Dialog with a Personal Assistant

Fabrício Enembreck[1], Jean-Paul Barthès[2]

[1] PUCPR, Pontifical Catholic University of Parana
PPGIA, Post-graduation Program in Applied Informatics,
1155 Imaculada Conceição St.
Curitiba PR - Brazil
fabricio@ppgia.pucpr.br

[2] UTC – Technological University of Compiègne,
HEUDIASYC – Royallieu Research Center,
60205 Compiègne - France
barthes@utc.fr

Abstract. This paper describes a new generic architecture for dialog systems which enable communication between a human user and a personal assistant, based on speech acts. Dialog systems are often domain-related applications; a system developed for specific applications and not reusable in other domains. A major problem concerns the development of scalable dialog systems that might be extended with new tasks without much effort. In this paper, we discuss generic dialog architecture for a personal assistant. The assistant uses both explicit task representation and knowledge to attain an "intelligent" dialog. The independence of the dialog architecture of both knowledge and tasks, allows the agent extension without dialog structure modification. The system has been implemented in a collaborative environment in order to personalize services and facilitate the interaction with collaborative applications, such as e-mail clients, document managers or design tools.

Keywords: Dialog Systems, Natural Language, Personal Assistants

1 Introduction

When using computers to work or to communicate, one observes three major trends: (i) the increasing complexity of user's environment; (ii) cooperative work growth; and (iii) knowledge management fast spread. Such complexity overwhelms users with tasks accomplished through many different tools (e-mail managers, web browsers, word processors, etc.). The resulting cognitive overload leads to disorganization, which has negative impacts, particularly when different people share some information. A major issue is thus, to develop better and more intuitive interfaces.

We are currently developing a Personal Assistant Agent in a project called AACC[1] for collaboration support between French and American groups of students, located at UTC (Université de Technologie de Compiègne) and at ISU (Iowa State University). The students must design electro-mechanical devices by using assistant agents. In this paper, we focus on the Personal Assistant (PA), discussing how a Natural Language interface allows the user to attain efficient interaction with the Assistant, and how this interaction can be used to increase the agent knowledge of the user. We developed a generic dialog system using several models: dialog model, tasks models, domain knowledge model and user model. We focus on the construction of the dialog model and show how speech acts can be used to make the dialog model independent of domain data (tasks and knowledge).

The paper is organized as follows: Section 2 presents some theory on natural language and dialog systems; Section 3 describes the architecture of our system. Section 4 discusses deployment and evaluation. Section 5 discusses related work and finally, Section 6 concludes with our observations.

2 Natural Language and Dialog Systems

Communications using natural language (NL) have already been proposed. Early attempts at it were done in the late sixties, early seventies. Szolovits et al. [1] or Goldstein and Roberts [6] developed formalisms and languages to stand for the knowledge contained in English utterances. The internal language would support inferences in order to produce answers. In the first project (OWL language), the application meant to draw inferences from an object database, in the second one (FRL-0 language), it meant to schedule meetings. The internal language was used to represent knowledge and translate utterances. Such an approach can simplify the representations and inferences, because only very specific applications are considered. However, for new domains, a major part of the application must be rewritten, an unfortunate fact in an environment involving several tasks (like collaborative work), when part of the dialog must be recoded every time a new task is added to the system. Later, several researchers, including Schank and Abelson [13], Sowa [16] or Riesbeck and Martin [12], proposed sophisticated knowledge representation techniques to handle natural language and represent meaning. They made it possible to express complex relationships between objects. The main difficulty toward such techniques is to define the right level of granularity for the representation, because even very simple utterances can produce very complex structures. Moreover, modeling concepts and utterances is a very time consuming non-trivial task. The field of NL and machine understanding has expanded since the early attempts. However, the techniques being used are fairly complex and most of the time unnecessary for the purpose of conducting dialogs, particularly in goal-oriented dialogs, since "it is not necessary to understand in order to act." Both NL techniques and dialog systems use internal structures, but the latter uses simpler structures to represent knowledge, e.g., ontology, semantic nets, or frame

[1] The AACC (Agents d'Aide à la Conception Coopérative) project is a project involving the CNRS HEUDIASYC laboratory of UTC, and the LaRIA laboratory of UPJV in France.

systems. The emphasis however, is not on the adequacy of the knowledge representation, but rather on the dialog coordination done by a dialog manager. In addition, the dialog systems are designed so that they can be used in other domains without changes in the dialog structure, in order to save development time.

