the abstract syntax trees. In particular, it has been shown that GF is a convenient framework for rapid and flexible implementation of multilingual CNLs – both those rooted in a formal language like the FOL-based Attempto Controlled English [7] and those rooted in a relatively informal language like standard touristic phrases [4].

In the ideal case, porting a GF application grammar to a new language or domain would only require introducing the domain lexicon – translation equivalents that are interlinked via common abstract terms. While it is possible for a highly restricted CNL, e.g. for authoring and verbalizing OWL ontologies (as implemented, for example, in [8]), developing and porting a less restricted CNL requires more linguistic knowledge about the particular language, and more experience in GF (particularly, in using RGL). Specifying a lexicon of nouns (incl. multi-word units) is mostly straightforward, however, specifying the lexicon of verbs is typically the most complex task in terms of both inflectional paradigms and argument structure, and may require specifying the whole clause (as in Phrasebook). Thus adding verbs to a GF application grammar's lexicon in a foreign language (or for a novice or less resourced GF developer even in his mother tongue) might not be a straightforward task and might present a stumbling block for potential GF multilingual application developers. Therefore in this paper we are focusing on verbs, investigating the possibility of creating a multilingual FrameNet-based GF resource grammar library.

The rest of the paper is organized as follows. In Section 2 we briefly re-capture the relevant architectural principles of FrameNet. In Section 3 we similarly re-capture some relevant GF application grammar development principles, demonstrating the current approach with a detailed example. Section 4 modifies the previous example, describing our solution for integrating FrameNet into GF application grammars. Finally we conclude with a brief discussion on the potential and benefits of the proposed FrameNet library for seamless multilingual CNL development.

2 FrameNet

FrameNet [9] is a lexicographic database that describes word meanings based on the principles of frame semantics. The central idea of frame semantics is that word meanings must be described in relation to semantic frames [10]. Therefore, the *frame* and the *lexical unit* are the key components of FrameNet. A lexical unit in FrameNet terms is the combination of a lemma with a specific meaning – each separate meaning of a word represents a new lexical unit. In FrameNet, each lexical unit is related to a semantic frame that it is said to *evoke* a frame (see Figure 1 and Figure 2).

The semantic frame describes a certain situation and the participants of that situation that are likely to be mentioned in the sentences where the evoking lexical unit (referred to as frame *target*) appears. The semantic roles played by these participating entities are called *frame elements* (FE). FrameNet makes a differentiation between *core* frame elements and *peripheral* frame elements. In general, frame elements that are necessarily realized are core elements. Peripheral elements represent more general information such as time, manner, place, and purpose and are less specific to the frame. Nevertheless all FrameNet frame elements are local to individual frames. This avoids the commitment to a small set of universal roles, whose specification has turned out to be controversial in the past [11]. In order to account for actual similarities between frame elements in different frames FrameNet includes also a rich set of frame to frame and FE to FE relations.

Residence		*This frame has to do with people (the Residents) residing in Locations, sometimes with a Co-resident.*
Core FEs	**Co_resident**	*A person or group of people that the Resident is staying with or among.*
	Location	*The place in which somebody resides.*
	Resident	*The individual(s) that reside at the Location.*
Lexical units		camp.v, dwell.v, inhabit.v, live.v, lodge.v, occupy.v, reside.v, room.v, squat.v, stay.v

Fig. 1. A sample FrameNet frame (only core frame elements shown)

The frame descriptions are coarse-grained and generalize over lexical variation. Therefore lexeme-specific information is contained within lexical unit entries that are more fine-grained and contain a definition of the lexical unit, the syntactic realizations of each frame element and the valence patterns. A sense of a lemma (word meaning) can evoke a frame, and thus form a lexical unit for this frame, if this sense is syntactically able to realize the core frame elements that instantiate a conceptually necessary component of a frame [12].

In Figure 2, a simplified (summarized) lexical entry of 'to live' (`Residence`) is given: information on non-core FEs is excluded (the rest is summed up); for each valence model only the most frequent realization pattern is given; valence models that contain multiple FEs of the same type are excluded; valence models that have ap-

peared in the corpus only once are excluded; prepositional phrase patterns (PP) are not distinguished by particular prepositions.

FE	Total	Pattern
Co_resident	14	PP.Dep (86%)
Location	131	PP.Dep (81%)
Resident	143	NP.Ext (90%)

(a)

Total	Patterns		
98		**Location**	**Resident**
71%		PP.Dep	NP.Ext
7	**Co_resident**		**Resident**
86%	PP.Dep		NP.Ext
7	**Co_resident**	**Location**	**Resident**
86%	PP.Dep	PP.Dep	NP.Ext

(b)

Fig. 2. A simplified lexical entry Residence.live. (a) Core FEs and their most frequent syntactic patterns in the FrameNet corpus. (b) Most frequent valence models of core FEs.

Our central point of interest in this paper is the multilingual dimension of FrameNet. A number of projects have investigated the use of English FrameNet frames for other languages, such as German (SALSA project [13]), Spanish [14], Japanese [15], and lately also for Thai, Chinese, Italian, French, Bulgarian, Hebrew [16]. A fundamental assumption of these projects is that English FrameNet frames can be largely re-used for the semantic analysis of other languages. This assumption rests on the nature of frames as coarse-grained semantic classes which refer to prototypical situations – to the extent that these situations agree across languages, frames should be applicable cross-linguistically. Also Boas [17] suggests the use of semantic frames as interlingual representation for multilingual lexicons.

While FrameNet multilinguality is clearly a very attractive assumption, its empirical validation comes primarily from the German SALSA project, which has found that the vast majority of English FrameNet frames can be directly applied to the analysis of German – a language that is typologically close to English. Meanwhile, some frames have turned out not to be fully interlingual and three main cross-lingual divergence types were found:

1. Ontological distinctions between similar frame elements.
2. Missing frame elements.
3. Differences in lexical realization patterns (e.g. German 'fahren' does not distinguish between Operate_vehicle and Ride_vehicle frames).

Nevertheless this empirical evidence shows that most of FrameNet frames indeed are language independent and therefore provide an opportunity for use as a multilingual coarse-grained lexicon in GF, as described in Section 4. Although FrameNet addresses all parts-of-speech, its strength and focus is on verbs for which the best coverage is provided. This is largely because while noun-phrase multi-word units are extensively "invented" to denote nominal concepts (especially in technical domains), phrasal verbs are more fixed, commonly reused (across domains) and are often captured in dictionaries of standard language. Since the advantage of valence structures is

more obvious for verbs, in this paper we consider only FrameNet frames with verbal frame evoking lexical entries.

3 GF Application Grammar Development: The Current Approach

GF facilitates reusability by splitting the grammar development in two levels:

1. A general-purpose *resource grammar* covers a wide range of morphological paradigms and syntactic structures and as such is highly ambiguous. GF provides a Resource Grammar Library (RGL) [6] implementing a common API for more than 20 languages.
2. Domain specific *application grammars* reuse the RGL, defining semantic structures and the subset of natural language (syntax and lexicon) that is used within a particular CNL. Application grammars reduce or even eliminate ambiguities.

Development of a resource grammar requires in-depth GF knowledge and in-depth linguistic knowledge about the particular language. Once a resource grammar is provided, application grammars are built on top of it significantly reducing the linguistic knowledge prerequisites (a non-linguist native or fluent speaker should be sufficient), as well as he or she can be less experienced with GF.

GF differentiates not only between general-purpose resource grammars and domain-specific application grammars, but also between *abstract syntax* and *concrete syntax*. The abstract syntax captures the semantically relevant structure of a CNL, defining grammatical categories and functions for building abstract syntax trees [5]. The concrete syntax defines the linearization of the CNL abstract syntax trees at the surface level for each language. Translation among languages (concrete syntaxes) is provided via abstract syntax[1].

We will describe the current approach to the RGL-based GF application grammar development using the MOLTO Phrasebook application [4] for multilingual translation of touristic phrases as an example. Phrasebook is a CNL implemented in 15 languages and is aimed to be usable by anyone without prior training. It has 42 categories and 290 functions. The number of phrases it can generate is infinite, but on a reasonable level of tree depth 3, Phrasebook has nearly 500,000 abstract syntax trees [4]. We will consider only a small subset of the Phrasebook grammar – categories (`cat`) and functions or constructors (`fun`) that are used to build the abstract syntax trees for the following sample sentences, and to generate these sentences from the corresponding abstract trees[2] as given in Figure 3.

In the next section we will modify the English implementation of the Phrasebook grammar by means of the proposed FrameNet-based resource grammar, acquiring a

[1] Note that in GF there is no concept of a language pair or a translation direction. Also there is no common semantic interlingua. Instead there are many application specific (i.e., CNL and domain specific) interlinguas.

[2] The provided abstract syntax trees are slightly simplified regarding the pronouns – their gender, number and politeness features – to avoid multiple variants.

simpler English Phrasebook (PhrasebookEng) implementation as the result, while preserving the same functionality. However, we will not make any changes neither in the Phrasebook functor (the common incomplete concrete syntax), nor in the abstract syntax, i.e., we will not impose any special requirements on application grammar design.

English sentences	Phrasebook abstract syntax
I like this pizza.	`PSentence (SProp (PropAction (ALike I (This Pizza))))`
I live in Belgium.	`PSentence (SProp (PropAction (ALive I Belgium)))`
I love you.	`PSentence (SProp (PropAction (ALove I You)))`
I want a good pizza.	`PSentence (SProp (PropAction (AWant I (OneObj (ObjIndef (SuchKind (PropQuality Good) Pizza))))))`
I want to go to a museum.	`PSentence (SProp (PropAction (AWantGo I (APlace Museum))))`

Fig. 3. Sample Phrasebook sentences along with their abstract syntax trees

The abstract syntax of Phrasebook that represents the syntactic and semantic model of the above phrases is given in Figure 4.

```
cat
  Action ;     -- proposition about a Person, e.g. "I love you"
  Phrase ;     -- complete phrase, e.g. "I love you."
  Country ;    -- e.g. "Belgium"
  Item ;       -- single entity, e.g. "this pizza"
  Kind ;       -- kind of an item, e.g. "pizza"
  Object ;     -- e.g. "a good pizza"
  Person ;     -- agent wanting or doing something, e.g. "I"
  Place ;      -- location, e.g. "a museum"
  PlaceKind ;  -- kind of location, e.g. "museum"
  Property ;   -- basic property of an item, e.g. "good"

fun
  Belgium : Country ;
  Good    : Property ;
  Museum  : PlaceKind ;
  Pizza   : Kind ;

  ALike   : Person -> Item    -> Action ;  -- Action(Person, Item)
  ALive   : Person -> Country -> Action ;
  ALove   : Person -> Person  -> Action ;
  AWant   : Person -> Object  -> Action ;
  AWantGo : Person -> Place   -> Action ;
```

Fig. 4. A fragment of Phrasebook abstract syntax (semantic model)

A fragment of the incomplete concrete syntax (aka functor) of Phrasebook is given in Figure 5. This is a technical intermediate layer between the abstract syntax and its implementation in concrete syntaxes. It defines language-independent syntactic categories and structures (e.g. `PSentence`, `SProp`, `PropAction`) that are common to all (or most) languages. Thus the concrete syntax of a particular language has to specify only language-dependent structures and the lexicon (e.g. `AWant`, `Good`, `Pizza`).

Note that the functor defines the mapping between the application-specific abstract syntax categories and the categories of the Resource Grammar Library. For instance, `Country`, `Item` and `Object` syntactically are realized as noun phrases (category `NP` in RGL). In Figure 5, there are also three Phrasebook categories that are not directly mapped to RGL categories (`Person`, `Place` and `PlaceKind`). Instead, they are defined as application-specific categories `NPPerson`, `NPPlace` and `CNPlace` that are specified as record types whose fields (e.g. `name`, `at`, `to`) are of RGL types.

```
lincat  -- category linearization types
  Phrase = Text ;
  Action = Cl ;
  Country, Item, Object = NP ;
  Person = NPPerson ;
  Place = NPPlace ;
  Kind = CN ;
  PlaceKind = CNPlace ;
  Property = A ;

oper  -- operations - functions in concrete syntax
  NPPerson : Type = {name : NP ; isPron : Bool ; poss : Quant} ;
  NPPlace  : Type = {name : NP ; pos : Adv ; dir : Adv} ;
  CNPlace  : Type = {name : CN ; pos : Prep ; dir : Prep} ;

  mkNPPerson : Pron -> NPPerson = \pron ->
    {name = mkNP pron ; isPron = True ; poss = mkQuant pron} ;

  mkCNPlace : CN -> Prep -> Prep -> CNPlace = \cn,prep1,prep2 ->
    {name = cn ; pos = prep1 ; dir = prep2} ;

  mkNPPlace : Det -> CNPlace -> NPPlace = \det,place ->
    let name : NP = mkNP det place.name in {
      name = name ;
      pos = mkAdv place.pos name ;  -- place - position
      dir = mkAdv place.dir name    -- place - direction
    } ;
```

Fig. 5. A fragment of Phrasebook incomplete concrete syntax (functor): common structures. To make the code more intelligible to readers unfamiliar with GF, it has been slightly modified.

The predication patterns (`Action`) with verbs at the centre are perhaps the most complex functions in Phrasebook (from the implementation point of view). Note that these actions are of type `Cl` (clause): this will be the gluing point for the integration of the FrameNet-based resource library (see Section 4). The linguistic (English) realization of the semantic model is specified by the concrete syntax (given in Figure 6),

which tells how abstract syntax trees are linearized (`lin`) into English strings. The same rules are also used for parsing.

```
lin  -- function linearization rules
  Belgium = mkNP (mkPN "Belgium") ;
  Good = LexiconEng.good_A ;
  Museum = mkPlaceKind "museum" "at" ;
  Pizza = mkCN (mkN "pizza") ;

  ALike pers item = mkCl pers.name (mkV2 (mkV "like")) item ;
  ALive pers country = mkCl pers.name (mkVP (mkVP (mkV "live"))
    (mkAdv SyntaxEng.in_Prep country)) ;
  ALove pers1 pers2 =
    mkCl pers1.name (mkV2 (mkV "love")) pers2.name ;
  AWant pers obj = mkCl pers.name (mkV2 (mkV "want")) obj ;
  AWantGo pers place = mkCl pers.name SyntaxEng.want_VV
    (mkVP (mkVP IrregEng.go_V) place.dir) ;

oper
  mkPlaceKind : Str -> Str -> CNPlace = \name,prep_pos ->
    mkCNPlace (mkCN (mkN name)) (mkPrep prep_pos) SyntaxEng.to_Prep ;
```

Fig. 6. A fragment of Phrasebook concrete syntax for English

Verbs, in general, are at the centre of a sentence, both syntactically and semantically. They have the most complex inflectional paradigms (at least in inflective languages). The syntactic and semantic valence of a verb is defined via its argument and modifier structure. This inevitably requires solid linguistic knowledge.

RGL differentiates among V (intransitive), V2 (transitive) and V3 (ditransitive) verbs, as well as some more specific types of verbs with syntactically fixed argument structure. The syntactic valence patterns for the predefined verb types are fixed when defining a verb in the application lexicon; these patterns do not depend on the argument (i.e. the case or preposition of the argument does not depend on a particular NP). Other valences are specified while constructing a verb phrase – as adverbial modifiers (`Adv`); their syntactic patterns are specified by the application developer for each target language, depending on the semantic role of the argument and syntactic properties of the language.

If compared to nouns, there are much less verbs and they are more ambiguous (see WordNet statistics[3], for example), thus verbs are also more reusable linguistic units than nouns. This suggests that a reusable lexicon of verbs would be helpful for GF application developers. However, this requires not only a dictionary, but also additional information about the basic syntactic valences for the direct and indirect objects. Even more helpful would be a multilingual resource grammar of verb valences.

There is a small, limited multilingual lexicon provided by the RGL, but it does not provide systematic means for scaling and expanding beyond the V-, V2- and V3-like

[3] `http://wordnet.princeton.edu/wordnet/man/wnstats.7WN.html`

valences. The basic lexicon also does not support polysemous verbs – valences often are different for various meanings of the same verb, and vice versa.

An impression of the syntactic coverage of the GF Resource Grammar Library can be obtained from its API documentation[4]. Figure 7 illustrates some of the constructors for clauses, verb phrases, noun phrases, common nouns, and adverbial modifiers that are referred in Figure 5 and Figure 6. For instance, Figure 7 illustrates that a clause can be built from a subject noun phrase with a verb and appropriate arguments. In general, a clause can be built from a subject noun phrase and a verb phrase.

Function	Type	Example
mkCl	NP -> VP -> Cl	*she always sleeps*
mkCl	NP -> V2 -> NP -> Cl	*she loves him*
mkCl	NP -> VV -> VP -> Cl	*she wants to sleep*
mkVP	VP -> Adv -> VP	*to sleep here*
mkNP	Det -> CN -> NP	*the old man*
mkNP	PN -> NP	*Paris*
mkNP	Pron -> NP	*we*
mkCN	N -> CN	*house*
mkAdv	Prep -> NP -> Adv	*in the house*

Fig. 7. A fragment of the Resource Grammar API documentation

In order to use the proposed FrameNet library (in addition to the Resource Grammar API), the application grammar developer will have to consult the FrameNet API as presented in the next section.

4 Our FrameNet-Based Approach

The current split of functionality between the application and the common libraries expects applications to define the domain specific knowledge in all languages by using specific verbs and defining their syntactic valences in each target language.

Our proposal is to raise the abstraction level for the common GF clause construction API from the current syntactic definition to a more semantic one. As we discussed in Section 2, the research on frame semantics suggests that an exhaustive cross-domain linguistic model of semantic frames and roles is possible, and it has been implemented for multiple languages. We believe that it is possible and reasonable to facilitate the development of multilingual application grammars in GF by referencing a common API of semantic frames that provide language-specific linearization for whole clauses or verb phrases (VP), and can optionally provide a default choice of a lexical unit that evokes the frame and default syntactic valence patterns.

