SPIN: A Semantic Parser for Spoken Dialog Systems

Ralf Engel

DFKI GmbH
Stuhlsatzenhausweg 3, Saarbrücken, Germany
ralf.engel@dfki.de

Abstract

This paper presents SPIN, a semantic parser developed for spoken dialog systems. The parser provides a powerful rule language for an easy and efficient creation of the rule set. Important features of the rule language include order-independent matching, built-in support for referring expressions, rule ordering, constraints and action functions. On the basis of an example utterance the advantages of the introduced features are shown. The increased processing complexity caused by the powerful rule language is handled by a new parsing approach that delivers sufficient performance for rule sets that are typical for dialog systems. We also show how the parser can be used for text generation. The paper closes with an evaluation of the parser performance showing that the approach is well suited for dialog systems.

SPIN: Pomenski parser za sisteme govorjenega dialoga

V članku je predstavljen SPIN, semantični razčlenjevalnik, ki je bil razvit za sisteme govorjenega dialoga. Razčlenjevalnik ima zmogljiv jezik za tvorjenje pravil, ki enostavno in učinkovito tvori nabor pravil. Pomembne značilnosti jezika za tvorjenje pravil so ujemanje ne glede na besedni red, vgrajena podpora referenčnim izrazom, razvrstitev pravil, omejitve in opravilne funkcije. Na podlagi primera izjave so prikazane prednosti vpeljanih lastnosti. Povečana kompleksnost procesiranja zaradi zmogljivega jezika za tvorjenje pravil obvladujemo z novim pristop k skladenjski analizi, ki ima zadosten učinek pri naboru pravil, značilnih za sisteme dialoga. Prikažemo tudi, kako je lahko parser uporabljen za tvorjenje besedila. Članek zaključimo z vrednotenjem delovanja parserja, ki pokaže, da je pristop primeren za sisteme dialoga.

1. Introduction

This paper presents SPIN, a semantic parser which is especially designed for spoken dialog systems. The parser operates directly on typed feature structures whereby the available types and features are taken from the system-wide ontology. A syntactic analysis of the input utterance is not performed, but the ontology instances are created directly from word level. The typical advantages of such an approach are that processing is faster and more robust against speech recognition errors and disfluencies produced by the user, and the rules are easier to write and maintain. Also, multilingual dialog systems are easier to realize as a syntactic analysis is not required for each supported language. A disadvantage is that the complexity of the possible utterances is somewhat limited, but this is acceptable for most dialog systems.

Most semantic parsers use as underlying formalisms context free grammars (CFGs), e.g., (Gavaldà, 2000) or finite state transducers (FSTs), e.g., (Potamianos and Kuo, 2000) or variants of them, e.g., (Ward, 1991; Kaiser et al., 1999). The SPIN parser uses a more powerful rule language to simplify writing of rules and to reduce the amount of required rules.

Properties of the rule language include:

- Direct handling of nested typed feature structure is available, which is important for processing more complex utterances.
- Order-independent matching is supported, i.e., the order of matched input elements is not important. This feature helps processing of utterances in free word order languages, like German, Turkish, Japanese, Russian or Hindi, and simplifies writing of rules that are robust against speech recognition errors and disfluencies produced by the user.
- Built-in support for referring expressions is available.
- Regular expressions are available. Formulating the rules in a more elegant way is supported by this feature whereby the amount of required rules is reduced. Furthermore, writing of robust rules is simplified.
- Constraints over variables and action functions are supported providing enough flexibility for real-world dialog system. Especially, if the ontology is developed without the parsing module in mind, flexibility is highly demanded.

SPIN's powerful rule language requires an optimizing parser, otherwise processing times would not be acceptable. Principally, the power of the rule language avoids the development of a parser which delivers sufficient performance for an arbitrary rule set. Therefore, the parser is tuned for rule sets that are typical for dialog systems. A key feature to achieve fast processing is pruning of results that can be regarded as irrelevant for further processing within the dialog system.

