Dialogue System Localization with the GF Resource Grammar Library

Nadine Perera
Department of Man Machine Interaction
BMW Group Research and Technology
Munich, Germany
nadine.perera@bmw.de

Aarne Ranta
Department of Computing Science
Chalmers University of Technology
and Göteborg University
Gothenburg, Sweden
aarne@cs.chalmers.se

Abstract

We present two experiments in the localization of spoken dialogue systems. The domain of the dialogue system is an MP3 application for automobiles. In the first experiment, a grammar in Nuance GSL format was rewritten in Grammatical Framework (GF). Within GF, the grammar was extended from two to six languages, giving a baseline for semantically complete grammars. In the second experiment, the German version of this baseline GF grammar was extended with the goal to restore the coverage of the original Nuance grammar.

1 Credits

Part of this work was done under the TALK[1] research project, funded by EU FP6 [ref. 507802]. The Nuance grammar was written by Jochen Steigner, Peter Poller, and Rosemary Stegmann. The first GF experiment was made together with Björn Bringert. The Spanish grammar was written together with Libertad Tansini.

2 Introduction

Spoken dialogue systems for cars emerged in the late 1990s with the appearance of advanced information and communication systems. Driving a car is a classical visual manual task, as the driver should keep his hands on the steering wheel and his glance on the surrounding traffic and the street. Speech interaction is very well-suited for secondary-level tasks such as handling information and entertainment systems.

The current spoken dialogue system in the automobiles of the BMW Group is a Command&Control-based system (Hagen et al., 2004). For the interaction with the entertainment and information functions of the iDrive system (Haller, 2003), the paradigm pursued is You-Can-Speak-What-You-See, i.e. every menu item or option that is shown on screen can be spoken. The localization of that spoken dialogue system for currently eight languages is done manually by translators, without advanced automation methods or special tools. The Command&Control-based approach has its disadvantages, as the system can only handle a fix set of commands. This makes it difficult for system novices to interact with the dialogue system because they may not know the commands they need to say to reach their goal.

Advanced conversational dialogue systems that allow a more flexible input and let the user decide about the form and the amount of the communicated information are being investigated. In order to implement such a flexible spoken dialogue system in the automobiles of the BMW Group, not only one dialogue system, but at least eight would have to be built - one for each language. The different, localized versions of the system would have to be designed in a way that allows for the generic addition of use cases, i.e. changes and additions to the German grammar (which is viewed as the initial source grammar) must be ported to the localized versions consistently and without the need to change the whole localized grammar.

2.1 Grammar Writing

Linguistic experts who write grammars for companies whose focus is not in language technology usually have to possess profound technical competence and programming skills in addition to linguistic expertise. For those grammar engineers who are computer scientists or engineers with little university ed-

[1]Tools for Ambient Linguistic Knowledge, www.talk-project.org

Proceedings of SPEECHGRAM 2007, pages 17–24,
Prague, Czech Republic, June 2007. ©2007 Association for Computational Linguistics

ucation in linguistics, a programming paradigm enabling them to avoid dealing with the morphological inflection paradigms of several languages would certainly be welcome. Writing consistent grammars for multiple languages is quite challenging: Writing one grammar requires the grammar engineer to be at least a fluent speaker of the language the grammar covers. If he also knows another language quite well, he may be able to localize a grammar from that language to his native language. This implies that for every language which requires a localized grammar, a person who knows the source language and is a native speaker of the target language is needed. At the moment, there is no commercial tool available that helps grammar engineers with the localization of spoken dialogue systems.

2.2 The Nuance SAMMIE grammar

Within the TALK project, an in-car spoken dialogue system for the MP3 domain was created and integrated into a BMW 3-Series Coupe (Becker et al., 2007). For the speech understanding component, a German corpus named SAMMIE (SAarbrücken Multi-Modal Interface Experiment) was collected by Saarland University and DFKI[2] using a Wizard of Oz experiment.