In particular, we envision a resource grammar library that is built on top of the current RGL offering each of the FrameNet's semantic frames as a function that builds a

[4] http://www.grammaticalframework.org/lib/doc/synopsis.html

clause from given parameters: fillers of the core elements of that frame, and an optional list of elements filling the peripheral roles. The FrameNet API would be implemented semi-automatically by generating GF code from FrameNet data providing a set of overloaded functions for each frame – mapping the frame (its elements) to the default or specific syntactic realization (linearization). Our observation is that, in the current approach to GF application development, a miniature ad-hoc 'framenet' is actually implemented for each application. Moreover, it is often 'reused' in a copy-paste-edit manner from previous applications or from concrete syntaxes of other languages that implement the same application. We would like to promote systematic means for reusing this language-specific knowledge via common language-independent frames.

The following simplified assumptions underlie the default behaviour of our approach (the default behaviour can be overridden for specific syntactic patterns and lexical units – see systematic "exceptions" illustrated below):

1. For each frame element there is a typical syntactic pattern that is used in most cases – independently of the verb that evokes the frame. I.e., both semantic and syntactic valences can be defined at the frame level.
    1.1. There is a common syntactic realization of a frame (a clause or a verb phrase) that is reused by most verbs that evoke the frame.
2. It is possible to specify a default lexical unit (the most general and/or the most frequently used verb) that evokes the frame, so that it can be used in the linearization (translation) of the frame, if no specific verb is provided.
3. In the CNL settings, it is often sufficient that only core semantic valences (core frame elements according to FrameNet) are available.

These assumptions, of course, do not hold in general, but they help us to keep the presentation of our approach simpler. Even then we cannot fully isolate the application developer from providing some language-specific features. For example, the Phrasebook application in English (and similarly in Russian) needs to distinguish between locations that are "at place" or "in place" – the preposition does not depend on the frame and not even on the specific verb, but on the particular noun (the filler of a frame element). In contrast to the highly analytical English, in many languages it might be necessary to customize the realization of the whole clause, depending on the verb. For example, in Latvian (and similarly in Russian, Italian and German) there are verbs (systematic "exceptions") that instead of the subject in the nominative case and the object in the accusative case require the subject in the dative case and the object in the nominative case[5] (see Figure 8). In the actual implementation of the FrameNet RGL, such agreement variations have to be handled by alternative verb-specific clauses implemented in the frame functions.

[5] Here we use the term 'case' in a broad sense: in Italian, for example, there are no cases for nouns; cases are expressed by prepositions, pronouns and implicitly by word order.

	LOVE	LIKE
English	$I_{[NOM]}$ love $pizza_{[ACC]}$	$I_{[NOM]}$ like $pizza_{[ACC]}$
German	$Ich_{[NOM]}$ liebe $Pizza_{[ACC]}$	$Ich_{[NOM]}$ mag $Pizza_{[ACC]}$ $Mir_{[DAT]}$ gefällt $Pizza_{[NOM]}$
Italian	$Io_{[NOM]}$ amo la $pizza_{[ACC]}$	A $me_{[DAT]}$ piace la $pizza_{[NOM]}$
Latvian	$Es_{[NOM]}$ mīlu $picu_{[ACC]}$	$Man_{[DAT]}$ patīk $pica_{[NOM]}$
Russian	$Я_{[NOM]}$ люблю $пиццу_{[ACC]}$	$Мне_{[DAT]}$ нравится $пицца_{[NOM]}$

Fig. 8. Verb-specific realization of the frame elements `Experiencer` and `Content` in different languages. All these verbs belong to the `Experiencer_focus` frame.

As in the case of the syntactic RGL, the proposed semantic resource grammars of the FrameNet library will also be ambiguous as such: the same verb can evoke different frames, and the same frame might be evoked by contradicting verbs (e.g. both 'to love' and 'to hate' evoke the same `Experiencer_focus` frame). However, the intuition is that the developer of a domain-specific CNL will reduce or eliminate the semantic ambiguity by avoiding ambiguous mappings between lexical units and frames, by specifying concrete verb lexemes instead of relying on the default ones etc. – analogically as it is currently done at the syntactic level.

To illustrate the use of the FrameNet API, we provide a sample re-implementation of some clause-building functions from the MOLTO Phrasebook application (see Figure 9) in contrast to the current `PhrasebookEng` implementation of the same functions as shown earlier in Figure 6.

Before:

```
ALike pers item = mkCl pers.name (mkV2 (mkV "like")) item ;
ALive pers country = mkCl pers.name (mkVP (mkVP (mkV "live"))
  (mkAdv SyntaxEng.in_Prep country)) ;
ALove pers1 pers2 =
  mkCl pers1.name (mkV2 (mkV "love")) pers2.name ;
AWant pers obj = mkCl pers.name (mkV2 (mkV "want")) obj ;
AWantGo pers place = mkCl pers.name SyntaxEng.want_VV
  (mkVP (mkVP IrregEng.go_V) place.dir) ;
```

After:

```
ALike pers item =
  Experiencer_focus (mkV "like") pers.name item NIL NIL ;
ALive pers country = Residence pers.name NIL country ;
ALove pers1 pers2 =
  Experiencer_focus (mkV "love") pers1.name pers2.name NIL NIL ;
AWant pers obj = Possession (mkV "want") pers.name obj ;
AWantGo pers place =
  Desiring pers.name (Motion_VP IrregEng.go_V NIL place.name) ;
```

Fig. 9. Changes to the `PhrasebookEng` syntax using the proposed FrameNet API

As seen in Figure 9, the application grammar developer still has to provide the domain-specific knowledge that the application requires, and some simple constructors of the GF RGL are still used, but the code is more intelligible and 'flat' – it is not specified how the parameters (frame elements) are glued together to build up verb phrases and clauses[6]. The proposed API refers to the semantic roles only: if the user specifies, for example, the resident of the `Residence` frame (`ALive` action in Phrasebook), the FrameNet library maps it to the relevant syntactic role (subject in this case). Thus the verb and clause building part of application grammars such as Phrasebook is in essence reduced to mapping domain-specific concepts to the appropriate general FrameNet frames, and to specifying the omitted core frame elements, if any (`NIL`[7]).

In multilingual applications, there is a general issue of selecting lexical units – translation equivalents. For example, for the `Residence` frame, there are many possible verbs that describe the same situation with various semantic nuances (e.g. 'to camp', 'to dwell', 'to live', 'to stay'; see Figure 1). If these differences are relevant to the application domain, then a particular lexical unit can be explicitly specified. If the differences are not considered important for a particular use-case or concept, the preferred lexical unit for the chosen frame can be omitted, resulting in a robust system that would use a default verb (e.g. 'to live') when generating a text, and that would allow all frame-relevant verbs in parsing.

An advantage of this approach is the ability to build robust multilingual CNL applications without expertise in all covered languages. The benefit of using GF is that it would be possible to port such applications to other languages without going into details of their grammars – as they are already implemented in the common RGL. Furthermore, it is possible to omit the details about how the semantic roles are mapped to syntactic elements, as the same semantic element may be expressed by different syntactic means when translating the same clause to another language.

The API of the proposed FrameNet RGL is illustrated in Figure 10 (similarly as the API of GF RGL in Figure 7). The function names match the FrameNet frame names, thus the API can be automatically documented by FrameNet data providing definitions and examples for each frame (function) and each frame element (argument of the function).

Although currently we have handcrafted the code of the sample FrameNet library, we have done it systematically using the actual FrameNet data that is well structured and includes statistics from a FrameNet-annotated corpus. This has given confidence that FrameNet data can be used to automatically generate both the abstract syntax of the FrameNet API and its implementation for English and other languages using the current GF RGL syntactic categories and constructors, and properly addressing verb-specific valence patterns.

---

[6] Note that the implementation of the `AWantGo` function is not 'flat' – there are nested frames. I.e., it might be necessary to specify the semantic tree structure, but not the syntactic structure.

[7] We have not specified the implementation of `NIL` arguments yet, but this is only a technical mater.

We have performed some initial experiments on automatic GF code generation from FrameNet data, but the development of a more elaborated convertor is pending. Nevertheless, there are only about 1000 frames in FrameNet, therefore the generated code can also be manually debugged and improved afterwards.

Function/Frame	Type	Mapping to FEs
Residence	V -> NP -> PP -> Adv -> Cl	Resident → Co_resident → Location
	V -> NP -> NIL -> Adv -> Cl	
	NP -> NIL -> NP -> Cl	
Possession	V -> NP -> NP -> Cl	Owner → Possession
	NP -> NP -> Cl	
Desiring	VV -> NP -> VP -> Cl	Experiencer → Event
	NP -> VP -> Cl	
Motion	V -> NP -> NP -> NP -> Cl	Theme → Source → Goal
	NP -> NP -> NP -> Cl	
Motion_VP	V -> NP -> NP -> VP	Source → Goal
	V -> NIL -> NP -> VP	
	NP -> NP -> VP	
	Adv -> Adv -> VP	
Experiencer_focus	V -> NP -> NP -> VP -> NP -> Cl	Experiencer → Content → Event → Topic
	V -> NP -> NP -> NIL -> NIL -> Cl	
	NP -> NP -> NIL -> NIL -> Cl	

Fig. 10. A simplified fragment of the proposed FrameNet API. The `Desiring` frame has actually four core elements, and `Motion` – seven. Also all the possible combinations of `NP`, `PP`, `Adv` and `NIL` argument types are not included. Note that the `Motion_VP` is a special case of `Motion` – generated for use as a nested frame (as the `VP` object of a `VV` verb).

The manually generated code for several FrameNet frames, as shown in Figure 11, implements the features in a very similar manner as the Phrasebook application shown earlier in Figure 6 – which is to be expected, as it needs to realize similar syntactic structures with the same GF resources. However, a major difference is that this code would be reusable for multiple applications, and it could cover larger domains in a scalable way.

There are still some technical issues that need to be addressed, such as a more convenient way for specifying omitted core frame elements, but we believe that these are minor challenges. A particular concern is common peripheral semantic roles such as Time, Place and Manner that are encountered in nearly all frames (if they are not among the core roles for that frame). Again, the current Resource Grammar API deals with them on a syntactic level – providing means to attach various adverbial modifiers. We propose adding them as a (possibly empty) list of peripheral parameters, allowing the language-specific API implementation to handle the word order changes as needed.

```
-- Residence : NP -> NIL -> NP -> Cl
Residence resident NIL location = Residence (mkV "live") resident NIL
  (mkAdv SyntaxEng.in_Prep location) ;

-- Residence : V -> NP -> NIL -> Adv -> Cl
Residence verb resident NIL location =
  mkCl resident (mkVP (mkV "live") location) ;

-- Residence : V -> NP -> PP -> Adv -> Cl
Residence verb resident co_resident location = mkCl resident
  (mkVP (mkVP (mkV2 verb co_resident.prep) co_resident.np) location) ;

-- Possession : V -> NP -> NP -> Cl
Possession verb owner possession =
  mkCl owner (mkVP (mkV2 verb) possession) ;

-- Desiring : NP -> VP -> Cl
Desiring experiencer event =
  Desiring SyntaxEng.want_VV experiencer event ;

-- Desiring : VV -> NP -> VP -> Cl
Desiring verb experiencer event = mkCl experiencer verb event ;

-- Motion : V -> NP -> NP -> NP -> Cl
Motion verb theme source goal = mkCl theme (Motion_VP verb source goal) ;

-- Motion_VP : NP -> NP -> VP
Motion_VP source goal = Motion_VP (mkV "move") source goal ;

-- Motion_VP : V -> NP -> NP -> VP
Motion_VP verb source goal = mkVP (
  (mkVP (mkVP verb) (mkAdv SyntaxEng.from_Prep source))
  (mkAdv SyntaxEng.to_Prep goal)) ;

-- Motion_VP : V -> NIL -> NP -> VP
Motion_VP verb NIL goal =
  mkVP (mkVP verb) (mkAdv SyntaxEng.to_Prep goal) ;

-- Experiencer_focus : V -> NP -> NP -> NIL -> NIL -> Cl
Experiencer_focus verb experiencer content NIL NIL =
  mkCl experiencer (mkV2 verb) content ;
```

Fig. 11. English implementation of the proposed FrameNet API (a simplified fragment). `PP` extends the RGL set of categories; its linearization type is `{prep : Prep ; np : NP}`.

5 Discussion and Future Work

Currently the GF toolset provides a reusable syntactic framework for the development of multilingual domain-specific CNLs. When one acquires a solid understanding of

the RGL structure and design principles, and gets used to the RGL-based application grammar design patterns, it is a rather rapid development to provide a concrete syntax for a language he or she knows well[8]. However, the process still might not be straightforward, especially when porting a third-party application, as it might not be enough to look at the code of e.g. English implementation to (immediately) understand the intended meaning of a specific abstract word or clause to provide an appropriate translation. In addition, different application grammars that cover related domains will more or less overlap, so that the same structures are re-implemented for each application.

In the current approach, GF application grammar developers essentially provide a miniature domain-specific framenet for each application. We make a case for basing application development on a common, reusable semantic framework, and argue that it is reasonably possible to develop such a framework by leveraging the existing FrameNet data. Working on the semantic level requires specific knowledge and training as well[9], but the resulting systems are more generic and easier to reuse across languages and across applications and domains.

The proposed approach is aimed to lower the entrance barrier of the GF application grammar development by moving it from the language-specific syntactic level towards the language-independent semantic level. The long-term goal is to facilitate the development of multilingual applications by providing robust means for automatic alignment of translation equivalents (particularly verbs), and by reducing syntactic and lexical ambiguities that appear in the parsing and generation of less restricted CNLs. GF has been chosen as an advanced and well-resourced framework for this purpose, but the proposed general principle could be applied also to other grammar formalisms.

The main limitations of the proposed approach to some extent are related to the limitations of FrameNet, particularly its coverage (in terms of lexical units). Furthermore, the coverage might differ among languages. The list of the lexical units for each frame could be extended via WordNet, as it has been shown by Johansson and Nugues [18], however then we would have to fall back to the frame-specific (vs. verb-specific) valence patterns. Another limitation is that even the most frequently used verb-specific valence patterns might not be appropriate in specific cases.

Although we have tested our proposal only on the English FrameNet data and English Phrasebook grammar, considering other languages only theoretically, we believe that in overall this would ease the multilingual GF application development, and that the limitations can be overcome by using the RGL syntactic structures directly where necessary. By relying on the default syntactic realization and the default lexical units, one can quickly obtain the first working version of a multilingual aplication for further testing and tuning. In fact, the default behaviour can be specified already in the

[8] The experience of the MOLTO team shows that adding a new language to Phrasebook takes 1.5 days on average.

[9] GF developers would have to explore and consult the documentation of FrameNet data (`https://framenet.icsi.berkeley.edu/fndrupal/framenet_data`) while designing or porting an application grammar.

functor that defines the common language-independent structures of an application grammar.

Apart from the development of the GF code generation facility from FrameNet data and apart from a wider evaluation taking into account both more languages and more applications, future work is also to investigate the possibilities for semi-automatic multilinguality by choosing the most appropriate lexical units automatically, and by aligning lexical units (translation equivalents) among different languages.

An additional direction for future work is the application of this semantic layer to relatively unrestricted natural language – in line with the naturalist approach [1]. Angelov [19] has demonstrated the potential of the current GF Resource Grammar Library in statistical parsing of unrestricted texts (using weights extracted from a treebank). FrameNet data would provide additional means in disambiguation and would provide mapping of parse results to semantic categories. Also Barzdins [20] has addressed bridging the gap between the CNL and full natural language through use of FrameNet on the discourse level. Integration with frame semantics thus provides additional means towards semantic parsing of less controlled text.

Acknowledgments. This work has been supported by the European Regional Development Fund under the project No. 2011/0009/2DP/2.1.1.1.0/10/APIA/VIAA/112. The authors would like to thank the reviewers for the detailed comments and constructive criticism.

References

1. Clark, P., Murray, W.R., Harrison, P., Thompson, J.: Naturalness vs. Predictability: A Key Debate in Controlled Languages. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 65–81. Springer, Heidelberg (2010)
2. Schwitter, R., Kaljurand, K., Cregan, A., Dolbear, C., Hart, G.: A Comparison of three Controlled Natural Languages for OWL 1.1. In: Proceedings of the 4th International Workshop on OWL Experiences and Directions (OWLED), CEUR, vol. 496 (2008)
3. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
4. Ranta, A., Enache, R., Détrez, G.: Controlled Language for Everyday Use: The MOLTO Phrasebook. In: Rosner, M., Fuchs, N.E. (eds.) CNL 2010. LNCS (LNAI), vol. 7175, pp. 115–136. Springer, Heidelberg (2012)
5. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011)
6. Ranta, A.: The GF Resource Grammar Library. Linguistic Issues in Language Technology 2(2) (2009)
7. Angelov, K., Ranta, A.: Implementing Controlled Languages in GF. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 82–101. Springer, Heidelberg (2010)

8. Gruzitis, N., Barzdins, G.: Towards a More Natural Multilingual Controlled Language Interface to OWL. In: Proceedings of the 9th International Conference on Computational Semantics (IWCS), Oxford, pp. 335–339 (2011)
9. Baker, C.F., Fillmore, C.J., Lowe, J.B.: The Berkeley FrameNet project. In: Proceedings of the COLING-ACL, Montreal, pp. 86–90 (1998)
10. Fillmore, C.J., Johnson, C.R., Petruck, M.R.L.: Background to FrameNet. International Journal of Lexicography 16(3), 235–250 (2003)
11. Burchardt, A., Erk, K., Frank, A., Kowalski, A., Pado, S., Pinkal, M.: Using FrameNet for the semantic analysis of German: Annotation, representation, and automation. In: Boas, H.C. (ed.) Multilingual FrameNets in Computational Lexicography: Methods and Applications, pp. 209–244. Mouton de Gruyter, Berlin (2009)
12. Ruppenhofer, J., Ellsworth, M., Petruck, M.R.L., Johnson, C.R., Scheffczyk, J.: FrameNet II: Extended Theory and Practice (2010)
13. Burchardt, A., Erk, K., Frank, A., Kowalski, A., Pado, S., Pinkal, M.: The SALSA corpus: a German corpus resource for lexical semantics. In: Proceedings of the 5th International Conference on Language Resources and Evaluation (2006)
14. Subirats, C.: Spanish FrameNet: A frame-semantic analysis of the Spanish lexicon. In: Boas, H.C. (ed.) Multilingual FrameNets in Computational Lexicography: Methods and Applications. Mouton de Gruyter, Berlin (2009)
15. Kyoko, O., Fujii, S., Ishizaki, S., Ohori, T., Saito, H., Suzuki, R.: The Japanese FrameNet Project: an Introduction. In: Proceedings of the Workshop on Building Lexical Resources from Semantically Annotated Corpora (at LREC), Lisbon, pp. 9–11 (2004)
16. Leenoi, D., Jumpathong, S., Porkaew, P., Supnithi, T.: Thai FrameNet Construction and Tools. International Journal on Asian Language Processing 21(2), 71–82 (2011)
17. Boas, H.C.: Semantic frames as interlingual representations for multilingual lexical databases. International Journal of Lexicography 18(4), 445–478 (2005)
18. Johansson, R., Nugues, P.: Using WordNet to Extend FrameNet Coverage. In: Proceedings of the Workshop on Building Frame Semantics Resources for Scandinavian and Baltic Languages (at NODALIDA), Tartu, pp. 27–30 (2007)
19. Angelov, K.: The Mechanics of the Grammatical Framework. PhD Thesis, Chalmers University of Technology and University of Gothenburg (2011)
20. Barzdins, G.: When FrameNet meets a Controlled Natural Language. In: Proceedings of the 18th Nordic Conference on Computational Linguistics (NODALIDA), Riga, pp. 2–5 (2011)

Legislative Drafting Guidelines: How Different Are They from Controlled Language Rules for Technical Writing?

