Rapid Prototyping for Spoken Dialogue Systems

Matthias Denecke
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, 15213
denecke@cs.cmu.edu

Abstract

We implemented a spoken dialogue system architecture for rapid prototyping. The features that support rapid prototyping include a clear separation of generic dialogue processing algorithms from domain and language specific knowledge sources. In an experiment, it could be shown that six individuals could specify these domain and language specific knowledge sources within 8 to 12 hours to come up with a prototypical implementation of a spoken dialogue system. To that end, no dialogue strategy had to be specified. Rather, it was sufficient to provide an ontology, a description of the services offered by the system, parsing grammars, database conversion rules and generation templates. Furthermore, the experiment shows that it is possible to formulate dialogue strategies in a domain and language independent manner; thus not requiring a system designer to be knowledgeable about dialogue processing.

1 Introduction

There are several approaches to reduce the effort to design and implement spoken dialogue systems. One approach to development environments consists of graphical editors for finite state automata (see for example (Cole, 1999)). Another approach to development environments emphasizes reusability of the domain model over graphical design interfaces. Here, fine tuning of dialogue strategies require separate fine-tuning in each module. In addition to the modules, dialogue flow is specified by a finite state automata whose nodes consist of modules. A third approach consists in designing a library of reusable dialogue strategies. Based on the observation that the behavior of a dialogue manager should be predictable in similar situations across several domains, (Araki et al., 1999) propose a library of dialogue strategies to be reused. (Kölzer, 1999) proposes a reusable dialogue system architecture based on specifications of knowledge sources for the different components.

In this paper, we describe the implementation of a generic spoken dialogue system that can be distinguished from the previous approaches by several features. We emphasize the separation of task and language dependent knowledge sources from generic dialogue processing algorithms. Each dialogue system can be specified by a set of knowledge sources, such as domain model, task model, grammar rules, and so on. The specifications are declarative rather than procedural, leaving to the dialogue manager the decision how to best interpret them in the context of the dialogue.

We conducted an experiment to evaluate the extent to which this approach generalizes to new domains and new languages. It could be shown that individuals could design a prototypical implementation of a spoken dialogue system within 8 to 12 hours simply by providing the domain and language specific knowledge sources. No alteration of dialogue strategies was necessary.

2 The Architecture

We give a brief overview of the system architecture. The architecture of the dialogue system Ariadne was designed specifically to support rapid prototyping. To this end, an architecture providing three levels of abstraction has been devised. The lowest level, called *Abstract Dialogue Engine*, provides a set of domain and language independent algorithms (e.g. to generate clarification questions) which make use of language and/or domain dependent knowledge sources. The intermediate layer, called *Interaction Pattern Layer*, provides a set of so-called interaction patterns which are domain and language independent. Interaction patterns are instantiated from the highest layer, called *Dia-*

logue Control Layer, to interact with the user and obtain representations that are to be added or removed from the discourse.

2.1 Abstract Dialogue Engine

2.1.1 Parsing

The output of the speech recognizer is parsed using the semantic parser SOUP (Gavalda and Waibel, 1998). The generated parse tree is converted into a typed feature structure by means of conversion rules with which the grammar rules are annotated (Denecke, 2000b). As the grammar rules and conversion specifications can be checked for well-typedness at compile-time, a non-welltyped or partially inconsistent feature structure generated at run-time indicates a possible misrecognition or skipping of the robust parser. In this case, the part causing the inconsistency is discarded from the representation (see below).

2.1.2 Discourse

The discourse history is represented in a tree structure where each node represents an utterance. The representation of an utterance consists of the text as provided by the recognizer, its semantic representation and the objects being referred to (if any). The formalism used to represent semantics and descriptions of objects is typed feature structures. Functionality provided by the ADE includes procedures to add nodes to the tree, and to maintain reference to the current node.

2.1.3 Dialogue Goal Descriptions

For each service offered by the back end application, there is a dialogue goal description that describes what kind of information is necessary to invoke that service. This is similar to a form in form-filling dialogue systems (Papineni et al., 1999). In addition, the dialogue goal descriptions also contain a list of references to services to be invoked once the goal is reached.

2.1.4 Semantic Representations

The ADE provides several algorithms to operate on typed feature structures. These include the selection of feature paths whose values disambiguate most efficiently on average a set of feature structures (for details see (Denecke and Waibel, 1997)), algorithms that determine which of the services offered by the system are compatible with what has been said (see section 3.1.4), and so forth.

2.1.5 Generation

Finally, the ADE provides the possibility to select and instantiate templates for natural language generation (see section 3.1.7).