A grammar in Nuance GSL format was written to specify well-formed sentences complying with the corpus data. The GSL formalism is a variant of BNF (context-free grammar), with Extended BNF additions such as disjunctions and Kleene closures. The grammar was structured according to syntactical motivations and interaction type coherence. To minimize overgeneration, nonterminals were instantiated with usual grammatical features. For instance, genitive definite forms of artist expressions were generated by the disjunction

```
NP_ARTIST_CASE_GEN[
 (DET_NUM_SING_CASE_GEN_GEND_NEUT
  N_ARTIST_NUM_SING_CASE_GEN_GEND_MASC)
 (DET_NUM_SING_CASE_GEN_GEND_FEM
  N_ARTIST_NUM_SING_CASE_DATIV_GEND_FEM)]
```

For a more detailed description of the grammar, see (Becker et al., 2007).

The German Sammie grammar in Nuance format (NuanceGer) was checked and extended continuously while the dialogue system was built. User evaluation results were analyzed and missing utterances were added to the grammar. In addition to that, an English version of the grammar, called "NuanceEng" here, was built by a near-native speaker of English. This grammar is the starting point for our experiments. Figure 1 shows a graph of the grammar development for the first experiment, Figure 2 for the second experiment.

[2]German Research Center for Artificial Intelligence

2.3 Outline of the paper

Section 3 gives an introduction to GF and its resource grammar library, by working through the implementation of a fragment of the Sammie grammar. Section 4 describes the first experiment, in which a baseline Sammie grammar was ported to six languages. Section 5 describes the second experiment, in which the German grammar was extended towards the coverage of the original grammar. Section 6 concludes with statistics on the experiments, related work, and some general lessons learnt.

3 Multilingual grammars in GF

GF (Grammatical Framework, (Ranta, 2004)) is a grammar formalism based on ideas from type theory and functional programming. Originally designed for written technical documents, GF focuses on language-independent semantic representations and their multilingual renderings. These features have proved useful in dialogue systems as well, and a support for dialogue applications is completed by translators from GF to various speech recognition formats, such as Nuance (Bringert, 2007).

A **grammar**, in the sense of GF, has an **abstract syntax** and a set of **concrete syntaxes**. The abstract syntax is a semantic description of an application domain. Each concrete syntax is a mapping of the semantics into a language, typically a natural language. To give an example from the GF implementation of the Sammie grammar, the abstract syntax has objects such as

```
identify ( currently_playing_object )
```

The six concrete syntaxes map the abstract object into the strings

```
vad heter den här sången
wie heißt dieses lied
comment s'appelle cette chanson
como se llama esta canción
mikä on tämän laulun nimi
what is the name of this song
```

of Swedish, German, French, Spanish, Finnish, and English, respectively.

The abstract syntax is specified by a set of **categories** (`cat`) and **constructor functions** (`fun`), in the same way as an inductive family of datatypes in a functional programming language. Here is a fragment of the Sammie abstract syntax, with five categories and five constructor functions:

```
cat
  Action ; ToIdentify ; Object ;
  Playlist ; Artist ;
fun
  create : Action ;
  identify : ToIdentify -> Action ;
  play : Object -> Action ;
  remove : Playlist -> Object -> Action ;
  currently_playing_object : ToIdentify ;
```

The concrete syntax is specified by defining a **linearization type** (`lincat`) for each category, as well as a **linearization function** (`lin`) for each constructor. A baseline concrete syntax can be obtained by just assigning the type of strings to each category, and defining:

```
lincat
  Action, ToIdentify,
  Object, Playlist, Artist = Str ;
lin
  create = ["create a new playlist"] ;
  identify x = x ;
  play = "play" ++ x :
  remove x y = "remove"++ y ++"from"++ x ;
  currently_playing_object =
    ["what is the name of this song"] ;
```