Stefan Höfler

University of Zurich, Institute of Computational Linguistics,
Binzmühlestrasse 14, 8050 Zurich, Switzerland
hoefler@cl.uzh.ch

Abstract. While human-oriented controlled languages developed and applied in the domain of technical documentation have received considerable attention, language control exerted in the process of legislative drafting has, until recently, gone relatively unnoticed by the controlled language community. This paper considers existing legislative drafting guidelines from the perspective of controlled language. It presents the results of a qualitative comparison of the rule sets of four German-language legislative drafting guidelines from Austria, Germany and Switzerland with a representative collection of controlled language rules published by the German Professional Association for Technical Communication. The analysis determines the extent to which the respective rule sets control the same or similar aspects of language use and identifies the main differences between legislative drafting guidelines and controlled language rules for technical writing.

Keywords: human-oriented controlled natural language, legislative drafting, technical writing.

1 Introduction

Controlled languages have been described as restricted versions of natural languages: they constrain the words, phrases, syntactic constructions etc. that may be used in the composition of a text. Such restrictions have been put in place with different aims in mind: (i) to make it easier for humans to read and interpret a text, (ii) to facilitate translation (manually or by machine) into other languages, or (iii) to allow for a direct mapping onto some formal semantic representation accessible to automated reasoning.[1] Some researchers have consequently proposed an ideal-typical distinction between human-oriented controlled languages and machine-oriented controlled languages [14,20,21,24,27,29]. While controlled languages grounded in formal logic have mostly been developed for the purpose of requirements engineering and computer-assisted knowledge representation [9,26], controlled languages aimed at improving the understandability and translatability of texts have mainly been applied in the context of technical writing [11,17]. Some controlled languages, among them the ones based on formal logic, prescribe a relatively restrictive set of words and syntactic constructions that may

[1] Occasionally, a fourth aim has been mentioned: to make texts more consistent. It can be seen as a means to achieving any one of the three aims listed above.

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 138–151, 2012.

be used, and prohibit the use of anything else; others, especially most human-oriented controlled languages, take a more permissive approach and confine themselves to designating words and constructions that must *not* be used or only be used in a certain way, thus implicitly allowing all the rest [15].

While human-oriented controlled languages developed and applied in the domain of technical documentation have received considerable attention, the writing standards set up by guidelines for legislative drafting have gone relatively unnoticed by the controlled language community. This is somewhat surprising given that, at first glance, legislative drafting guidelines pursue aims that are similar to those pursued by controlled languages for technical writing: improving the understandability of texts containing instructions (legal instructions in the case of the former, technical instructions in the case of the latter). In this undertaking, both are bound to finding a trade-off between keeping things simple and being precise.[2]

This paper presents the results of a comparison of the rule sets of German-language legislative drafting guidelines from Austria, Germany and Switzerland with the compilation of controlled language rules published by the German Professional Association for Technical Communication (*Gesellschaft für Technische Kommunikation e.V. – tekom*). The analysis was aimed at determining the extent to which the respective rule sets control the same or similar aspects of language use and at identifying the characteristics that distinguish legislative drafting guidelines from controlled language rules for technical writing. The motivation behind providing such information was to get a better idea of whether and how the two domains can inform each other: whether, for instance, it makes sense for one domain to borrow rules from the other,[3] and whether some functions offered by automated language checkers that were developed for the domain of technical writing could also be of use to the process of legislative editing.[4]

The paper is organised as follows. Section 2 introduces the rule sets that were analysed for the current study. Section 3 introduces and compares the linguistic phenomena the rule sets control. Section 4 discusses the results of the comparison with regard to the scope of the rule sets, their domain-specificity and the operationalisability of the respective rules.

2 The Rule Sets

2.1 The Tekom Standard

In 2011, the German Professional Association for Technical Communication (*Gesellschaft für Technische Kommunikation e.V. – tekom*[5]) published a compilation of

[2] The conflict between precision and simplicity and the extent to which legislation can and should be understandable to non-expert citizens has been the subject of extensive and controversial debates [7,16].

[3] The research question approached by the present work is loosely related to the question investigated in the comparative analysis presented by O'Brien [20]: the objective of O'Brien's study was to find out to what extent a range of (mostly machine-oriented) controlled language rule sets for English shared common rules.

[4] The challenges in developing such a domain-specific controlled language checker for Swiss German-language legislative drafts have been described in [13].

[5] `http://www.tekom.de`

the most common field-tested controlled language rules for technical writing [1]. The compilation represents the state of the art in German-language technical writing and is intended to serve both as an industry standard and as a source of reference for professionals and researchers. The rules it provides constitute building blocks from which companies can develop their own in-house controlled languages. The tekom standard has been chosen as a reference rule set for the present comparison because (i) it is representative of the language control typically employed in German-language technical writing, (ii) it is recent and reflects the state of the art and (iii) it is both grounded in professional experience and backed up by linguistic research.

The compilation comprises 39 rules concerned with sentence construction and 29 rules dealing with issues at the textual level. In addition, it provides 27 rules on spelling and word formation. However, as the standardisation of spelling is only of marginal interest to controlled language research, these last rules have not been considered in the present study. With the exception of suggestions referring to the use of specific function words, the tekom standard does not deal with terminology and vocabulary control; it focuses on issues that can be captured in the form of rules. The same holds for the analysis presented in this paper.

2.2 The Legislative Drafting Guidelines

Four sets of German-language legislative drafting guidelines have been included in the comparison, namely those contained in the legislation manuals of:

- the Austrian federal administration [4],
- the German federal administration [5],
- the state administration of the Swiss canton of Bern [22],
- the state administration of the Swiss canton of Zurich [23].

The legislation manuals of the Swiss federal administration [3] and of the European Parliament, Council and Commission [8] were also considered originally. However, in contrast to the four texts listed above, the passages on language contained in these two manuals mostly remain at the level of abstract writing principles and offer only few specific drafting rules. Thus, they were less suited for a comparison with the tekom standard and have not been included in the analysis.

Rules concerned with language only make up a small part of the aforementioned legislation manuals; the bulk of the rules deal with the formal and legal requirements that newly drafted statutes and regulations must fulfil. For the present study, however, only language-related rules have been considered. These are typically compiled in a specific chapter on legislative language and in sections on text organisation and the use of intra- and inter-textual cross references.

3 Analysis

In order to be able to compare the five rule sets introduced above, the rules were grouped into the two broad categories proposed in the tekom standard: sentence-level rules and text-level rules. The rules in each group were then further sub-categorised with regard

to the linguistic phenomena they control. The sub-categories are listed in Tables 1 and 2. This classification made it possible to determine whether certain phenomena are controlled (i) by all rule sets (even if the specific rules applied to these phenomena may differ) or (ii) only by some or all of the legislative drafting guides but not by the tekom standard or (iii) only by the tekom standard but not by any or all of the legislative drafting guides. The results of the comparison of sentence-level rules and text-level rules are discussed in sections 3.1 and 3.2 respectively.

3.1 Sentence-Level Rules

Table 1 gives an overview of the distribution of sentence-level rules in the five rule sets analysed and lists the linguistic phenomena they control.[6] Four sub-categories of sentence-level rules have been identified, namely rules aimed at controlling (i) ambiguity, (ii) complexity, (iii) modality and tense and (iv) information structure. The remainder of this section will discuss the main characteristics that the present comparison has revealed for these four rule classes.

Ambiguity. Controlling ambiguity can be considered one of the prototypical tasks of a controlled language. The tekom standard provides rules addressing issues such as attachment ambiguity (arising, for instance, when a modifier precedes or follows a coordinated phrase), anaphoric ambiguity (arising when a pronoun has more than one possible antecedent), functional ambiguity (arising when, due to the relatively free German word order, it is unclear which noun phrase is the subject and which is the direct object of a sentence) and relational ambiguity (present, for instance, in possessive phrases such as *die Untersuchung der Behörde* 'the inspection of the agency'). To avoid certain types of scope ambiguities, the tekom standard also contains a rule that prohibits the use of non-restrictive modifiers.

While all of the analysed legislative drafting guidelines name the avoidance of ambiguity as one of their aims, they provide only few actual drafting rules to address the problem. This finding is somewhat surprising in light of the fact that ambiguity in laws has received considerable attention in the literature [6,25,28]. The German guidelines contain general statements urging for the avoidance of attachment ambiguity, anaphoric ambiguity and relational ambiguity but offer no specific instructions. The Zurich guidelines are a bit more specific: together with a general rule on avoiding ambiguity, they list an example of attachment ambiguity and an example of plural ambiguity – note that plural ambiguity is not addressed in the tekom standard – and explain how the respective situations can be remedied. They also contain rules stating that the antecedent of a pronoun must be located within the same article – according to the tekom standard, it must even be within the same sentence – and that a new paragraph may only begin with a pronoun if that pronoun refers to the subject of the sentence contained in the preceding paragraph.

However, all four legislative drafting guides are very specific about the use of the conjunctions *und* ('and') and *oder* ('or') – an issue that is not covered by the tekom

[6] The acronyms TK, AT, DE, BE and ZH denote the tekom standard and the legislative drafting guidelines of Austria, Germany, Bern and Zurich respectively. Ticks in brackets denote rules that occur only implicitly, e.g. in the form of an example to a more general rule.

Table 1. Distribution of sentence-level rules

ID	Rule class	TK	AT	DE	BE	ZH
Rules aimed at reducing ambiguity						
1	Rules addressing *attachment ambiguity*	✓		✓		(✓)
2	Rules controlling the antecedents of *pronouns*	✓		✓		✓
3	Rules addressing *functional ambiguity (word order)*	✓				✓
4	Rules addressing *relational underspecification*	✓		✓		
5	Rules barring *non-restrictive modifiers*	✓				
6	Rules addressing *plural ambiguity*					(✓)
7	Rules controlling the use of *"and"* and *"or"*		✓	✓	✓	✓
Rules aimed at reducing complexity						
8	Rules limiting *sentence length*	✓	✓	✓		
9	Rules controlling the *position of the verb*	✓	✓	✓	✓	✓
10	Rules barring *embedded subordinate clauses*	✓	✓	✓	✓	
11	Rules barring *multiple subordinate clauses*	✓	✓	✓	✓	✓
12	Rules barring *chains of noun phrases*	✓	✓	✓	✓	✓
13	Rules barring *participle phrases*	✓	✓	✓	✓	✓
14	Rules barring *nominalisations*	✓		(✓)		✓
15	Rules barring *light-verb constructions*		✓	(✓)	✓	✓
16	Rules controlling the representation of *lists*	✓	✓	✓	✓	✓
17	Rules barring *double negation*	✓	✓		✓	
Rules aimed at controlling modality and tense						
18	Rules controlling the use of *modal verbs*	✓	✓	✓	✓	✓
19	Rules controlling the use of the *imperative*	✓				
20	Rules barring the use of the *subjunctive*	✓				
21	Rules stipulating the use of *present tense*	✓			✓	✓
22	Rules barring *unspecific provisos and exceptions*		✓		✓	✓
Rules aimed at controlling information structure						
23	Rules controlling the use of *passive voice*	✓	✓	✓	✓	✓
24	Rules controlling the representation of *conditions*	✓		✓		✓
25	Rules barring *multi-propositional sentences*	✓	✓	✓	✓	✓

standard at all. The distinction between *and* and *or* becomes particularly relevant if, in a law, an obligation, prohibition or permission is associated with a list of conditions. In such cases, it is crucial to know whether the legal consequence already takes effect if any one of the conditions is fulfilled or only if all of them are met. The analysed guidelines for legislative drafting consequently define that, unless the situation is absolutely clear from the context, the conjunction *and* must be put before the last element of a cumulative list and the conjunction *or* before the last element of an alternative list respectively. The Austrian guidelines even demand that in alternative lists, the conjunction *or* be put between all list elements. In addition, all four legislative drafting guidelines contain rules inhibiting the use of *und/oder* ('and/or') and *beziehungsweise* ('respectively').

Complexity. The rules that the tekom standard and the legislative drafting guides apply to reduce the linguistic complexity are very similar and often even identical. Among other things, they all contain rules urging for the main verb of a sentence to be introduced as early as possible and for split verb forms, accumulations of subordinate clauses, chains of noun phrases, and complex participle phrases to be avoided. In addition, some of them put an upper boundary to sentence length and discourage authors from using nominalisations, light-verb constructions and double negations.

They also share rules requiring lists to be broken up into explicit enumerations and defining the syntactic structure that such enumerations must exhibit. These rules state, for instance, that a sentence must not be continued after an enumeration, that all elements of an enumeration must have the same syntactic structure, that no additional sentences may be inserted in the enumeration elements, and that the lead-in to an enumeration must not consist of a single pronoun. Example (1) shows a sentence that contains such an explicit enumeration:

(1) Die Wahlbehörde kann eine Richterin oder einen Richter vor Ablauf der Amtsdauer des Amtes entheben, wenn diese oder dieser:
   a. vorsätzlich oder grobfahrlässig Amtspflichten schwer verletzt hat; oder
   b. die Fähigkeit, das Amt auszuüben, auf Dauer verloren hat.

   'The electoral authorities may remove a judge from office before he or she has completed his or her term if he or she:
   a. has wilfully or through gross negligence committed serious breaches of his or her official duties; or
   b. has permanently lost the ability to perform his or her official duties.'

Incidentally, most of the rules comprised in this category are also available to automatic assessment by state-of-the-art controlled language checkers [1,10]. Here, legislative drafters could thus benefit more or less immediately from the technical advances brought about by the domain of technical writing.

Modality and Tense. The expression of modality is central to both technical writing and legislative drafting as it defines the pragmatic effect that the texts have in the real world. Accordingly, the rules controlling modality are well developed both in the tekom

standard and in the four legislative drafting guides. However, while the two problems discussed above – ambiguity and complexity – have resulted in rules that are more or less domain-independent, modality is an issue that has led to rules that are highly specific to the respective text type. While in technical writing, the imperative plays a crucial role in expressing instructions, legislative writing employs the indicative and modal verbs.

The tekom standard contains several rules concerned with the use of the imperative; the four legislative drafting guidelines, in contrast, make no mention of the imperative. This does not mean that the use of the imperative would be permitted in legislative texts; it is rather so unlikely that anybody would try to use the imperative mood in a legislative text that providing a rule explicitly prohibiting it must have seemed unnecessary.

Conversely, while the tekom standard simply interdicts the use of modal verbs, the four legislative drafting guides contain several rules on how to use such verbs: obligations must either be marked by *müssen* ('must'), *haben zu* ('have to') or *sein zu* ('be to'), or they can simply be put in indicative mood since statements contained in a law are, by definition, obligational. Permissions must be expressed with the modal *können* ('can'). The guidelines of Austria, Zurich and Bern prohibit the use of the modal *sollen* ('should'), whereas the German guidelines restrict its use to a special, less binding type of provision ("Soll-Vorschriften"). On a related note, the former three guidelines furthermore interdict the introduction of unspecified provisos and exceptions as expressed, for instance, by the adverbs *grundsätzlich* ('principally') and *in der Regel* ('as a general rule').

The use of present tense is explicitly stipulated by the tekom standard as well as the legislative drafting guides of Zurich and Bern. They also include rules prohibiting the use of future tense and, in the case of the Zurich guidelines, the use of future-related adverbs such as *neu* ('now') in example (2):

(2) Die Benutzung der Anlagen unterliegt *neu* einer Gebühr von Fr. 150.
'The use of the premises is *now* subject to a fee of 150 francs.'

Information Structure. Writing principles relating to information structure, as crucial as they are for the composition of accessible texts, are difficult to boil down to concrete controlled language rules. The relative abstractness and indeterminacy of rules aimed at controlling the use of passive voice is symptomatic of this fact. Both the tekom standard and the four legislative drafting guidelines contain general statements discouraging authors from using passive voice. However, the tekom standard and the legislative drafting guides of Germany, Bern and Zurich relativise this rule by saying that, under certain circumstances, passive voice is to be preferred to active voice: for instance, if there is no specific addressee or if the focus of the sentence should be on the action rather than the agent.[7] The tekom standard and the German legislative drafting guide further specify that sentences in passive voice with the agent added as a prepositional object with *von, durch* or *seitens* ('by') are to be avoided.

The problem of insufficient specificity also arises with the rule that a sentence should not make multiple statements. Although this rule seems to be relatively straightforward,

[7] The legislative drafting guide of the Swiss federal administration mentions the rule that passive voice must be avoided as an example of an overly simplistic writing principle [3].

the detection (manually or by machine) of sentences violating it is far from trivial. In order to become operational as a controlled language rule, this writing principle needs to be concretised by explicitly prohibiting specific structures indicating the presence of a multi-propositional sentence. The tekom standard contains one such concretisation in the form of a rule barring main clause coordination. It has been shown that there is a number of further constructions that reveal the presence of more than one statement in a sentence [12]. Relative clauses introduced by the relative adverb *wobei* ('whereby'), for instance, usually introduce an additional statement, as illustrated in example (3).