Currently, the parser is used in the SMARTWEB project[1] (Wahlster, 2004). Earlier versions were successfully used in the MIAMM project (Reithinger et al., 2005) and the SmartKom project (Reithinger et al., 2003). SMARTWEB is a multimodal dialog system whose purpose is to provide a mobile and unified access to semantic databases, web services and internet search. The semantic databases include a database containing information about current and

[1] http://www.smartweb-project.org

previous football World Cups, the web services include, among others, information about POIs (point of interests), weather forecast, route planning and traffic information. Supported languages are German and English (with a reduced functionality). SMARTWEB is a joint project of several industrial and academic partners mainly located in Germany. A client-server architecture is used whereby the clients are ordinary smartphones or special onboard-units built into motor-bikes or cars. The clients are connected to the server via UMTS or WLAN. The multimodal recognizers, the dialog system, and the access subsystems are located on the server. All modules communicate using either XML messages based on the EMMA standard[2] or RDF messages based on the system-wide used ontology SWIntO (SmartWeb Integrated Ontology)[3] (Cimiano et al., 2004). SWIntO combines the DOLCE ontology (Gangemi et al., 2002) and the SUMO ontology (Niles and Pease, 2001) and contains also domain specific classes, properties and instances.

The paper is structured in the following way: Section 2 presents the rule language, section 3 contains a processing example, section 4 describes the parsing approach, and section 5 discusses how the parser can also be used for text generation. Section 6 reports on an evaluation of the parsing performance.

2. Rule language

2.1. Working memory

The rules operate on a working memory (WM) which consists of typed feature structures. The allowed types and features are extracted from the system-wide ontology, e.g., SWIntO in the SMARTWEB system, plus an additional type `Word` for representing words.

The top types are automatically extended with internal features which are defined in a reserved namespace. A list of the available internal features is shown in table 1.

The WM is initially filled with instances of the type `Word` representing the recognized words. The type `Word` has the predefined features `orth` (for the orthography of the word), `stem` and `pos` (part of speech). The features are filled by a lexicon lookup. If a feature is not provided in the lexicon, it remains unspecified.

2.2. Rule format

Like in classic rewriting systems, each rule consists at least of a set of conditions matching elements in the WM and a set of actions replacing the matched elements. Furthermore, constraints over the content bound to variables and processing options can be specified.

2.2.1. Conditional part

The conditional part consists of one or more conditions. Default mode is order-independent matching, i.e., the order within the WM and the features `leftMargin` and `rightMargin` are ignored. Order-independent matching simplifies the writing of rules for free-word order languages and the writing of rules that are robust against speech recognition errors and disfluencies produced by the user. Order-dependent matching can be activated by using square brackets. In this case, the values of `leftMargin` and `rightMargin` are considered.

Feature	Description
leftMargin	the index of the leftmost word
rightMargin	the index of the rightmost word
words	contains the input words used to create this instance
syn	contains syntactic information, like gender, number and case
scoreClass	contains the class used for scoring
score	contains the score (used in combination with scoreClass)

Table 1: List of internal features which are added automatically to each top type.

A single condition checks if an instance within the WM is of a certain type and if the specified features are also set in the tested instance. The type test considers the type hierarchy specified in the system-wide ontology. The tested type of the instance in the WM may be a subtype of the type in the condition, but also a supertype. The latter supports processing of referring expressions. For example, a pronoun can be mapped to a general type representing objects, like `PhysicalObject` in the SMARTWEB project. If the matched instance is inserted again in the WM, the type is refined to the type of the condition. The refined type can support reference resolution if several candidates are available in the dialog history.

Substructures within the conditions can be associated with variables. The variables can be reused in the constraints and in the action part. Within the conditions, disjunctions and negations are possible.

A test on an instance representing a word can be abbreviated with the orthography of the word. This test is replaced internally with a test on the stem. This avoids that individual rules must consider inflectional variants, a special advantage for languages with a rich usage of inflections like German.

An example for a condition that tests on the country with the name Brazil and assigns the name to the variable *N* is

```
Country(name:$N=Brazil)
```

2.2.2. Constraints

Constraints enable additional tests on the content bound to variables. Some built-in constraints are already available, but it is also possible to add user defined constraints.[4] Built-in constraints include a constraint that checks if the content is exactly of the specified type, ignoring the type hierar-

[2] `http://www.w3.org/TR/EMMA`

[3] `http://www.smartweb-project.org/ontology_en.html`

[4] User defined constraints have to be written as Java classes which have to be specified in the configuration options of the parser.

chy (`!isTypeOf`)[5], a constraint that checks if the words responsible for the content satisfy the specified syntactic property (`!syn`) and a constraint that checks if the content contains a specified substructure (`!contains`).

An example for a constraint that checks if the content bound to the variable *$V* contains an instance of the type `Country` is

```
!contains($V,Country())
```

2.2.3. Action part

The action part specifies the elements which replace the matched elements in the WM. Possible elements in the action part are typed feature structures, variables and action functions. Action functions allow to post-process the content bound to variables.

