SEMI-AUTOMATED INCREMENTAL PROTOTYPING OF SPOKEN DIALOG SYSTEMS

Stefan Kaspar and Achim Hoffmann
School of Computer Science and Engineering
University of New South Wales, Sydney 2052 NSW, Australia
Email: {chebe,achim}@cse.unsw.edu.au

ABSTRACT

Cost-effective development and widespread deployment of spoken language systems is only possible if their development process is substantially supported by intelligent design tools. In this paper, we present PIA a new approach for fast prototyping of complex spoken dialog systems. Dialog structures are specified in a specialised fully declarative design language. Dialog design in PIA is considered a knowledge acquisition task, where knowledge about possible user input and the desired system reaction is incrementally derived from actual interactions. PIA strongly supports the designer to keep the fine balance between robustness of recognition and the naturalness of the dialog.

1. INTRODUCTION

In this paper, we present PIA a new approach for designing spoken dialog systems which is particularly effective for rapid prototyping. Dialog design in PIA is based on the following observation: At any time in a dialog there is only a limited number of tasks a user could ask for or perform. However, for every task there may be a virtually unlimited number of ways in which a user could ask for this task or execute it. Modelling ungrammatical input and out-of-vocabulary words is very difficult if not impossible. On the other hand for every task there may only be a limited number of keywords or keyword phrases that specify this task and distinguish it from another task. For the purpose of creating a spoken dialog application we are not interested in grammars that define how users can express themselves. Our approach is rather based on the set of keywords that distinguishes one task from another. Our basic idea is the following: Treat dialog design as a knowledge acquisition process [3]. Use utterances from users and and their interactions with the system to add knowledge about a spoken dialog application. The knowledge describes the tasks a user typically performs at each stage of a spoken dialog. This includes the keyword phrases that select this task and the ways in which users can be guided if they need help.

2. SYSTEM OVERVIEW

PIA, see figure 1, consists of three main components: The dialog editor, the dialog engine and the dialog knowledge wizard. The dialog editor is a graphical design interface similar to [5]. It allows designers to define a spoken dialog using the elements of our language, the *dialog units*, also abbreviated DUs. For all non interface related functionality the dialog editor provides a scripting tool based on Visual Basic. The dialog engine interprets the dialog units and their hierarchical composition. The dialog engine itself consists of four components. The *speech recognition engine*, the *text-to-speech engine*, the *grammar compiler* and the *control component*. PIA supports standard interfaces to access speech recognition and text-to-speech engines. This allows us to experiment with different third-party state-of-the art speech tools. The control component is the core of the dialog engine. It initiates the dialogs, sends ASCII texts to the text-to-speech tool, changes the state of the dialog after every speech recogniser input and controls the grammar compiler. The grammar compiler generates context-free grammars at run time using the knowledge encoded in our dialog units and the current state of the dialog. The third and last component, the dialog knowledge wizard uses knowledge extracted from utterances of real users to enhance, modify and correct spoken dialog applications. A spoken dialog designer will use the dialog editor to create an initial basic bootstrap system. Subsequently, the dialog knowledge wizard will assist the designer in revising and refining the initially provided dialog system.

3. PIA'S DESIGN LANGUAGE

The definition of our design language used in PIA was motivated by the idea of encapsulating all dialog information in a simple easily understandable data structure [1]. Our design language is based on our notion of *task*. By *task* we mean subdialogs such as those for the acquisition of various pieces of information from the user or providing requested information

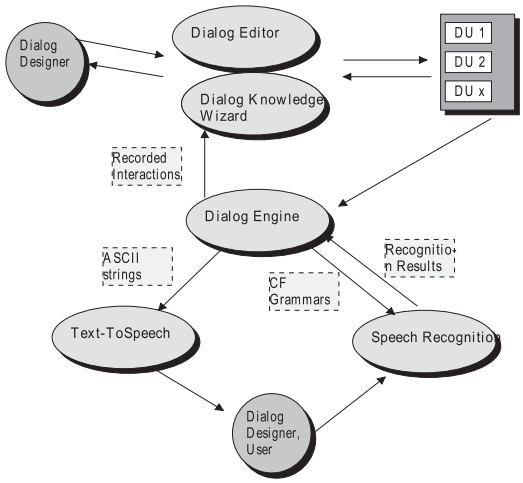

Figure 1: An overview of PIA.