(3) Die berufliche Vorsorge wird durch die Beiträge der Versicherten finanziert, *wobei* die Arbeitgeberinnen und Arbeitgeber mindestens die Hälfte der Beiträge ihrer Arbeitnehmerinnen und Arbeitnehmer bezahlen.

'The occupation pension scheme shall be funded from the contributions of those insured, *whereby* employers must pay a minimum of one half of the contributions of their employees.'

Provided that there is specialised linguistic research into the properties of legislative language, there is thus still room for the development of more specific controlled language rules that can help legislative drafters avoid multi-propositional sentences.

The construction of conditional clauses is controlled both by the tekom standard and by the four legislative drafting guides in question. While the tekom standard only allows conditional clauses introduced by the conjunction *wenn* ('if'), the legislative drafting guides are more permissive and allow for the whole palette of options available in natural language: conditional clauses introduced by various conditional conjunctions (e.g. *wenn*, *falls*, *sofern*), conditional clauses in the form of relative clauses (e.g. *wer* ... 'whoever ...') and conditional clauses constructed by means of inversion (*Sind alle Auflagen erfüllt,* ... 'Have all requirements been met, ...'). The tekom standard also prohibits the use of *sobald* ('as soon as') to introduce conditions because they could be mistaken for temporal relations. This rule does not appear in the legislative drafting guidelines but would certainly make sense there too.

The German and the Zurich guidelines, in turn, further specify the use of other conditional conjunctions: while *wenn* and *falls* ('if') are to be used to introduce absolute conditions (the consequence comes into effect if the condition has been met), the conjunctions *soweit* and *solange* ('to the extent') must only be used to express gradual conditions (the consequence comes into effect to the extent to which the condition has been met); the conjunction *sofern* ('so far as') is used in the former sense in Zurich and in the latter sense in Germany.

3.2 Text-Level Rules

Table 2 gives an overview of the distribution of text-level rules in the five rule sets analysed and lists the linguistic phenomena they control. Four sub-categories of text-level rules have been distinguished, namely rules controlling (i) text structure, (ii) cross references, (iii) discourse structure and (iv) content types.

Text Structure. Understandability does not end at the level of sentence construction: text structure is also an important factor. All four legislative drafting guides contain

Table 2. Distribution of text-level rules

ID	Rule class	TK	AT	DE	BE	ZH
Rules aimed at controlling text structure						
26	Rules controlling the *levels of text divisions*		✓	✓	✓	✓
27	Rules controlling the *length of text divisions*		✓	✓	✓	✓
28	Rules controlling the *form of division headers*	✓		(✓)		(✓)
Rules aimed at controlling cross references						
29	Rules controlling the form of *cross references*	✓	✓	✓	✓	✓
30	Rules barring unnecessary or vague *cross references*	✓	✓	✓	✓	✓
31	Rules barring *cross reference chains*		✓			✓
32	Rules barring *cataphoric cross references*				✓	✓
Rules aimed at controlling discourse structure						
33	Rules stipulating *general-specific order*		✓	✓	✓	✓
34	Rules stipulating *rule-exception order*				✓	✓
35	Rules stipulating *chronological order*				✓	✓
36	Rules stipulating the use of *discourse markers*		✓	✓	✓	
Rules aimed at controlling content types						
37	Rules barring *declarative statements*		✓		✓	✓
38	Rules controlling the use of *statements of purpose*		✓	✓	✓	✓
39	Rules controlling the use of *definitions of terms*	✓	✓	✓	✓	✓
40	Rules controlling the use of *subject-matter definitions*	✓		✓		✓

rules that determine what levels of text divisions (parts, chapters, sections, subsections, articles, paragraphs) are available and at what point a level of text division should be introduced or removed. The most common rules in that regard define that a text division has to be broken up into smaller units if it comes to contain more than a certain number of articles (20 according to the Austrian and German rules; 12 according the rules of Bern and Zurich), that an article should not consist of more than a certain number of paragraphs (8 in Austria, 5 in Germany, and 3 in Bern and Zurich) and that a paragraph should not contain more than a certain number of sentences (3 in Germany, 1 in Bern and Zurich). While the tekom standard contains a rule restricting the length of sentences, it does not make any suggestions as to the ideal length of supra-sentential text units.

However, the tekom standard does not ignore text structure completely: it contains a range of rules defining the linguistic properties of good division headers. Such rules are in turn mostly missing from the legislative drafting guides. Only the drafting guides of Germany and Zurich contain a rule stating, in a very unspecific way, that article headers and marginal titles should be short and keyword-like.

Cross References. Rules controlling the use and form of intra- and inter-textual cross references take up a comparatively large section in all four legislative drafting guides. These rules represent domain-specific concretisations of the more general principles set

up in the tekom standard demanding that cross references be marked consistently, that they be sufficiently specified and that vague or unnecessary cross references be avoided. Some of the legislative drafting guides additionally rule out cross reference chains and cataphoric cross references.

Discourse Structure. Ambiguity can also arise if the discourse relation holding between two statements or groups of statements is unclear. Furthermore, a well-organised discourse structure makes a text easier to access and understand. Legislative drafting guidelines thus discuss the relations that may hold between discourse segments and the order in which they are to be arranged. All four legislative drafting guidelines stipulate that general statements are to precede more specific ones, and the guidelines of Bern and Zurich concretise this principle by stating in addition that rules must precede their exceptions. They also stipulate the arrangement of discourse segments in the chronological order in which the actions described therein are meant to occur.

The guidelines of Austria and Bern further recommend the use of adverbs such as *jedoch* ('however') and *ferner* ('moreover') to mark the discourse relations holding between consecutive sentences, and all but the Zurich guidelines stipulate the use of adverbs such as *insbesondere* ('particularly') and *beispielsweise* ('for example') to mark the discourse relation holding between abstract rules and concrete examples. In contrast to the guidelines for legislative drafting, the tekom rule set does not address the issue of ambiguity arising from unclear discourse relations.

Content Types. All four legislative drafting guides contain rules about the types of content that do or do not belong in legislative texts. Declarative statements such as descriptions, explanations, justifications, background information or appeals are to be avoided. Statements of purpose are not permitted either, unless they occur in a special article at the beginning of the text or if they are necessary for a provision to be applied correctly.

The tekom standard does not include such information in the form of rules. However, in the commentary it provides with its rule set, it refers to functional design [18,19] as a technique to standardise text structure and content types in the domain of technical communication. It clarifies that some of these content types may allow declarative statements and statements of purpose while others would not, and that users must decide which of the rules proposed by the standard should be enforced in which context. Thus, parallels to the different content types defined in the legislative drafting guides do in fact exist: there too, different contexts require the application of different rules. The rule that the modal *sollen* ('should') must be avoided, for instance, only applies to the normative parts of a legislative text but not to statements of purpose.

Finally, both the tekom standard and the four legislative drafting guides contain rules about special meta-textual content types: subject-matter definitions and definitions of terms. The use of subject-matter definitions, i.e. putting a short overview of its main topics at the beginning of a text, is encouraged by the tekom standard and the legislative drafting guide of Germany but discouraged by the legislative drafting guide of the canton of Zurich. Both types of rule sets also set up formats for the definition of terms: the tekom standard suggests the introduction of a glossary whereas legislative texts may define terms throughout the text. The German guidelines additionally state that, in long

texts, all definitions of terms should be contracted in a specially designated article at the beginning of the text. However, although legal definitions are checked for a whole number of requirements in editorial practice [2], specific rules explicitly controlling them are relatively sparse in all four legislative drafting guides.

4 Discussion

The motivation behind the current study was to get a better idea of whether and how guidelines for legislative drafting and controlled language rules for technical writing can inform each other: whether it makes sense for one domain to adopt rules devised by the other, and whether automated language checkers developed for technical writing could also be employed in legislative drafting.

The analysis presented in the previous section has shown that, by and large, rules for German-language technical writing and guidelines for German-language legislative drafting pursue the same goals and attempt to control the same range of linguistic phenomena. Perhaps the most striking differences lie in the emphasis the two domains put on individual aspects of language. Controlled language rules for technical writing, for instance, provide detailed instructions regarding phenomena that can cause sentence-level ambiguity, whereas legislative drafting guidelines have relatively little to offer in way of controlling this aspect of language. As sentence-level ambiguity is not only ubiquitous in legislative language but can positively cause serious legal problems, it would only seem sensible if authors of future legislative drafting guidelines considered borrowing some of the rules that technical writers have put in place to control this phenomenon.

Legislative drafting guidelines, on the other hand, are relatively explicit about avoiding discourse-level ambiguity and about breaking texts up into manageable divisions and sub-divisions, while the tekom standard remains silent on these two issues. Here, it is thus legislative drafting that can inform technical writing: the prevention of ambiguity and the reduction of complexity does not end at the level of sentences. Taking language control to the level of discourse must be the logical next step for research in the field of controlled language, and legislative drafting offers some important suggestions as to how this task can be approached.

The analysis has also shown that with regard to the goal of reducing syntactic complexity, the two domains have adopted more or less the same rules. Incidentally, these are rules for which state-of-the-art controlled language checking is available. At present, controlled language checkers are almost exclusively employed in the domain of technical writing, but the findings of the present study suggest that they could be applied to check for the respective features in legislative texts too. Adaptations would be necessary where the two domains control the same linguistic phenomena but have come up with different rules. The most obvious example falling under this category are the ways in which modality is expressed in legislative texts and in technical documentation respectively. Here, domain-specific alterations of the rules applied by a controlled language checker that was developed for the domain of technical writing would be necessary (but also feasible) before the tool could be used to support legislative drafting.

The main area, however, in which legislative drafting guidelines can benefit from the example of controlled languages for technical writing is the specificity of the rules.

Legislative drafting guidelines have a tendency to content themselves with stating abstract writing principles rather than providing specific rules. By doing so, they run the risk of not being able to take full effect as users may struggle applying these abstract principles to concrete linguistic structures. While all analysed drafting guides provide examples with their rules, these examples often only mention a small percentage of the linguistic structures falling under the respective writing principle. Users are on their own when it comes to deciding to what other structures the principle may possibly apply. In such cases, the application of the rule easily fails – especially since language-related issues are usually not among the most pressing matters that legislative drafters have to keep in mind: legal technicalities will most likely take precedence. Controlled languages for technical writing demonstrate that it is useful and possible to concretise general writing principles in the form of more specific rules that are easy for users to memorise and apply. The present analysis suggests that for several abstract principles of legislative drafting, such concretisations can in fact be carved out – provided that appropriate linguistic research into the peculiarities of legislative language is undertaken.

Occasionally, the concretisations required for a particular writing principle can already be found in one of the other legislative drafting guides. The current study has shown that the individual legislative drafting guides are relatively eclectic when it comes to the selection of rules they include: some phenomena may be treated in detail in one guide but completely ignored or only briefly touched upon in another. However, where two guides have both decided to address a particular issue, they propose more or less concurring rules – albeit possibly to different degrees of specificity. Legislative drafting could thus benefit from what the tekom standard offers for technical writing: a collection of possible rules that covers all relevant issues and from which authors of drafting guidelines can pick the building blocks they need. The present study is also a first step in that direction.

Acknowledgments. The current work was funded by SNSF grant No. 134701. The author thanks Alexandra Bünzli and Kyoko Sugisaki for her helpful comments on the paper.

References

1. Bellem, B., Dreikorn, J., Drewer, P., Fleury, I., Haldimann, R., Jung, M., Keul, U.P., Klemm, V., Lobach, S., Prusseit, I.: Regelbasiertes Schreiben: Deutsch für die Technische Kommunikation. Gesellschaft für Technische Kommunication e.V. – tekom, Stuttgart (2011)
2. Bratschi, R.: Frau im Sinne dieser Badeordnung ist auch der Bademeister. Legaldefinitionen aus redaktioneller Sicht. LeGes 20(2), 191–213 (2009)
3. Bundesamt für Justiz (ed.): Gesetzgebungsleitfaden: Leitfaden für die Ausarbeitung von Erlassen des Bundes, Bern, 3rd edn. (2007)
4. Bundeskanzleramt (ed.): Handbuch der Rechtsetzungstechnik, Teil 1: Legistische Leitlinien. Wien (1990)
5. Justiz, B.f. (ed.): Handbuch der Rechtsförmlichkeit: Empfehlungen zur Gestaltung von Gesetzen und Rechtsverordnungen. Bundesanzeiger Verlag, Köln (2008)
6. Bünzli, A., Höfler, S.: Controlling Ambiguities in Legislative Language. In: Rosner, M., Fuchs, N.E. (eds.) CNL 2010. LNCS, vol. 7175, pp. 21–42. Springer, Heidelberg (2012)

7. Eichhoff-Cyrus, K.M., Antos, G. (eds.): Verständlichkeit als Bürgerrecht? Die Rechts- und Verwaltungssprache in der öffentlichen Diskussion. Duden, Mannheim (2008)
8. European Communities (ed.): Joint Practical Guide of the European Parliament, the Council and the Commission for persons involved in the drafting of legislation within the Community institutions. Office for Official Publications of the European Communities, Luxemburg (2003)
9. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2004. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
10. Geldbach, S.: Neue Werkzeuge zur Autorenunterstützung. MDÜ 4, 10–19 (2009)
11. Göpferich, S.: Sprachstandard oder Kontrollmechanismus? Textqualität steuern mit kontrollierter Sprache. Technische Kommunikation 29(4) (2007)
12. Höfler, S.: "Ein Satz – eine Aussage." Multipropositionale Rechtssätze an der Sprache erkennen. LeGes: Gesetzgebung und Evaluation 22(2), 275–295 (2011)
13. Höfler, S., Sugisaki, K.: From drafting guideline to error detection: Automating style checking for legislative texts. In: Proceedings of the EACL 2012 Workshop on Computational Linguistics and Writing, Avignon, pp. 9–18 (2012)
14. Huijsen, W.O.: Controlled language: An introduction. In: Mitamura, T., Nyberg, E., Adriaens, G., Schmandt, L., Wojcik, R., Zajac, R. (eds.) Proceedings of the Second International Workshop on Controlled Language Applications (CLAW 1998), Pennsylvania, pp. 1–15 (1998)
15. Lehmann, S.: Kontrollierte Sprachen und Sprachtechnologie in der Industrie: das Autorenwerkzeug acrolinx. Universität Zürich, Seminar Computerlinguistik: Making Word Processors Process Words, Sprachtechnologie für Autorenwerkzeuge (2009), `https://cast.switch.ch/vod/clips/2pjsz3usia/link_box`
16. Lerch, K.D. (ed.): Recht verstehen. Verständlichkeit, Missverständlichkeit und Unverständlichkeit von Recht. de Gruyter, Berlin (2004)
17. Muegge, U.: Controlled language: The next big thing in translation? ClientSide News Magazine 7(7), 21–24 (2007)
18. Muthig, J.: Technical documentation needs standardization. tcworld: Magazine for International Information Management (May 2011)
19. Muthig, J., Schäflein-Armbruster, R.: Funktionsdesign: Methodische Entwicklung von Standards. In: Muthig, J. (ed.) Standardisierungsmethoden für die Technische Dokumentation, tekom-Hochschulschriften, vol. 16. Schmidt-Römhild, Lübeck (2008)
20. O'Brien, S.: Controlling controlled English: An analysis of several controlled language rule sets. In: Proceedings of the Joint Conference combining the Eighth International Workshop of the European Association of Machine Translation and the Fourth Controlled Language Applications Workshop (EAMT-CLAW 2003), Dublin, pp. 105–114 (2003)
21. Pool, J.: Can controlled languages scale to the web? In: Proceedings of the Fifth International Workshop on Controlled Language Applications, CLAW 2006 (2006)
22. Regierungsrat des Kantons Bern (ed.): Rechtsetzungsrichtlinien des Kantons Bern. Bern (2000)
23. Regierungsrat des Kantons Zürich (ed.): Richtlinien der Rechtsetzung. Zürich (2005)
24. Reuther, U.: Two in one – can it work? Readability and translatability by means of controlled language. In: Proceedings of the Joint Conference combining the Eighth International Workshop of the European Association of Machine Translation and the Fourth Controlled Language Applications Workshop (EAMT-CLAW 2003), Dublin (2003)
25. Schane, S.: Ambiguity and misunderstanding in the law. T. Jefferson L. Rev. 25, 167–649 (2002)
26. Schwitter, R., Tilbrook, M.: Let's talk in description logic via controlled natural language. In: Proceedings of the 3rd International Workshop on Logic and Engineering of Natural Language Semantics, Tokyo, pp. 193–207 (2006)

27. Schwitter, R., Tilbrook, M.: Annnotating websites with machine-processable information in controlled natural language. In: Orgun, M.A., Meyer, T. (eds.) Advances in Ontologies 2006: Proceedings of the Second Australasian Ontology Workshop (AOW 2006), pp. 75–84. Australian Computer Society, Hobart (2006)
28. Solan, L.: Vagueness and ambiguity in legal interpretation. In: Bhatia, V., Engberg, J., Gotti, M., Helier, D. (eds.) Vagueness in Normative Texts: Linguistic Insights, pp. 73–96. Peter Lang, Bern (2005)
29. Wyner, A., Angelov, K., Barzdins, G., Damljanovic, D., Davis, B., Fuchs, N., Hoefler, S., Jones, K., Kaljurand, K., Kuhn, T., Luts, M., Pool, J., Rosner, M., Schwitter, R., Sowa, J.: On Controlled Natural Languages: Properties and Prospects. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 281–289. Springer, Heidelberg (2010)

Portuguese Controlled Language: Coping with Ambiguity

Palmira Marrafa, Raquel Amaro, Nuno Freire, and Sara Mendes

CLG – Group for the Computation of Lexical and Grammatical Knowledge,
Centro de Linguística da Universidade de Lisboa,
Av. Prof. Gama Pinto, nº 2, 1649-003 Lisbon, Portugal
palmira.marrafa@netcabo.pt, {ramaro,sara.mendes}@clul.ul.pt,
nfreire@gmail.com

Abstract. This paper focuses on strategies to avoid lexical related ambiguity, induced by polysemy or by syntactic function effects, in the context of a system to control Portuguese as a source language for machine translation. This system, which is being developed under wider scope ongoing research, involves two main components - a controlled language for Portuguese and a tool to evaluate the conformity of texts with the controlled language. In a subsidiary way, it also makes use of the Portuguese WordNet (WordNet.PT).

