INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

Model-Based User Interaction for Accessing Electronic News

Vitaly Iourtchenko

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

December 1998

ABSTRACT

Model-Based User Interaction for Accessing Electronic News

Vitaly Iourtchenko

Natural language interfaces support human-machine interactions in a way that is natural and that does not require elaborate user training. The two main problems in human-computer dialog are resolution of ambiguity and incompleteness. A model-based architecture is introduced in this thesis that provides a solution to these problems. The uniqueness of the solution is based on five interacting models: a language model responsible for the processing of individual user queries, a domain-independent dialog model providing dialog management, a user model keeping track of the user's tasks and preferences, a world model resolving references to both common-sense and domain-dependent knowledge, and an error model handling erroneous user input. The implementation and testing have been carried out in the context of electronic news delivery as the chosen application domain. The design of the dialog and error models is sufficiently generic to be used in different application domains; the language, world, and user models are application-specific and encapsulate domain-dependent information. The language model is based on a set of syntactic classes that put restrictions on user input. These classes were derived using a brief study of a sample population of users. A Java-based implementation of a prototype of the five models has shown the feasibility of our approach to dialog management.

Acknowledgements

I would like to take this opportunity to extend my gratitude to **Dr. T. Radhakrishnan**, my thesis supervisor and mentor, for the motivation and financial support he has given me throughout my studies. He has been an inexhaustible source of ideas and suggestions, and has always been there for me whenever I needed it most. The countless hours he spent with me, above and beyond the call of duty, discussing my work or suggesting improvements to the never-ending drafts, has made a world of difference in the final appearance of this thesis. His professionalism and dedication to his students has earned him a new level of respect and admiration – a level only reserved to a handful of people in my life.

I wish to thank my parents for encouraging me to pursue graduate education and for their moral and financial support during this period of my life; I am deeply indebted for the countless sacrifices they made for me. Without their love, respect and support, none of this would have been made possible.

I would also like to thank the many people at Nortel, especially **R. Kenworthy** and **H. Cameron**, for funding the industrially-oriented research grant that made this work a reality.

Many thanks go out to **Priya Parimelalagan** for putting in a great deal of effort in the user survey presented in Appendix I.

Finally, I wish to thank **Ian Pavelko** for providing the much-needed technical support with the collection of video data for the news database.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Electronic News and Internet

The term "electronic news" is used to refer to digitally stored and distributed news content in the form of multimedia (such as text, audio, or video combined with audio). The need to access or synthesize information in the present era of Information Technology (IT) is fast growing. The popularity of Internet is bringing the ability to access computer-stored information within the reach of millions of people spread all over the world. Several sectors of industry and business are already benefiting from this IT revolution, including the financial sector and banking, travel industry, entertainment industry, digital libraries, publishing industry, communication industry, electronic commerce, and the education sector.

In this context, we identify the following three aspects to be important for providing the right information to the seekers:

(a) ***Information acquisition aspects:*** this deals with diverse factors such as gathering information, validation, indexing and classification for the purposes of storage and retrieval, quality assurance, time constraints, etc.

(b) ***User aspects:*** specification of user needs, query languages, I/O modalities, authentication of users, presentation of information to suit the user needs, ease of use and ease of learning, preservation of privacy and security.

(c) ***Information management aspects:*** organization and storage of information, maintenance of the currency of the information stored, sharing, distribution, search

and retrieval, cost effectiveness in the provisioning of information, accountability, etc.

In each aspect mentioned above, there are several issues and open problems yet to be solved. At the same time, a great amount of progress has been already made to make the IT industry usable today. The work reported in this thesis falls into the category of user aspects. In this work, accessing electronic news is used as a case study. The goal of the thesis is to investigate how a set of interacting models can provide better user-interaction possibilities. Several daily newspapers are available today in digital format to the Internet users. However, broadcast news sources are delivered in continuous streams, and converting them from their traditional audio or video format to digital form that is amenable for computerized search is not a simple task. In the Informedia project at Carnegie Mellon University, on-the-fly techniques were developed for video skimming from TV news and audio skimming from the radio news [Smith98].

The Internet based access to electronic news can be characterized as follows:

1. It provides world-wide reachability at the electronic speed.
2. Customizing the news to an individual user's needs is quite possible.
3. News from multiple sources can be combined or contrasted.
4. It is easy to follow through a chain of related news items over a time period.
5. The stored news is available to the users at all times.
6. News can be in multiple languages and in multimedia form.
7. Based on the user's interest profile, a news item can be delivered even without an initiation from the user's side (proactive delivery).

In order to view TV news or listen to radio, one needs a television set or a radio, whereas reading a newspaper does not require any special devices. In contrast, several things are needed in order to receive electronic news through the Internet. There has to be a computer connected to an Internet service provider. The computer must be equipped with multimedia capability. There must be sufficient bandwidth and resources available to deliver large video files. The software support for user interaction must be simple and versatile. Security must be enforced to maintain the user's privacy.

It should be cost effective for the user to receive news in this manner when compared to the traditional means of receiving news. The success of providing news through Internet would depend upon how these problems are tackled in the coming years.

1.2 User Categories

In the traditional study of human-computer interaction (HCI), different factors are used to categorize the users [Mayhew92]. Based on the user's experience, they are classified into naive users or expert users; based on the regularity of use they are classified into casual users or regular users. They are also categorized based on other factors, such as educational level, computer literacy, task complexity, the importance of the task, the primary training the users have received, the inherent motivation to do the task, etc. In the case of electronic news access, the users are quite varied. In this case, we can include other dimensions for classifications: the age of the user, the preferred medium for receiving news (newspaper, radio, TV, Internet, word of mouth), the regularity in accessing the news, whether the user is stationary or mobile while receiving the news, the physical handicap, if any. In accessing the electronic news, we find users of all categories and hence the user interface to a system that provides electronic news should have the ability for adapting to the individual user needs. Such an adaptation may be provided at the time of initial registration of the user into the system, or in an ongoing basis as an *adaptive user interface.*

Every news medium has its own way of organizing the news into categories so that the users can benefit in their browsing and selections. A typical daily newspaper divides the news items into the following categories:

- Headlines
- Local News
- National News
- World News
- Editorial Columns
- Weekly Reviews
- Letters to the Editor
- Business
- Sports
- Life Style
- Entertainment
- Classified Ads
- Cartoons
- Front page
- Comics

Different classes of users are interested in different categories of news. Their interests could be self-driven or need-driven. Moreover, the users' interests may vary from time to time. The user interface, whether it is adaptive or not, should be capable of accommodating such wide variations found from user to user, or variations found over time even within a single user.