2.2 Interaction Pattern Layer

2.2.1 Classification of Dialogue State

As noted in (Seneff and Polifroni, 2000), the explicit representation of dialogue state (indicating information on missing slot fillers) is cumbersome. In addition, it is impossible to formulate task independent dialogue strategies based on such a representation. For this reason, we chose to represent dialogue state by a set of features that describe the progress of the ongoing dialogue based on the information available in the discourse. Currently, we are using the following six features.

CURRENTQUALITY. This variable represents the quality of the representation of the current utterance. The value of this variable depends on the confidence measure of the speech recognizer, whether the semantic representation is consistent and whether the robust semantic parser skipped parts of the utterance.

OVERALLQUALITY. The value of this variable is a cumulation of the value of CURRENTQUALITY to detect deteriorating dialogue.

CURRENTSPEECHACT. This variable indicates the speech act of the current utterance, drawn from a speech act repertoire of 12 domain independent speech acts.

REFERENCE. This variable indicates if database access needs to take place to resolve reference of representations of noun phrases in the discourse. If so, the variable also indicates if enough information is available to actually perform that database request.

REFERRINGEXPRESSIONS. The value of this variable indicates if referring expressions could be resolved. If so, it indicates if the reference is unique or not. If reference (through database access) could not be resolved because no object in the database satisfied the constraints given by the user, the variable also indicates if a close match of similar objects is possible.

INTENTION. Finally, this variable indicates how many dialogue goal descriptions are compatible with the information in the discourse, and, if there is only one compatible if all necessary information to invoke the associated services has been acquired.

2.2.2 Interaction Patterns

Interaction patterns are sequences of utterances designed to obtain information to be added to or to be removed from the discourse. There are four types of interaction patterns.

1. The QUESTION interaction pattern seeks to obtain information from the user to be added to the discourse. An example is `"Would you like a,b or c?"`.
2. The UNDO interaction pattern causes to remove information from the discourse. This interaction pattern can be triggered through user utterances such as `"Undo"` or `"No, not a."`.
3. The CORRECTION interaction pattern both adds and removes information from the discourse. Examples are `"I said a not b"` (for a user initiated example) or `"I do not know a b but I do know a c or d b. Which one would you like?` for cooperative system initiated example.
4. The STATE interaction pattern does not add to nor remove information from the discourse but has influence on the dialogue flow. This interaction pattern can be triggered through user utterances such as `"I don't know"`, `"Repeat"` or `"Help"`.

Please note that interaction patterns can be instantiated in various shapes. The concrete shape of the interaction pattern is determined as the dialogue develops and depends on the abstract classification of dialogue state. For example, the dialogues `"Would you like a,b or c?" "a."` and `"Would you like a,b or c?" "a." "Please say again." "a." "Did you say a? Please say 'yes' or 'no'." "Yes."` are two instantiations of the same interaction pattern. It is the responsibility of an instantiation of an interaction pattern to guarantee that the information it is asked to obtain is reliable. If this is not possible, the instantiation of the interaction pattern may fail.

2.3 Dialogue Control Layer

The Dialogue Control Layer instantiates interaction patterns depending on the abstract classification of dialogue state. The functionality of the dialogue control layer is given by a constraint logic program. It is again domain and language independent. The dialogue control layer requests the instantiation of interaction patterns and is responsible for updating and maintaining dialogue state.

3 The Experiment

In order to demonstrate the ability of this architecture to support rapid prototyping, an experiment was designed in which the participants were to built their own dialogue system in a domain and a language of their choice.

It should be stressed that it was not the goal of the experiment to build stable robust dialogue systems. Rather, the dialogue systems should be used for data collection and as a basis for iterative improvement.

3.1 Execution of the Experiment

The design process was split into seven steps, described in more detail below. In each step, the participants were expected to generate specifications responsible for one aspect of the resulting dialogue system. The participants received a description of the goals of that step, in addition to documentation how to specify the desired properties. The participants were free to choose the domain of their application, and were free to choose between the target languages English and German. A specification of a working small-scale dialogue application was provided to complement the documentation.

3.1.1 Step 1: Back End Application

In this step, the participants are supposed to create a JAVA class that implements the functionality of their application. To do that, tools are provided that automatically generate a skeleton class implementation.

3.1.2 Step 2: Databases

In the second step, participants are expected to create databases that store descriptions of objects in the domain. The result of this step is an actual SQL database created with MS ACCESS. For example, the directory application contains a database with one table storing information on the employees.[1]

[1]Please note that while the given example is rather simple, the dialogue manager supports databases with multiple tables and joins. For more information, see (Denecke, 2000a).

```
desc obj_employee inherits object {
  string :   FirstName;
  string :   LastName;
  string :   Title;
  string :   DepartmentName;
  string :   EmailName;
  string :   HomePhone;
  string :   WorkPhone;
  string :   OfficeLocation;
};
```

Figure 1: Part of the ontology.