Many dialog systems implementing NL interfaces have been developed in applications like speech-to-speech translation [8], meeting schedule, travel books [2] [15], telephone information systems, transportation and traffic information, tutorial systems, etc. Flycht-Eriksson [5] has classified dialog systems into query/answer systems and task-oriented systems. Query/answer systems include consultation systems such as tourist information, time information, traveling, etc. Task-oriented systems guide the user through a dialog to execute a task. Tasks range from very simple ones like "find a document" to complex ones decomposed into several subtasks. We argue that, for a dialog system to support collaborative work, it must be both of query/answer and task-oriented types, because user problems can involve questions ("Where does Robert work?", "What does electrostatic mean?") and tasks ("Find a document for me", "Send a message to the project leader"). We present our approach in the next section.

3 A Personal Assistant that Participates in Dialogs

We discuss the different models that compose our dialog system and pay special attention to the dialog model.

```
1  User:    Send a mail to Marco for me.
2  System:  Who are the Carbon Copy receivers?
3  User:    What is Leila's address?
4  System:  The address of Leila is "25 rue de Paris".
5  System:  Who are the Carbon Copy receivers?
6  User:    none
7  System:  What is the subject?
8  User:    Ohh I'm sorry I made mistake!
9  System:  Who are the Carbon Copy receivers?
10 User:    Mary's husband.
11 System:  What is the subject?
...
```

Fig. 1. Open Dialog.

3.1. Dialog Model

Our approach uses a speech act system. According to Searle [14], the speech act is the basic unit of language used to express meaning through an utterance that expresses an intention of doing something (to act). In our system, the users' utterances express questions and requests. Then, a PA starts a dialog to reach a state where an action is triggered according to the intention of the user. The dialog states are nodes of a dialog graph in which most speech acts are available at all times. For instance, consider the dialog on Fig. 1. In lines 1-5, the user requests the task "send mail" and the system asks for additional information. The user enters a new question during the task dialog

(lines 3-4), the system answers it and returns to the previous dialog context. To accomplish this, the system keeps a stack of states. When a new task is requested, the system pushes a number of states in the stack equal to number of slots required to accomplish the task. When a slot is successfully filled, the system marks it as "popped." This strategy also allows the user to return to previous states (Fig. 1 lines 5-10).

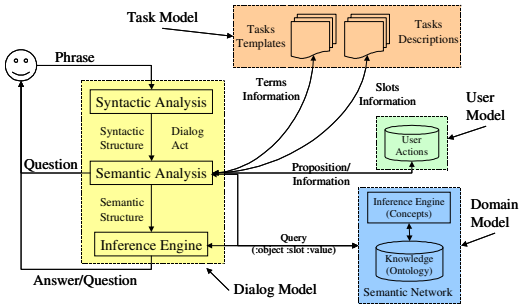

Fig. 2. Model-Based Architecture.

Our system (architecture Fig. 2) has been developed for dialog-based and question/answer interaction. In a task-oriented dialog the system asks the user to fill slots of a given task (like *send e-mails* or *locate documents*). Then, the system runs the task and presents the result. In a question/answer interaction, the user asks the system for information. In this case, the assistant uses its knowledge base to provide correct answers.

Fig. 2 shows that, when the system receives a simple question or information, the syntactic analyzer produces a syntactic representation. The representation gives the grammatical structure of the sentence (verbal phrase, nominal phrase, prepositional phrase, etc). We developed a grammatical rule base, where each rule refers to a single dialog act. The semantic analyzer uses this structure to build requests. The role of the semantic analyzer is to identify objects, properties, values and actions in the syntactic structure by using the object hierarchy and relations defined in the domain model. The information is used to create a formal query. During the semantic analysis, the system can either ask the user for confirmation or request additional information to resolve conflicts. Finally, the inference engine uses the resulting formal query to retrieve the required information and the system presents it to the user.