Available built-in action functions include a function that acts differently in cases where a specified variable is bound or not (`@if`)[6], a function that insert a specified syntactic property (`@syn`) and functions that provide string operations (`@concat`, `@toUpperCase`).

An example for an action inserting an instance of the type `Country` with the feature `name` set to the value of the variable *$N* in upper case is

```
Country(name:@toUpperCase($V))
```

2.2.4. Processing options

Processing options include an option that the test is not performed only on top level, but also within embedded instances (∼`deepMatch`), an option that a rule is always applied optional (∼`opt`), and the possibility to specify an ordering label. The ordering label can be used to force that a rule is applied before or after other rules. This allows, e.g., to write clean-up rules that are performed when parsing is finished.

3. Processing example

In this section, we will demonstrate how the SPIN parser can be used to process the utterance *Wie spielte diese Mannschaft gegen Brasilien? (How did this team play against Brazil?)*.[7]

The country *Brasilien (Brazil)* is handled by the following rule[8]:

```
(R1) Brasilien
     → Country(name:BRAZIL)
```

The queried database is language independent and uses English identifiers in uppercase. Therefore, the country name has to be set to *BRAZIL*.

In our domain, a country name can stand for a national football team stemming from that country. A rule performs this transformation:

```
(R2) ∼opt $C=Country()
     → FootballNationalTeam(origin:$C)
```

As the country instance is used also in its original meaning, the rule is marked as optional (∼`opt`). Otherwise, the parser optimizations may cause that the solution without the rule being applied is not produced.

The word *Mannschaft (team)* is simply mapped to an empty instance of the type `Team`.

```
(R3) Mannschaft → Team()
```

The next rule handles the determiner *dieser (this)*.

```
(R4) [ dieser $O=PhysicalObject() ]
     → $O(lingInfo:RefProp(type:def,
            gender:@syn($O,gender),
            number:@syn($O,number)))
```

In this case, the order of the matched elements is relevant, so order-dependent matching is activated, indicated by the square brackets. This rule exploits the hierarchy of the system-wide used ontology as all objects that can be referred to inherit from the type `PhysicalObject`. In the SMARTWEB system, the reference resolution module uses gender and number as a criterion to find a suitable referent. The action function `@syn` examines the words that have been used to create the instances bound to the specified variables and computes the specified features `gender` and `number`. The corresponding entry in the lexicon is

```
Mannschaft,syn:female-singular
```

Although not required for processing of the example utterance, we present a rule processing *sie* (*it* in this case) to show how pronouns are processed.

```
(R5) sie
     → PhysicalObject(lingInfo:RefProp(
           type:det, gender:female,
           number:singular))
```

Questioned instances are marked in the SMARTWEB query language with a variable that contains the requested media type; it is also possible to asked explicitly for images or videos. The rule processing *welches (which)* is

```
(R6) [ welches $PO=PhysicalObject() ]
     → $PO(var:Variable(
           focus:Text()))
```

A corresponding rule for *wann (when)* is

```
(R7) wann
     → TimePoint(var:
         Variable(focus:Text()))
```

The verb phrase *wie spielte <Team1> gegen <Team2> (how did <Team1> play against <Team2>)* is handled by the rule

```
(R8) %wie spielte $T1=Team()
     %[ gegen $T2=Team() ]
     %[ %bei $T=Tournament() ]
     %$TP=TimePoint()
     %[ %in $R=TournamentRoundStage() ]
     → Match(team:T1, team:T2,
           tournament:$T,
           inRound:$R,
           @if($TP,happensAt:
              TimeInterval(begins:$TP)))
```

[5] All constraints are prefixed with `!`.

[6] Action functions are prefixed with `@`.

[7] As processing of free-word order phenomena should be shown, the example utterances are in German.

[8] The expression `(RX)` is not part of the rule and is only used for referring purposes.

This rule is able to integrate further information, like a specified tournament, a time point or a round stage like final. The additional information is matched by optional conditions, indicated by the prefix `%`.

Besides our example utterance, this single rule (together with other preprocessing rules for tournaments, rounds, etc.) can process a lot of other utterances including

Wie spielte Brasilien im Finale 1990?
(How did Brazil play in the 1990 final?)

Wie spielte Brasilien bei der WM in Spanien?
(How did Brazil play at the World Cup in Spain?)

Gegen welche Mannschaften spielte Brasilien bei der WM 1974?
(Which teams did Brazil play against at the World Cup 1974?)

Wann spielte Brasilien gegen Frankreich?
(When did Brazil play versus France?)