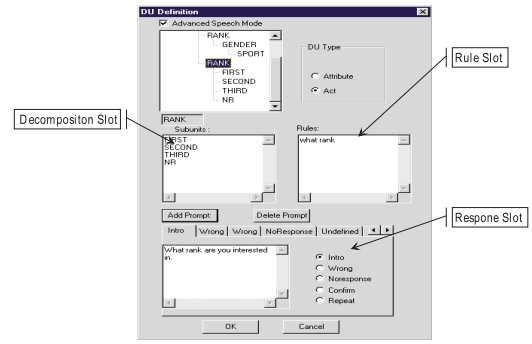

Figure 2: A dialog unit in PIA.

to the user. Our design language allows a hierarchical design of the task structure. The basic building block of our design language is the so-called dialog unit (DU). From the designer's point of view, a dialog unit represents a task a user can perform or a piece of information that the system needs in order to achieve its goal. Every dialog unit has three slots: The *rules slot*, the *decomposition slot*, and the *prompts slot*. The **rules slot:** Rules are keyword phrases with which users typically refer to this dialog unit. For example, for a DU named MOTIVATETOGIVEPRICE the rules slot may contain the two keyword phrases "* what's the price *", "* how much *". An '*' is a wild-card and stands for any sequence of words or unrecognised sounds in an utterance. A rule *matches* a spoken utterance, if the keyword phrase is contained in the utterance. We say a *user calls a dialog unit*, if at least one of the rules in the dialog unit's rule slot matches the given utterance. The **decomposition slot** stores a DU's possible decompositions, i.e. a set of other DUs. The DUs in the decomposition slots specify the subtasks or the attributes for this dialog unit. To give an example, a DU ORDERPIZZA may have the decompositions GIVETYPE GIVESIZE and GIVESIZE GIVETYPE, to list both orders in which users could give the information. The **response slot** stores text structures or prompts from which the dialog engine constructs the appropriate system output. A DU can have any number of prompts. Every prompt is of one of five predefined types: *Intro prompts* will be used when the dialog engine assumes that the user needs guidance to give the required information or perform the next step in the application. *Confirmation prompts* will be used when recognition confidence is low. The remaining types are *repeat*, *no response* and *wrong* prompts, which will be explained later.

4. DIALOG EXECUTION

A PIA led spoken dialog consists of essentially a user calling dialog units and the system producing appropriate output. That is, our dialog engine has to handle three different tasks; recognizing which dialog units have been called, guessing what dialog units will be called next and producing system output. At the core of PIA's dialog engine is a state change algorithm. We use states to define three different things; what dialog units have already been called, what dialog units are likely to be called next and which dialog unit is the most likely dialog unit to be called next. By defining dialog execution at the state level, we get a very flexible way to define mixed initiative [4]. Depending on the quality of our speech recognizer the dialog units that can be called at a certain state of the dialog can be restricted to a suitable number. A dialog unit can be in one of four possible states: *sleeping*, *alert*, *activated*, or *set*. Initially all the dialog units are in the sleeping state. Alert dialog units are the ones that can be called by a user, i.e. if one of the DU's rules matches the utterance. The activated dialog unit is the one that is most likely to be called next. If the dialog stalls, i.e. there is no user input, the dialog engine will prompt the user for input relating to the activated dialog unit. We say *a dialog unit is set*, if all the dialog units in the decomposition slot have been set. If a dialog unit has an empty decomposition slot, it is set as soon as it is called. The execution of a spoken dialog in PIA is governed by three processes, *generate input predictions*, *generate output*, and *change dialog unit states*. The first two processes are triggered by state changes, the last process is triggered by the speech recogniser. We will give some more detailed information about these three processes in the following.