Whenever a task-oriented dialog starts, the semantic analyzer first tries to determine if a known task is concerned. If it is the case, it verifies the slots initially filled with information and continues the dialog to acquire important information to execute the task. To identify the task and concerned slots, the analyzer retrieves information from task models (see Section 3.3). The recursive stack strategy allows the user to use relations and concepts defined in the domain model (see Fig. 1, line 10) at all times.

In our system, the dialog coordination depends on the types of utterances denoted by speech acts. Schank and Abelson [13] proposed a categorization of messages, of which we keep the following:

- *Assertive*: message that affirms something or gives an answer (e.g., "Paul is professor of AI at UTC", "Mary's husband");
- *Directive*: gives a directive (e.g., "Find a document for me.");
- *Explicative*: ask for explanation (e.g. "Why?");
  Interrogative: ask for a solution (e.g., "Where does Paul work?").

To maintain a terminological coherence, the previous categories will be referred to by speech acts: Assert act for Assertive; Directive act for Directive; Explain act for Explicative and WH/Question (where, what or which) or Y-N/Question (yes-no) for Interrogative. Speech acts are used to classify nodes of the dialog state graph. The dialog graph represents a discussion between the user and the system where nodes are the user's utterances and arcs are the classification given to the node.

To improve communications we introduced new specialized speech acts:

- *Confirm*: used by the system to ask the user to confirm a given value;
- *Go-back*: used by the user to go to the previous node of the dialog. For instance, when the user made a mistake;
- *Abort*: used by the user to terminate the dialog;
- *Propose*: used by the system to propose a value to a question. This act can be followed by a Confirm act.

Fig. 3 shows the functional architecture of the dialog coordination. Fig. 3 shows how we implemented the semantic interpretation for each speech act. The interaction with the user always starts with the "Ask" system act. The default question is "What can I do for you?" Then, the user can ask for information or start a task. Based on the user phrase the Task Recognizer classifies the user phrase as a "General Utterance" or a "Task-Related Utterance." The task recognizer compares the verbs and nouns of the verbal and nominal parts of the utterance with the linguistic information previously stored into task templates (section 3.3).

The semantic analyzer simply analyses a general utterance by taking into account the speech act recognized during syntactic analysis. Four types of speech acts are possible: Assertive (assert act), Explicative (explain act), Directive (directive act) and Interrogative (wh/question or y-n/question acts). Finally, the Inference Engine can ask the knowledge base for the answer. The Inference Engine does a top-down search in the concepts hierarchy, identifying classes and subclasses of concepts, properties and values, and filtering the concepts that satisfy the constraints specified in the queries.

The interpretation of a task-related utterance is more complex. First, the Task Recognizer locates the correct task based on the terms present on the nominal phrases by using the terminological representation (Task Template on Section 3.3) about the tasks. Next, the task recognizer matches the modifiers of these nominal phrases with information about the parameters for filling the slots referred in the phrase. Then, the Task Engine will ask the user about other parameters sequentially. For each parameter an *Ask* act is executed by the system. At this point, the user can: simply answer the question (in this case the task engine fills the slot and passes to the next one), ask for *Explanation*, *Go-back* to the last slot or *Abort* the dialog. When a user asks for *Explanation*, the Task Explainer presents the information coded on the task description concerning the current parameter (*params-explains* on Section 3.3) and the task engine resumes the dialog concerning the current parameter. The *Go-back* act simply makes the task engine roll back the dialog flow to the last parameter filled. When the

user enters an *Abort* act the Task Eraser reinitializes variables concerning the current task and the system goes back to the default prompt or to the top task of the task stack.