Covering such a variety of utterances with a single rule is only possible because the rule language supports optional conditions and mixing of order-dependent and order-independent matching.

In addition, order-independent matching makes processing more robust against speech recognition errors, as misrecognized words can be simply skipped in many cases. If the recognition errors affect only words which are not essential for the understanding of the utterance, the utterance can be analyzed at least partially. Examples are:

spielte diese Mannschaft gegen Frankreich?
(did this team play against France?)
(*Wie (How)* was not recognized)

Wie spielte Brasilien Frankfurt 1990?
(*How did Brazil play Frankfurt 1990?*)
(*im Finale (in the final)* was misrecognized as *Frankfurt*)

As the query module for the semantic database of the World Cup data expects that at least one instance is marked as questioned, a cleanup rule checks whether an embedded instance is marked as questioned and adds the dialog act *Question*. If this is not the case the matched instance itself is marked as questioned (second rule):

```
(R9) cleanup1:  $M=Match()
     !contains($M,Variable())
     → Question(content:$M())

(R10) cleanup2:  $M=Match()
      → Question(content:$M(var:
            Variable(focus:Text())))
```

In the configuration options of the parser, it is specified that rules marked with the ordering label `cleanup1` are applied before rules marked with the ordering label `cleanup2`.

A rule that handles general utterances like *bitte (please)* is

```
(R11) cleanup3:  bitte $D=DialogAct()
      → $D(mode:polite)
```

Due to order-independent matching this works also if *bitte* is placed in the middle of the utterance, e.g.,

wie spielte bitte diese Mannschaft gegen Brasilien
(Please, how did this team play against Brazil?)

The generated result structure for the utterance *wie spielte diese Mannschaft gegen Brasilien* is finally

```
Question(content:Match(
     var:Variable(focus:Text()),
     team:Team(lingInfo:RefProp(
        gender:female, number:singular))
     team:Team(origin:Country(
        name:BRAZIL))))
```

4. Parsing algorithm

In this section, the main ideas of the parsing algorithm are presented, a more detailed description can be found in (Engel, 2005).

4.1. Parsing challenge

Fast CFG parsing approaches like *Earley parsing* or *Tomita's parsing approach* cannot be used because SPIN's rule language allows order-independent matching. (Huynh, 1983) has shown that parsing of rule languages that support order-independent matching is NP-complete.

Typically, parsing algorithms are optimized to avoid the generation of multiple identical (intermediate) results and intermediate results that cannot be further processed. The first issue can be addressed using a chart, the second one using top-down predictions.

But the main problem in parsing rule languages which support order-independent matching is that most of the generated WMs are irrelevant for further processing in other modules within the dialog system as they contain unprocessed elements. The basic idea of the presented approach is to avoid the generation of as many irrelevant results as possible. Two starting points have been discovered:

(1) For many rules it is not appropriate to be applied before some other rules, as in this case irrelevant results are generated. An example is the application of rule (R8) before rule (R4) is applied. The problem is that *dieses (this)* is not integrated and, even worse, the word cannot be integrated later on, as the instance `FootballNationalTeam` is embedded after the application of rule (R8) and therefore unreachable for rule (R4). To overcome this problem, the idea is to order the rules offline, so that rule (R4) is applied before rule (R8).

(2) In many cases, the original WM can be deleted after the application of a rule. In a standard bottom-up parser, the result of a rule application is always added to the already existing set of alternative WMs. This is necessary as otherwise relevant results may not be generated. But if alternative rules do not exist, maintaining the original WM is not necessary. So the idea is to detect offline which rules can match the same input and to maintain only the original WM in these cases. All other rules are marked as destructive, i.e., the original WM is deleted after the application of that rule. As the presented example rules are not ambiguous, all rules are marked as non-destructive with the exception of rule (R2) which is marked explicitly as optional.

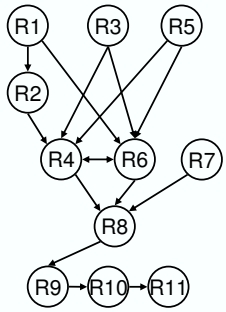

Figure 1: The generated dependency graph for the rules (R1) to (R11). One cycle exists containing rules (R4) and (R5). Transitive transitions are omitted.

4.2. Realization

First, the rules are ordered using a dependency graph. Therefore, each rule is compared with all other rules. If rule A creates instances that can be processed by rule B, a dependency transition is inserted between rule A and rule B. After all rules have been processed, the rules are linearized by walking through the graph and assigning each rule an application number.

