Eurospeech, pp. 1555-1558, ESCA, Aalborg, Denmark, 2001

ROBUST LANGUAGE UNDERSTANDING IN MIPAD

Ye-Yi Wang

Speech.Net Research Group
Microsoft Research
Redmond, Washington 98052, USA

Abstract

MiPad is an application prototype for the study of conversational, multi-modal interface in Microsoft Research. It has a *Tap and Talk* interface that allows users to effectively interact with a PDA device. The major Spoken Language Understanding (SLU) engine component behind MiPad is a robust chart parser. This paper discusses some novel features of the parser that enable it to take full advantage of the *Tap and Talk* interface and better support semantic based analysis. It also describes some implementation issues so that these new features can be accommodated without slowing down the parser. The new implementation speeds up the parser by a factor of three, making it more suitable for a SLU server.

1. Introduction

1.1. MiPad

MiPad (Multi-modal Interactive Pad) [1] is an application prototype of a research project in the Speech Technology Group of Microsoft Research and Microsoft Speech.Net Group. It offers a conversational, multi-modal interface to Personal Information Manager functionality, including calendar, contact list and e-mail. While our ultimate goal is an interaction model that spans across a number of different platforms and users, the initial target device is in the palmtop form factor. We chose this target device because it is very difficult to enter large amount of text, to fill a form, and to issue commands with multiple parameters with the current PDA devices. Multi-modal interaction with speech and pen can help address these problems, which can significantly improve the usability with the *Tap and Talk* interface. Figure 1 shows a screenshot of the Appointment card in MiPad. Seven fields appear on the card: Subject, Location, Attendees, Duration, Start Date, Start Time, and Body Message. A user interacts with it by tapping and holding on a field and speaking appropriate content into it. In the example in Figure 1, the user is tapping on the Start Time field and talking to the device. The bars in the field indicate that MiPad is carrying out speech recognition. Experienced users can also tap on the Command button and speak a full sentence that contains cross-field information, such as "Schedule a meeting with Kevin Larson and Derek Jacoby on Friday for two hours."

1.2. MiPad Spoken Language Understanding

While dialog management is an important topic in our research [2], it plays little role in the MiPad, for the *Tap and Talk* interface explicitly provides dialog state (tapped field) information. The major SLU component is a robust chart parser.

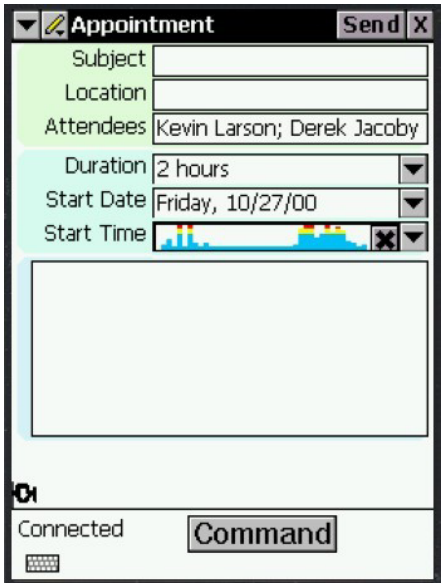

Figure 1. MiPad Screenshot. The user is tapping on the Start Time field and talking to the device.

The speech recognizer can use field-specific language models and the SLU component can employ field-specific grammars. This seemingly makes the SR and SLU task easier. However, on the other hand, in the typical MiPad usage scenario, users may use the built-in MiPad microphone that is very sensitive to environment noise. Using the iPaq device from Compaq as one of our prototypes, the word recognition error rate increased by a factor of two in comparison to a close-talk microphone (in the normal office environment), which shows that we need much better noise robust SR and SLU.