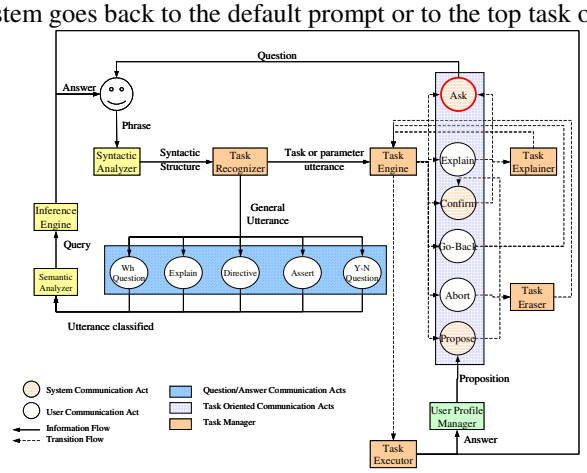

Fig. 3. Functional Architecture.

The system can also ask for confirmation and propose values with *Confirm* and *Propose* speech acts respectively. To confirm a given value, the system shows a default question like "Confirm the value?" and waits for a valid answer. If a positive answer is given, the system confirms the value and the dialog continues. Otherwise, the task engine asks the question concerning the current parameter again. The *Propose* act is executed before the *Ask* act. The User Profile Manager looks at the user model seeking a value to propose to the user. If a value is found, it is presented to the user and the system asks the user for a confirmation by executing a *Confirm* act.

Finally, when no more information is needed, the Task Executor executes the task and presents either the solution or a feedback to the user. It also sends information to User Profile Manager that saves the current task in the user model.

3.2. How to Interpret the User's Utterances

In our approach, we use a simple regular English grammar extended from Allen [1]. We divided the syntactic and semantic processing into two steps. The algorithm uses nominal and prepositional phrases to locate known objects and properties. We implemented an algorithm that analyzes the syntactic representation and the domain ontology and generates well-formed requests. The semantic analysis is complemented by a linguistic analysis of the phrase, where we try to identify if an action, e.g., "leave", or some general modifier, e.g., "time (when), quantity (how many)" is being asked by using a list of verbs denoting actions and modifiers. Finally, the inference engine takes the resulting formal query and does the filtering. The query is a conjunction of atomic queries. The format of each query can be "(:Object O: slot S: value V)" for object selection or "(:Object O: slot S)" for slot-value verification. "O" and "V" can be complex recursive structures.

3.3. Task Model

We divide a task into two parts: *template* and *description*. To identify the task requested by the user and the information related to parameters, the semantic analyzer uses the template part of the task. The template contains linguistic terms related to the parameters and the verbs used to start the task. The task description describes all the information required for the task execution. The data required in the task structure definition are:

- *Params*: the parameters of the task;
- Params-values: the values given by the user as parameters;
- Semantic-value: the specification of a function that must be executed on the value given by the user. For instance, the function "e-mail" can give the value "carvalho@utc.fr" for the term "carlos" given by the user;
- Params-confirm: it is true if a confirmation for the value given by the user is necessary;
- Params-labels: the question presented to the user;
- Params-save: the specification if the values of the parameters are used to generate the user model (see next Section);
- Params-explains: if true, (for a parameter) an explanation is given to the user;
- Global-confirm: if true, a global confirmation for the task execution is made.

3.4. User Model (UM)

We use a dynamic UM generation process. All the tasks and query executions are saved within the user model. Values are predicted with a weighed frequency-based technique. We use UM dynamic generation to avoid manual modeling of users. The main idea is to minimize the user's work during the execution of repetitive dialogs by predicting values and decreasing the needs for feedback. A more elaborated discussion about user model in dialog systems is beyond the scope of this paper.

3.5. Domain Model

To allow the system to identify users' problems and provide answers to particular questions, it is necessary to keep a knowledge base within the assistant. The knowledge of the agent is used to identify objects, relations and values required by the user. Such objects can represent instances of various object classes (*People*, *Task*, *Design*, etc.) and have a number of synonyms. Therefore, it is quite important to use efficient tools to represent objects, synonyms and a hierarchical structure of concepts. In our approach, we use the MOSS system proposed by Barthès [4] to represent knowledge. MOSS allows object indexing by terms and synonyms. Several objects can share the same index. MOSS has been developed at the end of seventies to represent and manipulate LISP objects. The objects can be versioned and modified simultaneously by several users. The MOSS concepts have been further used in object-oriented databases.

