Towards Development of Multilingual Spoken Dialogue Systems

Hartwig Holzapfel

Interactive Systems Laboratories
Universität Karlsruhe
Germany
hartwig@ira.uka.de

Abstract

Developing multilingual dialogue systems brings up various challenges. Among them development of natural language understanding and generation components, with a focus on creating new language parts as rapidly as possible. Another challenge is to ensure compatibility between the different language specific components during maintenance and ongoing development of the system. We describe our experiences with designing multilingual dialogue systems and present methods for multilingual grammar specification, as well as development and maintenance methods for multilingual grammars used for language understanding, and multilingual generation templates. We introduce grammar interfaces, similar to interface concepts known from object oriented languages, to improve compatibility between different grammar parts and to simplify grammar development.

1. Introduction

New applications are emerging that make use of customized speech software, such as humanoid robot interaction, telephony services, or information kiosks. Especially these services are (of course) not restricted to a single user, neither are they restricted to a single language. This can either mean that one system is deployed to interact with different users speaking different languages. But it can also mean that a company might be interested in deploying one system in a different country, and while keeping the functionality the same, porting the system to a new interaction language.

Here we want to present our efforts that we have so far undertaken within two projects developing multilingual systems and porting existing systems to new languages, as well as new work that we wish to undertake in the near future to ensure compatibility between different language modules. We present methods that reduce the difficulty in porting grammars to new languages and to ensure their compatibility during changes of the system.

We have been developing multilingual spoken dialogue systems for an intelligent room within the FAME project[1]. and a humanoid robot platform (Stiefelhagen et al., 2004; Gieselmann et al., 2003) within a German collaborative research effort named SFB588[2]. The languages created for the FAME system are English, Spanish and German; English and German for the humanoid robot.

Section two gives an overview of related work. Section three describes our dialogue system. Section four, five and six describe the natural language component with multilingual extensions and grammar interfaces, the multilingual generation part and generation of language resources from databases. Section seven covers maintenance, while section eight gives an outlook to future work.

[1]The FAME project: http://isl.ira.uka.de/fame, presented at IST 2004 Den Haag, and the Barcelona FORUM 2004 http://www.barcelona2004.org

[2]SFB588 - collaborative research effort on humanoid robots: http://www.sfb588.uni-karlsruhe.de

2. Related work

Some of the above parts, namely setting up a new system from scratch or porting an existing dialogue system to a new language, relate to the field of rapid prototyping. So far, some approaches have been taken to support rapid prototyping for spoken dialogue systems like in ARIADNE (Denecke, 2002). Other work focuses on the definition of meta grammars to design multilingual grammars (Kinyon and Rambow, 2003), and facilitate the generation of parallel multilingual grammars (Clement and Kinyon, 2003), by using a metagrammar that is annotated with syntactic properties. (Denecke, 2000) describes separation of language specific syntax and language independent semantics, (Kim et al., 2003) describes methods for grammar porting.

3. The Dialogue System

For dialogue management we use the TAPAS dialogue framework. TAPAS uses dialogue algorithms developed within the language and domain independent dialogue manager ARIADNE (Denecke, 2002) which is specifically tailored for rapid prototyping of spoken dialogue systems. The dialogue manager uses typed feature structures (TFS) (Carpenter, 1992) to represent semantic input and discourse information. A context-free grammar is used to parse the user utterance. The grammar is enhanced by information from the ontology defining all the objects, tasks and properties about which the user can talk. After parsing, the parse tree is converted into a semantic representation and added to the current discourse. The dialogue manager unifies compatible information in discourse or opens subdialogues to process new threads. If all necessary information to accomplish a goal is available in discourse, the dialogue system calls the corresponding service. If some information is still missing, the dialogue manager generates questions to request this information.

For speech recognition, we are using the Janus Recognition Toolkit (JRTk) (Finke et al., 1997) with the Ibis single pass-decoder (Soltau et al., 2001). We use the option of Ibis to decode with context free grammars (CFG) instead of statistical n-gram language models (LM). These context free grammars are generated by the dialogue manager that uses the same grammars to convert the resulting parse tree into typed feature structures. In the same way, the dialogue manager can be used in combination with other speech recognizers that can decode with context free grammars, by providing grammars in SOUP, PHOENIX, JSAPI, and Microsoft SAPI formats. In addition, the system offers a tighter integration with Janus by being able to weight (e.g. boost) different grammar rules depending on the dialogue context (Fügen et al., 2004).