The MiPad SLU is modeled with domain-specific semantic grammars. Normally, semantic grammars are CFGs with non-terminals representing semantic concepts instead of syntactic categories, like the one used for ATIS in [3]. Our grammar introduces a specific type of non-terminals called semantic classes to describe the schema of an application [4, 5]. The semantic classes define the conceptual structures of the application that are independent to linguistic structures. The linguistic structures are modeled with context free grammars. In doing so, it makes the linguistic realization of semantic concepts transparent to an application; therefore the application logic can be implemented according to the semantic class structure, in parallel with the development of linguistic context free grammar. We introduced the framework of a robust spoken language parser in [5]. It analyzes input sentences according to the linguistic grammar and maps the linguistic structure to the semantic conceptual structure. Recently, we have made

substantial modifications to the parser to fully take advantage of the form factor of MiPad and better support the semantic based analysis. The changes range from some enhanced features to improved performance.

2. Robust Chart Parser

The robust parsing algorithm is an extension of the bottom-up chart-parsing algorithm [6]. The robustness to ungrammaticality and noise can be attributed to its ability of skipping minimum unparsable segments in the input. The algorithm uses dotted rules, which are standard BNF CFG rules plus a dot in front of a right-hand-side symbol. The dot separates the symbols that already have matched with the input words from the symbols that are yet to be matched. Each constituent constructed in the parsing process is associated with a dotted rule. If the dot appears at the end of a rule like in $\mathbf{A}\rightarrow\alpha\bullet$, we call it a complete parse with symbol **A**. If the dot appears in the middle of a rule like in $\mathbf{A}\rightarrow\mathbf{B}\bullet\mathbf{CD}$, we call it a partial parse (or hypothesis) for **A** that is expecting a complete parse with root symbol **C**.

The algorithm maintains two major data structures --- A chart holds hypotheses that are expecting a complete constituent parse to finish the application of the CFG rules associated with those hypotheses; an agenda holds the complete constituent parses that are yet to be used to expand the hypotheses in the chart.

Initially the agenda is empty. When the agenda is empty, the parser takes a word (from left to right) from the input and puts it into the agenda. It then takes a constituent $\mathbf{A}[i,j]$ from the agenda, where **A** is the root symbol of the constituent and $[i,j]$ specifies the span of the constituent. The order by which the constituents are taken out of the agenda was discussed in [5]. The parser then activates applicable rules and extends appropriate partial parses in the chart. A rule is applicable with respect to a symbol **A** if either **A** starts the rule or all symbols before **A** are marked optional. The activation of an applicable rule may result in multiple constituents that have the same root symbol (the left-hand-side of the rule) but different dot positions, reflecting the skip of different number of optional rule symbols after **A**. If the resulting constituent is a complete parse, namely with the dot positioned at the end of the rule, the complete constituent is added into the agenda. Otherwise partial constituents are added into the chart; To extend the partial parses with the complete parse $\mathbf{A}[i,j]$, the parser exams the chart for incomplete constituent with dotted rule $\mathbf{B}[l,k] \rightarrow \alpha\bullet\mathbf{A}\beta$ for $k < i$, and constructs new constituents $\mathbf{B}[l,j] \rightarrow \alpha\mathbf{A}\beta$ with various dot positions in β, as long as all the symbols between **A** and the new dot position are optional. The complete constituent $\mathbf{B}[l,j] \rightarrow \alpha\mathbf{A}\beta\bullet$ is added into the agenda. Other constituents are put into the chart. The parser continues the above procedure until the agenda is empty and there are no more words in the input sentence. By then it outputs top complete constituents according to the heuristic scores described in [5].

In [5], we distinguished three different types of rule symbols: optional symbols that can be skipped without penalty; normal symbols that can be skipped with penalty; and mandatory symbols that cannot be skipped. We found the skip of normal symbols is very problematic, mainly due to the following reasons:

1. Grammar authors are generally very forgetful to mark a symbol mandatory. On the other hand they are very good at labeling optional symbols. We believe that in their mindset the symbols they have written down are deemed to be mandatory unless it is marked as optional.
2. The skip of normal rule symbols adversely increases the constituent space. This dramatically slows down the parser and results in a great number of bogus ambiguities.

Therefore the current parser does not skip rule symbols unless the symbol is explicitly marked as optional.