Keywords: CNL, machine translation, ambiguity.

1 Introduction

The so-called controlled natural languages (CNL) involve sets of restrictions on the lexicon, syntax and/or semantics which enable the reduction or elimination of ambiguity and complexity typical of natural language utterances. In this paper we present strategies to eliminate ambiguity (mainly lexical ambiguity) defined under the scope of ongoing large coverage work aiming at obtaining better quality results with machine translation (MT) systems. Instead of listing lexical units to be avoided or to be used, our approach makes use of WordNet.PT ([1]). Although profiting from previous work ([2]), the new system involves a larger scope and a shift of perspective.

CNLs are not defined in a univocal way in the literature. Although elaborating on this matter is out of the aims of this paper, it is worthwhile to clarify that we use CNL in the following sense: sets of linguistic restrictions to be applied to written texts in a given language. The nature of those restrictions depends on the purposes to be achieved.

As discussed in [3] there are two main approaches to the design of CNLs: "naturalist" approaches, which view controlled languages as sets of restrictions on the existing structures and lexicon of a given natural language, stating which structures and lexical items are not to be used; and "formalist" approaches, which view controlled languages as sets of vocabulary and rules to form utterances in a given natural language, determining the lexicon allowed as well as the syntactic and interpretation rules allowed.

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 152–166, 2012.

Choosing between any of these approaches depends highly on the goals of the task at hand, as there is a considerable tradeoff in terms of reduction of the complexity involved in natural language processing (NLP) tasks and CNL usability for common users. On the one hand, "naturalist" approaches can result in utterances that are too close to natural language, requiring a lot of heuristics to deal with different issues regarding NLP, which are sometimes so complex that they would make building an open-domain CNL too expensive. On the other hand, "naturalist" CNLs reduce the effort of users considerably, namely in terms of learning the controlled language and using it proficiently.

In our specific case, we are mostly concerned with making it manageably easy for the user to write using a controlled Portuguese, since we focus on improving the outputs of state-of-the-art MT systems available online. The work presented here is framed by a wider ongoing research, which aims at providing non-expert users of such systems with tools for improving the results of their outputs for the Portuguese-English (PT-EN) language pair and, thus, their usability. In this scenario, it is crucial for the controlled Portuguese to be relatively easy to learn and formulated in a straightforward way, allowing non-expert users to use it effortlessly. In the following sections we discuss our approach in more detail, presenting the preliminary study of the outputs of two different MT systems – a rule-based system (PROMT[1]); and a statistic-based system (Google Translator[2]) –, and how this allowed us to pinpoint a set of relevant phenomena to be covered by the controlled Portuguese developed. We focus on ambiguity issues and on how the use of WordNet.PT – the Portuguese WordNet ([4-5]) – allows us to cope with ambiguity issues.

In section 2 we discuss related work in an analytic perspective, underlying the differences in approaches and strategies, considering CNLs oriented for MT and CNLs developed for Portuguese. Section 3 is devoted to the Portuguese control system depicted in this paper. In section 3.1 we present and motivate the general goal and design options of the controlled language, addressing its empirical basis and the different phenomena encountered. Section 3.2 focuses on ambiguity issues and strategies put forward to control polysemy and ambiguity induced by specific syntactic and semantic properties of lexical items. Section 3.3 concerns the motivations and design options regarding checking tools for the control system presented in this paper, particularly tools that are able to identify lexical ambiguity and provide information on possible solutions based on WordNet.PT. Finally, in section 4 we present our final remarks.

2 Related Work

CNLs have been developed for some time now to tackle issues arising in contexts so distinct as human-human communication or human-machine communication. Both human-oriented CNLs (i.e., instructions easily usable by humans) and machine-oriented CNLs (i.e., instructions to produce utterances easily processable by machines) aim at reducing ambiguity and complexity of natural languages. Independently

[1] http://www.online-translator.com/site_translation.aspx
[2] http://translate.google.com/

of the different approaches regarding their design options, CNLs are used in many NLP tasks and applications, among which knowledge representation, technical text production, natural language simplification or improvement, and control of the performance of MT systems.

According to [3], machine-oriented CNLs tend to follow a formalist approach as they aim at translating cleanly and predictably to a formal representation, in which there is only a single sense for each word and one acceptable parse and interpretation for each sentence, following a logically defined path. Formalist CNLs result, thus, in utterances more easily handled by machines being, however, harder to master for humans. Human-oriented CNLs, on the other hand, tend to follow a more naturalist approach, allowing for ambiguity and complexity, although to a lesser extent than that of natural languages. Despite being easier to master (in the users' perspective), naturalist CNLs produce utterances harder to control.

Nowadays, however, this distinction may be irrelevant given the multiplicity of goals intended and the tools and applications available, as depicted in the many emerging subareas of research concerning CNLs: CNLs for knowledge representation, for question answering, for specifications, for business rules, for interactive systems, for MT, and so on. On the one hand, machine-oriented CNLs, being harder to learn and master by humans, often depend on sophisticated predictive editors to help humans to use it; on the other hand, human-oriented CNLs are easy to master but often rely heavily on more advanced NLP systems when used for human-machine interface.

2.1 CNLs and Machine Translation

The conception and design of a CNL are highly dependent on the specific goals pursued, particularly the use for which the CNL is intended. Considering the different possible applications of CNLs, this is particularly true for CNLs designed for MT purposes, since the phenomena to be tackled concern different natural languages (source and target languages) and already existing NLP processing tasks involved in the MT system.

The phenomena at stake in the conception of a controlled Portuguese for MT (from PT to EN) are not necessarily the same considered in KANT - Knowledge-based, Accurate Natural Language Translation ([6]), for instance, as it was to be expected, given the different issues concerning the processing of English and Portuguese, both at the level of structural and lexical correspondences and disambiguation, and with regard to possible causes for ambiguity and difficulties in processing the data. For instance, the absence of the singular neutral pronoun in Portuguese, a language in which determiners and the vast majority of pronouns are gender-marked, creates additional and/or different difficulties to be controlled, since MT systems have to decide in which cases the gender-marked pronoun corresponds to the neutral pronoun (*it*) and in which cases it corresponds to the gender-marked personal pronoun (*he*). On the other hand, several types of relative structures are accurately processed by the MT systems tested and do not require specific control, in opposition to what happens in KANT, as referred to by [6]. It is also important to notice that work developed on CNLs is mostly concerned with technical domains, while our goal is not bounded to a restricted semantic domain.

2.2 CNLs for Portuguese

As in many other domains in NLP, English is one of the more resourced languages in terms of controlled formulation. CNLs such as KANT ([7], [6], and further work), ACE - Attempto Controlled English ([8-9]), PENG - Processable English ([10-11]), CLCE - Common Logic Controlled English ([12]), Boeing's CPL - Computer-Processable Language ([13-14]) or ProjectIT-RSL – Project IT Requirements Specification Language ([15]) are examples of controlled languages for English and have been used in the development of a considerable amount of research and applications. However, this is not the case for many other natural languages, among which Portuguese.

In what concerns Portuguese, [16] presents a script for the production of technical instructions, based on the analysis of specific characteristics of this type of texts in Portuguese. Deploying from a similar initiative of the German company Huhn-Dialog, the author proposes writing guidelines such as: "write short sentences; write one sentence for each proposed action; write grammatically complete sentences; write sentences in the active voice; avoid deictics; if you have to use negation, use the n-word at the beginning of the sentence", etc. Although aiming at developing a CNL, [16]'s instructions are too vague to be successfully used to assure quality results with machine translators. Such generality can even induce undesirable effects, as in the case of the guideline regarding negation. As a matter of fact, when applied to sentence negation, neither the input nor, must importantly, the MT output express sentence negation. In both cases, negation does not have scope over the sentence but just over its first constituent.

In a different and more comprehensive approach, [17-19] present the PorSimples project (Simplification of Portuguese Text for Digital Inclusion and Accessibility), aiming at developing technologies for making access to information easier for low-literacy individuals by means of Automatic Summarization, Text Simplification, and Text Elaboration. For the purposes of this paper, the relevant part of this work concerns the text simplification (lexical and syntactic simplification) strategies proposed. Lexical simplification is carried out by replacing complex words with simpler ones using thesauri and a lexical ontology. With regard to syntactic simplification, it is accomplished by a rule-based system, which comprises seven operations (sentence splitting, replacement of discourse marks, passive/active transformations, clause order inversions, subject-verb-object ordering and topicalization/detopicalization) that are applied sentence-by-sentence to a text to make its syntactic structure simpler. The rule-based text simplification system is based on the Manual for Syntactic Simplification for Portuguese ([20]) and covers 22 linguistic phenomena ([18]).

Although concerning Portuguese, PorSimples aims at simplifying natural language utterances for native speakers with poor literacy skills, but with no specific processing handicaps, while we aim at assuring quality PT-EN translations using available machine translators, the conception and design of the controlled language depending on the performance of the systems. Also, with regard to lexicon control, the simplification strategies used in PorSimples are related to the gaps in the lexicon of low-literacy individuals, which are not necessarily the same of lexicon modules of machine translators, and thus lexicon control is hardly coincident in these two cases. Moreover, polysemy is far from being a main issue in PorSimples as it is in the work

we depict here. Regarding phrase level and sentence level control, given the processing devices at work in machine translators, not all syntactic phenomena addressed in PorSimples are relevant to our work. For instance, relative clauses are identified as a cause of structural complexity in PorSimples, i.e., they contribute to situations in which text content is misunderstood by low-literacy individuals, but are structures mostly accurately handled by the MT systems tested. Finally, there are significant lexical and structural differences in the Portuguese varieties considered: Brazilian vs. European Portuguese.

WordNet PortControl[3] constitutes the first system to control European Portuguese for MT and for Portuguese teaching/learning, involving the design of a CNL and of checking tools. It covers basic grammar phenomena such as order of constituents, null constituents, ellipsis, use of pronouns, proper noun identification, auxiliary verb (tense and aspect) system. The following examples illustrate some of the CLG – Controlled Portuguese instructions ([2]):

1. Instead of finite clauses with 3rd person null subjects use the corresponding infinitives
   (a) Cada cidadão deve escolher cuidadosamente a casa onde Ø mora e o computador com que Ø trabalha. (not controlled)/Cada cidadão deve escolher cuidadosamente a casa para morar e o computador para trabalhar. (controlled)
   (b) #Every responsible citizen should carefully choose the house you live in and the computer that works./Every responsible citizen should carefully choose a place to live and computer to work. (Google)
   (c) #Each citizen must choose carefully the house where it lives and the computer with which it works./Each citizen must choose carefully the house to live and the computer to work. (PROMT)
2. Instead of auxiliary verb *ir* (imminent event)+main verb use *quase*+main verb
   (a) A criança ia caindo. (not controlled)/A criança quase caiu. (controlled)
   (b) #The child was falling./The child nearly fell. (Google)
   (c) #The child was falling./The child almost fell. (PROMT)
3. Instead of 3rd person possessive pronouns *seu(s)*, *sua(s)* use *dele(s)*, *dela(s)*
   (a) Estas pessoas deveriam ter um contrato de trabalho, como os seus colegas.(not controlled)/Estas pessoas deveriam ter um contrato de trabalho, como os colegas delas. (controlled)
   (b) #These people should have an employment contract, as his colleagues. These people should have an employment contract, as their colleagues. (Google)
   (c) #These persons should have a contract of work, like his colleagues. These persons should have a contract of work, like their colleagues. (PROMT)

Although covering issues relevant to MT, providing us with grounds to pursue further work, WordNet PortControl is not concerned with a relevant set of phenomena involved in the establishment of a CNL for MT purposes only, since teaching/learning

[3] Project coordinated by Palmira Marrafa at CLG, CLUL, funded by Instituto Camões, `http://www.clul.ul.pt/clg/eng/projectos/ portcontrol.html`.

goals impose strict restrictions on the quality of source texts, conditioning the phenomena addressed as well as the instructions established in the CNL.

3 Portuguese Control System

As mentioned before, according to the goals in pursue, this system involves two main components – a controlled language for Portuguese and a checker. In a subsidiary way, it also makes use of WordNet.PT.

3.1 Controlled Language

The general goal of the controlled language discussed here is to improve the quality of MT outputs as much as possible. This means that its design is not oriented in a relevant way to the quality of the source texts, contrarily to what happens with several other controlled languages (e.g. KANT, [6]; CLG-Controlled Portuguese, [2]). Our option is justified by the fact that achieving both goals simultaneously is often not possible. To give an example, let us take the sentence *o rapaz viu as prendas triste* ('the boy saw the gifts sad'), which is a simple short non-ambiguous sentence. Nevertheless, when translating it into English both Google Translate and PROMT Translator results are unacceptable, as shown below.

4. (a) the boy saw the sad gifts (Google)
   (b) the boy saw the gifts sadly (PROMT)

On the contrary, we obtain accurate results with a paraphrase of that sentence – *o rapaz estava triste quando viu as prendas* ('the boy was sad when [he] saw the gifts') –, which is more complex, as it includes an embedded sentence with a null subject, and less economic and elegant.

5. (a) the boy was sad when he saw the presents (Google)
   (b) the boy was sad when he saw the gifts (PROMT)

This kind of problem requires special attention when in similar sentences the adjective in post-nominal position has modification and predication potential. We discuss this issue deeper in section 3.2.

Empirical Basis. The development of the Portuguese controlled language discussed here is supported by systematic research on real results of MT systems, freely available online and covering the two main paradigms in MT, knowledge-based and statistic based systems.

In order to cover the most representative structures of Portuguese, and to cover phenomena such as lexical and structural ambiguity, different orders of constituents, coordination and subordination structures, ellipsis, modality, among others, besides consulting related work, we compiled a *corpus* of real texts to test the performance of MT systems for the PT-EN language pair. This *corpus* is constituted mainly by newspaper articles, with diverse lexicon and structures, representative of common use of

Portuguese, as opposed to technical texts, on the one hand, and to literature, on the other, and was first used to build the CLG -Controlled Portuguese, referred to above.

The analysis of the *corpus* and of the resulting translations provided us with a first list of structures potentially problematic to MT systems, directly related to the language pair at stake, to the type of texts (open domain and not stylistically constrained) and to the processing performance of the MT systems used. In a first stage, the on-line MT systems tested were Google Translate, a statistic-based system, PROMT, a knowledge-based system, and Systran, a hybrid system. However, and in spite of being a hybrid system, the Systran results for the language pair considered (PT-EN) were consistently weaker than the results obtained with Google Translate and with PROMT. For this reason, the Systran MT was tested in an earlier stage, but its results were not taken as decisive for establishing the controlled language.

The potentially problematic structures identified concern ambiguity and/or complexity related to semantic issues, such as time and aspect, predication and verb classes, nominal reference and modality; and ambiguity and/or complexity related to syntactic issues, such as syntactic relations and word order, sentence structure (passive with *–se*, for instance), innacusative constructions, coordination structures, relatives and completives, adverbial subordination, degree and comparison constructions, negation, anaphora and long distance dependencies, ellipsis, secondary predication, null subject. Texts showing these structures were thus translated by the two MT systems in order to determine which phenomena were accounted for by these systems and which required specific control.

The results, as we illustrate below, were not always coincident with the structures commonly pointed out as problematic and/or complex in related works. Note that our goal is to assure quality results from MT systems by helping users to avoid using lexical items and syntactic structures that are incorrectly processed by the MT systems tested. This way, the complexity and/or simplicity of the source language structures or lexicon used are only relevant when they trigger low quality results. Let us consider, for instance, two phenomena typically treated in CNLs, passive and relative clauses (see [6], [20], [18] and [16], for instance).

Syntactic passive structures are typically controlled, usually being converted into active sentences. However, for our purposes, this conversion (and its correlate undesirable pragmatic effects) has no motivation, since MT systems generally have no problem processing them, as illustrated bellow.

6. (a) A redução das indemnizações foi causada pelo corte de despesas que o Governo prevê realizar em 2011.
   (b) The reduction of damages was caused by cutting expenses that the Government is planning to hold in 2011. (Google)
   (c) The reduction of the compensations was caused by the expenses cut that the Government predicts to carry out in 2011. (PROMT)
7. (a) As culturas foram totalmente destruídas pelas inundações.
   (b) The cultures were completely destroyed by floods. (Google)
   (c) The cultures were totally destroyed by the floods. (PROMT)

Concerning relative clauses, although we have identified a few problems related with prepositions/function markers introducing relative pronouns (further research is needed to capture generalizations), quality results are obtained, sensitive to the semantic of the relative pronoun antecedent and independent of its gap position, as the examples show.

8. (a) O Ministro viu os trabalhadores que protestaram.
   (b) The Minister saw the workers who protested. (Google/PROMT)
9. (a) Ele viu os cães que atacaram crianças.
   (b) He saw the dogs that attacked children. (Google/PROMT)
10. (a) As pessoas com quem ele se relaciona são inteligentes.
    (b) People with whom he relates are smart (Google)
    (c) The persons with whom he is connected are intelligent. (PROMT)
11. (a) Os animais de que ele gosta são selvagens.
    (b) The animals he loves are wild. (Google)
    (c) The animals which he likes are wild. (PROMT)

Other relative clauses, with issues involving pronoun reference ambiguity, were tested with quality results as well:

12. (a) A criança agarrou o estojo da caneta que estava em cima da mesa.
    (b) The child grabbed the case of the pen that was on the table. (Google)
    (c) The child seized the case of the pen that was on top of the table. (PROMT)

In this case, ambiguity is directly related to the definition of the pronoun antecedent: 'the case that was on top of the table' or 'the pen that was on top of the table'. Although ambiguous, this structure is well processed by the MT systems tested, resulting in a well-formed sentence that maintains the ambiguity in the target language.

In the next section, we discuss issues pertaining to lexical related ambiguity, which is the focus of this paper.

3.2 Controlling Ambiguity

Being targeted at the so-called non-marked common language, the controlled language developed under the scope of the work considered here is a wide coverage controlled language, in the sense that all the aspects of grammar (lexical, syntactic, semantic and pragmatic) have to be taken into account in its design. Nevertheless, in this paper we focus on ambiguity, a kind of cross-phenomena epiphenomenon.