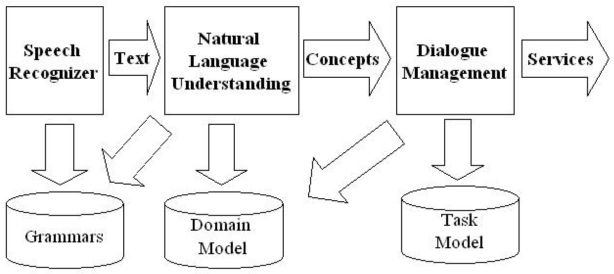

Figure 1: Flow diagram showing the integration of NLU component into the dialogue system.

The dialogue system uses semantic grammars (Gavalda, 2000) to interpret spoken (or typed) input. The integration of the grammars (natural language understanding) into the system is shown in figure 1. The processing of the dialogue algorithms and the discourse representation are language independent. This allows using general dialogue and discourse algorithms, without depending on language specific peculiarities.

The dialogue manager controls an application (e.g. robot control) through an API (application programming interface) that is defined by the application. This API is language independent and defines the services that can be executed by the system. Since the dialogue manager is task-oriented, most of the dialogue goals are created to collect information that is required to execute functions defined by the application's API. Additional dialogue goals (and subgoals) allow non task-oriented communication, such as greetings, error correction, etc. More language specific information is required to generate clarification questions or respond to the user. Generation is accomplished by generation templates. They have a language independent interface, representing the concepts of what the template generates, and a language specific generation component, that is hidden behind the language independent interface.

4. Natural Language Understanding

One important part of a multilingual dialogue system is the development of natural language understanding grammars. We use vectorized context free grammars that allow a combination of semantic and syntactic tags to be used (Denecke, 2000) for the definition of nonterminal symbols.

4.1. Multilingual Grammars with Grammar Inheritance

Vectorized context free grammars are similar to (semantic) context free grammars, in that they exist of a set of nonterminal symbols, a set of terminal symbols, a set of rules and a set of start symbols, and semantic annotation of right hand side elements of a rule. In addition, vectorized context free grammars allow multiple inheritance of nodes and definition of vectorized nonterminal symbols. A vectorized nonterminal consists of a vector $< e_1, e_2, ..., e_n >$ where e_i is an element of a partially ordered set V_i.

Our implementation defines a nonterminal as a quadruple with the elements semantic concept, syntactic category, subcategory, and language. The grammar inheritance mechanism follows the general approach of vectorized context free grammars. By using inheritance, the grammar writer can define general language independent rules and inherited rules for each language.

Vectorized grammars allow a separation and modularization of generic and specific grammar parts. Generalization of concepts includes domain-independent and language independent information. The above definition allows to create separated language resources as well as multilingual resources. Separated language resources facilitate that different grammar writers work on different language grammars. Multilingual resources are useful to specify declaration with similar structure, e.g. importing information from a database, where the import statements only differ in language, see example in section 5.

4.2. Grammar Interface Definitions

Similar to object oriented languages like Java, our grammar formalism allows definition of abstract classes and interface classes. An abstract class defines a mixture of implemented rules, as well as a set of frame definitions (similar to deferred rules). Abstract classes allow the definition of semantic frames that need to be implemented by rules in different languages. Thus, frame definitions resemble abstract methods known e.g. from Java.

A grammar interface is similar to an abstract class, but doesn't allow implemented rules, just frame definitions. While an interface is language independent, an abstract class implements language specific code, and is thus either bound to a language or contains multilingual grammar nodes.

The definition of grammar interfaces can thus be used to assure equivalent functionality in different language implementations. An example of defining and implementing a grammar interface is given in section 5.

The aim, when developing a new language, is to create new language-specific grammar parts that are converted to the same (language independent) semantic representation as existing languages. Besides using grammar interfaces that define what functionality must be covered, we want to assist the grammar writer in giving him examples from other, existing languages. Therefore, the system provides a list of all possible semantic input structures. Plus a mapping from existing parse-trees to the semantic structures they are converted to. The grammar developer can then follow this mapping and rewrite language specific grammar parts. The initial English grammar was developed by two researchers working in the field of grammar development. However, translating the robot system's grammar from English to German was accomplished by a non-expert student (German native speaker, with little experience in grammar development). The same scheme was also applied to the German generation component.