Before the dependency graph is linearized, the graph is checked for cycles. A cycle means that a rule A can process the result generated by a rule B, but rule B can also process the result generated by rule A. If a cycle is detected, all rules of that cycle get the same application number. Rules with the same number are applied in a loop until none of the rules can be applied anymore.

After the rules are ordered, each rule is examined if it is in "competition" with at least one other rule applied afterwards. If this is the case, the rule is marked as non-destructive, i.e., the original WM is kept in the set of alternative WMs, otherwise the rule is marked as destructive, i.e., the original WM is deleted from the set of alternative WMs.

If a rule A is in competition with a rule B depends on the following: The application number of rule A has to be greater or equal than the application number of rule B, and a WM must exist so that rule A and rule B can be applied to that WM, and at least one element of the WM is matched by both rules. The algorithm to detect if two rules can match partially the same input is quite complex and is not described in this paper.

Figure 1 shows the constructed dependency graph for the rules used in section 3.

5. Text generation

When the development of the parser was started, using the parser as part of a text generation module was not intended. But the parser has proven as flexible enough to support this task. An early version of the text generation module called NipsGen[9] is already used in the SMARTWEB project. The SPIN parser is used in combination with a TAG (tree adjoining grammar) module (Becker, 2006). The TAG grammar of this module is derived from the XTAG grammar for English developed at the University of Pennsylvania[10].

The input of the generation module is the result of a query and is represented as an instance of SWIntO. The input is transformed to a text string in three steps:

1. A derivation tree for the TAG-grammar is created using SPIN rules which are applied on the semantic input structure.

2. The actual syntax tree is constructed using the derivation tree. After the tree has been built up, the features of the tree nodes are unified.

3. The correct inflections for all lexical leafs are looked up in a lexicon. Traversing the lexical leafs from left to right generates the text string.

The focus of the further description is put on the first step, the creation of the derivation tree.

A direct generation of the TAG tree description would lead to too complicated and unintuitive rules. Instead, the generation process is split into two phases. First, an intermediate representation is built up on a phrase level. This phase is domain dependent. In a second step, the intermediate description is transformed to a derivation tree. The intermediate layer is domain independent and therefore the transformation rules for the second part are also domain independent.

The generation of a text string is illustrated by the input structure

```
VP(o:Match(
   team1:FootballNationalTeam(origin:
      Country(name:GERMANY))
   team2:  FootballNationalTeam(origin:
      Country(name:BRAZIL))
   result:  "1:0" ))
```

which should be verbalized as

Deutschland spielte gegen Brasilien 1:0
(Germany played against Brazil 1:0)

Two examplary rules of the first phase are presented. The first rule produces the verb phrase (VP) with *spielen (play)*, the second one verbalizes the teams.

```
$VP=VP(o:Match(
      team1:$T1,team2:$T2,
      result:$R,not(lex:))
→ $VP(lex:spielen,
      sub:NP(o:$T1),
      pp:PP(lex:gegen,np:NP(o:$T2)),
      adv:AdvP(lex:$R))

$NP=NP(o:FootballNationalTeam(origin:
      Country(name:Brazil)),not(lex:))
→ $NP(lex:Brasilien)
```

In the second phase, the phrase structure is converted to a derivation tree for the TAG grammar. Each tree in the TAG grammar has a corresponding type in the ontology.

[9] Nips in the name NipsGen stands for the reverse usage of SPIN parser, Gen stands for generation.

[10] http://www.cis.upenn.edu/∼xtag/

The features of a TAG tree type represent the type of operation (adjunction (a), substitution (s), lexical replacement (l)) and the position in the tree, e.g., 211.

An example for a rule transforming a verb phrase to the intermediate representation to the TAG tree anCnx0VADJ is

```
VP(lex:$L,sub:$S,adv:$A,%pp:$PP,%fvp:$F)
→ anCnx0V(
      l_211:$L,
      s_1:$S(fvp:Fvp(case:nom)),
      a_221:$A,
      a_222:@if($PP,$PP(mode:vpAdj)),
      fvp:$F)
```

For text generation, the parser is driven in a slightly different mode: The automatic ordering of rules is switched off, instead the order in which the rules are applied is taken from the file containg the rules. Regions that have to be applied in a loop and rules that have to be applied optionally are marked explicitly. In the current system, two loops exists, one for each phase. In cases where multiple solutions should be produced, the alternative rules have to be marked as optional.[11]

Currently, the generation module contains 179 rules for the first phase and 38 rules for the second phase.

