WIT: A Toolkit for Building Robust and Real-Time Spoken Dialogue Systems

Mikio Nakano*, Noboru Miyazaki, Norihito Yasuda, Akira Sugiyama, Jun-ichi Hirasawa, Kohji Dohsaka, Kiyoaki Aikawa
NTT Corporation
3-1 Morinosato-Wakamiya
Atsugi, Kanagawa 243-0198, Japan
E-mail: nakano@atom.brl.ntt.co.jp

Abstract

This paper describes WIT, a toolkit for building spoken dialogue systems. WIT features an incremental understanding mechanism that enables robust utterance understanding and real-time responses. WIT's ability to compile domain-dependent system specifications into internal knowledge sources makes building spoken dialogue systems much easier than it is from scratch.

1 Introduction

The recent great advances in speech and language technologies have made it possible to build fully implemented spoken dialogue systems (Aust et al., 1995; Allen et al., 1996; Zue et al., 2000; Walker et al., 2000). One of the next research goals is to make these systems task-portable, that is, to simplify the process of porting to another task domain.

To this end, several toolkits for building spoken dialogue systems have been developed (Barnett and Singh, 1997; Sasajima et al., 1999). One is the CSLU Toolkit (Sutton et al., 1998), which enables rapid prototyping of a spoken dialogue system that incorporates a finite-state dialogue model. It decreases the amount of the effort required in building a spoken dialogue system in a user-defined task domain. However, it limits system functions; it is not easy to employ the advanced language processing techniques developed in the realm of computational linguistics. Another is GALAXY-II (Seneff et al., 1998), which enables modules in a dialogue system to communicate with each other. It consists of the hub and several servers, such as the speech recognition server and the natural language server, and the hub communicates with these servers. Although it requires more specifications than finite-state-model-based toolkits, it places less limitations on system functions.

Our objective is to build robust and real-time spoken dialogue systems in different task domains. By robust we mean utterance understanding is robust enough to capture not only utterances including grammatical errors or self-repairs but also utterances that are not clearly segmented into sentences by pauses. Real time means the system can respond to the user in real time. The reason we focus on these features is that they are crucial to the usability of spoken dialogue systems as well as to the accuracy of understanding and appropriateness of the content of the system utterance. Robust understanding allows the user to speak to the system in an unrestricted way. Responding in real time is important because if a system response is delayed, the user might think that his/her utterance was not recognized by the system and make another utterance, making the dialogue disorderly. Systems having these features should have several modules that work in parallel, and each module needs some domain-dependent knowledge sources. Creating and maintaining these knowledge sources require much effort, thus a toolkit would be helpful. Previous toolkits, however, do not allow us to achieve these features, or do not provide mechanisms that achieve these features without requiring excessive efforts by the developers.

This paper presents WIT[1], which is a toolkit

*Mikio Nakano is currently a visiting scientist at MIT Laboratory for Computer Science.

[1]WIT is an acronym of *Workable spoken dialogue Inter-*

for building spoken dialogue systems that integrate speech recognition, language understanding and generation, and speech output. WIT features an incremental understanding method (Nakano et al., 1999b) that makes it possible to build a robust and real-time system. In addition, WIT compiles domain-dependent system specifications into internal knowledge sources so that building systems is easier. Although WIT requires more domain-dependent specifications than finite-state-model-based toolkits, WIT-based systems are capable of taking full advantage of language processing technology. WIT has been implemented and used to build several spoken dialogue systems.

In what follows, we overview WIT, explain its architecture, domain-dependent system specifications, and implementation, and then discuss its advantages and problems.

2 Overview

A WIT-based spoken dialogue system has four main modules: the *speech recognition module*, the *language understanding module*, the *language generation module*, and the *speech output module*. These modules exploit domain-dependent knowledge sources, which are automatically generated from the domain-dependent system specifications. The relationship among the modules, knowledge sources, and specifications are depicted in Figure 1.