Ambiguity is commonly accommodated in two main types: (i) lexical ambiguity, mostly limited to contrastive polysemy; and (ii) structural ambiguity, often concerned just with PP attachment (see section 3.3). However, neither these two types cover all the cases of ambiguity nor do they represent the more problematic ones. Although we do not elaborate explicitly on this matter, these assumptions are supported by the discussion carried out in the remainder of this section, which includes strategies to avoid ambiguity due to different phenomena, selected to illustrate our approach.

Polysemy. As a general lexical constraint, it is recommended that lexical polysemy be avoided. Being very general, this advice is limitedly helpful. However, in our approach, checking tools, besides pointing out polysemy, directly access WordNet.PT, extracting information on polysemy as well as non-ambiguous synonyms of the word forms at stake. Wordnets encode the different concepts a word form is associated to. On the other hand, each concept, corresponding to a node in the network, is represented by the set of synonyms lexical units that denote it. Let us take the example of *casa* ('house/home/buttonhole'). Whenever we search for casa in WordNet.PT, the system presents three lexical entries to the user, each one corresponding to one of the three possible meanings referred to above. The user can then choose the entry corresponding to the relevant meaning. If, for instance, we choose the entry corresponding to home, we immediately have access to the set of Portuguese lexicalizations of this concept, namely, *residência, domícilio, casa, lar*, with additional information concerning the register of each of the word forms. This way, the user can choose an alternative to *casa* to express the concept at stake, unambiguously.

The examples presented before illustrate contrastive polysemy, i.e., the case where two or more non-related senses are associated with a single expression. However, complementary polysemy, which regards related senses, as for instance, the two senses of *lamb* – animal; meat of that animal – is treated in a similar way, since the two senses are encoded in WordNet.PT in different nodes. This option is supported by semantic evidence but also by the fact that languages do not behave uniformly with regard to this kind of alternation, both within a specific language and in a cross-linguistic point of view. Let us examine an example.

Contrarily to what happens with *lamb*, *cow* means female bovine, but does not stand for the meat of the same animal, which is designated *beef*. Considering the lexical correspondences in Portuguese we use the word form *carneiro* for the male ovine and for the meat of male and female ovines, while we use *vaca* for the female bovine and for the meat of male and female bovines. Portuguese does not have a specific lexicalization corresponding to *beef*. In these cases, the solution does not involve the choice of non-ambiguous synonyms, since they do not exist. Instead, users should employ the immediate superordinate, to which they have access as they have for synonymous. This way, the desired results are obtained, as shown below.

13. (a) Hoje comi vaca. ('Today [$_{null}$ I] ate cow/beef')
    (b) #Today I ate cow. (Google/PROMT)
14. (a) Hoje comi carne de vaca. ('Today [$_{null}$ I] ate cow/beef')
    (b) Today I ate beef. (Google/PROMT)

Other type of relations can help users to choose non-problematic lexical units, since WordNet.PT is a very high density network, as shown in the table 1 below.

Table 1. Average of relations (other than synonymy) per lexical unit encoded in WordNet.PT

average of relations	POS
4,6	noun
3,9	verb
4,7	adjective

It is also worthwhile to notice that at the present stage WordNet.PT covers an expressive set of common language semantic domains, such as art, clothes, communication, education, food, geography, health, housing, human activities, human relations, living things, sports, and transportation.

Inherent Syntactic Function Ambiguity. This kind of ambiguity usually does not receive much attention in the literature on CNLs, contrarily to what happens with structural ambiguity induced by PP attachment (see [6], among many others). This is particularly curious as the first case, but not the second one, is a complex issue. As a matter of fact, PP attachment seems not to be such a big deal for MT, unless the language pairs at stake dramatically differ in what concerns the order of constituents[4]. Otherwise, ambiguity is expected to be maintained in the target language without any problems arising from this fact. Disambiguation can also occur with minor problems induced by an inappropriate choice of prepositions, easily detectable and correctable by users, as illustrated in the examples below.

15. (a) Vi o rapaz de óculos amarelos na sala.
    ('[$_{null}$ I] saw the boy of/with glasses yellow in+the room';
    "I saw the boy with yellow glasses in the room")
    (b) I saw the guy in yellow glasses in the room. (Google)
    (c) I saw the boy of yellow glasses in the room. (PROMT)

16. (a) Vi o rapaz com óculos amarelos na sala.
    ('[$_{null}$ I] saw the boy with glasses yellow in+the room';
    "I saw the boy with yellow glasses in the room")
    (b) I saw the guy with yellow glasses in the room. (Google/PROMT)

Let us now concentrate on ambiguity due to the inherent capacity of certain lexical units to assume more than one syntactic function, as it happens here:

17. (a) O rapaz bebeu o chá frio ('The boy drank the tea cold')
    (b) "The boy drank the tea cold"
    (c) "The boy drank the cold tea"
    (d) The boy drank the cold tea. (Google/PROMT)

The sentence in 17(a) is ambiguous between the interpretation in 17(b), where *cold* is interpreted as a secondary predicate of *tea*, and the interpretation in 17(c), where *cold* is interpreted as a modifier of *tea*. The preferred interpretation is the first one (17(b)), the other one only being available in specific contextual conditions. Despite this fact, both Google and PROMT provide only the latter reading.

As we can infer from the examples, the syntactic problems raised by this kind of ambiguity are related in a significant way to the order of constituents. Nevertheless,

[4] The different treatment given to PP attachment probably arises from the fact that by having different goals with regard to the most common goals in CNLs (for instance we do not need to build a semantic representation), we have different requirements, as observed by an anonymous reviewer, whose comments we thank.

there are other important variables to be considered, such as the semantic properties of adjectives and nouns. To make it apparent, let us also consider the following sentences:

18. (a) O rapaz bebeu o chá verde. ('The boy drank the tea green')
    (b) "The boy drank the green tea"
    (c) "#The boy drank the tea green"
19. (a) O rapaz leu o livro triste. ('The boy read the book sad')
    (b) "The boy_i read the book sad_i"
    (c) "The boy read the sad book"
    (d) "#The boy read the $book_i$ sad_i"

The sentence 18(a) is unambiguous. Only the interpretation *the boy drank the green tea* is available, i.e., the interpretation in which *verde* ('green') would be a secondary predicate is excluded. Differently, 19(a) is ambiguous. The possible readings are given in 19(b) – in which *triste* ('sad') is interpreted as a secondary predicate applied to *o rapaz* ('the boy') –, and in 19(c) – in which *triste* ('sad') is interpreted as a modifier of *livro* ('book'). Here *triste* can function both as a modifier and as a secondary predicate, but the interpretation where *triste* is a secondary predicate applied to *livro* is excluded.

As shown, ambiguity related to the inherent potential of certain lexical units to assume more than one syntactic function is a somewhat puzzling issue. Therefore, the solutions are not so straightforwardly provided as in the case of polysemy.

Nonetheless, in a similar way to what happens with regard to polysemy, the user is warned by the checker that a given lexical unit is ambiguous in what concerns its syntactic function potential. This functionality implies the annotation of WordNet.PT lexical units with this kind of information.

3.3 Checking Tools

Given the burden imposed on CNL users, specifically the requirement that they learn and master the CNL, the development of CNLs is often accompanied by the development of tools to help users to employ them appropriately.

Putting it somewhat simplistically, it is possible to consider three main types of such tools: editors (that help users to write texts that conform to the CNL on the fly); converters (that convert natural language texts into texts in CNL format); and checkers (that check whether the text produced by the user conforms to the CNL).

[21], for instance, present a chart parser to improve the support of the writing process using a machine-oriented CNL for English in AceWiki. The ACE editor is a predictive system that helps users to construct valid sentences. This edition tool enables the creation of sentences by allowing the user to iteratively select the desired words, simultaneously ensuring that only well-formed sentences in terms of the CNL format are created.

In the context of the PorSimples project, [18] developed an authoring system, SIMPLIFICA, to help authors to produce simplified texts according to the PorSimples

CNL. SIMPLIFICA is an online converter that presents the converted text to the user, allowing him to make some decisions regarding the application of the CNL.

[6] presents an online checker that supports interactive disambiguation of lexical and structural ambiguities. When more than one valid analysis is found for a sentence, the checker indicates whether it is caused by a lexical or a structural ambiguity. The user is asked to choose the meaning intended for the word in question (in the case of lexical ambiguity), or the relevant structural relationship (in the case of structural ambiguity - PP attachment ambiguity, to be more precise, since this is the only kind of structural ambiguity referred to by the author). The user may also choose to rewrite the sentence in an unambiguous way.

Under the scope of the WordNet PortControl project, mentioned above, we developed a non-predictive checker that takes a text provided by the user as its input and verifies whether it conforms to the CLG-Controlled Portuguese rules and format, through an online web interface. The tool identifies problematic structures and lexical items and, in both cases, provides information that guides the user in the process of correcting the text in conformity with the CNL; also, it makes available a Web Service interface that allows other applications to use the checker functionality to create mashups[5] with other applications. For example, it can be combined with an online automatic translator that also provides a Web Services interface.

Being used for didactic purposes only, under the scope of the WordNet PortControl project, WordNet.PT potential for lexicon control is not explored, along the lines of the strategies depicted in this paper. Hence, in the context of the present proposal, the checking tool has to be extended so that it is able to identify lexical ambiguity and provide information on possible solutions based on WordNet.PT, as described in section 3.2, although not using it to produce sentence interpretation (on this issue, see, for instance, [3]), as this is not relevant for our purposes.

In what concerns checking the conformity of texts with the CNL, the checking tools put forward here replicate the sequence of operations already established for the checker previously built: text tokenization – to transform the source text into a series of paragraphs, sentences and, finally, tokens (words, punctuation marks, numbers, etc.); part-of-speech tagging – to classify each word in a sentence as noun, verb, adjective, preposition, etc.; token sequence analysis – to search, within each sentence, for sequences of tokens that indicate non-conformity of the text with the CNL; marking non-conformities – to add markups to the input text that point out the location of non-conformities and the instructions that were not followed by the user.

The most relevant step of the checking process is the one in which conformity with the CNL rules is verified. The complexity and heterogeneity of the specification of the rules requires the use of several flow control (*if*, *if-not*) and logical combination of sequences (*and*, *or*, *not*) as well as several token matching methods. While, in simpler rules, exact token matching is sufficient, more complex token matching is required for others. The following token matching methods are considered: exact token match (for specific words and punctuation marks); token suffix matching; regular verb token

[5] A mashup is an application that uses and combines data or functionalities from two or more sources to create new services.

matching after stemming; irregular verb token matching with either stemming or with a complete list of irregular stemmed forms of the verb; matching of the part-of-speech classification.

However, besides this sequence of operations which had already been defined and implemented in the CLG-Controlled Portuguese checker, the checking tools needed for the control system presented here involve the design of new processing modules. Specifically, the verification of lexical ambiguity within the text requires an algorithm that uses a search index populated with all the ambiguous word forms in WordNet.PT and performs word lookups (and word sequence lookups, since some concepts are denoted by multi-word expressions) for every word found in the text under analysis. The computation efficiency of the word lookups has to be addressed, since these are done for every word in the text. For this reason, the evaluation of different implementation strategies is necessary, in particular, the efficiency of memory based and disk based B-Tree and Hash indexes ([22] and [23], respectively).

4 Final Remarks

In this paper we present a system to control source texts for MT systems in order to obtain better outputs. We specifically focus on strategies to avoid lexical related ambiguity. i.e., ambiguity due to polysemy and structural ambiguity induced by semantic properties of lexical units.

Our approach differs substantively from related work in that it does neither involve lists of lexical constraints nor semantic representations. Rather, the system makes use of WordNet.PT both to identify problematic lexical units and to provide non problematic alternatives.

Although the system presented here is conceived for the PT-EN language pair, the strategies purposed to avoid undesirable polysemic lexical units are target language independent. However, in the cases where ambiguity can or should be preserved, namely when there is a similarly ambiguous lexical unit in the target language (e.g. nominal alternations), access to a wordnet-like resource is needed. Since WordNet.PT is linked to Princeton Wordnet by means of an Inter-Lingual-Index, information for sustainable decisions on this matter for the pair of languages at stake is available. Concerning other languages, for now, the system does not integrate any information, but a similar solution can be encountered by sharing the type of resources referred to, which exist for a considerable number of languages (cf. http://www.globalwordnet.org/gwa/wordnet_table.html). Regarding structural ambiguity induced by semantic properties of lexical units, it is related to head-complements/modifiers order, which although not universal is easily predictable.

Syntactic rules of the Portuguese CNL discussed here, which are out of the focus of this paper, are also conceived under a cross linguistic point of view, making use of constituency parameters as much as possible.

We do not put aside possible shortcomings of the work depicted here, mostly related with its wide scope. However, although a more systematic evaluation of possible drawbacks is needed, we think our approach conciliates two relevant desiderata: innovation and usefulness.

References

1. Marrafa, P., Amaro, R., Chaves, R.P., Lourosa, S., Martins, C., Mendes, S.: WordNet.PT – Rede Léxico-Conceptual do Português 1.6. CLUL (2006), http://www.clul.ul.pt/clg/wordnetpt/index.html
2. Marrafa, P., Amaro, R., Mendes, S., Ibrahimo, N.: CLG - Controlled Portuguese: Controlled Portuguese for Machine Translation and for Portuguese teaching/learning. CLUL/Instituto Camões (2011), http://www.clul.ul.pt/clg/eng/projectos/portcontrol.html
3. Clark, P., Murray, W.R., Harrison, P., Thompson, J.: Naturalness vs. Predictability: A Key Debate in Controlled Languages. In: Fuchs, N.E. (ed.) CNL 2009. LNCS, vol. 5972, pp. 65–81. Springer, Heidelberg (2010), http://www.cs.utexas.edu/users/pclark/papers/cnl09.pdf
4. Marrafa, P.: WordNet do Português - Uma base de dados de conhecimento linguístico. Instituto Camões (2001)
5. Marrafa, P.: The Portuguese WordNet: General Architecture and Semantic Internal Relations. In: DELTA (2002)
6. Mitamura, T.: Controlled Language for Multilingual Machine Translation. In: Proceedings of Machine Translation Summit VII, Singapore (1999), http://www.mt-archive.info/MTS-1999-Mitamura.pdf
7. Mitamura, T., Nyberg, E.: Controlled English for Knowledge-Based MT: Experience with the KANT System. In: Proceedings of the 6th International Conference on Theoretical and Methodological Issues in Machine Translation, TMI 1995 (1995)
8. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English – Not Just Another Logic Specification Language. In: Flener, P. (ed.) LOPSTR 1998. LNCS, vol. 1559, pp. 1–20. Springer, Heidelberg (1999)
9. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
10. Schwitter, R.: English as a Formal Specification Language. In: Proceedings of DEXA 2002, pp. 228–232. Aix-en-Provence, France (2002)
11. Schwitter, R., Tilbrook, M.: Meaningful Web Annotations for Humans and Machines using Controlled Natural Language. Expert Systems 25(3), 253–267 (2008)
12. Sowa, J.F.: Common Logic Controlled English. Draft (2004), http://www.jfsowa.com/clce/specs.html
13. Clark, P., Harrison, P., Jenkins, T., Thompson, T., Wojcik, R.: Acquiring and Using World Knowledge Using a Restricted Subset of English. In: Proceedings of FLAIRS 2005, pp. 506–511 (2005)
14. Clark, P., Harrison, P., Thompson, J., Wojcik, R., Jenkins, T., Israel, D.: Reading to Learn: An Investigation into Language Understanding. In: Proceedings of AAAI 2007, Spring Symposium on Machine Reading, pp. 29–35 (2007)
15. Videira, C., Rodrigues da Silva, A.: Patterns and metamodel for a natural-language-based requirements specification language. In: CAiSE Short Paper Proceedings (2005), http://isg.inesc-id.pt/alb/static/papers/2005/i40-cv-caise2005.pdf
16. Guimarães, M.J.: Interculturalidade na Produção Textual. In: XI Seminário de Tradução Científica e Técnica em Língua Portuguesa (2008), http://dtil.unilat.org/XIseminariofct_ul/guimaraes.html

17. Aluísio, A., Specia, L., Pardo, T., Maziero, E., Fortes, R.: Towards Brazilian Portuguese Automatic Text Simplification Systems. In: Proceedings of The Eighth ACM Symposium on Document Engineering, São Paulo, pp. 240–248 (2008), http://www.icmc.usp.br/~taspardo/DocEng2008-AluisioEtAl.pdf
18. Candido Jr., A., Maziero, E., Gasperin, C., Pardo, T., Specia, L., Aluísio, S.M.: Supporting the Adaptation of Texts for Poor Literacy Readers: a Text Simplification Editor for Brazilian Portuguese. In: Proceedings of the NAACL HLT Workshop on Innovative Use of NLP for Building Educational Applications Boulder, Colorado, pp. 34–42 (2009)
19. Aluísio, S.M., Gasperin, C.: Fostering Digital Inclusion and Accessibility: The PorSimples project for Simplification of Portuguese Texts. In: Proceedings of the NAACL HLT 2010 Young Investigators Workshop on Computational Approaches to Languages of the Americas, Los Angeles, pp. 46–53 (2010), http://aclweb.org/anthology-new/W/W10/W10-1607.pdf
20. Specia, L., Aluísio, S.M., Pardo, T.: Manual de Simplificação Sintática para o Português. Technical Report NILC-TR-08-06, São Carlos-SP (2008)
21. Kuhn, T., Schwitter, R.: Writing Support for Controlled Natural Languages. In: Proceedings of the Australasian Language Technology Association Workshop, Hobart, Australia, pp. 46–54 (2008), http://aclweb.org/anthology-new/U/U08/U08-1.pdf
22. Bayer, R., McCreight, E.: Organization and Maintenance of Large Ordered Indices. In: Mathematical and Information Sciences Report No. 20. Boeing Scientific Research Laboratories (1970)
23. Mehta, D.P., Sahni, S.: Handbook of Datastructures and Applications, pp. 9–15 (1953)

Multilingual Verbalisation of Modular Ontologies Using GF and *lemon*

Brian Davis[1], Ramona Enache[2],
Jeroen van Grondelle[3], and Laurette Pretorius[4]

[1] DERI/NUIG, Ireland
brian.davis@deri.org
[2] Chalmers University, Sweden
ramona.enache@chalmers.se
[3] Be Informed, The Netherlands
j.vangrondelle@beinformed.com
[4] University of South Africa, South Africa
pretol@unisa.ac.za

Abstract. This paper presents an approach to multilingual ontology verbalisation of controlled language based on the Grammatical Framework (GF) and the *lemon* model. It addresses specific challenges that arise when classes are used to create a consensus-based conceptual framework, in which many parties individually contribute instances. The approach is presented alongside a concrete case, in which ontologies are used to capture business processes by linguistically untrained stakeholders across business disciplines. GF is used to create multilingual grammars that enable transparent multilingual verbalisation. Capturing the instance labels in *lemon* lexicons reduces the need for GF engineering to the class level: The *lemon* lexicons with the labels of the instances are converted into GF grammars based on a mapping described in this paper. The grammars are modularised in accordance with the ontology modularisation and can deal with the different styles of label choosing that occur in practice.