3.1.3 Step 3: Ontology

The goal of the third step is to provide an ontology containing all the concepts necessary to represent the semantics of utterances. To facilitate things, a set of concepts (such as $object, action, state$ and $property$) is provided in a generic ontology and can be reused. The ontology is internally represented as a type hierarchy for typed feature structures (Carpenter, 1992). Part of the ontology of the directory application is given in figure 1.

3.1.4 Step 4: Dialogue Goals

Dialogue goals establish a link between information in the discourse and the services that the back end application should invoke once the dialogue goal is reached. For each service implemented in step 1, the participants need to specify the amount of information that is necessary for the service to be invoked. The dialogue goal for the call forwarding service is given below. Please note that the service only requires the phone number to be present. Consequently, this is the only information that the feature structure of the dialogue goal requires to be present. The dialogue system, however, will ask for first or last names (as opposed to the phone number). This means there is no direct link between the information that is necessary to invoke the service and the clarification questions that are asked.

3.1.5 Step 5: Database Conversion Rules

Database conversion rules map the names of tables and fields given in the database (step 2) onto semantic representations of typed feature structures. In addition to the conversion information, it is possible to specify database guards. Database guards are lower bounds on information that need to be satisfied before a database takes place. This is to ensure that for example at least the first name or the last name of the person to be called is known before a database request takes place so as to avoid that the entire database is copied into the discourse representations. The conversion rules for the directory application are given in figure 3.

```
goal call {
  description:
    [ act_call
      EMPLOYEE [ obj_employee
                 WorkPhone [ string ]
               ]
    ]
  min:
    1
  max:
    1
  binding:
    callforward : [EMPLOYEE|WorkPhone] ;
};
```

Figure 2: The dialogue goal establishing the link between information in the discourse and invocation of the $callforward$ service

```
dbtable Employees obj_employee {
  dbfield First_Name      = [FirstName];
  dbfield Last_Name       = [LastName];
  dbfield Work_Phone      = [WorkPhone];
  dbfield Office_Location = [Location];
};
```

Figure 3: Database conversion rules for the directory application.

3.1.6 Step 6: Parsing Grammars

Semantic parsing grammars are used to convert the output of the speech recognizer into typed feature structures representing the meaning of the utterance. Grammar rules are separated in four categories: (i) lexical rules of the form $A \rightarrow w_1 \dots w_n$, (ii) lexical database rules of the form $A \rightarrow < databasename >< tablename >< fieldname >$ to automatically import strings from a database, (iii) structural rules of the form $A \rightarrow B_1 \dots B_n$, and (iv) derived rules where generic rules that do not contain domain specific semantic information can be reused in order to speed up grammar development (see (Denecke, 2000b) for more information on derived rules). Nonterminal symbols of the grammar consist of vectors of partially ordered symbols (for example $\langle obj\_person, N, sg \rangle$ represents a nonterminal symbol that would expand to constituents describing a person), where the first element is a concept of the ontology, and the second to last elements provide syntactic information. These nonterminal symbols can be considered as simple non-reentrant feature structures of depth one. Grammar rules are annotated with conversion information that allows to generate typed feature structures automatically from the rule specifications (see figure 4).

3.1.7 Step 7: Generation Templates

Generation templates establish a link between the dialogue state (recall that the dialogue state is given by the abstract classification of dialogue state, the discourse and the stack) and a speech act on one hand and a partially specified utterance and information on the expected answer of the user on the other. The constraints restrict the abstract dialogue state and the information in the discourse that allow a proper instantiation of the associated template. Figure 5 illustrates the use of constraints. There are three variables used in the templates. The variable `$sem` refers to the semantic representation of the last utterance, `$objs` refers to the representations of the objects retrieved from the database. `$db` refers to a database given by the name following the variable.

Both clarification questions shown in figure 5 seek to get the first name of a person. The first template can only be applied after a database request has taken place (otherwise `$objs` would be undefined), the first name of the retrieved employees is ambiguous and the last name is uniqe. The generated question then enumerates the known names using the `.first, .middle` and `.last` functors (returning the first, the second to previous to last, and last elements of the representations, respectively.) Possible options are passed on to the speech recognizer using the `options` keyword. In addition, "always active" commands are also allowed. The `location` keyword indicates semantic information on the expected answer, namely where to place it in the discourse and under which nonterminal symbol it should be parsed. If, on the other hand, the first name of the person to be called has not yet been mentioned by the user, then the system should prompt the user for it. The possible options for the speech recognizer are generated based on the filler of the field `FirstName` in the table `Employees` of the database `DirectoryDB`.