3. New Features of the Parser

To take advantage of the MiPad form factor and better support semantic analysis, we have enhanced the parser with the following novel features:

3.1. Dynamic Grammar Support

Dynamic grammar support provides the parser with the capability of modifying the grammar it is using on the fly. It is necessary because:

1. Different users may have different data; therefore the parser should be able to customize the grammar online. For example, users may have different and changing contact list, therefore the parser should dynamically modify the rule for the contacts of different users.
2. Different dialog states need different grammar too. If MiPad is showing a New-Email card, then the PersonByName semantic class should only contain those names in the user's contact list, since they are the only names that the user can specify as recipients --- otherwise the user has to specify an e-mail address. On the other hand, if MiPad is showing a New-Contact card, then we should use a name set with much greater coverage, or even introduce a spelling grammar.

We have devised an API for application developers to dynamically and efficiently change the grammar used by the parser. The change can be made at different granularity, from the entire grammar to every single rule. This enables the parser to adapt to different users and dialog states.

3.2. Wildcard Parsing

In a semantic grammar, some semantic concepts are free form text and can hardly be modeled with semantic rules. For instance, meeting subjects can hardly be predicated and modeled with semantic grammar rules. To model this type of semantic units, the parser is augmented to handle rules with wildcards like the following one:

```
<Meeting-Property>::=<about> <Subject:Wildcard>
<about> ::= about | regarding | to discuss
```

Here "<Subject:Wildcard>" represents the non-terminal semantic class "<Subject>" that can match free-form text. "<about>" serves as a context cue that triggers the wildcard rule for "<Subject>". Anything that does not match other appropriate rules in the context and that matches the wildcard

rules with appropriate trigger will be parsed as a wildcard semantic component in the appropriate context.

3.3. Parsing with Focus

The parser can effectively take advantage of the context/dialog state information to reduce parsing ambiguities and the search space, hence improve its performance. For example, if the parser knows that the system is in the middle of a dialog with the user, talking about the attendee of a meeting, then the parser will only parse "John Doe" as Attendee, although it could be E-mail Recipient or new Contact according to the grammar. The parser can get the context information (focus) either from the dialog manager in the general Dr. Who architecture, or directly from the applications with a *Tap and Talk* interface. The focus is specified as a path from the root to the semantic class that the system is expecting an input for, like Root/ScheduleMeeting/MeetingProperty/StartTime).

The parser can override the focus mode in case the user volunteers more information in mixed initiative dialogs. In the case, if the system is expecting an input of the aforementioned focus and the user speaks "from 3 to 5pm', the parser will be smart enough to identify both Start-Time and End-Time, and return the semantic class that is the closest common ancestor of the identified semantic classes, in this case, Meeting-Property.

3.4. N-best Parsing

The parser can take n-best hypotheses from the speech recognizer, together with their scores, as input. The parser parses all the n-best hypotheses; ranks them with a heuristic score that combines the speech recognition score and the parsing score. The best parse will be forwarded to the application.

4. Implementation Issues

We found that the parser slowed down with the support of wildcard parsing, so we have redesigned the data structure and sped up the parser dramatically. Here we briefly describe some of the implementation issues.

4.1. Lexicon Representation

The lexicon contains both terminals and non-terminals. The non-terminals include names for semantic class, groups and productions [5]. Each entry in the lexicon is represented with an ID. The lexicon can map a terminal or non-terminal string to its ID. An ID is an integer with the least signification 3 bits devoted to the type of the ID (word, semantic classes, types, productions, groups etc.) The rest bits can be used as index for the direct access to the definition of the grammar components of the specific type. Each lexical entry points to a set Φ that contains the information of the IDs of the context free rules it can activate, as well as the position of the lexical item in these rules. Set Φ is used to locate the applicable rules after a complete is taken out of the agenda**.**