WIT can also display and move a human-face-like animated agent, which is controlled by the speech output module, although this paper does not go into details because it focuses only on spoken dialogue. We also omit the GUI facilities provided by WIT.

3 Architecture of WIT-Based Spoken Dialogue Systems

Here we explain how the modules in WIT work by exploiting domain-dependent knowledge and how they interact with each other.

3.1 Speech Recognition

The speech recognition module is a phoneme-HMM-based speaker-independent continuous speech recognizer that incrementally outputs word hypotheses. As the recognition engine, either VoiceRex, developed by NTT (Noda et al., 1998), or HTK from Entropic Research can be used. Acoustic models for HTK is trained with the continuous speech database of the Acoustical Society of Japan (Kobayashi et al., 1992). This recognizer incrementally outputs word hypotheses as soon as they are found in the best-scored path in the forward search (Hirasawa et al., 1998) using the ISTAR (Incremental Structure Transmitter And Receiver) protocol, which conveys word graph information as well as word hypotheses. This incremental output allows the language understanding module to process recognition results before the speech interval ends, and thus real-time responses are possible. This module continuously runs and outputs recognition results when it detects a speech interval. This enables the language generation module to react immediately to user interruptions while the system is speaking.

The language model for speech recognition is a network (regular) grammar, and it allows each speech interval to be an arbitrary number of *phrases*. A phrase is a sequence of words, which is to be defined in a domain-dependent way. Sentences can be decomposed into a couple of phrases. The reason we use a repetition of phrases instead of a sentence grammar for the language model is that the speech recognition module of a robust spoken dialogue system sometimes has to recognize spontaneously spoken utterances, which include self-repairs and repetition. In Japanese, *bunsetsu* is appropriate for defining phrases. A bunsetsu consists of one content word and a number (possibly zero) of function words. In the meeting room reservation system we have developed, examples of defined phrases are bunsetsu to specify the room to be reserved and the time of the reservation and bunsetsu to express affirmation and negation.

When the speech recognition module finds a phrase boundary, it sends the category of the phrase to the language understanding module, and this information is used in the parsing process.

It is possible to hold multiple language models and use any one of them when recognizing a speech interval. The language models are

face Toolkit.