4 Results

4.1 Applications

There were altogether a set of six application implemented four of which use English and the remaining two use German. The applications implemented were the following: (i) a beer purchase system (English), (ii) a telephone directory service (English), (iii) a ticket purchase system (English), (iv) a roulette system (German), (v) a cd player (English) and (vi) a video rental system (German),

4.2 Temporal Effort

In each step, the participants were given the instructions detailing what to do in this step. They were allowed to ask questions at any time, and to consult the example specification at any time. The effort it took the participants to come up with the specifications are given in table 1.

4.3 Size of specifications

Table 2 shows the size of the specifications that were created in the experiment.

4.4 Other Observations

Almost all of the designed systems exhibit a lack of coverage in the natural language resource area. In other words, not all sentences the systems should understand could be parsed, and there was not an appropriate template for all of the dialogue states. The first problem is a direct consequence from the decision not to do any data collection ahead of time. To the contrary, the idea behind the presented approach is to generate a rudimentary system in a short time and to use this system for data collection rather than a wizard of oz setup. This is particularly beneficial in the light of the short development time of 8 to 12 hours and the fact that the data collection can then be done semiautomatically, not requiring the presence of a human wizard. The second problem is due to the fact that the template constraints do not cover the entire state space. Future work addresses the question how to detect lack of template coverage automatically by examining the template constraints and determining the subset of the state space that is covered.

The conducted experiment was limited to English and German languages; however, it should be possible in principle to extend the described approach to other languages as well. In order for the dialogue algorithms to work, the following requirements need to be fulfilled: (i) the meaning of an input needs to be able to be represented in the chosen formalism (typed feature structures), and (ii) two inputs with different syntactic structure but same meaning need to be represented in equivalent feature structures.

```
public <action:VP:_> = <act_lookup:V:_> <obj_employee:NP:_>* { EMPLOYEE obj_employee };
<act_lookup:V:_>  lmclass  = 'look' 'up' : 'find';
```

Figure 4: A structural and a lexical grammar rule with conversion rule specifications.

```
enumqst {
  state:(determined = lookup),
  path:(ambiguous   = $objs@[EMPLOYEE|FirstName]),
  path:(unique      = $objs@[EMPLOYEE|LastName])  ->
    text:     "There are $objs.num employees with the last name $sem@[EMPLOYEE|LastName]
               called $objs.first@[EMPLOYEE|FirstName], $objs.middle@[EMPLOYEE|FirstName]
               and $objs.last@[EMPLOYEE|FirstName]. What is the first name of the person
               you would like to lookup?"
    options:  "$objs@[EMPLOYEE|FirstName]"
    commands: 'undo' 'repeat' 'start over'
    location: [EMPLOYEE] <obj_employee:NP:_>
};
infoqst {
  state:(determined = lookup),
  path:(undefined   = $sem@[EMPLOYEE|FirstName]) ->
    text:     "What is the first name of the person you would like to lookup?"
    options:  "$db.DirectoryDB.Employees.FirstName "
    commands: 'undo' 'repeat' 'start over' 'i dont know it'
    location: [EMPLOYEE] <obj_employee:NP:_>
};
```

Figure 5: Templates. The syntax has been slightly simplified for expository reasons.

These requirements (and the chosen representations) are not too different from those employed in interlingua-based machine translation which has been proven to work reliably across a multitude of languages, including Asian languages.

4.5 Example Dialogue

In order for this example dialogue to be processed, the grammar specified by the participant is written out to a file. The grammar file is then fed into a script which generates sentences out of the grammar, and treats the resulting sentences as data for standard language model generation. The resulting language model is then used for speech recognition. This process is automatic and does not provide any manual intervention provided that the lexicon used contains phonetic transcriptions for each of the words in the grammar.

User:	Call Lori.
System:	There are two Bobs called Myers and Miller. What is the last name of the person would you like to call?
User:	I said Lori not Michael.
System	I am calling Lori Baker. Please wait.

In this dialogue, the dialogue system initiates a question interaction pattern to disambiguate the referents of the (misunderstood) noun phrase Lori. The initiation of that interaction pattern is triggered by generic constraints in the dialogue control layer. The user initiates a correction interaction pattern which terminates successfully and triggers a reevaluation of the classification of dialogue state. Since in this case the referent is unique and the constraints of the intended dialogue goal are fulfilled, the first interaction pattern is succesfully terminated and control is passed on to the back-end application to place the call. It should be noted that the designer of this application did not have to specify how the discourse should be updated or how the repair should be processed. This is handled by the interaction patterns.