Each non-terminal entry **A** has a Boolean element, specifying if it can derive a wildcard as its left-most descendant in a parse, or more formally, if $\mathbf{A} \xrightarrow{*}$ wildcard α. The value of this element has to be pre-computed at grammar load time, and recomputed every time after dynamic grammar modification. The speed to set this Boolean value is hence very important. Fortunately we already have the information of rules that can be activated by a symbol in set Φ. With that information, we can define the relation $\Re = \{(x, b) \mid b \Rightarrow \alpha \mathrm{x} \beta\}$, where α is empty or a sequence of optional symbols. Then a non-terminal can derive a wildcard on its leftmost branch if and only if it is in the transitive closure of the wildcard with respect to relation $\Re$. This transitive closure can be computed in time linear to the number of non-terminal symbols.

4.2. Chart and Agenda

The chart consists of a dynamic array of n elements and a dynamic programming structure of $n*(n+1)/2$ (n is the length of an input sentence) cells that corresponds to the $n*(n+1)/2$ different span of constituents. Each array element corresponds to a position in the input sentence, and it contains a heap that maps from a symbol **A** to a list of partial parses. The partial parses cover the input sentence to the position that the element represents for, and they expect a complete parse of a constituent with root symbol **A**. With this the parser can quickly find the partial parses to extend when a complete parse with root **A** is popped from the agenda; Each cell of the dynamic programming structure contains a heap that maps a grammar symbol **A** to a pointer to the complete constituent tree with root **A** and the span that the cell represents for. This enables the parser to quickly find out if a new constituent has the same root name and span as an existing constituent. If so, the parser will safely prune the constituent with lower score.

The agenda is implemented as a priority queue. An element of the queue has a higher priority if it has a smaller span and higher heuristic score. This guarantees that the parser does not miss any parses with high heuristic score [5].

4.3. Wildcard Support

Since wildcard match is expensive, we would like to treat input words as wildcard only when it fits in the context. Therefore we added some top down guidance for the creation of a wildcard constituent. With the wildcard derivation information available for non-terminal symbols as described in 4.1, this can be implemented efficiently: during the parsing process, after a partial constituent with dotted rule $\mathbf{A} \rightarrow \alpha \bullet \mathbf{B} \beta$ is added to the chart, if **B** can derive a wildcard on its leftmost branch, we then set a flag that allows the next input word to be introduced as a wildcard.

After we introduce a wildcard to the parser, theoretically it can build m different wildcard constituents with different coverage, where m is the number of remaining words in input. This adversely increases the search space drastically, since each of these wildcard constituents can be combined with other constituents to form much more constituents. Instead of generating these m constituents, we assume that the wildcard only covers a single word. After the parser has built the parse for the complete sentence, it expands the wildcard coverage to all the skipped words adjacent to the word covered by a wildcard in the initial parse. The parser always prefers non-wildcard coverage to wildcard coverage. So wildcard will be used only when there is no no-wildcard parse of the input segment that fits the context.

5. Experimental Results

We have conducted some preliminary experiments to study the performance of the robust parser. The test data contains 624 sentences spoken to the Command button. Each sentence is manually annotated with its semantic card (top level semantic category) and slots like in the following example:

Sentence: *Schedule a meeting with Peter at 3 pm*
Card: Schedule-Meeting
Slots: Participant=*Peter*, Start-Time=*3pm*

The robust parser is used to parse the input sentences and the results are compared with the human annotated data. The card switching (topic ID) accuracy is obtained by comparing the parser-found semantic category with the human annotated one. The slot accuracy is obtained by computing the slot (labels plus contents) insertion-deletion-substitution errors that the parser has made.

Our semantic grammar contains 13453 rules; among them 6406 are lexical rules with the form $A \rightarrow a$. There are 908 non-terminals and 8546 terminals, and 14 top-level non-terminals representing the 14 major semantic categories (topics). The uni-gram perplexity of the topics is 8.4 and the average number of slots per topic is 4.9.