A concrete syntax like this is essentially a system of templates with chunks of canned text. While it is easy to produce for small applications, it does not scale up well, especially in languages that have rich morphology and require agreement in syntactic structures. Thus GF also supports user-defined **parameter types**, which can be used to control inflection and word order in linearization. For instance, the German version of the above grammar needs a type of Case, and the linearization of `Object` and `Playlist` depends on case:

```
lincat
  Object, Playlist = Case => Str ;
lin
  remove x y = "nimm" ++ y ! Acc ++
    "aus" ++ x ! Dat ++ "heraus"
```

3.1 The GF resource grammar library

Having to think about parameters requires linguistic knowledge from the grammar writer. Moreover, accurate descriptions tend to become long and complex. The GF solution to this problem is a **resource grammar library**. Like any software library, this library can be used via a high-level API (an abstract syntax for linguistic structures) that hides the implementation details (the concrete syntaxes for each language). The GF resource grammar library is currently available for 10–15 languages (10 languages support the full API, 5 just parts of it). Its first applications were in the domain of written technical language (Burke and Johannisson, 2005, Caprotti et al., 2006), but its use was extended to spoken dialogue systems in the TALK project (Johansson 2006, Ljunglöf & al. 2006).

Let us rewrite the Sammie grammar fragment by using the library,

```
lincat
  Action     = Phr ; -- phrase
  ToIdentify = QS ;  -- question
  Object, Playlist,
    Artist   = NP ;  -- noun phrase
lin
  create = imperative (mkVP create_V2
    (indef (mkCN new_A playlist_N))) ;
  identify x = mkPhr x ;
  play x = imperative (mkVP play_V2 x) ;
  remove x y =
    imperative (mkVP remove_V3 y x);
  currently_playing_object =
    mkQS whatSg_IP (mkNP name_N2
      (mkNP this_Quant song_N)) ;
```

This grammar uses the language-independent resource grammar API with categories such as `Phr`, `QS`, `NP` and constructors such as `mkVP`, `indef`, `this_Quant`. The ones provided by the resource grammar are syntactic combination rules and structural words, which are independent of the domain of application.

In addition to the resource API elements, a concrete syntax also needs a lexicon of domain-specific words, such as `new_A`, `play_V2`, `remove_V3` above. The resource library provides for each language a set of operations for constructing lexical entries with all morphosyntactic information they need. Thus the three mentioned objects are defined as follows in English:

```
  new_A = regA "new" ;
  play_V2 = dirV2 (regV "play") ;
  remove_V3 = dirV3
    (regV "remove") from_Prep ;
```

Here are the German definitions:

```
new_A = regA "neu" ;
play_V2 = dirV2 (regV "spielen") ;
remove_V3 = dirV3
  (prefixV "heraus" nehmen_V) aus_Prep ;
```

The lexicon definitions are gathered into a separate **interface** module, which the concrete syntax module depends on. All that is needed to add a new language to the system is a new implementation of the interface module, with lexical entries belonging to that language.

3.2 Beyond baseline grammars

A baseline multilingual grammar system can be obtained by defining the syntax in a language-independent way using the resource API, and only letting the lexical entries vary from one language to another. Such a system is guaranteed to be grammatically correct, as regards to word order and agreement. But the different languages often come out unidiomatic. For instance, the above rule for `currently_playing_object` produces the translations

```
vad är namnet på den här sången
was ist der name von diesem lied
quel est le nom de cette chanson
mikä on tämän laulun nimi
what is the name of this song
```

These translations are OK for Finnish and English, but very clumsy for the rest of the languages, which have special verbs for expressing the name of a subject (the proper forms were shown above; the closest corresponding English idiom is *what is this song called*).