User:	Call Lori Miller.
System:	I do not know a person called Lori Miller, but I do know Lori Baker and Michael Miller. What is the last name of the person you would like to call?
User:	Baker.
System	I am calling Lori Baker. Please wait.

Here, the system initiates a Navigation interaction pattern to resolve a close match of referents of a noun phrase. Again the conditions under which the system needs to instantiate that

Application	Step 3	Step 4	Step 5	Step 6	Step 7	Sum
Beer Purchase System	30 min	15 min	40 min	10 min	70 min	165 min
Directory Service	60 min	20 min	60 min	20 min	60 min	220 min
Ticket Purchase System	15 min	5 min	10 min	60 min	60 min	150 min
Roulette	30 min	30 min	30 min	30 min	45 min	165 min
CD-Player	190 min	130 min	130 min	10 min	60 min	520 min
Video Rental System	75 min	30 min	120 min	40 min	120 min	385 min

Table 1: Temporal effort for each of the steps (does not include reading of documentation)

Application	Ontology Concepts/Features	Goals	Grammar rules lexical/structural	Databases Tables / Fields	Templates Quest. / Statem.	File Length Lines of code
Beer Purchase system	57/7 (193/24)	2 (7)	256/200	1/5	5/2	456
Directory Service	4/9 (88/26)	2 (7)	77/39	1/8	4/2	163
Ticket Purchase System	5/5 (89/22)	3 (8)	11/50	1/ 3	4/4	236
Roulette	12/12 (96/29)	4 (9)	129/65	1 /3	9/4	349
CD-Player	26/12 (108/29)	6 (11)	106/66	1/4	7/0	308
Video Rental system	9/5 (93/22)	1 (6)	236/72	2/5	4/1	330

Table 2: Size of the knowledge sources. Shown are the sizes of the created knowledge sources. If knowledge sources are reused, the total size (including the reused knowledge sources) is shown in brackets. The Beer Purchase System was the only system that reused two existing dialogue packages instead of only one.

interaction pattern are generic; the system designer did not have to specify them.

5 Conclusion

We described an experiment in which six individual designed prototypical dialogue systems using the ARIADNE dialogue system. All created systems could successfully process dialogues and instantiate all interaction patterns. It could be shown that the chosen system design allows a successful separation of the specification of dialogue strategies and domain and language dependent resources. This implies that system designers do not need to be knowledgeable about dialogue processing in order to design a dialogue system. The problems that did occur with the created systems are mainly due to lack of grammar coverage or failure to provide a generation template for a specific dialogue state (for example, to provide templates that ask for the first name but not the last name). Further research will determine how the lack of template specification can be determined automatically.

References

M. Araki, K. Komatani, T. Hirata, and S. Doshita. 1999. A Dialogue Library for Task-Oriented Spoken Dialogue Systems. In *Workshop on Knowledge and Reasoning in Practical Dialogue Systems*.

B. Carpenter. 1992. *The Logic of Typed Feature Structures.* Cambridge Tracts in Theoretical Computer Science, Cambridge University Press.

R. Cole. 1999. Tools for Research and Education in Speech Science. In *Proceedings of the International Conference of Phonetic Sciences, San Francisco, USA*.

M. Denecke and A.H. Waibel. 1997. Dialogue Strategies Guiding Users to their Communicative Goals. In *Proceedings of Eurospeech, Rhodos, Greece.* Available at http://www.is.cs.cmu.edu.

M. Denecke. 2000a. An Integrated Development Environment for Spoken Dialogue Systems. In *Workshop on Toolsets in NLP, Coling, Saarbrücken.* Available at http://www.is.cs.cmu.edu.

M. Denecke. 2000b. Object-oriented techniques in Grammar and Ontology Specification. In *Proceedings of the MSC 2000 Workshop, Kyoto.* Available at http://www.is.cs.cmu.edu.

M. Gavalda and A. Waibel. 1998. Growing semantic grammars. In *Proceedings of the COLING/ACL, Montreal, Canada.*

A. Kölzer. 1999. Universal Dialogue Specification for Conversational Systems. In *Workshop on Knowledge and Reasoning in Practical Dialogue Systems.*

K.A. Papineni, S. Roukos, and R.T. Ward. 1999. Free-Flow Dialogue Management Using Forms. In *Proceedings of EUROSPEECH 99, Budapest, Hungary.*

S. Seneff and J. Polifroni. 2000. Dialogue Management in the Mercury Flight Reservation System. In *ANLP Conversational Systems Workshop.*