Dialog unit states State changes are triggered by the speech recogniser or by the application itself. For example in order to initiate a spoken dialog interaction the application must activate one of the dialog units. An application can also set dialog units if e.g. the in-

formation represented by this dialog unit is no longer needed. If the speech recognition engine returns the *response* or *noresponse* event, the dialog engine will activate or set the dialog units whose rules are part of the speech recognition result. If the state of a dialog unit is changed, our control strategy causes the states of other dialog units to be changed too. Our control strategy builds on a set of heuristic rules that define how state changes of one dialog unit causes state changes of others. We demonstrate the three main rules on a small example, taken from our Sydney2000 application . The Sydney2000 application lets user ask for information about upcoming Olympic events and results. In our application we have a dialog unit EVENTS containing the decomposition SPORT GENDER and the rules "what time", "starting times". SPORT in turn might contain the decomposition CYCLING SWIMMING. **Rule 1:** if a dialog unit changes to set, activate the next following step in the decomposition. If a user speaks something like "what time does the cycling start", the system will activate GENDER because this is the only missing information in order to execute the request. The word cycling will set the dialog unit CYCLING which in turn sets SPORT. The next following step is GENDER. **Rule 2:** If an utterance does not set any dialog unit, activate the dialog that was called last in this utterance. A sentence like "can you tell me about starting times" calls the dialog unit EVENTS, as it contains the rule "starting times". **Rule 3:** If a dialog unit is activated or alerted, alert the dialog units of its decomposition slot. Alerting dialog units that go beyond the current focus allows users to execute more than one task at a time. If the DU EVENTS is activated, it may ask what sport you are interested in? A user may respond with "women's swimming", calling the DU SPORT and GENDER with a single utterance.

Generate Spoken Output As stated above only one dialog unit is activated at a time, namely the dialog unit that is most likely to be called next. If the user does not take initiative, PIA has to prompt the user for input or guide them to the next task. For that purpose, PIA uses the text-strings in the response slot of the activated dialog unit. The first text string to use is the *intro prompt* which contains information about the choices the user has. If there is no valid input the dialog engine takes one of the *wrong* or *no response* prompts to provide further help. In some situations an input will be confirmed with a *confirm* or *repeat prompt*. Wrong prompts should be defined so as to gradually constrain the possible user input. If all the defined wrongs and noresponses have been used and the user still has not given a valid input, the dialog engine terminates the execution of this dialog unit and returns a *failed* signal.

Generate Input Predictions PIA's dialog engine constrains the speech recognizer's search space by constructing context-free grammars (CFGs). The compilation algorithm in PIA adds for every dialog unit that is alert a variable to the CFG. If the dialog unit's decomposition slot is empty but the rules slot is not empty we add the following definition:

```
DlgunitVar={rule1| ... |rule N };
```

If the DU has decompositions that are alert add

```
DlgunitVar=[rule1| ... |rule N] {decomp
DlgunitVar | decomp DlgunitVar ... };
```

If the DU has an empty decomposition and rule slot add:

```
DlgunitVar=dialog unit name;
```

5. DIALOG DESIGN

A dialog structure is declaratively described by defining a set of dialog units. The dialog knowledge wizard guides the user through the dialog design process, i.e. through enhancing, modifying or updating an initially crude bootstrap application. Our dialog knowledge wizard ensures that all interactions which have been covered at some stage will also be covered at all later stages of the design process. For that purpose the wizard maintains a phrase database specifying what phrases have previously called what dialog units. The wizard has to deal with the trade-off of producing rules that are general enough to cover a broad spectrum of possible user input without being too general and compromising the recognition robustness. In every cycle of the dialog design process the designer either picks a recorded utterance or provides a sample phrase himself. If the utterance is wrongly recognized, according to the designer, the designer is asked to type in the uttered phrase. Dialog design consists of two processes: the mapping of words to dialog units and the generalization of rules. The first process is run in every cycle of the knowledge acquisition process. The second process only when the number of rules in a dialog unit reaches a certain threshold.

Mapping of words to dialog units From the dialog units that were alert or activated when the error occurred the wizard constructs a CFG as described above. The original (typed) utterance is then repeatedly modified by removing words till it passes an LR-Parse with this CFG. The string that is parsed defines an initial mapping of words to dialog units. The mapping is graphically presented to the designer. Our dialog knowledge wizard tool now guides the designer through the process of assigning the unmapped words

1. Wizard creates CFG from dialog units.
2. Removes word from utterance.
3. Until modified string passes CFG.
4. Presents mapping to the designer as graph.
5. Designer removes or adds words to the mapping graph.
6. If parts of the utterance remain unmapped designer.
   (a) defines new dialog unit or inserts dialog unit defined at different dialog location.
   (b) specifies neighbour of new dialog unit.
7. Wizard adds phrases to phrase database.
8. If phrase is not matched by existing rules, wizard adds phrase to rule base.