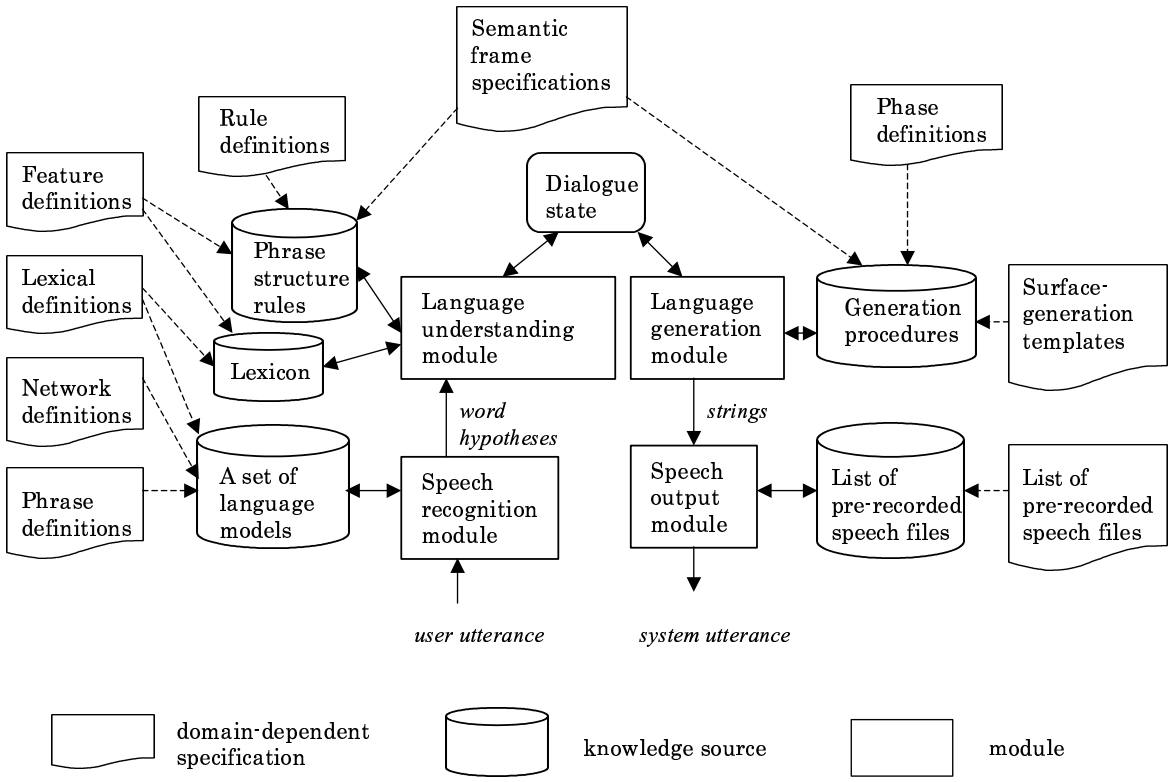

Figure 1: Architecture of WIT

switched according to the requests from the language understanding module. In this way, the speech recognition success rate is increased by using the context of the dialogue.

Although the current version of WIT does not exploit probabilistic language models, such models can be incorporated without changing the basic WIT architecture.

3.2 Language Understanding

The language understanding module receives word hypotheses from the speech recognition module and incrementally understands the sequence of the word hypotheses to update *the dialogue state*, in which the result of understanding and discourse information are represented by a frame (i.e., attribute-value pairs). The understanding module utilizes ISSS (Incremental Significant-utterance Sequence Search) (Nakano et al., 1999b), which is an integrated parsing and discourse processing method. ISSS enables the incremental understanding of user utterances that are not segmented into sentences prior to parsing by incrementally finding the most plausible sequence of sentences (or *significant utterances* in the ISSS terms) out of the possible sentence sequences for the input word sequence. ISSS also makes it possible for the language generation module to respond in real time because it can output a partial result of understanding at any point in time.

The domain-dependent knowledge used in this module consists of a unification-based lexicon and phrase structure rules. Disjunctive feature descriptions are also possible; WIT incorporates an efficient method for handling disjunctions (Nakano, 1991). When a phrase boundary is detected, the feature structure for a phrase is computed using some built-in rules from the feature structure rules for the words in the phrase. The phrase structure rules specify what kind of phrase sequences can be considered as sentences, and they also enable computing the semantic representation for found sentences. Two kinds of sentences can be considered; domain-related ones that express the user's intention about the reser-

vation and dialogue-related ones that express the user's attitude with respect to the progress of the dialogue, such as confirmation and denial. Considering the meeting room reservation system, examples of domain-related sentences are "I need to book Room 2 on Wednesday", "I need to book Room 2", and "Room 2" and dialogue-related ones are "yes", "no", and "Okay".

The semantic representation for a sentence is a command for updating the dialogue state. The dialogue state is represented by a list of attribute-value pairs. For example, attributes used in the meeting room reservation system include task-related attributes, such as the date and time of the reservation, as well as attributes that represent discourse-related information, such as confirmation and grounding.

3.3 Language Generation

How the language generation module works varies depending on whether the user or system has the *initiative* of turn taking in the dialogue[2]. Precisely speaking, the participant having the initiative is the one the system assumes has it in the dialogue.