1.3 User Needs

Accessing the current news is a professional need for some people whereas for others it satisfies a social need or a curiosity. At the beginning of this thesis work, we conducted a small sample survey among the students, faculty and staff of the Department of Computer Science to examine the types of user needs. A questionnaire shown in Appendix I was circulated and we received about 15 responses from various categories of users. Although this survey was not carried out to provide statistical significance, it gave us an idea of the types of user needs. The summary is as follows:

- Those who receive news in the mornings use the radio or the daily newspaper as the source for news. In the evening, people use the TV as an additional medium.
- More than half of the people spend less than 30 minutes per day in getting their news. About a quarter spend between 30 and 60 minutes on their news.
- Front page, national and international news are the sections most people read/view regularly.
- Several people get their news while doing other activities (hands-free and eyes-free mode)
- People would prefer a personalized news delivery system for the following reasons: ability to look for specific news of interest, easy access to older news, possibility of selecting between condensed and elaborate news versions, round the clock availability.

Further interviews with the people who responded to our survey gave more insights. Those findings can be summarized in the form of following scenarios (the user is referred to as U):

Scenario 1 After waking up, U listens to the radio news about weather, headlines and traffic.

Scenario 2 U continues to listen to the radio while in his bathroom.

Scenario 3 Sitting at the dining table, U reads the daily newspaper while eating.

Scenario 4 U is on his way to work (walking, taking a bus, or driving) and continues catching-up on the news (either radio or newspaper).

Scenario 5 During his break at work, U is watching the stock market and checking out how his portfolio is doing.

Scenario 6 In a lunch meeting, U hears other people talking about a crisis which U missed to read about.

Scenario 7 On his way back home, U reads some local news.

Scenario 8 After supper, U watches TV news for a while, but he is unable to skip through some news items and advertisements.

Scenario 9 Occasionally, before going to sleep, U catches up with important international news.

1.4 A Model Based Approach for User Interaction

A model is an abstracted representation of some real world entity, and in this abstraction we keep only those aspects of the entity which are relevant to the study undertaken. In accessing electronic news, we identify the following entities to be essential for our research: A *user* is an entity who accesses news about some *world* or *domain*. The user interactions take place between the user and a software system through the use of an interaction *language*. Normally, such interactions take place in the form of a conversation or a *dialog*. The input device used by the user could be the conventional keyboard or *speech*. When speech input is used, the user's hands and eyes are free and he can simultaneously engage in multiple activities.

Thus, we have identified five entities or five models for this research: user model describing the user's characteristics, domain or world model describing the domain about which news is provided, language model, dialog model describing the relevant

aspects of the dialog that takes place between the user and the system, and speech-recognition-error model that captures the types of errors occurring in speech input. This error model is taken into account because the speech recognizers are not as good as the human understanding of speech input and correct understanding of the user's utterances is essential in a dialog. We use the two terms 'utterance' and 'user interaction' interchangeably, although one might like to distinguish the term utterance as what comes out of the user's mind and the term user interaction as what goes into the computer as per the utterance. The models mentioned above are used in finding a solution for an important problem in user interactions, namely, resolving the ambiguity and incompleteness that might possibly be present in the user's utterance. A solution to this issue involves two steps; the first step is recognition of the presence of the problem, and the next step is its resolution.

Models have been used for different purposes. Simulation models are intended to facilitate simulation of the behavior of a system. Analytical models are used for analysis purposes, and mathematical models are used for proving certain properties in a formal manner. Physical models in engineering are used for conducting tests that may possibly destroy the model in the process of testing. Programming models are used for several different purposes, such as to show the proof of the concept, to use as a prototype in usability testing, to study the functional behavior in a time warped manner (accelerated or decelerated physical time). In the research reported in this thesis, the five models are used as the sources of information for reasoning and resolution. Reasoning is involved when we recognize if there is ambiguity or incompleteness that needs to be resolved. During resolution, the models provide the necessary information at a sufficient level of detail.

1.5 Organization of the Thesis

In Chapter 2 of the thesis, we provide an overview of several spoken-language systems that use various forms of discourse management to provide a natural speech interface to the user. The systems that we examine include a telephone-based conversational tool, a yellow-pages information seeking application, and several others. In Chapter 3, we

introduce our own model-based architecture aimed at providing a conversational interface for electronic news delivery. We examine the interactions between models, and get into design details of individual models. In Chapter 4, we describe a Java implementation of the system prototype, and get into implementation details of various system components. Finally, in Chapter 5, we examine the contributions of the thesis and provide directions for future work.

2. An Overview of Related Literature

Natural language interfaces (NLI) allow users to interact with computers using a natural language in a conversational mode [Cole95]. Combined with speech recognition, NLI systems revolutionize the way that people interact with machines. Because such spoken-language systems support human-machine interaction in a natural way that requires no special training, these interfaces make computer-based resources available to many new groups of users such as casual users, telephone users, hands-busy or eyes-busy users. New systems have recently emerged that allow speech-based access to electronic mail, yellow pages, catalogues, movie information, electronic news, and many others [Wyard96]. Spoken-language access allows users to quickly get to the required information, without having to traverse complex prompts and menus.

The main goal of such systems is to facilitate a robust human-machine dialog [Brennan95]. However, in order for speech to be recognized perfectly, the recognizer needs to disambiguate what was said. In the context of this thesis, we will concentrate on the conversational aspects of such a system. The question of input modality (be it typed input or speech input) remains an open issue.

2.1 Need for Intelligent Components in NLI Systems

Interactive voice response systems have been available for a long time over the telephone network [Wyard96]. They were typically restricted to interactive TouchTone input from the user, with the system providing voice responses. While being suitable for simple applications, complex services are difficult to use due to complicated menu structures. Information could only be provided one at a time, with a "prompt and response" type of dialog guiding the interaction. This could result in the user becoming lost in the navigation, or obtaining the wrong information. In addition, experienced

users would get irritated by the unnecessarily large number of responses, even though they knew exactly where they were and what they wanted from the system. By moving from menu-based input to speech-based interaction, users would be able to express their needs directly and avoid the time-consuming navigation through menus. In addition, such an approach would allow users to take control of the interaction, thus giving them confidence in the system.