Table 1: The mapping algorithm.

to new or existing dialog units. The mapping algorithm is given in table 1.

Generalize rules The purpose of the Generalize Rules algorithm is to speed up the development process by shortening the rules. Shorter rules will cover more utterances as rules represent the keyword phrases of our keyword spotter. On the other hand we have to make sure that rules that are too short do not match phrases intended for other dialog units and also that the modified rules don't compromise the speech recognition robustness. To describe our generalize rules algorithm we have to introduce some new definitions. The competing dialog units of du_1 are DUs belonging to the same decomposition slot as du_1 and which are in parallel to du_1. The phonetic distance of a du_1 to du_2 $D(du_1, du_2)$ intuitively defines how unlikely it is that a phrase that intends to call du_2 calls du_1. We will give the exact definition of the phonetic distance in the next chapter. The overall phonetic distance of du_1 to all its competing dialog units C, $DT(du_1)$ is defined as:

$$DT(du_1, C) = min_{i \in C} D(du_1, du_i)$$

$\Delta DT(du_1, C, w)$ is the difference in the phonetic distance after removing the word w from the rules of du_1. $L(w, du_1, c)$ is ratio of occurrence of a a word w in du_1 divided by the average ratio of occurrence in the competing dialog units. Now we rank all words in du_1 according to $L(w, du_1, C)$ and $\Delta DT(du_1, C, w)$. The word with the highest value of L or DelatDT gets the rank 1 . The generalization degree $G(du_1)$ is defined as the number of rules divided by the number of entries in the dialog unit's phrase database. The generalize algorithm now removes words from the rule slot of du_1 till the generalization degree reaches a certain threshold. It does this by removing the words with worst rank first, where as the overall rank is taken as the product of the two ranks.

The phonetic distance DT The purpose of the phonetic distance is to measure how likely or unlikely it is that the rules of a dialog unit match the phrases belonging to competing dialog units The basic idea is to use the speech recogniser to try to recognise spoken phrases that were produced by our text-to-speech engine. For each phrase in the phrase database of the competing dialog units we create an acoustic representation using a text-to-speech tool. Subsequently, we activate our speech recogniser with a grammar that has all the rules of du_1 in parallel. $S(p_i, CFG(du_1))$ is the likelihood measure returned by the speech recogniser's Viterbi algorithm that tells us how likely it appears that phrase p_i matches the Hidden Markov Network constructed from the grammar $CFG(du_1)$. The distance measure between du_1 and du_i is then defined as:

$$D(du_1, du_i) = (max_{p_i \in C} S(p_i, CFG(du_1))^{-1}$$

6. DISCUSSION

This paper introduced PIA, a new framework for fast prototyping of complex spoken dialog systems. By treating dialog design as a knowledge acquisition process, we avoid the need for highly skilled experts that specify formal grammars that cover possible user input. The structure of our design language proved intuitive to our dialog designers and well-suited to the task. PIA was used to build a ticket ordering and a movie information application. Currently we are implementing a sport result information system. The so called Sydney2000 application is still being extended and contains more than 80 dialog units to date. Further work will have to be done to compare and measure [2] the quality of the produced applications whith other related research. Furthermore, our future research will investigate ways of dealing with other well-known problems of natural language processing, such as sudden topic changes and anaphoric reference.

7. REFERENCES

1. A. Abella, M. Brown, and B. Buntschuh. Development principles for dialog-based interfaces. In *ECAI96: Dialogue Processing in Spoken Language Systems*, 1996.
2. L. Hirschman and H. Thomson. Overview of evaluation in speech and natural language processing. *Survey of the State of the Art in Human Language Technology. Cole R., Editor*, 1996.
3. S. Kaspar and A. Hoffmann. Using knowledge acquisition to build spoken language systems. In *European Knowledge Acquisition Workshop*, 1997.
4. R. W. Smith. Spoken variable initiative dialog: An adaptable natural-language interface. *IEEE Expert*, 2:45–50, 1994.
5. S. Sutton, D. Novick, R. Cole, and M. Fanty. Building 10,000 spoken-dialogue systems. *Proceedings of the Internatiaonal Conference on Spoken Language Processing, Philadelphia*, 1996.