1 User: Joe is a professor of Artificial Intelligence at UTC.
2 System: OK.
3 User: What does Joe do?
4 System: Joe is a professor of Artificial Intelligence at UTC.
5 User: What is Joe's profession?
6 System: Joe is a professor of Artificial Intelligence at UTC.
7 User: Where does Joe works?
8 System: At UTC
9 User: Is Joe a professor of Artificial Intelligence?
10 System: Yes
11 User: Is Joe a professor of artificial intelligence at UTC?
12 System: Yes
13 User: Is Joe a professor of artificial intelligence at PUC?
14 System: I don't know. None information about this.
15 User: What does Joe teaches?
16 System: Artificial Intelligence

Fig. 4. Intelligent Dialog.

Knowledge is important because it increases the system capability of producing rational answers. Consider the dialog reproduced in 4. Initially, the system has no information on Joe's occupation. The user starts the dialog with an "Assert" dialog act by stating Joe's occupation. Afterwards, the user asks several questions related to the initial utterance and the system is able to answer them. The system can identify and interpret correctly very different questions related to the same concepts (lines 3 and 5) and answer questions about them. This is possible because the semantic meanings of the slots are explored in the queries. Thus, a slot can play a role that is referenced in different ways.

4 Deployment and Evaluation

We currently develop a personal assistant (PA) in the AACC project. We hope to use the mechanisms discussed on this paper to improve the interaction with the present assistant prototype. Then, students will have an assistant able to execute services, help them with mechanical engineering tasks, and answer questions using natural language.

The current state of the prototype did not allow its immediate application because the current interface is not good enough. The interface is being redesigned to test our dialogue approach during the mechanical engineering courses given to students. During the Spring semester of 2004, we will evaluate the results of students either using or not the assistant, and we will measure the quality of the information provided by the assistant. A formal evaluation of the system can be accomplished, for instance, with the criteria presented by Allen et al. [2], however, for us; the main criterion is the acceptation or the non-acceptation of the system by the students.

5 Related Work

Grosz and Sidner [7] discuss the importance of an explicit task representation for the understanding of a task-oriented dialog. According to the authors, the discourse is a

composite of three elements: (i) linguistics (utterances); (ii) intentions and (iii) attentional states (objects, properties, relations and salient intentions at any given point of the discourse). Our system presents some very close elements such as: linguistic information (template of tasks), intentions (given by speech acts) and specific information about tasks and task properties. Very often, assistants communicate by using ACLs (Agent Communication Languages) like KQML or FIPA (Foundations for Intelligent Physical Agents) ACL. However, such messages are based on *Performatives* rather than *speech acts*. A basic difference between performatives and speech acts is that they tell what to do when something is said (action) and do not express the meaning of what is said (intention). In other words, ACL messages cannot express a Go-back like speech act because there is not an explicit action in the utterance. Unlike most dialog systems, the dialog flow implemented in Section 3.1 is completely generic. Thereby, new tasks and knowledge can be added to the system (Assistant) without changing or extending the dialog structure. Generic dialog systems are relatively rare. Usually, the developer specifies state transition graphs where a dialog flow should be entirely coded like the dialog model discussed by McRoy and Ali [10]. Kölzer [9] discusses a generic dialog system generator. In the Kölzer's system, the developer must specify the dialog flow using state charts. Such techniques make the development of real applications quite hard. In contrast, in our approach, we need to specify only task structures and domain knowledge concerned. Rich and Sidner [11] also used the concept of generic dialog systems. The authors used the core of the COLLAGEN system to develop very different applications. COLLAGEN is based on a plan recognition algorithm and a complex model of collaborative discourse. The problem is that most part of the collaborative discourse must be coded using a language to model the semantic of communicative acts. The representation includes knowledge concerning the application. So, the knowledge of the system is intermixed with the dialog discourse, which makes the application domain dependent. Allen et al. [3] used speech acts to model the behavior and the reasoning of a deliberative autonomous agent. Speech acts are separated into three groups: Interaction, Collaborative Problem Solving (CPS) and Problem Solving (PS). Assuming we do not intent to model interaction with the user like a problem solving process, the PS and CPS speech acts proposed by Allen are not relevant to our work, because they are domain-related. However, the interactions acts are very similar to the speech acts that we proposed.