Fortunately, GF is a functional programming language that permits functions, instead of just words, to appear in an interface. An improved way to implement the rule above is

```
lin currently_playing_object =
  mkQS (what_name
    (mkNP this_Quant song_N))
```

where the function `what_name` has different implementations in different languages: here, for instance, German and English:

```
what_name x =
  mkQCl how_IAdv (pred heißen_V x)
what_name x =
  mkQCl whatSg_IP (mkNP (regN2 "name") x)
```

A similar refinement is needed in the GF Sammie grammar to express imperatives. A baseline, language-independent definition would be

```
imperative vp = UttImpSg vp
```

which produces the second-person singular imperative form of a verb phrase. In German, as shown by the corpus collected for Sammie, both the familiar singular and the polite imperative are appropriate, and should be accepted in user input. GF has the `variants` construct to express such free variation:

```
imperative vp = variants {
  UttImpSg vp ;
  UttImpPol vp
  }
```

When extending the different languages of the Sammie grammar in GF, above the baseline, adding variants was the prominent method used.

3.3 Using GF in dialogue systems

In the TALK project, GF was used for building various components of dialogue systems at three different sites. The most relevant features of GF in this work were the following:

- a common abstract syntax guarantees that the same semantics is implemented for all languages
- the resource grammar library makes it easier to port systems to new languages
- the GF grammar compiler supports the production of many other formats from the GF source

The first two features have been covered in the preceding sections. The third feature, the grammar compiler, is what in practice can integrate GF in the work flow of different projects. Language models for speech recognition are the most crucial formats in dialogue systems. GF supports several such formats, including the GSL format used in the Nuance system, which in turn is used in the Sammie dialogue system. Porting the Sammie grammar to new languages with GF would thus automatically produce the required speech recognition grammars.

4 The first experiment

The starting point of the work was Nuance-Sammie, a pair of hand-written Nuance grammars used in the Sammie system, one for English (NuanceEng) and one for German (NuanceGer). The goal was to produce GF-Sammie, a GF grammar with the same coverage as Nuance-Sammie, but for more languages.

This was to be produced by using the resource grammar library, and share as much code as possible between the languages.

The experiment was aimed to test the hypotheses that a grammar for basic communication is easy to produce using the library; adding a new language should be a matter of a few hours.

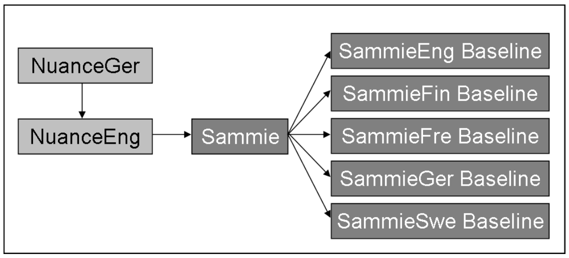

Figure 1: First experiment: The baseline grammar development. The modules on the left are handwritten Nuance grammars used in the Sammie system. The module in the middle is a GF abstract syntax defining the semantics implicit in the Nuance grammars. The modules on the right are GF concrete syntaxes implementing the semantics in a minimal but complete way.

4.1 The phases of the work

Before the baseline grammar, an abstract syntax must of course be produced. It was written by Björn Bringert on the basis of NuanceEng, which was richly commented with information indicating what actions should be covered by the grammar. The abstract syntax was produced in five hours, which includes the work needed to write a string-based English concrete syntax to test the abstract syntax.

To prepare for a multilingual localization, the string-based English concrete syntax was first **globalized** by rewriting it in terms of the recource grammar API and moving lexical items and some other obviously English-dependent constructs to an interface. This work took two hours.

After the globalization, the grammar was localized by writing new instances of the interface. This was done for Swedish, Finnish, French, and German. The work took half an hour for each language.

Did we now have a satisfactory baseline grammar for five languages? This was tested by generating sentences in all languages, and led to some fine-tuning to get satisfactory (grammatical and idiomatic) results. But now we did have a grammar that permitted user input in five languages, with the same semantics as NuanceEng, but with more limited variation in expressions. Spanish was added later to the system. Summary of the time consumption for this work is as follows:

- abstract syntax and string-based English: 5h
- globalized English by use of resource API: 2h
- five new languages: 5h

A baseline grammar, as we have defined it, covers the abstract syntax with a minimal, grammatically correct and stylistically acceptable concrete syntax. Such a grammar can be used for communication by users who are willing to learn to speak in a certain way. Notice that this can still be richer than a Command&Control system, because the dialogue manager is based on the language-independent abstract syntax and works quite as well with a minimal concrete syntax.