Unfortunately, the design of spoken-language systems is a relatively new field and is not as straightforward as it seems. To be truly successful, spoken language systems must be designed to accommodate the unique features of human spoken language, which should address the following issues [Oviatt96]:

(a) disfluencies (spontaneous self-corrections that interrupt the smooth flow of a coherent sentence)
(b) run-on constructions
(c) prosodic and nonverbal language modulation (fluctuations in stress and intonation across spoken words)
(d) errors and repairs
(e) confirmation requests and feedback
(f) dialog control and turn-taking (between the user and the machine)

Dealing with these important issues requires proper modeling of human speech as an input modality. These are not the only issues in the processing of spoken language – other important and challenging problems also arise. They include the following:

(1) meaning extraction – the extraction of information from the user utterances
(2) dialog management – the coordination of dialog between the user and system, as well as the final interpretation of the user utterances in the context of past utterances
(3) response generation – the generation of output as a result of a user query

Efficiently dealing with these issues (a)-(f) and (1)-(3) requires the assistance of intelligent components of the SLS and the knowledge of users. As a result, all systems described in this chapter follow a user-centered modeling approach to provide a natural speech interface to the user.

The following sections examine three different families of spoken-language systems that have been recently developed. These systems are SpeechActs [Yankelovich95], a telephone-based conversational system, PLUS, a "yellow pages" information seeking dialog application [Prince95], and a set of systems under development at BT Laboratories [Wyard96].

2.2 Case Study 1: SpeechActs

SpeechActs is an experimental conversational speech system that has been developed as the result of joint efforts between Sun Microsystems and the MIT Media Lab. The set of current SpeechActs telephone-based applications targets business travelers, allowing them to read their email, retrieve stock quotes, look up calendar entries, set up appointments, ask for time around the world, and convert currency amounts. The sample dialog given below captures the essence of a SpeechActs conversation.

Dialog 1: Sample SpeechActs interaction

SpeechActs:	Welcome to SpeechActs. Would you like mail, calendar ...
User:	I'd like mail please.
SpeechActs:	You have 47 old messages and 78 new messages; 14 are high priority. The first new message is from Paul Martin regarding "Sailing this weekend".
User:	Skip that one.
SpeechActs:	Message 2 is from Nicole Yankelovich regarding "Change in itinerary"
User:	Let me hear it.
SpeechActs:	"Ric, can you meet me in Paris on the first Friday after New Year's instead of in Brussels next week?"
User:	Switch to Calendar.
SpeechActs:	Switching to Calendar. Today you have "Lunch with visitors".
User:	What do I have the Friday after New Year's?
SpeechActs:	On Friday, January 5th, you have no appointments.

User:	How about the day before that?
SpeechActs:	On Thursday, January 4th, you have "Remember to call florist".
User:	What does Sandy have on Wednesday?
SpeechActs:	On Wednesday, December 11th, at 11:00 Sandy has "Staff Meeting".
User:	No, I meant the Wednesday after New Year's.
SpeechActs:	Sorry. On Wednesday, January 3rd, Sandy has "Peter's Birthday".
User:	What's on Tom's calendar the next day?
SpeechActs:	Do you mean Tom Ricardo or Tom Jones?
User:	I mean Tom Jones.
SpeechActs:	On Thursday, January 4th, at 9:00 Tom Jones has "Meet with architect". From 3:00 to 4:00 he has "Performance review meeting".
User:	Switch back to Mail.
SpeechActs:	Switching back to Mail. You left off with message 2 from Nicole Yankelovich.
User:	Send a message to Tom.
SpeechActs:	Preparing message to Tom Jones ...

This example illustrates the natural conversation flow between the user and system. it is important to note how SpeechActs allows the user to correct only a portion of the query when the system selects the wrong Wednesday. Other things worth mentioning are the extensive use of relative dates by the Calendar application, as well as the request by the system for disambiguation of the reference to Tom.

2.2.1 Design Overview

SpeechActs consists of an audio server, a natural language processor, a discourse manager, and a text-to-speech manager.

The speech recognizer recognizes only the words contained in the relevant lexicon, and once it decides that the user has completed an utterance, it sends a list of recognized words to the natural language processor, called Swiftus.

Swiftus parses the words and uses a grammar written by the application developer to return a set of feature-value pairs that encode the semantic content of the utterance. For example, a calendar application might require the following set of pairs:

```
USERID=pmartin
DATE=6 January 1996
ACTION=appointment-lookup
```

As the application processes these pairs, it might ask the discourse manager for help. The discourse manager maintains a stack of information for the current conversation, including data that helps disambiguate partial information and resolve references that use pronouns.

2.2.2 Lexicon

The lexicon used by SpeechActs is a vocabulary containing all the words that are needed for the current application. Each entry is a word sense, and contains a set of feature-value pairs that define such things as its part of the speech (noun, verb, etc.), its root form ("was" is a root form of "to be"), and semantic features ("shirt" means "clothing"). In addition, there may be feature-value pairs for defining synonyms (in order to cut down on the size of the vocabulary) and irregular forms. For example, an entry for the word "show" might look as follows:

```
(show
  ( (category verb)(root show)
    (semantics display)
    (irregular (past-participle shown)))
```

This entry would allow the lexicon loader to produce the derivative forms "shows", "showed", "shown" automatically.

2.2.3 Grammar

Today's recognizers are required to have grammars that have a rule for each possible utterance that the user can say. Since SpeechActs does not rely on one particular type of recognizer, an intermediate unified grammar form was introduced that could be compiled into a recognizer-specific grammar.

Each unified grammar rule consists of a pattern followed by a set of augmentations. Augmentations can be one of the following:

(1) head declarations – a way of passing groups of feature-value pairs

(2) tests – a way of further constraining the match

(3) actions – a way of passing feature-value pairs out of the parsed rule

The following example illustrates a unified grammar rule that matches utterances of the type "What is on Nicole's calendar?".

```
{CalendarQuestion :=

          #PATTERN
   "what" root=be ("in" | "on")
   namePossessive sem=calendar;

          # AUGMENTATIONS
          # Head declaration
   head namePossessive;

          # Tests
   be.past-participle != t;
   be.ing-form != t;

          # Actions
   action := `lookup; }
```

Each word in the pattern matches a word of the utterance. The first augmentation sets the head of the rule to be the results of the `namePossessive` rule. The tests ensure that the verb is not of the form "being" or "been". Finally, the action augmentation adds a feature "action" with the value of "lookup".