Keywords: Controlled Natural Languages, Ontology Verbalisation, Multilingualism, Ontology Lexicalisation, Natural Language Generation.

1 Introduction

As the adoption of ontologies into enterprise application environments grows, new audiences have to deal with ontologies beyond the traditional disciplines that have done so in the past, such as knowledge engineers and ontologists. These audiences range from business users, who need to take ownership of the ontologies, to end users, such as customers or citizens, who are presented with the services based on these ontologies. As the formalisms themselves are often inaccessible to these new audiences, appropriate visualisations are important. Our experience in practice is that business users often overcome their perception of graph oriented visualisations being too technical when gaining experience,

T. Kuhn and N.E. Fuchs (Eds.): CNL 2012, LNCS 7427, pp. 167–184, 2012.

but they remain a challenge for incidental reviewers and end users. Therefore, verbalisation of ontologies into natural language is one of the approaches that is crucial to make ontologies accessible to new audiences.

Being able to provide verbalisation in a multilingual manner is important: Governments and enterprise often offer their products and services in international contexts or to customers of different languages. For instance, Dutch Immigrations offers many of its services based on ontologies [1], and it typically needs to interface with people that do not speak Dutch. Also, governments have to deal with numerous international aspects in legislation when drafting their national laws. Specifically in Europe, large parts of national legislation is either heavily influenced by or originates in European legislation. Sharing ontologies capturing such international legislation and being able to refer to them from local ontologies offers important benefits in areas of productivity and traceability across local practices.

In this paper, we present an approach for ontology verbalisation based on the Grammatical Framework (GF) and the Lexicon Model for Ontologies (*lemon*) that is essentially multilingual and addresses the particular challenges when classes are chosen up front based on consensus, while multiple parties contribute often changing instances individually.

We use the Be Informed business process platform[1] as an example throughout this paper. The verbalisation examples described in [2] are built upon and we show how they can be generated across languages transparently. Finally, we illustrate how the challenges in this example generalise across other examples, including the generic case of verbalising Linked Open Data.

2 Related Work

Ontology Lexicalisation. The *lemon* model builds on previous research for designing lexica for interfacing with ontologies, in particular that of the LexInfo [3] and LIR [4] models, as well as existing work on lexicon (meta-)models, in particular the Lexical Markup Framework (ISO-24613:2008) [5]. In addition, it builds on efforts to link resources via the Web to the ISOcat meta-data registry [6]. *lemon* seeks to scale the modelling of lexical and linguistic information related to concepts in ontologies from very simple to quite complex lexical entries. *lemon* is closely related the SKOS project[2], which attempts to model simple knowledge organisation systems such as thesauri, classification schemes and taxonomies on the Semantic Web.

Ontology Verbalisation and CNLs. The application of CNLs for ontology authoring and instance population is an active research area. *Attempto Controlled English*[3] (ACE) [7], is a popular CNL for ontology authoring and generation. It is a subset of standard English designed for knowledge representation

[1] `http://www.beinformed.com/`
[2] `http://www.w3.org/2004/02/skos/`
[3] `http://www.ifi.unizh.ch/attempto/`

and technical specifications, and is constrained to be unambiguously machine-readable into DRS - Discourse Representation Structure, a form of first-order logic. WYSIWYM (*What you see is what you meant*)[8], involves direct knowledge editing with natural language directed feedback. A domain expert can edit a knowledge base reliably by interacting with natural language menu choices and the subsequently generated feedback, which can then be extended or re-edited using the menu options. With respect to GF, there have been a number of GF grammars using ontologies/databases, such as the grammar representing the structure of SUMO, the largest open-source ontology [4] as a type-theoretical grammar and verbalise it in 3 languages [9]. Moreover, an ontology describing paintings and a database describing over 8,000 artifacts from the Gothenburg City Museum were used for generating descriptions of the paintings in 5 languages [10].

3 Challenges in Verbalising Modular Ontologies

3.1 New Audiences for Ontologies

An example where ontologies are used in business applications is Be Informed's business process platform. It allows representatives of many disciplines and domain experts to capture the definitions and constraints that the business process must meet and have engines infer executable business processes and decision services from them automatically. This provides business users with a large degree of control and agility, as the constraints are easily changed and the process is updated accordingly. The resulting processes are highly contextual and deal well with exceptions: They allow professionals to influence their own work, and are resilient to the effects of overrides as is described in [2].

When ontologies are used to infer business processes in this way, its stakeholders get to interact with the underlying ontologies on a number of levels. The first is obviously understanding the content of the ontology and the consequences for the inferred business process. The ontology becomes part of the system documentation and is used for validation and reviewing at specification time or for reference when using and maintaining the services and its specification/documentation. Also, the ontology driven services have user interfaces that provide dialog to its end users and the ontology typically is the source of many of the textual elements in that dialog, such as the questions asked, the captions in forms, etc. Finally, the end results of the ontology driven services are also based on the underlying ontology. In applications, these results might be represented by generated documents or letters sent to customers. Specifically when decisions are taken based on models, explaining the end result and the underlying argument have strong ties with the concepts and their textual representations in the ontology.

Be Informed's business processes are inferred from an ontology capturing all relevant activities, artifacts, involved roles etc. and the conditional relations

[4] `http://www.ontologyportal.org/`

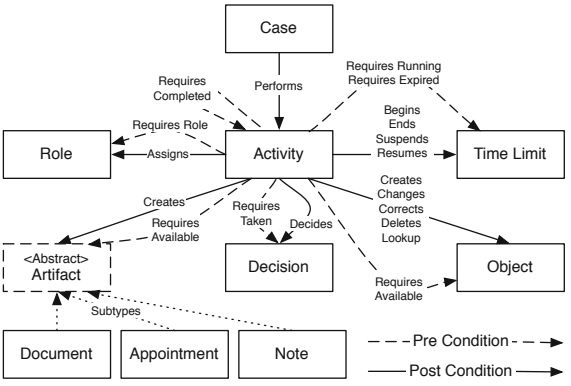


Fig. 1. Summary of the classes used to capture business processes

between these. The meta model, as shown in Figure 1, contains the conceptualisation of the business process domain, that instead of using flow semantics to capture who does what and when, is based on pre- and post-conditions. Each activity may have pre-conditions that have to be met before it may be performed, and post-conditions that capture what conditions have to be met in order for the activity to be complete. Figure 2 contains an of example how this meta model is used to capture the business process of applying for a grant.

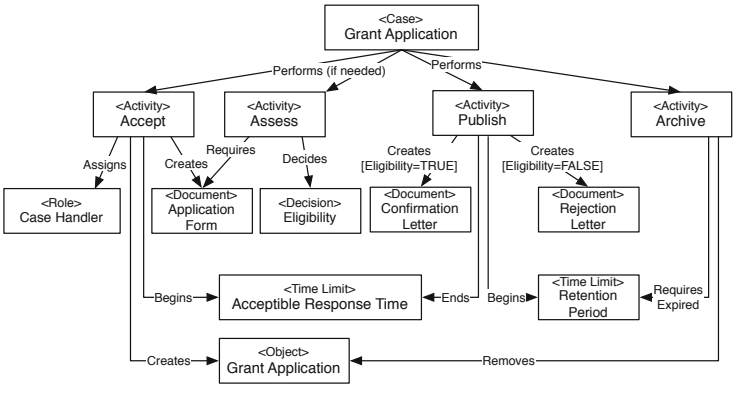


Fig. 2. An example of an grant application process

In [2], this example is discussed in more depth and the model from Figure 2 is verbalised using a pattern sentence approach. The result when verbalising the example used throughout the paper with the pattern sentence approach is the following.

1. The activity Publishing the Result may be performed if
   (a) a document of type Document with Details is available.

2. The activity PUBLISHING THE RESULT is completed if
   (a) a document of type SUBMISSION FORM has been created.

Although the sentences generated have proven useful in practice, some additional requirements exist in areas such as grammatical correctness and fluency. For instance, the first sentence part is grammatically incorrect, as the label inserted is a verb phrase in itself. Also, the pattern sentences approach proves not to scale in terms of number of supported languages. Every grammar translation needs to deal with all lexical aspects (and their peculiarities) of their specific language.

3.2 Differences between Classes and Instances

Within the field of knowledge representation, both classes and instances are typically treated as equal means of expression when creating an ontology and both can be used by the knowledge engineer as he sees fit to best capture the conceptualisation of his choice.

In the use case presented in this paper however, class and instance information are typically in separate, but linked ontologies and these ontologies differ greatly in characteristics related to ownership and rate of change:

- Classes are chosen based on consensus across multiple parties, while the instances are provided by individual parties without requiring consensus on the data;
- The classes are often determined once and are fixed, whereas the instances of these classes are introduced over time and may change often;
- Classes may use an ontology formalism as its native/original representation, where instances often have existing databases or other information stores as primary sources, and the ontology is used exclusively for sharing the data;
- The classes are typically created by knowledge engineers or other information professionals, while instances are created by a wide range of people, who enter data in the original databases, and might not deal with ontologies at all.

In the Be Informed use case provided throughout this paper, the product contains default meta models consisting of the classes that are used in modelling. Although these might be adjusted or extended in individual implementations, this is typically done early in the design phase. An example of such a meta model is shown in Figure 1. When adopting the product, the majority of effort is spent on creating detailed models based on the classes in the meta model. Meta models and their extensions are typically created by knowledge engineers, while the models themselves are built by both knowledge engineers and domain experts. Moreover, it is considered crucial that business users, who adopt the complete model for validation, can make changes and updates.

The choices made in the lexicalisation/verbalisation of classes and instances respectively, follow a similar pattern as the classes and instances themselves:

- In general the labels and lexical representations for the classes are created alongside the ontology that contains the classes and may require guidelines

to be met. The labels of the instances are often chosen by people less or unskilled in knowledge representation or may even originate from existing databases. This compromises the coherent adherence to guidelines for the lexical properties of instance labels;
- Classes may have complex lexical and grammatical consequences, and also introduce sentence patterns and planning. Instances have only labels with limited complexity at a lexical level.

The separation between reaching consensus on the types of things considered and identifying the individual things generalises well across other scenario's of both the use of ontologies and their verbalisation. For instance, ontologies used in Linked Open Data typically expose these characteristics. An example of this is the verbalisation of structured knowledge about 8000 artifacts from the Gothenburg City Museum into natural language descriptions [10]: The conceptualisation of painters, paintings and materials is codified in an ontology with classes, the facts about the individual paintings are exported out of a database into ontologies with instances.

3.3 Practices in Choosing Labels

In practice, different styles of choosing labels are found when modellers are, for instance, modelling and naming concepts within a business process model.

In practice, there is a difference between concepts that are commonly referred to by a proper name (term) and concepts that do not have a term associated with them and are referred to by description. For example, when modelling business processes, the ontology typically contains activities. Some activities may have names (terms), but more often a label describes what is done in the activity. Examples are "Publishing the result", "Publish the result" or "The result is published" (when the activity is completed). This phenomenon also occurs in multilingual contexts when a name for the concept exists in the source language of a model but not in the target language, in which case some form of description is necessary.

Another source of label choosing practices can, especially in business use of ontologies, be found in the diverse background of the people involved in modelling. Typically, the domain experts may have experience in other structured conceptualisation approaches and the naming conventions or practices that come with it. For instance, many information professionals have backgrounds in systems development disciplines and are familiar with techniques like UML and Entity Relational modelling, influencing their naming practices.

Providing guidelines and practices for systematically choosing good labels contributes to reducing the number of label variants used in modelling. However, for industry adoptability and robustness in practice, ontology verbalisation techniques should be able to deal systematically with these challenges and practices, both in terms of the number of constraints required for accurate modelling and also the implications of language variation and multilingualism for any given ontology concept.

4 Ontology Verbalisation Using GF and *lemon*

We present a multilingual ontology verbalisation approach that addresses the challenges discussed in Section 3. It is based on the Grammatical Framework, currently developed in the Molto Project[5], and *lemon*, the Lexicon Model for Ontologies, currently developed in the Monnet Project[6].

Both projects have a strong focus on ontologies and multilingualism, and complement each other well in our approach: GF provides sophisticated multilingual grammar support while *lemon* provides state of the art ontology lexicalisation techniques.

4.1 Introduction to *lemon* and GF

Lexicon **M**odel for **On**tologies, or *lemon* is a model for representing lexical information about words and terms relative to an ontology on the Web. *lemon* is what we term an *ontology-lexicon* in that it expresses how the elements of the ontology, i.e. classes, properties, and individuals, are realised linguistically. The model follows a principle called *semantics by reference* whereby it is assumed that the (lexical) meaning of the entries in the lexicon are expressed exclusively in the ontology. Hence the lexicon merely points to the appropriate concepts. *lemon* is designed to be a basic model supporting the exchange of ontology-lexica on the Semantic Web. The core of *lemon* contains the basic elements required to define lexical entries and their association to their lexical forms and, moreover, to concepts in the ontology representing their meaning. It consists of:

- **Lexicon:** This object represents the lexicon as a whole. It must be marked with a language, and all objects in the lexicon belong to this language.
- **Lexical Entry:** An entry for a given lexicon is a container for one or several forms. It also contains one or more meanings of a lexeme. All forms of an entry must have the same part of speech. An entry may have multiple meanings.
- **Lexical Form:** This is the inflectional form of an entry. It must have one canonical form, but may have any number of other forms. Stems and other partial morphological units can also be modeled as abstract forms.
- **Representation:** A lexical form may have several representations, ranging from different orthographies, to phonetic representation as well as standard written representation.
- **Lexical Sense:** This links a lexical entry to the **reference** in the ontology i.e. a concept, property or instance.
- **Component:** A lexical entry may also be broken up into a number of components.

[5] `http://www.molto-project.eu/`
[6] `http://www.monnet-project.eu/`

The following example gives a simple lexicon with a single lexical entry:

```
@prefix lemon: <http://www.monnet-project.eu/lemon#>.
@prefix bpo: <http://www.beinformed.com/resource/>.

:lexicon lemon:entry:adult_applicant;
  lemon:language "en".

:adult_applicant
  lemon:canonicalForm [lemon:writtenRep "Applicant is Adult"@en];
  lemon:sense [lemon:reference bpo:adult_applicant].
```

This simple English lexicon has a single entry, with canonical form "Applicant is Adult", and a sense that refers to the entry in Be Informed Business Process ontology.

The **G**rammatical **F**ramework (GF) [11] is a formalism for describing multilingual grammars. A GF grammar consists of an abstract syntax, acting as a semantic interlingua and a number of concrete grammars that verbalise the abstract syntax in multiple languages. In this way the semantic level is the central part of the grammar, connecting any language pair of concrete syntaxes.

Most GF grammars are used to describe fragments of natural language. The largest and most general such grammar is the resource grammar library, which contains *resource grammars* for 24 languages, and implements the most common syntactic constructions (such as predication, complementation, etc.) for these languages.

The resource grammar library supports the development of so-called *application grammars* for more restricted domains by providing standard language-specific technicalities so that they do not need to be described again in the new (application) grammar. This makes it easier to develop application grammars by assembling the primitive constructs from the resource grammar and obtain syntactically-correct text in languages from the library.

In the work reported on in this paper GF is used to develop an application grammar for the Be Informed use case, discussed in section 3.

4.2 Modular GF Grammars Based on Decoupling of Classes and Instances

The ontology verbalisation approach exploits the complementary strengths of GF and *lemon*. GF is used to capture ontological information as well as the required sentence structure while *lemon* is the source of concrete label information. The approach is explained by using as example the Be Informed business process ontology and the pre- and post-condition sentence patterns of [2]. The aim of the business processes ontology in Figures 1 and 2 is to specify business processes in terms of pre- and post-conditions, which in turn requires the verbalisation of such conditions.

A challenge resulting from the strict separation of (ownership of) TBox and ABox is that the lexical information required in verbalisation of the complete

ontology also needs modularisation along these boundaries and that restrictions with respect to availability/feasibility of knowledge engineering to the different ontology parts also apply to the lexical information associated with those parts.

The grammar modularisation follows the structure of the meta model in Figure 1 (a similar approach was followed in [10]). The verbalisation proceeds according to the sentence patterns described in [2]. In particular, each activity may have pre-conditions and post-conditions verbalised as *conditional statements* ("A if B"), where A and B are simple *propositional statements* with modalities, as appropriate. All *concept labels* are to be verbalised as propositional statements in accordance with specifications, as explicated in [2].