6 Conclusions

In this paper we addressed the problem of communication between User and Personal Assistant Agent (PA). In the AACC project, users need to communicate with a PA to do collaborative work. We argued that natural language should be used to provide a better interaction. A user assistant communication module was developed as a modular dialog system. To execute services and to ask for knowledge, the user enters a dialog with his PA. In this application, the dialog coordination model should be generic to support the scalability of the system concerning the addition of new tasks without much effort. Then, we introduced a new generic model of dialog based on

speech acts. Simple tasks and questions have been used to highlight the effectiveness of the system and the advantages in relation to traditional collaborative work tools.

References

1. Allen, J. F., *Natural Language Understanding*, The Benjamin/Cummings Publishing Company, Inc, Menlo Park, California, 1986. ISBN 0-8053-0330-8
2. Allen, J. F.; Miller, B. W. et al. *Robust Understanding in a Dialogue System*, Proc. 34 th. Meeting of the Association for Computational Linguistics, June, 1996.
3. Allen, J. ; Blaylock, N. ; Ferguson, G., *A Problem Solving Model for Collaborative Agents*, Proc. of AAMAS'02, pp. 774 – 781, ACM Press New York, NY, USA , 2002. ISBN 1-58113-480-0
4. Barthès, J-P. A., *MOSS 3.2*, Memo UTC/GI/DI/N 111, Université de Technologie de Compiègne, Mars, 1994.
5. Flycht-Eriksson, A., *A Survey of Knowledge Sources in Dialogue Systems*, Proceedings of the (IJCAI)-99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems, International Joint Conference on Artificial Intelligence, Murray Hill, New Jersey, Jan Alexandersson (ed.), pp 41-48, 1999.
6. Goldstein, I. P.; Roberts, R. B., *Nudge, A Knowledge-Based Scheduling Program*, MIT AI memo 405, February, 23 pages, 1977.
7. Grosz, B. J., Sidner, C. L.. *Attention, intentions, and the structure of discourse,* Computational Linguistics, 12(3):175--204, 1986.
8. Kipp, M.; Alexandersson, J.; Reithinger, N., 1999. *Understanding Spontaneous Negotiation Dialogue*, Linköping University Electronic Press: Electronic Articles in Computer and Information Science, ISSN 1401-9841, vol. 4, nº 027.
9. Kölzer, A., *Universal Dialogue Specification for Conversational Systems*, Linköping University Electronic Press: Eletronic Articles in Computer and Information Science, ISSN 1401-9841, vol. 4, nº 028, 1999.
10. McRoy, S., Ali, S. S., *A practical, declarative theory of dialog*. Electronic Transactions on Artificial Intelligence, vol. 3, Section D, 1999, 18 pp.
11. Rich, C.; Sidner, C. L.; Lesh, N., *COLLAGEN: Applying Collaborative Discourse Theory to human-Computer Interaction*, AI Magazine, Special Issue on Intelligent User Interfaces, vol 22, issue 4, pp. 15-25, Winter 2001.
12. Riesbeck, C., Martin, C., *Direct Memory Access Parsing*, Yale University Report 354, 1985.
13. Schank, R. C., Abelson, R. P., *Scripts, Plans, Goals and Understanding, Lawrence Erlbaum Associates*, Hillsdale, NJ, 1977.
14. Searle, J., *Speech Acts: An Essay in the Philosophy of Language*, Cambridge, Cambridge University Press, 1969.
15. Seneff, S.; Polifroni, J., *Formal and Natural Language Generation in the Mercury Conversational System*, Proc. 6th Int. Conf. on Spoken Language Processing, Beijing, China, October, 2000.
16. Sowa, J. F., *Conceptual Structures. Information Processing and Mind and Machine*, Addison Wesley, Reading Mass, 1984.
17. Szolovits, P; Hawkinson L. B.; Martin W. A., *An Overview of OWL, A Language for Knowledge Representation*, Technical Memo TM-86, Laboratory for Computer Science, MIT, 1977.