2.2.4 Discourse Management

Without a discourse manager, a SpeechActs application would understand the user only if she fully specified a command every time she spoke ("Show the calendar for Nicole Yankelovich on January 3, 1996"). Thus, discourse management is absolutely necessary to support more natural speech.

SpeechActs' discourse manager keeps track of what has been previously said by using the theories of discourse-segment pushing and popping [Yankelovich95]. The discourse is tracked at the application level, as well as across multiple applications.

The application-level discourse is maintained as a stack of data structures. When an application enters a new dialog, it informs the discourse manager, which pushes a new structure for the dialog on the discourse stack. When the discourse manager is told that the current dialog has been resolved, it is popped off the stack, exposing the underlying dialog. Consider the following example dialog [Yankelovich95]:

Dialog 2: SpeechActs discourse management

SpeechActs:	Please state a currency which you'd like conversion to.
User:	What's the rate for the franc?
SpeechActs:	Do you mean Belgian, French, or Swiss?
User:	Swiss.
SpeechActs:	There are 1.15 Swiss francs in one dollar or 0.87 dollars in one Swiss franc.

Here, the currency conversion application does not have sufficient information to reply to the user's query, so the discourse manager is asked for disambiguation. If the nature of the franc was established in the previous context, the discourse manager would have inserted the needed information. However, in the example case, the user is asked the question "Do you mean Belgian, French, or Swiss?", and a new discourse structure is pushed into the discourse stack. The structure is popped off the stack when the ambiguity is resolved (as in the example case), or if the user cancels the activity.

In order to keep track of the discourse across multiple applications, SpeechActs maintains a common stack of referenced items. As a particular concept is being discussed, the application can push it on top of the stack. When an application needs to resolve a reference, it asks the discourse manager to search the context stack for the most recent entry of a particular type. Consider the following dialog, a continuation of the preceding example:

Dialog 3: Discourse management across applications

User:	How about 1 franc in Belgian francs?
SpeechActs:	There are 25.58 Belgian francs in 1 Swiss franc.
User:	How much is 57 dollars there?
SpeechActs:	Do you mean Belgium or Switzerland?
User:	Belgium.
SpeechActs:	There are 1,450.65 Belgian francs in 57 U.S. dollars.

When the user asks for a conversion between Swiss and Belgian francs, both Switzerland and Belgium are placed as locations on the top of the stack. But when the user asks for the rate "there", the reference is ambiguous, and the application asks the discourse manager to search the stack for a geographic location. Both Switzerland and Belgium are returned, so the ambiguity remains. At this point, the disambiguation discourse is entered, and the choices are more limited. Once the user specifies that he is interested in Belgium, this location is pushed into the context stack.

2.2.5 Challenges and Limitations

As impressive as SpeechActs is, some issues remain to be addressed to be able to convincingly simulate a natural human conversation. The more important issues include pacing, error recovery, and definition of application boundaries.

Pacing is not an obvious challenge. Humans attach a great deal of importance to pauses in a conversation, yet current spoken-language systems produce pauses (due to

processing delays) that are inappropriate by human standards. The resulting gaps in conversation can be incorrectly interpreted by users as recognition problems, prompting them to restate their queries. The use of audio cues to fill processing delays can remedy the problem.

Recognizer errors are another important issue in SpeechActs. It is difficult to make a system robust in the presence of such errors, and more work is required in this direction. Ideally, the user should be able to make full-replacement corrections ("I said what time it is"), partial-replacement corrections ("No, I meant this Friday"), elimination of options ("No, not that one"), undoing of state-changing mistakes ("Oops!"), and probing the system state ("Where am I?").

Finally, another issue in SpeechActs is letting users know of an application's functional boundaries, which are hidden behind the speech interface. Since speech is a slow output channel, extensive spoken help is not always an option. A possible solution to the problem is to establish a common ground with the user that suggests possible next utterances. However, this approach falls short of letting them know the range of legal utterances possible at a particular moment.

2.3 Case Study 2: ESPRIT II/PLUS

The ESPRIT II/PLUS (Pragmatics-based Language Understanding System) [Prince95] is a European Research project whose goal is to design a yellow pages dialog application. The user utterances are in a written natural language, and are assumed to be constraint-free in both form and content. There are three main capability requirements for the system:

- understand the content of an utterance in language
- react properly to the current utterance content while maintaining topical coherence
- behave adequately when topical coherence is broken by the user or due to any disturbance

The system consists of several knowledge models dealing with various aspects of dialog handling:

(i) world knowledge

(ii) application knowledge

(iii) conversation knowledge

(iv) topic knowledge

(v) language knowledge

All these models cooperate by using a blackboard-based architecture. This architecture allows for different models to interact without redundancy or contradiction.

2.3.1 Dialog Modeling

The system is composed of two parts (see Figure 1): a natural language engine (NLE) and a dialog manager (DM). The NLE itself consists of two modules – one is used to parse the user's utterances and the other to create system responses.

The DM, which will be focused on, consists of three major components:

1. the *Cognitive Analyzer* (CA), which extracts the user's goal from the semantic representation of a user 's utterance
2. the *Goal Formulator* (GF), which determines the best system goal as a reaction to the current utterance. This module is the one to use the knowledge sources and blackboard memory the most
3. the *Response Planner* (RP), which creates the semantic representation of system utterances

The data flow in the dialog manager proceeds as follows:

- the cognitive analyzer (CA) receives the semantic representation of the speech act and changes it into intention and belief in the user's model according to the current beliefs state, and communicative goal and user's plan development according to the dialog structure.