The domain-dependent knowledge used by the language generation module is *generation procedures*, which consist of a set of *dialogue-phase definitions*. For each dialogue phase, an *initial function*, an *action function*, a *time-out function*, and a *language model* are assigned. In addition, phase definitions designate whether the user or the system has the initiative. In the phases in which the system has the initiative, only the initial function and the language model are assigned. The meeting room reservation system, for example, has three phases: the phase in which the user tells the system his/her request, the phase in which the system confirms it, and the phase in which the system tells the user the result of the database access. In the first two phases, the user holds the initiative, and in the last phase, the system holds the initiative.

Functions defined here decide what string should be spoken and send that string to the speech output module based on the current dialogue state. They can also shift the dialogue phase and change the holder of the initiative as well as change the dialogue state. When the dialogue phase shifts, the language model for speech recognition is changed to get better speech recognition performance. Typically, the language generation module is responsible for database access.

The language generation module works as follows. It first checks which dialogue participant has the initiative. If the initiative is held by the user, it waits until the user's speech interval ends or a duration of silence after the end of a system utterance is detected. The action function in the dialogue phase at that point in time is executed in the former case; the time-out function is executed in the latter case. Then it goes back to the initial stage. If the system holds the initiative, the module executes the initial function of the phase. In typical question-answer systems, the user has the initiative when asking questions and the system has it when answering.

Since the language generation module works in parallel with the language understanding module, utterance generation is possible even while the system is listening to user utterances and that utterance understanding is possible even while it is speaking (Nakano et al., 1999a). Thus the system can respond immediately after user pauses when the user has the initiative. When the system holds the initiative, it can immediately react to an interruption by the user because user utterances are understood in an incremental way (Dohsaka and Shimazu, 1997).

The time-out function is effective in moving the dialogue forward when the dialogue gets stuck for some reason. For example, the system may be able to repeat the same question with another expression and may also be able to ask the user a more specific question.

3.4 Speech Output

The speech output module produces speech according to the requests from the language generation module by using the correspondence table between strings and pre-recorded speech data. It also notifies the language generation module that speech output has finished so that the language generation module can take into account the timing of the end of system utterance. The meeting room reservation system uses speech files of short

[2]The notion of the initiative in this paper is different from that of the dialogue initiative of Chu-Carroll (2000).

phrases.

4 Building Spoken Dialogue Systems with WIT

4.1 Domain-Dependent System Specifications

Spoken dialogue systems can be built with WIT by preparing several domain-dependent specifications. Below we explain the specifications.

Feature Definitions: Feature definitions specify the set of features used in the grammar for language understanding. They also specify whether each feature is a *head feature* or a *foot feature* (Pollard and Sag, 1994). This information is used when constructing feature structures for phrases in a built-in process.

The following is an example of a feature definition. Here we use examples from the specification of the meeting room reservation system.

```
(case head)
```

It means that the case feature is used and it is a head feature[3].

Lexical Descriptions: Lexical descriptions specify both pronunciations and grammatical features for words. Below is an example lexical item for the word *1-gatsu* (January).

```
(1-gatsu ichigatsu month nil 1)
```

The first three elements are the identifier, the pronunciation, and the grammatical category of the word. The remaining two elements are the case and semantic feature values.

Phrase Definitions: Phrase definitions specify what kind of word sequence can be recognized as a phrase. Each definition is a pair comprising a *phrase category name* and a *network of word categories.* In the example below, `month-phrase` is the phrase category name and the remaining part is the network of word categories. `opt` means an option and `or` means a disjunction. For instance, a word sequence that consists of a word in the `month` category, such as *1-gatsu* (January), and a word in the `admoninal-particle` category, such as *no* (of), forms a phrase in the `month-phrase` category.

[3]In this section, we use examples of different description from the actual ones for simplicity. Actual specifications are written in part in Japanese.

```
(month-phrase
 (month
  (opt
   (or
     expression-following-subject
     (admoninal-particle
      (opt
       sentence-final-particle))))))
```

Network Definitions: Network definitions specify what kind of phrases can be included in each language model. Each definition is a pair comprising a network name and a set of phrase category names.