5. Multilingual Generation

The other main part that has to be developed for a multilingual dialogue system, is the spoken output of the system, which is called speech generation. To allow language independent dialogue management, the system uses template functions. They are defined with language independent concepts representing the output they generate. The dialogue manager only uses the language independent concept representation. The template functions are implemented in the language-specific part and generate text in the given language. Therefore they have to translate concepts that are passed as parameters and variables pointing to discourse elements, which are also semantic concepts.

Since the variability of the system's spoken output is much smaller than the possible user input, this part means less effort to translate than the input grammars. However, there might be a long list of objects loaded from a database. Their generation information is read from the database, equivalent to the generation of natural language understanding grammars from database.

Depending on the type of language that is required for the system, these template functions can be very simple or very complex. Languages like English can be added very easily. However, highly inflecting languages require more attributes such as case and gender to correctly generate the sentence including the inflected noun and its propositions.

The following code line shows an invocation of a template function informing the user that a given object is available.

```
#Available {$objs.first.[OBJ|NAME]}
```

The String "$objs.first.[OBJ|NAME]" calls a dialogue variable by accessing discourse information with the given typed feature structure path "[OBJ|NAME]". The English generation template looks like this: "{0} is available" and generates the response "Foo is available". Since the object is read from a database it is only referenced by its language independent semantic representation, e.g. object-id like a unique name. The generation components converts the object id to the string "Foo" by using a translation table. The above example is very simple, also for other languages than English, since only the nominative is applied and the word is not inflected. For other cases the object string is extended with additional case-tags in the generation string. For example, "{0}[akk]" can be specified to extended the table lookup using case and/or gender information for text generation. More complex sentences (for inflecting languages as German and especially Slavic languages) where gender or case agreement is required (e.g.) for adjectives and nouns require more complex generation capabilities. This can be handled with unidirectional references (generation of adjective references gender of noun) or full unification of case and gender tags.

6. Grammar-Generation from Database Information

The system supports generation of grammar rules from database information. This mechanism is used both in the FAME system, and in the humanoid robots system. The FAME system offers a multimodal interface to retrieve and discover information about previously recorded lectures during a seminar or a conference. There, all information about lectures, speakers, topics, keywords, etc. is stored in a database. The system uses this information to generate grammar rules that allow the user to query e.g. for a speaker. The big advantage is that new data can be added to the database, without changing the dialogue manager code or its grammar files. Similar, in the humanoid robot system, databases are used to describe the robot's environment model including the description of all objects it knows about.

Furthermore, in this database also translation information can be found to support grammar-development for multiple languages and multilingual grammars. It contains multi-lingual information for all stored objects. When adding new objects, such as titles of the talks, weekdays, information on lecture places, etc., this information is added for all existing languages. The language specific information is then used by the language specific grammar parts and integrated as terminal symbols. Using grammar interfaces and rule inheritance, the definition of a rule containing the name of a lecture looks as follows. A rule interface (frame definition)

```
iframe <act_askPlace,VP,_> =
<obj_Place,N,_,_> {PLACE obj_Place}
```

requires that the rule $< obj\_Place, N, \_, \_ >$ is defined in all implementing language specific grammars. Then two rules, one for English (with language tag EN) $< obj\_Place, N, \_, EN >$ and one for Spanish (with language tag ES) $< obj\_Place, N, \_, ES >$ are specified. The right hand side of both rules contains a database import instruction. The import statement specifies the database location, table, field, and semantic value.

```
<obj_Place,N,_,ES> = import
$db Lecture PlaceES { PLACE import }
```

7. Maintenance

Our current efforts are to support maintenance of multilingual grammars. The basic idea is to record changes that are done in one language and to project these changes to grammar parts of the remaining languages. The projection results in an information ("todo"-item) annotating which grammars are likely to require improvement. Similar to the mapping described above, the todo-items are sought by computing the mapping between changed nodes and their projected TFS. The affected typed feature structures are marked. Then, for all remaining languages the inverse mapping is computed between grammar nodes/parse trees and the projected semantic representation. Note that the inverse mapping is no longer a mathematical function any more, but still a mapping. When thinking of a graph, one can imagine edges from grammar nodes to TFSes, then finding the inverse mapping is simply looking at all incoming edges in a TFS-list. All grammar nodes that are connected to a marked TFS via an edge are then marked with todo-items and the original change.