The next phase was to grow the coverage of one of the baseline grammars, SammieGer Baseline, to match the corpus defined by NuanceGer. This work was expected to take a few days, as carried out by a non-linguist programmer who first had to learn GF.

5 The second experiment

As expected, the SammieGer Baseline grammar covered less user utterances than the NuanceGer grammar. The purpose of our experiment was to find out how much time and effort a GF-novice grammar engineer needed to extend the SammieGer Baseline grammar to match the coverage of the NuanceGer grammar. The top level grammars involved can be seen in Figure 2.

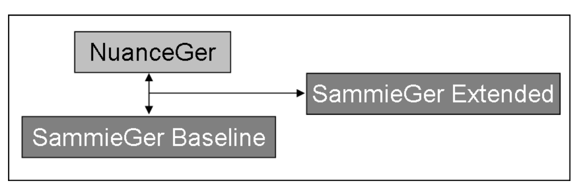

Figure 2: Second experiment: The SammieGer Baseline was extended to SammieGer Extended, to match the coverage of the original NuanceGer.

5.1 Experimental plan

For the extension of the SammieGer Baseline grammar, we were in the fortunate position of already having a grammar at hand that defined the terminal symbols and the grammar rules which the SammieGer Extended grammar would have to include. We planned the extension experiment in the following way: Comparing the coverage of SammieGer with the original NuanceGer grammar by generating sentences from the Nuance grammar and checking if they are covered by the GF grammar. If a generated sentence is grammatically correct but contains words that are missing in the lexicon, the GF lexicon has to be extended. If the syntactic structure is not covered, the concrete syntax has to be extended, and if the semantic structure of the sentence is missing in the abstract grammar, it has to be added.

5.2 Adding words to the lexicon

Before generating sentences from the NuanceGer grammar, we started with a simple word count. The NuanceGer grammar contained 463 single words, counting all inflected forms of the same stem individually. The SammieGer Baseline grammar contained 100 words, so it was clear that our first action had to be the extension of the SammieGer lexicon. Wherever this was possible using the **variants** construct (cf. Section 3.2), i.e. when adding a word that is a synonym of a word which was already modeled in the SammieGer grammar, this was most comfortable. 46 words could be added in this fashion, this time counting morphological infinitive forms that added more than one inflected form to the grammar. In fact, the 46 infinitive forms extended the word count to 215, so that the adding of 46 infinitives extended the grammar by 115 inflected word forms.

Some of these words had to be added because the starting point for the SammieGer Baseline grammar was in fact an English (NuanceEng) grammar. When translating from German to English, some words got lost, for instance, the words "Sänger" and "Sängerin" united to the word "singer" in English, as there is no gender distinction in English. The word "Sängerin" is missing in the SammieGer Baseline grammar, as "Sänger" only becomes translated to "singer".

Another source of words are verbs with their respective removable prefixes. German is rich in prefixes that can be combined with verbs to gain new meanings, for instance "an-gehen", "auf-gehen", "aus-gehen" [...], which are all related verbs sharing word stem and inflection paradigms, but each mean something else. These prefixes can be severed from the verb in certain utterances, and fortunately, GF accounts for that. By extending play_V (cmp. above) to:

```
play_V2 = variants {
  dirV2 (regV "spielen") ;
  dirV2 (prefixV "ab" (regV "spielen"))
  } ;
```

the extended grammar is able to parse an utterance like "spiele einen Titel von U2 ab" ("play a title by U2"), as well as an utterance without the "ab" in the end. The linearization rules in GF place the severed prefix in the syntactically correct position.