	Topic ID		Slot ID	
	Original	Enhanced	Original	Enhanced
Transcription	7.9	7.5	35.4	29.8
Recognition	12.0	10.6	51.6	43.7

Table 1. ***Topic ID/Slot ID Error Rate or the Original and the Enhanced Parsers.***

Table 1 compares the enhanced parser's topic ID and slot ID error rates with the original parser when it takes the transcribed data and speech recognizer's outputs as input. While the n-best parsing improves the topic ID performance with SR input, the wildcard parsing feature also helps topic ID, as shown with the error rate on the transcribed data. The wildcard parsing improves the Slot ID accuracy significantly simply because it covers more free form semantic slots like subject, street names, etc.

Many topic ID errors can be attributed to the intrinsically ambiguous sentences. Rudimentary error analysis (with about half of the erroneous parses) shows that many topic ID errors can be attributed to the intrinsically ambiguous sentences. Around 1/3 of the slot errors are side effect of the card-switching errors. Another 1/3 slot ID errors are directly related to the free-form text. The last 1/3 "errors" are due to some other reasons, such as transcription errors and uncovered proper names. Many errors in this category are actually not real errors. They are due to the inconsistent human annotations (e.g., time expressions were labeled with or without *o'clock, a.m.* or *p.m.*, etc.)

Speed-wise, the newly implemented parser on average spends 0.2 millisecond per word on a Pentium-III 600MHz machine, which is a speedup by a factor of three compared to the original parser that does not support wildcard parsing. The speedup may be even more significant for sentences spoken to the non-Command fields, since the focused parsing can prune more hypotheses.

6. Discussions and Summary

The applications of our robust parser are not limited to MiPad SLU. The same parser is used in the unified language modeling technique [7] that we are investigating for the same application. It takes advantage of both statistical language model such as n-gram and semantic context free grammar to alleviate the sparse data problem in a domain-specific application. We are also using the parser together with some name entity grammars and statistical learning algorithms to aid the authoring of semantic grammar.

While the parser is working satisfyingly, some improvements can still be made:

1. Our error analysis shows that slots with free-from text are a major source of slot ID errors. This is due to the aggressive natural of the wildcard-parsing algorithm: it puts everything that is not covered and fits the context as a wildcard match and identifies word sequences like "latest build at" as a wildcard e-mail subject. We are planning to integrate syntactic information from partial parsing [8] algorithms into our semantic framework, and only allow legal syntactic phrases to match wildcard semantic slots.
2. We are going to use the unified language model [7] to score parses. With the unified language model, we can score parses with probability instead of heuristics. This enables a tight integration of speech recognition and SLU in a statistical framework --- we can use the acoustic score from the speech recognizer and the language model score from the SLU to rank the parses of n-best input.

In this paper we have introduced the SLU component of MiPad, our first application prototype in the study of conversational, multi-modal interface. We have described the new features of the robust chart parser that can fully take advantage of the *Tap and Talk* interface of MiPad. We also discussed some of the implementation issues that speed up the parser for the use in a SLU server.

7. References

[1] X. Huang and et al., "MiPad: A Next Generation PDA Prototype," ICSLP, Beijing, China, 2000.

[2] K. Wang, "A Plan-based Dialog System with Probabilistic Inference," ICSLP, Beijing, China, 2000.

[3] W. Ward, "Understanding Spontaneous Speech: the Phoenix System," ICASSP, Toronto, Canada, 1991.

[4] B. Alabiso. A. Kronfeld, "LEAP: Language Enabled Applications," the First Workshop on Human-machine Conversation, Bellagio, Italy, 1997.

[5] Y.-Y. Wang, "A Robust Parser For Spoken Language Understanding," Eurospeech, Budapest, Hungary, 1999.

[6] J. Allen, *Natural Language Understanding*: Benjamin-Cummings Publishing Company, Inc., 1995.

[7] Y.-Y. Wang, M. Mahajan, and X. Huang,, "A Unified Context-Free Grammar And N-Gram Model For Spoken Language Processing," ICASSP, Istanbul, Turkey, 2000.

[8] S. Abney, "Partial Parsing via Finite State Cascades.," *Natural Language Engineering*, vol. 2, pp. 337-344, 1996.