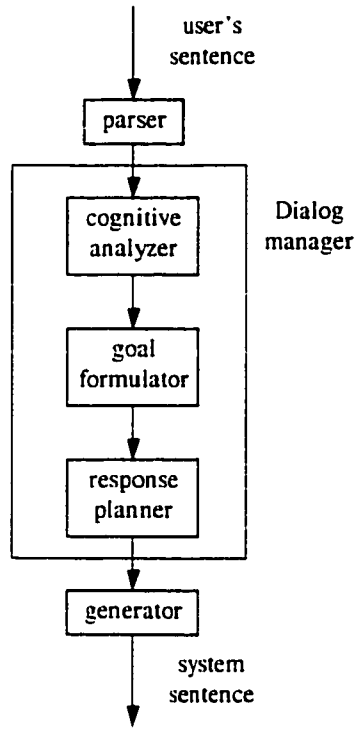


Figure 1: PLUS system architecture

- the goal formulator (GF) receives the complete set of CA outputs, and generates the system communicative goal to answer to the user in the best way according to its ultimate goal.
- the response planner (RP) receives the system communicative goals and creates a semantic representation according to the current user's beliefs and the dialog state.

2.3.2 Dialog Model Knowledge Sources

Every dialog in natural language is based on materializing communication between two or more partners. This materializing results in a *speech act*. Uttering is a process producing a statement in language and assigning a structural position to this statement in the dialog. This statement has the following three properties:

- a propositional content (of the statement)
- the speaker's intentions (derivable from the statement)
- a context of appearance (of the statement)

For example, the utterance "I need a car to go to the airport" could be seen as expressing the following elements:

Propositional content:	Person(I) & need(I,x) & Car(x) & go-to(I,y) & Airport(y)
Intention:	Want(I,x) & Car(x) & Want(I,z) & is-a(z, go-to) & go-to(I,y) & Airport(y)
Context of appearance:	Speaker(I) Utterance is a request The speaker wants the address and phone number of professional(s) who could provide the object

World Model. The world model contains the necessary general concepts to understand properly every task-oriented interaction. For example, such terms as "trip", "car", "action" are general knowledge. They are implemented as a hierarchy where each node is a class containing properties and rules. These elements of general knowledge model common sense. They allow the system to make inferences to update other pragmatic bases.

Application Model. The application model is there to capture specific application knowledge, which includes a database model (defining the structure and content of the application base), an interface model (defining how to find information stored in the database), and a concept-matching model (defining the structure of information available in the database).

User Model. The user model records the wishes or intentions of the user. It also models the user's weak beliefs, the user's knowledge, and strong beliefs.

Topics Model. The topics model takes an important part in determining dialog themes related to the application. It helps to precise the structural and intentional connections between utterances.

Dialog Structure Model. The dialog structure model is there to build a current dialog structure by using a dialog grammar.

Semantic Representation Model. The semantic representation model contains objects defined in the world or application models, or used by speakers in their utterances. It also contains the syntactic and semantic representations themselves and the kinds of speech acts used. This facilitates the management of the user's model.

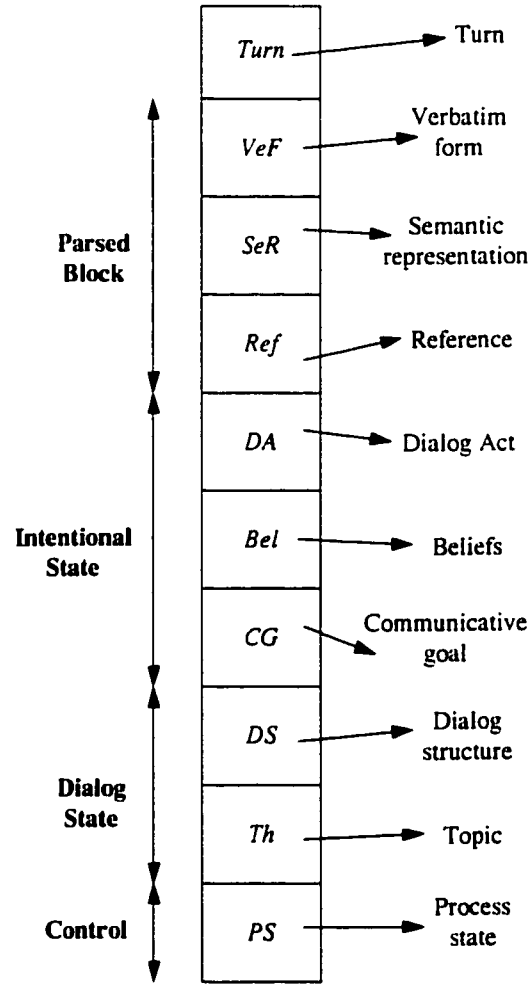


Figure 2: Representation of a DH record

2.3.3 Dialog History

The design for dialog history has several goals:

- to synchronize the different conclusions of the models about the same utterance
- to access the different states of the different models without implementing complex research algorithms in each of the models log files
- to retrace the distinct branches of alternative interpretations that may occur when difficulties arise

The dialog history is implemented as a dynamic database accessible from every knowledge base and process in the system. Every module is able to read and write into it, but no operation of deletion is allowed. The dialog history consists of dialog history records (DHR), each being a bundle of pointers to the current state of every model involved in utterance understanding and generation (see Figure 2).

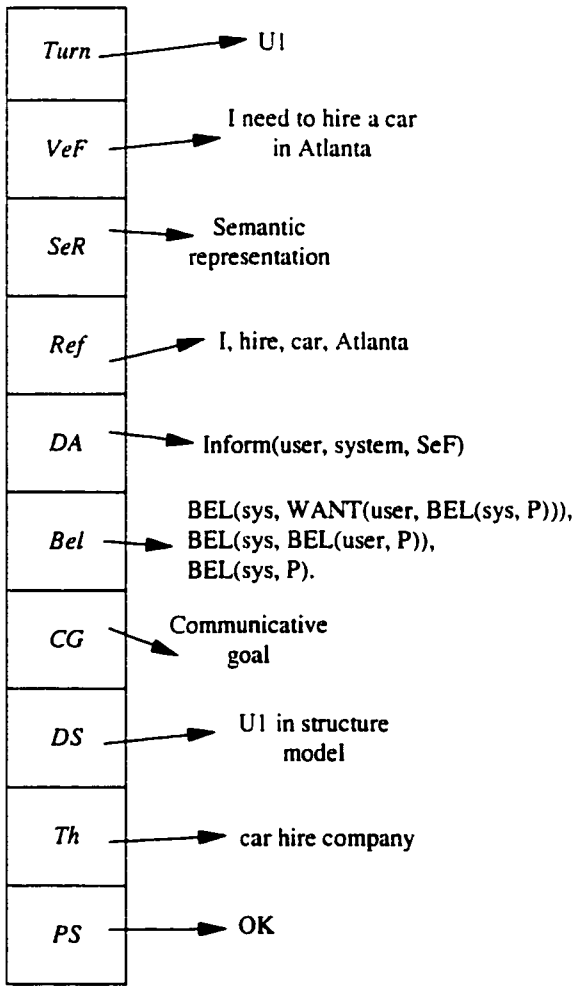


Figure 3: Instantiation of a DH record

The content of the DHR is aimed at designating:

- a structure of the dialogic event itself
- the set of the cooperating model states that results from these actions
- an event valuation in term of processing issue or strategic marks to preserve in case of partial failure

Consider the DH record corresponding to the sentence "I need to hire a car in Atlanta" (see Figure 3). This example illustrates how a DH record is filled when parsing a user utterance.

By putting together both semantic content and management aspects, the dialog history is able to account for the true communicative function of a dialog act. A partially understandable utterance and a completely understood utterance have a very different role from the communications point of view. The partial utterance will generate strategies from the control module, modify the dialog structure, and continue with other recovery experiments. All the computational effort cannot be forgotten and excluded from the system. Because the models cannot keep track of the aborted paths, colliding exchanges, and partial features, the use of such a dialog history presents to be a necessity.

2.3.4 Challenges and Limitations

The presented PLUS system is able to process ill-formed utterances and misspellings. One of the shortcomings of the system is its inability to process several communicative goals in one utterance, even though this feature occurs quite frequently in human speech. In order to deal with this problem, more sophisticated planning needs to be devised so that proper multiple goal resolution can be ensured.

2.4 Case Study 3: BTL Spoken-Language Systems

British Telecom Laboratories are involved in the development of several spoken-language systems to provide a key competitive advantage to its customers [Wyard96]. Multimodal spoken-language systems, which combine several modes of input and output, allow far greater freedom of expression to users who, as a result, would feel more comfortable and less as though they were "talking to a computer".

The goal of such systems is to permit users to occasionally access a particular service without taking the time to learn the language that the machine understands. The systems under development include a multimodal SLS for access to the BT Business Catalogue, a speech system for remote email access, and a system for accessing information about films.

2.4.1 System Overview

Even the simplest form of SLS requires the following major components:

- speech recognition
- meaning extraction
- database query
- dialog manager
- response generation
- speech output

In the simplest form, processing consists of a linear sequence of calls to each component. However, this approach is often insufficiently flexible. It is better for the dialog manager to have more control over other components. This allows the dialog manager to act in an intelligent manner, coordinating the other components and combining their outputs in a nonlinear manner (Figure 4).

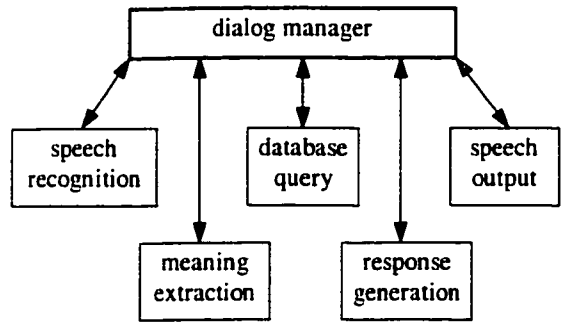


Figure 4: Central role of the dialog manager in SLS

2.4.2 Components of a Spoken-Language System

Speech Recognition. The speech recognizer is usually described as a component that converts a speech utterance into a set of words. Recognizers use language models (LMs) to hold the information about the language being recognized. Speech recognition is not yet accurate enough, and in order to get a working system it is desirable to keep language models relatively small, and give the recognizer as much help as possible about what the user is likely to say next. There are several possible options when it comes to loading the language model into the recognizer.

The LM can be loaded once at the start of the application. This approach, however, is too simplistic and results in only one LM for the whole application. Another possibility is to load different LMs based on the current dialog context. This results in greatly increased recognition rates, since the recognizer makes use of the fact that the relative probabilities of different words change significantly at different points in the dialogue. Finally, the LM can be changed just as the utterance is being decoded. If the user says "I want to go to ...", the recognizer can increase its recognition chances significantly by loading an LM containing the set of possible user destinations. However, this approach is often prohibitive due to high computation costs.

Meaning Extraction. The purpose of the meaning extraction module is to extract and represent the meaning of the user's utterance. There are two major difficulties that need to be overcome during this step: ambiguity and ill-formed input.

Ambiguity may be present as lexical ambiguity ("saw" can be either a noun or a verb) or sense ambiguity ("bank" can be a shore, or a place to store money). Ill-formed input can be the result of misrecognition, or the user making an ungrammatical utterance.

Assuming that these two problems can be overcome, a suitable meaning representation has to be chosen. Several issues need to be considered:

- suitability for further stages of processing
- explicitness versus conciseness
- extendibility

One possible way of representing meaning is to use an extended logical form (ELF). This representation has three fields corresponding to three types of information present in the utterance. An example ELF, representing the utterance "Read the message from Mike" is as follows:

Type:	read(A)
Expectations:	salient(A), singular(A)
Entities and relationships:	message(A), named(B, Mike), from(A, B)

Dialog Manager. The dialog manager coordinates all the activities of the SLS. The DM also knows about the generic structure of the conversations. As the conversation progresses, it builds a dynamic model of the dialog between the user and system.

Humans use words to refer to things in the real world. Anaphoric references are a specific type of referring expression and are used when a speaker refers to something mentioned earlier. For example, consider the following exchange:

Dialog 4: Anaphoric references

User:	Are there any messages from Peter?
System:	You have two messages from Peter Wyard.
User:	Read his second one.