There were also words missing from the SammieGer Baseline grammar that could not be included with a simple variants construct. They were added to the lexicon under new identifiers and integrated into the concrete grammar by writing new linearization rules. In order to accomodate some of the missing words, new abstract syntax rules had to be defined.

5.3 Adding rules to the concrete grammar

One example of additions to the concrete syntax are the rules for interrogative and infinitive forms. Utterances follow certain patterns which are also reflected in the NuanceGer grammar (see Table 1 for an overview). In the Baseline SammieGer, only the imperative construct was modeled. The detour we took in localizing the system over English accounts for one missing utterance type: the infinitive and the imperative type are identical in English, but not in German. The interrogative forms are phrased like questions, but contain an implicit but politely expressed imperative. We managed to include the other utterance types by adding four rules to the concrete SammieGer grammar and renaming rule identifiers in one referenced library grammar.

5.4 Adding rules to the abstract grammar

Some user intentions modeled in the NuanceGer grammar were missing in the abstract SammieGer Baseline grammar, for instance scrolling a list presented on the screen up or down. These additions

Table 1: *Utterances Types.* The types of user utterances for German and English. Note that the imperative and the infitive forms in are the same in English, but not in German.

Type	German Example	English Example
Imperative	Spiele Vertigo von U2.	Play Vertigo by U2.
Interrogative	Kannst du Vertigo von U2 spielen?	Can you play Vertigo by U2?
Indicative	Ich möchte Vertigo von U2 hören.	I want to listen to Vertigo by U2.
Infinitive	Vertigo von U2 spielen.	Play Vertigo by U2.

took one day to accomplish. Summary of the time needed for the grammar extension is as follows:

- Installing and learning GF: 4 days
- Adding words: 3 days
- Adding concrete syntax rules: 3 days
- Adding abstract syntax rules: 1 day

6 Results

In this section, we compare the SammieGer Baseline/Extended and the NuanceGer grammar.

The goal set for the first experiment to build prototypical grammars for six languages was fulfilled quite successfully. However, the aim of the second experiment to match the coverage of the NuanceGer grammar with the SammieGer Extended grammar was not reached as quickly as we had hoped. It took a substantial time for the programmer to learn GF well, and the the development cycle was slowed down by fairly long compilation times. The resource library was difficult to navigate and contained some bugs that were fixed during the experiment, which caused waiting time. Nevertheless, the SammieGer Extended grammar's coverage increased considerably compared to SammieGer Baseline. Moreover, most of the extensions made to the German grammar can be ported to the other languages with very little work, due to the common resource library API.

6.1 Statistics

The original German grammar NuanceGer was written in approximately 18 days. In the GF experiments, 12 hours were needed to create the six baseline grammars from the NuanceEng original, and about 7 days for the SammieGer Extended grammar (not counting the time needed for installation and learning to use GF). If we sum up the SammieGer Baseline and the SammieGer Extended grammar writing time, we end up with 8 days for the SammieGer combined. This is faster than the 18 days spent on the original NuanceGer grammar, but we had of course the advantage of already having NuanceGer available: its authors had to start from scratch and continuously add words and rules after user evaluations. Moreover, the full coverage of NuanceGer was not reached, mostly because of colloquial forms of speech that were not covered by the resource library. Statistics of the coverage of the three grammars (SammieGer Baseline, SammieGer Extended, and NuanceGer) can be seen in Table 2.

6.2 Related work

The idea of generating speech recognition grammars from higher-level formats was first implemented in the Regulus system (Rayner et al., 2006). The source format of Regulus is a unification-based grammar formalism, and the target is GSL (the format used in Nuance); GF supports many other formats as well, such as the SLF format used in HTK (Young et al., 2005); see (Bringert, 2007). Regulus also has a resource grammar library currently covering five languages.

GF was previously used for dialogue system localization in the TALK project, where seven languages were covered (Johansson, 2006, Ljunglöf et al., 2006).