Semantic-Frame Specifications: The result of understanding and dialogue history can be stored in the *dialogue state*, which is represented by a flat frame structure, i.e., a set of attribute-value pairs. Semantic-frame specifications define the attributes used in the frame. The meeting room reservation system uses task-related attributes. Two are `start` and `end`, which represent the user's intention about the start and end times of the reservation for some meeting room. It also has attributes that represent discourse information. One is `confirmed`, whose value indicates whether if the system has already made an utterance to confirm the content of the task-related attributes.

Rule Definitions: Each rule has one of the following two forms.

```
( ⟨rule name⟩
  ⟨child feature structure⟩
      ...  ⟨child feature structure⟩
  => ⟨mother feature structure⟩
      ⟨priority increase⟩ )
( ⟨rule name⟩
  ⟨child feature structure⟩
      ...  ⟨child feature structure⟩
  => ⟨frame operation command⟩
      ⟨priority increase⟩ )
```

These rules are similar to DCG (Pereira and Warren, 1980) rules; they can include logical variables and these variables can be bound when these rules are applied. It is possible to add to the rules constraints that stipulate relationships that must hold among variables (Nakano, 1991), but we do not explain these constraints in detail in this

paper. The priorities are used for disambiguating interpretation in the incremental understanding method (Nakano et al., 1999b).

When the command on the right-hand side of the arrow is a frame operation command, phrases to which this rule can be applied can be considered a sentence, and the sentence's semantic representation is the command for updating the dialogue state. The command is one of the following:

- A command to set the value of an attribute of the frame,
- A command to increase the priority,
- Conditional commands (*If-then-else* type command, the condition being whether the value of an attribute of the frame is or is not equal to a specified value, or a conjunction or disjunction of the above condition), or
- A list of commands to be sequentially executed.

Thanks to conditional commands, it is possible to represent the semantics of sentences context-dependently.

The following rule is an example.

```
(start-end-times-command
 (time-phrase :from *start)
 (time-phrase (:or :to nil) *end)
 => (command (set :start *start)
             (set :end *end)))
```

The name of this rule is `start-end-times-command`. The second and third elements are child feature structures. In these elements, `time-phrase` is a phrase category, `:from` and `(:or :to nil)` are case feature values, and `*start` and `*end` are semantic feature values. Here `:or` means a disjunction, and symbols starting with an asterisk are variables. The right-hand side of the arrow is a command to update the frame. The second element of the command, `(set :start *start)`, changes the `:start` attribute value of the frame to the instance of `*start`, which should be bound when applying this rule to the child feature structures.

Phase Definitions: Each phase definition consists of a *phase name*, a *network name*, an *initiative holder specification*, an *initial function*, an *action function*, a *maximum silence duration*, and a *time-out function*. The network name is the identifier of the language model for the speech recognition. The maximum silence duration specifies how long the generation module should wait until the time-out function is invoked.

Below is an example of a phase definition. The first element `request` is the name of this phase, `"fmr_request"` is the name of the network, and `move-to-request-phase` and `request-phase-action` are the names of the initial and action functions. In this phase, the maximum silence duration is ten seconds and the name of the time-out function is `request-phase-timeout`.

```
(request "fmr_request"
         move-to-request-phase
         request-phase-action
         10.0
         request-phase-timeout)
```

For the definitions of these functions, WIT provides functions for accessing the dialogue state, sending a request to speak to the speech output module, generating strings to be spoken using surface generation templates, shifting the dialogue phase, taking and releasing the initiative, and so on. Functions are defined in terms of the Common Lisp program.

Surface-generation Templates: Surface-generation templates are used by the surface generation library function, which converts a list-structured semantic representation to a sequence of strings. Each string can be spoken, i.e., it is in the list of pre-recorded speech files.