The presented approach uses offline methods to detect changes. This is done by comparing changes (as detected by a cvs or subversion[3]) in the grammar tree to the previously stored version. Other possibilities are to use design-time listeners that are integrated into the editor and track complex changes while they occur. This is better suited, and offers better performance to track changes such as moving tree parts. However, this requires that the developer must use this editor, and changes cannot be computed offline.

8. Conclusion and Outlook

We have presented our experiences with creating and maintaining multilingual dialogue systems. The dialogue system contains a language independent core and language specific understanding and language generation components, and supports mechanisms to load language specific information from a database during runtime. We have presented new methods to multilingual grammar design by using grammar interfaces and abstract rules, resembling interfaces and abstract classes as known from object oriented programming languages.

8.1. Towards an Integrated Development Environment

After collecting experiences with the design of multilingual dialogue systems and still going to continue in this area, we are undertaking efforts, to integrate existing tools into an integrated development environment (IDE). So far we have been integrating some tools into the open source platform Eclipse, which currently is the most common IDE platform available. Existing tools for natural language processing do not cover enough IDE functionality for multilingual editing and object oriented grammars. The functionality shall cover methods for code navigation and runtime code-checking, resembling functionality from IDEs for object oriented languages. In addition we want to integrate tools for evaluation, testing, and comparison of multilingual resources, including change management of reference languages to automatically create translingual todo-items during maintenance and ongoing development of parallel multilingual resources.

9. Acknowledgements

We would like to thank Petra Gieselmann, Patrycja Holzapfel, Marta Tolos and Raquel Tato for doing major parts of the grammar development and Ulf Krum and Tang Ting for ongoing efforts in assisting me to provide grammar development tools.

This work was supported in part by the German Research Foundation (DFG) as part of the SFB 588 on humanoid robots and within the FAME project by the European Union as IST-2000-28323, and by the European Commission under project CHIL (contract #506909).

[3]cvs and subversion are commonly used version control systems. Both use ”diff-tools” to detect changes in files.

10. References

Carpenter, B., 1992. *The Logic of Typed Feature Structures*. Cambridge University Press.

Clement, Lionel and Alexandra Kinyon, 2003. Generating parallel multilingual lfg-tag grammars from a metagrammar. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Denecke, Matthias, 2000. Object-oriented techniques in grammar and ontology specification. In *Proceedings of the Workshop on Multilingual Speech Communication*.

Denecke, Matthias, 2002. Rapid prototyping for spoken dialogue systems. In *Proceedings of the 19th International Conference on Computational Linguistics*. Taiwan.

Finke, M., P. Geutner, H. Hild, T. Kemp, K. Ries, and M. Westphal, 1997. The karlsruhe-verbmobil speech recognition engine. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing, ICASSP-97*. Munich, Germany.

Fügen, Christian, Hartwig Holzapfel, and Alex Waibel, 2004. Tight coupling of speech recognition and dialog management - dialog-context grammar weighting for speech recognition. In *Proceedings of the International Conference on Spoken Language Processing, ICSLP 2004*.

Gavalda, Marsal, 2000. Soup: A parser for real-world spontaneous speech. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT-2000)*.

Gieselmann, P., C. Fügen, H. Holzapfel, T. Schaaf, and A. Waibel, 2003. Towards multimodal communication with a household robot. In *Proceedings of the International Conference on Humanoid Robots*.

Kim, Roger, Mary Dalrymple, Ronald M. Kaplan, Tracy Holloway King, Hiroshi Masuichi, and Tomoko Ohkuma, 2003. Multilingual grammar development via grammar porting. In *Proceedings of the ESSLLI 2003 Workshop on Ideas and Strategies for Multilingual Grammar Development*.

Kinyon, Alexandra and Owen Rambow, 2003. Using a metagrammar for parallel multilingual grammar development and documentation. In *Proceedings of the ESSLLI 2003 Workshop on Ideas and Strategies for Multilingual Grammar Development*.

Soltau, H., F. Metze, C. Fuegen, and A. Waibel, 2001. A one pass- decoder based on polymorphic linguistic context assignment. In *Proceedings of the Automatic Speech Recognition and Understanding Workshop, ASRU-2001*. Madonna di Campiglio, Trento, Italy.

Stiefelhagen, R., C. Fügen, P. Gieselmann, H. Holzapfel, K. Nickel, and A. Waibel, 2004. Natural human-robot interaction using speech, gaze and gestures. In *Proceedings of the International Conference on Intelligent Robots and Systems*. Sendai, Japan.