In this dialog, the user's utterance contains two anaphoric references: "his" and "second one". The meaning extracted from the user utterance can be represented as follows:

Extended Logical Form (ELF):

read(A),
[salient(A), singular(A)],
[ord(A,2), named(B,C), gender(B,masculine), from(A,B)]

The reference resolution process fills in missing information in the ELF. In order to do so, it goes through the preceding conversation in an attempt to find something that can be read, and a masculine person. Thus, the meaning extraction module (ME) ELF output is converted into a resolved extended logical form (RELF) by adding the missing information message(A) and name(B, Peter):

Resolved Extended Logical Form (RELF):

read(A),
[salient(A), singular(A)],
[message(A), from(A,B), ord(A,2),
name(B,Peter), gender(B,masculine)]

Another important aspect of conversational speech is the use of ellipses. Ellipses occur when things that could be determined from what has been previously said are left out of an utterance:

Dialog 5: Ellipses

User:	Are there any messages from Peter?
System:	You have two messages from Peter Wyard.
User:	And David?

The user has left out "Do I have any messages from ..." and has simply said "And David?". The ME component will produce the form:

ELF:

A,

[],
[name(B,David)]

and the RELF will include the missing information which was found in the previous logical form:

RELF:
list(A),
[],
[message(A), from(A,B), name(B,David), gender(B,masculine)]

Database Query. When the DM has processed a user utterance, the resulting query will be passed to the database query component. This component is responsible for converting the DM query into a language that can be used for information retrieval.

A number of difficulties may be encountered in this stage. The DQ component may need to consult general and domain-specific knowledge to satisfy the query. For example, if the user asked about "reptiles" but the database contained information about "snakes", the DQ could consult a taxonomy of reptiles to realize that there is a link between the query and available data.

The DQ component may also need to optimize the query to speed up its execution. In order to do so, an additional source of knowledge may be necessary.

Response Generation. The generation of responses as a result of user utterances may be a complex process. Text generation can consist of canned-text sentences, or a more flexible approach may be taken to generate the most appropriate responses as and when necessary. Multimodal systems can be enhanced by incorporating graphics into system responses, and text-to-speech engines may be used for generation of voice responses.

2.4.3 Challenges and Limitations

A lot of work needs to be done in order to convert the demonstration systems introduced earlier into usable applications. Namely, more robust speech recognition is needed that allows for speaker adaptation and spontaneous recognition. Further studies are needed to determine the users' language patterns. By incorporating into spoken-language systems what real users actually say, a more natural interaction can be achieved.

2.5 Conclusion

This chapter has introduced three sample spoken-language systems that have different architectures and designs. The approach followed in our project consists of five tightly coupled models, whose origins can be traced back to some of these systems:

- ***language model*** – provides a restriction on possible query types and, whenever it can, provides hints to the dialog model for resolution of ambiguity and incompleteness
- ***user model*** – stores user preferences and keeps track of the user state
- ***world model*** – stores common-world knowledge about geographic information
- ***dialog model*** – in addition to coordinating the interaction among the other four models, keeps the dialog history and resolves ambiguity and incompleteness in user queries
- ***error model*** – provides rudimentary error management in case of erroneous user input

From the three systems described earlier, SpeechActs is closest to our design. SpeechActs utilizes a sophisticated discourse and error model for handling ambiguous, and often erroneous, user input. Like in our system, its language model puts a restriction on the structure of user queries. In contrast with SpeechActs, which operates in a restrictive speech environment, our system uses a display screen for graphical presentation of query results. Not having to deal with the problems of a speech-only environment allows us to do more work in the area of model integration.

ESPRIT/PLUS separates the sources of knowledge into several knowledge models, and utilizes a sophisticated natural-language engine for processing of input queries. As a result, there are no constraints on query types. It uses the speech act theory to obtain the propositional content and the speaker's intentions from the utterance. The system is multimodal, and accepts both typed and spoken input. In contrast to ESPRIT, our system does not focus on natural language understanding, and rather goes in the direction of exploring model integration with the purpose of improving the quality of interaction with the user.

Finally, the systems from BTL also utilize a model-based approach, and use an NLE and speech act theory for extracting meaning from a user utterance. There are no restrictions on query structure, and multimodal input is used. This system is similar to ESPRIT in terms of functionality, and its natural language understanding module is the most prominent feature.

3. Proposed Architecture for News Access

Human dialogs are often full of ambiguous, incomplete, and ill-formed utterances. Ambiguity may exist as lexical ambiguity (words that have more than one meaning, e.g. "plane" can mean "aircraft" or "level surface"), structural ambiguity (e.g. "I saw her duck" can mean "I saw the duck belonging to her" or "I saw her when she ducked"), and may also be present in the relationships between phrases. In addition, dialogs often rely on what has been previously said, which makes understanding impossible without discourse management.

The system architecture proposed below is aimed to provide a conversational interface for electronic news delivery. The system is user-centered and allows "naïve" users to easily retrieve conventional news items (newspaper articles, television and radio broadcasts) at their convenience. The possibility of access to time-dependent media at the user's convenience and discretion makes the electronic news delivery system especially attractive.

3.1 Architecture

To address the difficulties of human language and, at the same time, make the user-system interaction more natural, the proposed system architecture is composed of 5 closely integrated models. A specialized *language model* (LM) is responsible for the processing of individual utterances, with vocabulary and grammar restrictions. A domain-independent *dialog model* (DM) is there to provide discourse management. A *user model* (UM), based on task graphs and a user profile, keeps track of the user's goals and preferences. A domain-oriented *world model* (WM) allows the resolution of references to knowledge pertinent to the subject of study (news access, in our case). Finally, an *error model* (EM) handles erroneous user input.

The models interact with each other through the architecture described in Figure 5. The central module of the software architecture is the dialog model, which receives all relevant query information from the language model. The output from the dialog model is a query Q_s with ambiguities resolved and incompleteness filled, such that Q_s can be used by a DBMS for search and retrieval.