6.3 Conclusion

GF provides elegant solutions for many grammar writing challenges. Based on the concept of one abstract and many concrete grammars for different languages, GF is well-suited for localization tasks and fast prototyping in multiple languages. One disadvantage of GF is that it is quite difficult to get a grasp

Table 2: Statistics of SammieGer Baseline, SammieGer Extended, and the original Nuance.

Grammar	Baseline	Extended	Original
top-level constructors	18	23	∼23
syntactic categories	17	17	419
German - specific source code	4kB	18kB	200kB
German + generic source code	14kB	33kB	200kB
Nuance code	18kB	31kB	200kB
distinct words	100	325	463

of the framework quickly, compared to the concept of a context free grammar format in BNF or EBNF form which is easier to understand, for computer scientists as well as for linguists. As GF is more of a programming language than a grammar format, it implements much more constructs than BNF, which also makes it more powerful. That power can be seen in the comparison of source code size between NuanceGer and SammieGer Extended in Table 2.

The elegance of the many resource files that hide the complexity leads to difficulties in error detection, as there is a tree of resource grammars referencing other grammars and to the novice programmer, it is not always transparent where an error occurred. This is of course a problem with all high-level programming languages using libraries. A more intuitive IDE and faster compilation times could improve the system's usability significantly.

Grammatically correct utterances can be modeled nicely in the GF resource grammar library, which also eliminated some of the grammatical errors present in the original hand-coded Nuance grammar. However, some spoken language oriented rules were not covered by the library, and were implemented by brute force by using strings in GF. In this experiment, the resource grammar was taken as it was (apart from bug fixes), and no new functions were added to it.

References

T. Becker, N. Blaylock, C. Gerstenberger, A. Korthauer, N. Perera, M. Pitz, P. Poller, J. Schehl, F. Steffens, R. Stegmann, and J. Steigner (Editor). 2007. *TALK Deliverable D5.3: In-Car Showcase Based on TALK Libraries.*

B. Bringert. 2007. *Speech Recognition Grammar Compilation in Grammatical Framework.* SPEECHGRAM 2007, Prague, 29 June 2007.

D. Burke and K. Johannisson. 2005. *Translating Formal Software Specifications to Natural Language / A Grammar-Based Approach.* P. Blache, E. Stabler, J. Busquets and R. Moot (eds). *Logical Aspects of Computational Linguistics (LACL 2005).* LNCS/LNAI 3407, pages 51–66.

O. Caprotti. 2007. *WebALT! Deliver Mathematics Everywhere.* Proceedings of SITE 2006. Orlando March 20-24.

E. Hagen, T. Said, and J. Eckert. 2004. *Spracheingabe im neuen BMW 6er.* ATZ.

R. Haller. 2003. *The Display and Control Concept iDrive - Quick Access to All Driving and Comfort Functions.* ATZ/MTZ Extra (The New BMW 5-Series), pages 51–53.

A. Ranta. 2004. *Grammatical Framework: A type-theoretical grammar formalism.* Journal of Functional Programming,14(2):145–189.

M. Johansson. 2006. *Globalization and Localization of a Dialogue System using a Resource Grammar.* Master's thesis, Göteborg University.

P. Ljunglöf, G. Amores, R. Cooper, D. Hjelm, O. Lemon, P. Manchón, G. Pérez, and A. Ranta. 2006. *Multimodal Grammar Library.* TALK Talk and Look: Tools for Ambient Linguistic Knowledge IST-507802 Deliverable 1.2b

M. Rayner, P. Bouillon, B. A. Hockey, and N. Chatzichrisafis. 2006. *REGULUS: A Generic Multilingual Open Source Platform for Grammar-Based Speech Applications.* In Proceedings of LREC, 24-26 May 2006, Genoa, Italy.

S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, G. Moore, J. Odell., D. Ollason, D. Povey, V. Valtchev, and P. Woodland. 2005. *The HTK Book (for HTK Version 3.3).* Cambridge University Engineering Department.