For example, let us consider the conversion of the semantic representation `(date (date-expression 3 15))` to strings using the following template.

```
((date
  (date-expression *month *day))
 ((*month gatsu) (*day nichi)))
```

The surface generation library function matches the input semantic representation with the first element of the template and checks if a sequences

of strings appear in the speech file list. It returns `(''3gatsu15nichi'')` (March 15th) if the string "3gatsu15nichi" is in the list of pre-recorded speech files, and otherwise, returns `(''3gatsu'' ''15nichi'')` when these strings are in the list.

List of Pre-recorded Speech Files: The list of pre-recorded speech files should show the correspondence between strings and speech files to be played by the speech output module.

4.2 Compiling System Specifications

From the specifications explained above, domain-dependent knowledge sources are created as indicated by the dashed arrows in Figure 1. When creating the knowledge sources, WIT checks for several kinds of consistency. For example, the set of word categories appearing in the lexicon and the set of word categories appearing in phrase definitions are compared. This makes it easy to find errors in the domain specifications.

5 Implementation

WIT has been implemented in Common Lisp and C on UNIX, and we have built several experimental and demonstration dialogue systems using it, including a meeting room reservation system (Nakano et al., 1999b), a video-recording programming system, a schedule management system (Nakano et al., 1999a), and a weather information system (Dohsaka et al., 2000). The meeting room reservation system has vocabulary of about 140 words, around 40 phrase structure rules, nine attributes in the semantic frame, and around 100 speech files. A sample dialogue between this system and a naive user is shown in Figure 2. This system employs HTK as the speech recognition engine. The weather information system can answer the user's questions about weather forecasts in Japan. The vocabulary size is around 500, and the number of phrase structure rules is 31. The number of attributes in the semantic frame is 11, and the number of the files of the pre-recorded speech is about 13,000.

6 Discussion

As explained above, the architecture of WIT allows us to develop a system that can use utterances that are not clearly segmented into sentences by pauses and respond in real time. Below we discuss other advantages and remaining problems.

6.1 Descriptive Power

Whereas previous finite-state-model-based toolkits place many severe restrictions on domain descriptions, WIT has enough descriptive power to build a variety of dialogue systems. Although the dialogue state is represented by a simple attribute-value matrix, since there is no limitation on the number of attributes, it can hold more complicated information. For example, it is possible to represent a discourse stack whose depth is limited. Recording some dialogue history is also possible. Since the language understanding module utilizes unification, a wide variety of linguistic phenomena can be covered. For example, speech repairs, particle omission, and fillers can be dealt with in the framework of unification grammar (Nakano et al., 1994; Nakano and Shimazu, 1999). The language generation module features Common Lisp functions, so there is no limitation on the description. Some of the systems we have developed feature a generation method based on hierarchical planning (Dohsaka and Shimazu, 1997). It is also possible to build a simple finite-state-model-based dialogue system using WIT. States can be represented by dialogue phases in WIT.

6.2 Consistency

In an agglutinative language such as Japanese, there is no established definition of words, so dialogue system developers must define words. This sometimes causes a problem in that the definition of word, that is, the word boundaries, in the speech recognition module are different from that in the language understanding module. In WIT, however, since the common lexicon is used in both the speech recognition module and language understanding module, the consistency between them is maintained.

6.3 Avoiding Information Loss

In ordinary spoken language systems, the speech recognition module sends just a word hypothesis to the language processing module, which

speaker	start time (s)	end time (s)	utterance
system:	614.53	615.93	*donoyôna goyôken deshô ka* (how may I help you?)
user:	616.38	618.29	*kaigishitsu o yoyaku shitai ndesu ga* (I'd like to make a reservation for a meeting room)
system:	619.97	620.13	*hai* (uh-huh)
user:	622.65	624.08	*san-gatsu jûni-nichi* (on March 12th)
system:	625.68	625.91	*hai* (uh-huh)
user:	626.65	627.78	*jûyo-ji kara* (from 14:00)
system:	629.25	629.55	*hai* (uh-huh)
user:	629.91	631.67	*jûshichi-ji sanjup-pun made* (to 17:30)
system:	633.29	633.57	*hai* (uh-huh)
user:	634.95	636.00	*dai-kaigishitsu* (the large meeting room)
system:	637.50	645.43	*san-gatsu jûni-nichi, jûyo-ji kara, jûshichi-ji sanjup-pun made, dai-kaigishitsu toyû koto de yoroshî deshôka* (on March 12th, from 14:00 to 17:30, the large meeting room, is that right?)
user:	645.74	646.04	*hai* (yes)
system:	647.05	648.20	*kashikomarimashitâ* (all right)