The modular layered approach to verbalisation is illustrated in Figure 3 by means of the pre-condition triple

```
(Activity,Requires_Available,Artifact subtyped as Document).
```

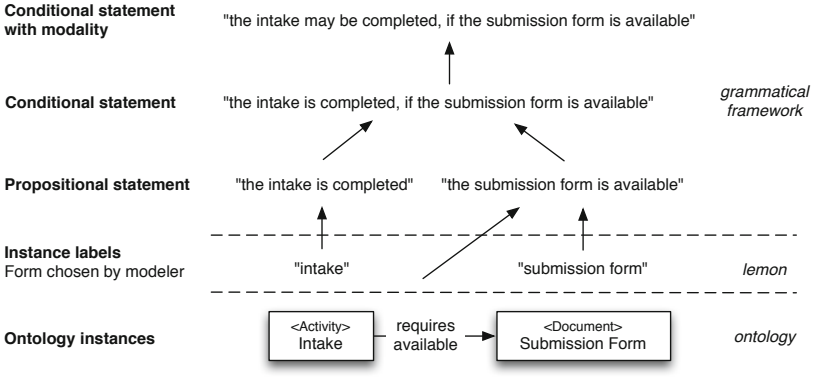


Fig. 3. Layered verbalisation procedure

The TBox abstract syntax caters for the pattern sentence structure (the conditional and propositional statements), the modalities, the concept classes and the relations between them. The polarities and modalities are introduced as separate functions, allowing for the addition of more modalities by merely creating two additional functions per modality (positive and negative form), using the primitive operation already defined. The (meta model) classes are modelled as GF categories and the relations become GF functions with a return type used for verbalisation, as shown in the code segment taken from the TBox abstract syntax module:

```
cat
 Activity;  -- meta model type
 Document;  -- meta model type
 Artifact;  -- meta model type
```

```
  Fragment;  -- type for proposition
  BIText;    -- type for language generation

fun
  requires_available : Activity -> Artifact -> Fragment;
  subDocArtifact : Document -> Artifact;
  FCan : Fragment -> BIText;
```

The conceptual modularisation of the grammar is completed by the ABox abstract syntax, which uses the TBox abstract syntax, and contains the label/instance definitions (ABox information) as GF function declarations with category types, for example,

```
 AIntake : Activity;
 SubmissionForm : Document;
```

The verbalisation of the information captured by the ontology, as well the pattern sentences in which they may occur, is addressed in the concrete syntaxes for English and Dutch. The linearisation categories for the (abstract syntax) categories provide detailed linguistic structure in terms of parts of speech etc, for rendering correct verbalisations. Complex (GF record) types are used to facilitate label variants. The predicates that are built with transitive verbs (creates, deletes, corrects, changes) are rendered in the passive voice, as required by the Be Informed application. The TBox concrete syntax also provides primitives for creating the main categories, for example `mkActivity` (see next section) for `Activity` from basic parts of speech from the resource library, and hides the implementation details from the users, thereby ensuring that the optimisations are as seamless as possible.

The TBox concrete grammar (code segment below) provides the linearisation of both the propositional and the conditional statements. In particular, the overloaded function `mkFragm` implements both the simple propositional statement (as complete sentence pattern) (`hasExt = False`) and the conditional statement (`hasExt = true`) by means of pattern matching on the number and types of the arguments. The overloaded function `mkBIText` completes the verbalisation by finally adding modality and polarity, as necessary. We show `mkFragm` and a part of `mkBIText` by way of illustration.

`Utt` and `S` are GF resource grammar library categories for sentences, questions, etc. and declarative sentences, respectively, while `Fragm` is a user defined complex (GF record) type. `VV` is the the resource grammar library category for verb-phrase-complement verbs.

```
lincat
  Activity  = {noun : NP; subj : NP; vp : VP; hasVerb : Bool};
  Document, Artifact = NP;

lin
```

```
  requires_available ac ar = mkFragm ac.subj ac.vp (mkS (mkCl ar
  (mkVP available_A)));
  subDocArtifact d = d;
  FCan frag = mkBIText frag positivePol may_VV;

oper
  Fragm = {subj : NP; pred : VP; ext : {s : S; hasExt : Bool}};

  ifExt : {s : S; hasExt : Bool} -> S -> S = \ext,s -> case
  ext.hasExt of {
    True  => Sentence.SSubjS s if_Subj ext.s;
    False => s
    };

  mkFragm = overload {
  mkFragm : NP -> VP -> Fragm =
    \np, vp ->
      {subj = np;
      pred = vp;
      ext = {s = dontCareS; hasExt = False}};

  mkFragm : NP -> VP -> S -> Fragm =
    \np, vp, sub ->
      {subj = np;
      pred = vp;
      ext = {s = sub; hasExt = True}};
  };

  mkBIText = overload {
  mkBIText : Fragm -> Pol -> Utt =
    \frag, pol ->
      mkUtt (ifExt frag.ext (mkS pol (mkCl frag.subj frag.pred)));
.......

  mkBIText : Fragm -> Pol -> VV -> Utt =
    \frag, pol, vv ->
      mkUtt (ifExt frag.ext (mkS pol (mkCl frag.subj
      mkVP  vv frag.pred))));
  };
```

In the ABox concrete syntax the labels are defined according to the required types using either provided primitives or basic parts of speech, for example,

```
AIntake = mkActivity (mkNP the_Quant intake_N);
DSubmissionForm = mkNP the_Quant (mkCN submission_N (mkNP form_N));
```

where the functions `mkN`, `mkCN`, `mkNP`, `mkVP`, `mkCl`, `mkS`, `mkUtt`, etc., and many more, are available in the GF resource grammar library for supporting and facilitating application grammar development.

The function `mkActivity`, used in linearising the label `AIntake`, converts the label "intake" to a propositional statement, where the verb to be used with activities expressed as nouns or noun phrases is always "to complete', i.e. "the intake is completed" as prescribed by the meta model in Figure 1. We return to the `mkActivity` function in Section 4.3 where the handling of label variants is discussed.

The final (customary) component in the suite of modules that constitute a GF application grammar is a dictionary of application specific lexical items that do not occur in the resource grammar lexicon and additional linguistic constructs that are language specific and/or are not included in the resource grammar, for example, verb nominalisation. Examples (showing the abstract as well as the concrete syntax) are as follows:

```
oper
 intake_N:N;
 submission_N:N;
 form_N:N;
```

and

```
oper
 intake_N = mkN "intake";
 submission_N = mkN "submission";
 form_N = mkN "form";
```

An example of a pre-condition triple and its verbalisation in English and Dutch by means of the GF grammar is follows:
(`Activity`, `Requires_Available`, `Artifact`, subtyped as `Document`), instantiated with the labels "intake" and "submission form", is rendered as the pre-condition, containing the propositional statements "the intake may be completed" and "the submission form is available", with modality added:

```
The intake may be completed , if the submission form
is available.
De inname kan worden afgerond , als het aanvraagformulier
beschikbaar is.
```

Other typical examples of linearisations are as follows:

Pre-conditions:

```
44b may be completed , if the document with details
is available.
44b kan worden afgerond , als het document met details
beschikbaar is.
```

Post-conditions:

```
If 44b has been completed , the submission form has been created.
Als 44b afgerond is , is het aanvraagformulier aangemaakt.
```

Propositional statements (intermediate layer):

```
44b has been completed.
44b is afgerond.
```

In summary, the modularisation of the grammar was illustrated by showing how concepts and relations between them are introduced as categories and functions in the TBox abstract syntax and reused in the in the ABox abstract syntax to instantiate abstract instances. The concrete syntaxes were shown to provide linearisations, using different variants, for the instance labels, the propositional, and the conditional statements that consitute the pre- and post-conditions in the ontology.

4.3 Dealing with Different Variations of Labels

In the previous section the focus was on modularisation as a mechanism to support efficient and effective ontology development. By separating the TBox and ABox information in different grammar modules the modeller is allowed to concentrate on application dependent information while generic business process modelling support is provided by the grammar. In this section we discuss the extent to which the BI grammar may allow for label variants and what kinds of variation is accommodated at present.

Basically, the grammar allows two broad kinds of labels. Firstly labels in the form of nouns or compound nouns (names or terms) are permitted, for example, "Intake" and "Equality principle". Secondly, labels may take the form of a proposition such as "The Result is published" or other verb oriented style labels such as "Publishing the result" or "Publish the result".

While allowing the modeller some freedom of choice in label selection, the verbalisation of label variants that refer to the same concept or relation in the ontology should be unique.

The following code fragment from the TBox concrete syntax shows how some of these design choices may be implemented:

```
mkActivity = overload {
mkActivity: N -> V2 -> NP -> Activity = \n,v,o -> lin Activity {
   noun = mkNP the_Quant (mkCN n (mkAdv of_Prep o));
   subj = o;
   vp = passiveVP v;
   hasVerb = True
};
mkActivity: V2 -> NP -> Activity = \v,o -> lin Activity {
   noun = nominalize (mkVPSlash v) o;
```

```
    subj = o;
    vp = passiveVP v;
    hasVerb = True
 };
 mkActivity: NP -> Activity = \o -> lin Activity {
    noun = o;
    subj = o;
    hasVerb = False
 }
};
```

In this way, the ABox concrete syntax may be used to create an object of type `Activity` by, for example, either providing an `NP` for simple cases like "Intake" or a `V2` or an object `NP` for labels such as "Publishing the result" and "Publish the result".

There are a number of other fields that are used in the English TBox concrete syntax that ensure an optimized natural language generation such as "The Results are published" or "Intake is completed if the results are published", but these details are just handled in the grammar, and the users of the ABox syntax do not need to be aware of them.

It should be noted that these choices of implementation are, to some extent, language specific since in labels of the latter kind the English gerund is used while in Dutch the infinitive form of the verb plus "van" is customary.

In the following example the label variant "publish the result" is rendered as in the sentences below:

Post-condition triple:

```
(Activity, Creates_Artifact, Document)
```

Linearisation:

```
If the result has been published , the submission form
is available.
Als het resultaat gepubliceerd is , is het aanvraagformulier
beschikbaar.
```

4.4 Multilingual Aspects

In the GF grammar the multilingual correspondence of the bilingual system is via the ontological labels. For instance, "Publishing the result" in English corresponds to "Publiceren van het resultaat" in Dutch because they both map to the instance `APublishingOfResult`. However, word-wise there is no one-to-one correspondence, since this would not scale up when adding more languages and extending the ontology with more instances. From this perspective it resembles the *lemon* notion of multilingualism and supports the automated mapping from *lemon* to GF, as discussed in a subsequent section.

Moreover, since the TBox syntax mainly uses the resource library and alignable primitives from there, it can be abstracted in a functor, so that the new languages can inherit it, at least partially. In this way, adding a new language would be a simpler process, since it would mainly require the writing of a new lexicon and translating the instances from the ABox concrete syntax. An investigation into this aspect forms part of future work.

4.5 Generating the Instance Grammars from *lemon*

We reiterate that the exploratory work presented in this paper focuses on three aspects of modular ontology verbalisation, as also illustrated in Figure 3. In section 4.2 we addressed the modularisation of the GF grammars, used for the verbalisation, in accordance with the modular ontologies. It was noted that the labour intensive component of the product is the creation of the ABox grammars (that contain instances and their labels) and that the flexibility in specifying labels is of strategic importance. Section 4.3, therefore, focused on label variants currently allowed by the GF grammar. In this section we now briefly turn our attention to the third aspect viz. an automated way of populating the ABox grammars by making use of *lemon*. The approach is based on the observation that the *lemon* entries contain all the essential information required to construct GF lexicon entries and linearisation rules for the instance labels in the ABox grammars.

We illustrate this by means of an example in which essential parts of the linguistic information, necessary to create an instance label, are shown. We consider the entry for the instance label "Sketch of the situation", which is represented as a *lemon* `decomposition`:

```
 #Sketch of the situation
lemon:decomposition ( :sketch_component :prep_component
                :det_component :sit_component );

lemon:phrase_Root
[ lemon:constituent :NP;
 lemon:edge [ lemon:constituent :NN; lemon:leaf:sketch_component];
 lemon:edge [ lemon:constituent :PP;
  lemon:edge [ lemon:constituent :P; lemon:leaf:prep_component];
  lemon:edge [ lemon:constituent NP;
   lemon:edge [ lemon:constituent :DET; lemon:leaf:det_component];
   lemon:edge [ lemon:constituent :NN; lemon:leaf:sit_component
         ]]]];

:sketch_component lemon:element sketch.
:sketch a lemon:Word .
:sketch isocat:partOfSpeech isocat:noun .

:prep_component lemon:element of.
```

```
:of a lemon:Word .
:of isocat:partOfSpeech isocat:preposition .

:det_component lemon:element the.
:the a lemon:Word .
:the isocat:partOfSpeech isocat:determiner.

:sit_component lemon:element situation.
:situation a lemon:Word .
:situation isocat:partOfSpeech isocat:noun .
```

It consists of a `phrase_Root` entry which describes the syntactic decomposition of the entry. The *lemon* entry is decomposed into a noun phrase, in turn decomposed into a noun, a preposition, a determiner and another noun. Each component in the *lemon* decomposition is comprised of `elements`, for example the preposition `of` a the noun `situation`.

In GF the same information is encoded as follows, making use of functions from the resource grammar library:
ABox abstract syntax:
`DSketchOfSituation : Document` where `Document` is a class in the ontology and a category in GF.
ABox concrete syntax:
`DSketchOfSituation = mkNP a_Quant (mkCN sketch_N (PrepNP of_Prep (mkNP the_Det situation_N)))` where `mkNP`, `mkCN` and `PrepNP` are functions from the resource grammar library.
Dictionary abstract syntax: `sketch_N : N` and `situation_N : N`
Dictionary concrete syntax: `sketch_N = mkN ‘‘sketch’’` and `situation_N = mkN ‘‘situation’’`.

The constituent structure (syntax tree) of the label “Sketch of the situation” is given in the first part of the *lemon* entry and by the GF ABox concrete syntax entry, while the lexical information is available in the second part of the lemon entry and the GF dictionary entry. An appropriate generalisation of this correspondence (mapping) will cover all possible *lemon* entry syntax trees and their GF equivalents. This will form the basis for an automated procedure to generate ABox concrete syntax entries for instance labels from *lemon* and is part of our future work. Furthermore, this approach is well suited to multilingual entries in *lemon* and their representation and verbalisation in GF, the investigation of which also forms part of future work.

5 Discussion and Future Work

In this paper we demonstrated how GF and *lemon* can be combined to provide multilingual verbalisation in a specific class of problems, where ontologies are modularised into ontologies containing consensus based classes and (multiple) instance ontologies. In these scenarios, the instances are typically not created

by knowledge engineers and are often based on information stores other than ontologies.

We showed an approach to grammar development that leads to grammars being modular along the same boundaries as the ontologies, with the grammar modules also having the same dependency structure as the ontology parts.

Additionally, we introduced mechanisms in the TBox grammar that deal automatically with the different styles of label choosing that are encountered in practice. This is particularly relevant in cases where different disciplines are involved in creating the instances and in multilingual cases where concepts may have a name in one language and can only be referenced by describing them in another language.

By generating the instance grammars from the ontology labels encoded in *lemon*, the need for GF engineering at the instance level is reduced. This is crucial for the adoption of this mechanism, as the instance data either already exists in databases or is created by people who may not be expected to engineer GF representations of the labels they choose.

In terms of multilingualism this approach supports and suggests two ways of verbalisation. In cases where translations of the labels are available inside *lemon* lexica, multiple ABox syntaxes can be generated for the different languages. This approach has the benefit that labels may differ quite significantly across languages, but obviously requires translation of the ontology. Alternatively, translation could be performed at the GF grammar level if the label styles align across languages. In cases where aligned dictionary grammars are available, this approach could prove particularly efficient.

Future work also includes the extension of this approach to include different levels of paraphrasing and advanced sentence planning in order to achieve improved fluency in and across sentences. Finally, we envisage investigating label choosing practices as encountered amongst professionals in the different fields that may benefit from a framework as described in this paper. The importance of robustness under lexical variance as found in real world applications suggests an in depth study of label style variation and its impact on ontology verbalisation.

Acknowledgements. The authors wish to gratefully acknowledge the support for this work by the European Commission (EC). This work was partially funded by the EC within the EU FP7 Multilingual Ontologies for Networked Knowledge (MONNET) Project under Grant Agreement No. 248458 and the MOLTO Project under Grant Agreement No. 247914.

References

1. Heller, R., van Teeseling, F.: Knowledge Applications for Life Events: How the Dutch Government Informs the Public about Rights and Duties in the Netherlands. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 846–850. Springer, Heidelberg (2009)

2. van Grondelle, J.C., Gülpers, M.: Specifying Flexible Business Processes Using Pre and Post Conditions. In: Johannesson, P., Krogstie, J., Opdahl, A.L. (eds.) PoEM 2011. LNBIP, vol. 92, pp. 38–51. Springer, Heidelberg (2011)
3. Buitelaar, P., Cimiano, P., Haase, P., Sintek, M.: Towards Linguistically Grounded Ontologies. In: Aroyo, L., et al. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 111–125. Springer, Heidelberg (2009)
4. Montiel-Ponsoda, E., de Cea, G., Gómez-Pérez, A., Peters, W.: Modelling multilinguality in ontologies. In: Proceedings of the 21st International Conference on Computational Linguistics (COLING) (2008)
5. Francopoulo, G., George, M., Calzolari, N., Monachini, M., Bel, N., Pet, M., Soria, C.: Lexical markup framework (LMF). In: Proceedings of the 2006 International Conference on Language Resource and Evaluation (LREC) (2006)
6. Kemps-Snijders, M., Windhouwer, M., Wittenburg, P., Wright, S.E.: ISOcat: Corralling data categories in the wild. In: Proceedings of the 2008 International Conference on Language Resource and Evaluation (LREC) (2008)
7. Fuchs, N.E., Kaljurand, K., Kuhn, T.: Attempto Controlled English for Knowledge Representation. In: Baroglio, C., Bonatti, P.A., Małuszyński, J., Marchiori, M., Polleres, A., Schaffert, S. (eds.) Reasoning Web 2008. LNCS, vol. 5224, pp. 104–124. Springer, Heidelberg (2008)
8. Power, R., Scott, D., Evans, R.: What you see is what you meant: direct knowledge editings with natural language feedback. In: Prade, H. (ed.) 13th European Conference on Artificial Intelligence (ECAI 1998), pp. 677–681. John Wiley and Sons, Chichester (1998)
9. Angelov, K., Enache, R.: Typeful Ontologies with Direct Multilingual Verbalization. In: Rosner, M., Fuchs, N.E. (eds.) CNL 2010. LNCS, vol. 7175, pp. 1–20. Springer, Heidelberg (2012)
10. Dannélls, D., Enache, R., Damova, M., Chechev, M.: Multilingual online generation from semantic web ontologies. In: WWW 2012, EU projects track, Lyon (April 2012)
11. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011) ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth)

Author Index