6. Evaluation of parsing performance

The rule set used for the SMARTWEB project consists of 1069 rules where 363 rules are created manually, and 706 are generated automatically from the linguistic information stored in SWIntO, e.g., country names. The lexicon contains 2250 entries.

In the offline rule ordering 12 loops are generated with an average size of 4.2 rules, the largest loop contains 20 rules. 242 rules are marked as non-destructive.

The parser is written in Java 1.5. We tested the performance on a Pentium IV 3.2GHz computer with a test corpus of 175 utterances with an average length of 6.5 words and a maximal length of 13 words. The average processing time was 45.9 ms, the largest one 183.4 ms.

7. Conclusion and outlook

In this paper we presented SPIN, a semantic parser providing a powerful rule language. After a short description of the rule language, a processing example was provided showing some of the advantages of the powerful rule language. A parsing approach which provides fast processing with rule sets that are typical for dialog systems was outlined, and the inclusion of the SPIN parser in the text generation module was presented. The evaluation of the parsing performance shows that the parser provides sufficient performance for real-world dialog systems.

The current research focus is on the development of tools for efficient rule writing and maintaining, and on further optimizations of the parser, like pruning of irrelevant results caused by optional conditions.

[11] A separate component selects one of the generated solutions.

8. Acknowledgments

This research was funded by the German Federal Ministry for Education and Research under grant number 01IMD01A. The views expressed are the responsibility of the authors. Points of view or opinions do not, therefore, necessarily represent official Ministry for Education and Research position or policy.

9. References

Becker, Tilman, 2006. Natural language generation with fully specified templates. In Wolfgang Wahlster (ed.), *SmartKom: Foundations of Multi-modal Dialogue Systems*. Heidelberg: Springer, pages 401–410.

Cimiano, Philipp, Andreas Eberhart, Pascal Hitzler, Daniel Oberle, Steffen Staab, and Rudi Studer, 2004. The smartweb foundational ontology. Technical report, Institute for Applied Informatics and Formal Description Methods (AIFB) University of Karlsruhe, Karlsruhe, Germany. SmartWeb Project.

Engel, Ralf, 2005. Robust and efficient semantic parsing of free word order languages in spoken dialogue systems. In *Proc. of Interspeech-2005*. Lisboa.

Gangemi, Aldo, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schcneider, 2002. Sweetening Ontologies with DOLCE. In *Proc. of EKAW02*, volume 2473 of Lecture Notes in Computer Science. Sigünza, Spain.

Gavaldà, Marsal, 2000. SOUP: A parser for real-world spontaneous speech. In *Proc. of 6th IWPT*. Trento, Italy.

Huynh, Dung T., 1983. Communicative grammars: The complexity of uniform word problems. *Information and Control*, 57(1):21–39.

Kaiser, Edward C., Michael Johnston, and Peter A. Heeman, 1999. PROFER: Predictive, robust finite-state parsing for spoken language. In *Proc. of ICASSP-99*, volume 2. Phoenix, Arizona.

Niles, Ian and Adam Pease, 2001. Towards a Standard Upper Ontology. In Chris Welty and Barry Smith (eds.), *Proc. of FOIS-2001*. Ogunquit, Maine.

Potamianos, Alexandros and Hong-Kwang Kuo, 2000. Statistical recursive finite state machine parsing for speech understanding. In *Proc. of 6th ICSLP*. Beijing, China.

Reithinger, Norbert, Jan Alexandersson, Tilman Becker, Anselm Blocher, Ralf Engel, Markus Löckelt, Jochen Müller, Norbert Pfleger, Peter Poller, Michael Streit, and Valentin Tschernomas, 2003. SmartKom - adaptive and flexible multimodal access to multiple applications. In *Proc. of ICMI 2003*. Vancouver, B.C.

Reithinger, Norbert, Dirk Fedeler, Ashwani Kumar, Christoph Lauer, Elsa Pecourt, and Laurent Romary, 2005. Miamm: A multi-modal dialogue system using haptics. In L. Dybkjaer and J. van Kuppevelt (eds.), *Natural, Intelligent and Effective Interaction in Multimodal Dialogue Systems*. Kluwer.

Wahlster, Wolfgang, 2004. SmartWeb: Mobile Applications of the Semantic Web. In Peter Dadam and Manfred Reichert (eds.), *GI Jahrestagung 2004*. Springer.

Ward, Wayne, 1991. Understanding spontaneous speech: the Phoenix system. In *Proc. of ICASSP-91*.