Figure 2: An example dialogue of an example system

must disambiguate word meaning and find phrase boundaries by parsing. In contrast, the speech recognition module in WIT sends not only words but also word categories, phrase boundaries, and phrase categories. This leads to less expensive and better language understanding.

6.4 Problems and Limitations

Several problems remain with WIT. One of the most significant is that the system developer must write language generation functions. If the generation functions employ sophisticated dialogue strategies, the system can perform complicated dialogues that are not just question answering. WIT, however, does not provide task-independent facilities that make it easier to employ such dialogue strategies.

There have been several efforts aimed at developing a domain-independent method for generating responses from a frame representation of user requests (Bobrow et al., 1977; Chu-Carroll, 1999). Incorporating such techniques would decrease the system developer workload. However, there has been no work on domain-independent response generation for robust spoken dialogue systems that can deal with utterances that might include pauses in the middle of a sentence, which WIT handles well. Therefore incorporating those techniques remains as a future work.

Another limitation is that WIT cannot deal with multiple speech recognition candidates such as those in an N-best list. Extending WIT to deal with multiple recognition results would improve the performance of the whole system. The ISSS preference mechanism is expected to play a role in choosing the best recognition result.

7 Conclusion

This paper described WIT, a toolkit for building spoken dialogue systems. Although it requires more system specifications than previous finite-state-model-based toolkits, it enables one to easily construct real-time, robust spoken dialogue systems that incorporates advanced computational linguistics technologies.

Acknowledgements

The authors thank Drs. Ken'ichiro Ishii, Norihiro Hagita, and Takeshi Kawabata for their support of this research. Thanks also go to Tetsuya Kubota, Ryoko Kima, and the members of the Dialogue Understanding Research Group. We used the speech recognition engine VoiceRex developed by NTT Cyber Space Laboratories and thank those who helped us use it. Comments by

the anonymous reviewers were of great help.

References

James F. Allen, Bradford W. Miller, Eric K. Ringger, and Teresa Sikorski. 1996. A robust system for natural spoken dialogue. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL-96)*, pages 62–70.

Harald Aust, Martin Oerder, Frank Seide, and Volker Steinbiss. 1995. The Philips automatic train timetable information system. *Speech Communication*, 17:249–262.

James Barnett and Mona Singh. 1997. Designing a portable spoken language system. In Elisabeth Maier, Marion Mast, and Susann LuperFoy, editors, *Dialogue Processing in Spoken Language Systems*, pages 156–170. Springer-Verlag.

Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd. 1977. GUS, a frame driven dialog system. *Artificial Intelligence*, 8:155–173.

Jennifer Chu-Carroll. 1999. Form-based reasoning for mixed-initiative dialogue management in information-query systems. In *Proceedings of the Sixth European Conference on Speech Communication and Technology (Eurospeech-99)*, pages 1519–1522.

Junnifer Chu-Carroll. 2000. MIMIC: An adaptive mixed initiative spoken dialogue system for information queries. In *Proceedings of the 6th Conference on Applied Natural Language Processing (ANLP-00)*, pages 97–104.

Kohji Dohsaka and Akira Shimazu. 1997. System architecture for spoken utterance production in collaborative dialogue. In *Working Notes of IJCAI 1997 Workshop on Collaboration, Cooperation and Conflict in Dialogue Systems*.

Kohji Dohsaka, Norihito Yasuda, Noboru Miyazaki, Mikio Nakano, and Kiyoaki Aikawa. 2000. An efficient dialogue control method under system's limited knowledge. In *Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP-00)*.

Jun-ichi Hirasawa, Noboru Miyazaki, Mikio Nakano, and Takeshi Kawabata. 1998. Implementation of coordinative nodding behavior on spoken dialogue systems. In *Proceedings of the Fifth International Conference on Spoken Language Processing (ICSLP-98)*, pages 2347–2350.

Tetsunori Kobayashi, Shuichi Itahashi, Satoru Hayamizu, and Toshiyuki Takezawa. 1992. Asj continuous speech corpus for research. *The journal of the Acoustical Society of Japan*, 48(12):888–893.

Mikio Nakano and Akira Shimazu. 1999. Parsing utterances including self-repairs. In Yorick Wilks, editor, *Machine Conversations*, pages 99–112. Kluwer Academic Publishers.

Mikio Nakano, Akira Shimazu, and Kiyoshi Kogure. 1994. A grammar and a parser for spontaneous speech. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, pages 1014–1020.

Mikio Nakano, Kohji Dohsaka, Noboru Miyazaki, Jun ichi Hirasawa, Masafumi Tamoto, Masahito Kawamori, Akira Sugiyama, and Takeshi Kawabata. 1999a. Handling rich turn-taking in spoken dialogue systems. In *Proceedings of the Sixth European Conference on Speech Communication and Technology (Eurospeech-99)*, pages 1167–1170.

Mikio Nakano, Noboru Miyazaki, Jun-ichi Hirasawa, Kohji Dohsaka, and Takeshi Kawabata. 1999b. Understanding unsegmented user utterances in real-time spoken dialogue systems. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, pages 200–207.

Mikio Nakano. 1991. Constraint projection: An efficient treatment of disjunctive feature descriptions. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL-91)*, pages 307–314.

Yoshiaki Noda, Yoshikazu Yamaguchi, Tomokazu Yamada, Akihiro Imamura, Satoshi Takahashi, Tomoko Matsui, and Kiyoaki Aikawa. 1998. The development of speech recognition engine REX. In *Proceedings of the 1998 IEICE General Conference D-14-9*, page 220. (in Japanese).

Fernando C. N. Pereira and David H. D. Warren. 1980. Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278.

Carl J. Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. CSLI, Stanford.

Munehiko Sasajima, Yakehide Yano, and Yasuyuki Kono. 1999. EUROPA: A generic framework for developing spoken dialogue systems. In *Proceedings of the Sixth European Conference on Speech Communication and Technology (Eurospeech-99)*, pages 1163–1166.

Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. 1998. GALAXY-II: A reference architecture for conversational system development. In *Proceedings of*

the Fifth International Conference on Spoken Language Processing (ICSLP-98).

Stephen Sutton, Ronald A. Cole, Jacques de Villiers, Johan Schalkwyk, Pieter Vermeulen, Michael W. Macon, Yonghong Yan, Edward Kaiser, Brian Rundle, Khaldoun Shobaki, Paul Hosom, Alex Kain, Johan Wouters, Dominic W. Massaro, and Michael Cohen. 1998. Universal speech tools: The CSLU toolkit. In *Proceedings of the Fifth International Conference on Spoken Language Processing (ICSLP-98)*, pages 3221–3224.

Marilyn Walker, Irene Langkilde, Jerry Wright, Allen Gorin, and Diane Litman. 2000. Learning to predict problematic situations in a spoken dialogue system: Experiments with how may I help you? In *Proceedings of the First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-00)*, pages 210–217.

Victor Zue, Stephanie Seneff, James Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen, and Lee Hetherington. 2000. Jupiter: A telephone-based conversational interface for weather information. *IEEE Transactions on Speech and Audio Processing*, 8(1):85–96.