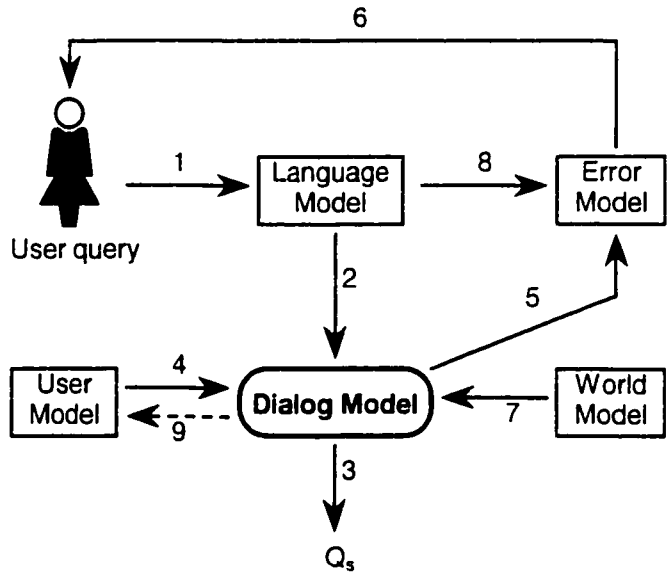


Figure 5: System architecture

To help illustrate the interactions between the five models, consider what happens during the processing of each of the following queries:

(a) *Get me the headlines from CNN of October 11th*

This is an ideal case – the query is free of incompleteness and ambiguities. The LM hands the parsed query to the DM, which is immediately able to produce Q_s. Looking at Figure 5, the corresponding data flow is 1-2-3.

(b) *Is there anything about the flood in China?*

The query is ambiguous – the DM does not know which media types to return, and needs to consult the UM for disambiguation. The data flow is 1-2-4-3.

(c) *Anything in headlines?*

The DM does not know which news sources should be consulted to answer the query, and asks the EM to issue a request for clarification from the user (*Do you mean CNN, BBC, or the Gazette?*). The data flow is 1-2-5-6-1-2-3.

(d) *Anything from Canada?*

To ensure better recall, the query needs to be modified – in addition to searching for news from Canada, it is necessary to look at a finer level of granularity (in this case, looking for news from the different provinces and cities of Canada). The WM supplies the needed information, and the data flow is 1-2-7-3.

(e) *How's the weather tomorrow?*

The query is of an unknown type, and the LM immediately passes it to the EM, which asks the user to reformulate the query. The data flow is 1-8-6.

(f) *Set my preferred video channel to CNN*

This is a preference statement by the user, who defines his favorite news provider for video data. The value of the property is stored in the UM, and the associated data flow is 1-2-9.

3.2 Language Model

Consider the sample dialog in an electronic news delivery system shown below:

Dialog 6:

new session starts

User:	*Anything new in politics?*	
System:	CNN, CBC, or Gazette?	
User:	*Headlines from Gazette*	– clarification provided by user
System:	(shows a set of headlines related to politics)	
User:	*What about the election debates?*	– additional focus
System:	(displays the requested information)	
User:	*Another one*	
System:	(displays the next news item)	
User:	*Anything from CBC on that?*	– with reference to context
System:	(shows a video clip from CBC related to the previous news item)	

the context of the dialog changes

User:	*Now, get me the latest sports results*	– switching focus
System:	(displays sports headlines from TSN, the user's preferred sports channel)	
User:	*Get me more details on Mario's resignation*	
System:	(displays latest information on coach's resignation)	

a sub-dialog is introduced

User:	*Set the preferred sports channel to ESPN*	– switching modes
System:	(visual confirmation that ESPN has been set as preferred sports channel)	
User:	*Set preferred media type to video*	
System:	(visual confirmation that audio is the preferred media type)	
User:	*No, I said video!*	
System:	(visual confirmation that video is the preferred media type)	

back to the previous dialog context

User:	*What else is new?*	– relative to what has been seen
System:	(displays headlines for other important news)	

the context of the dialog changes

User:	*Show me the Business section of the Gazette*	
System:	(presents an overview of the business section)	
User:	*What about that IBM merger?*	
System:	(displays the article from the business section)	
User:	*Give me a summary*	– choice of format
System:	(displays the article in condensed form)	
User:	*Follow this news item*	
System:	(visual acknowledgment)	
User:	*Goodbye*	
System:	(visual confirmation)	

end of current session

...

new session is started

User:	*Any developments?*
System:	(displays audio, video, and newspaper items relevant to the IBM merger)
User:	*Show the CNN item*
System:	(plays a video clip from CNN)
User:	*Drop this news item*
System:	(visual confirmation)

the context of the dialog changes

User:	*Is there anything...*

It would be too unnatural for the user to have to specify a complete query at each step of the above interaction. Therefore, to provide naturalness in the conversation between the user and system, we must allow users to employ potentially incomplete and ambiguous statements. A precise definition of incompleteness and ambiguity will be given later in the chapter (Section 3.3).

A generalized NLI approach to query processing is an extremely difficult task, and the approach taken in this thesis work consists of introducing a restricted set of query types that the system would recognize. Any query whose structure is not in this set would not be recognized as valid, and the system would force the user to restate his query.

The definition of proper query types is a critical task, directly influencing the usability of the resulting system. Based on a user survey conducted earlier (the survey results are presented in Appendix I) and further discussions with a selected population sample, we were able to determine the structural and functional requirements of these queries. We arrived at a set of 13 query types, which are shown below:

Query type 1 (sem **show**)[1] *NewsMediaReference* [(sem **story**)]

Get me | yesterday's Gazette

Query type 2 (sem **show**) *NewsTopic* [(sem **in**) *GeographicLocation*]

Anything about | the flood | in | China?

Query type 3 (sem **show**) (sem **headlines**) [(sem **in**) *NewsMediaReference*]

Search for | headlines | from | CNN of three days ago

Query type 4 (sem **show**) *MediaType* [(sem **about**) *NewsTopic*]

Get me | the video

Query type 5 (sem **show**) *NewsCategory* (sem **segment**) [(sem **in**) *NewsMediaReference*]

Show | the business | section | from | the Financial Post of October 15th

Query type 6 (sem **show**) (sem **about**) *NewsTopic* [(sem **in**) (*MediaType* | *NewsMediaReference*)]

Anything | about | the plane crash | from | CNN?

Query type 7 (sem **show**) (sem **new**) (sem **in**) *NewsTopic*

Anything | new | on | peace talks?

Query type 8 (sem **show**) (sem **developments**)

Any | developments?

Query type 9 (sem **show**) (sem **in**) *GeographicLocation*

Anything | from | India?

---

[1] The notation (sem **show**) is used to indicate that we are expecting an expression whose meaning is semantically equivalent to the verb *show*. The definition of query syntax is given in detail in Chapter 4.

Query type 10 (sem **show**) (sem **in**) (*NewsMediaReference* | *MediaType*) (sem **about**) (sem **this**)

Is there anything from yesterday's BBC on this?

Query type 11 (sem **what else**) (sem **new**)

What else is new?

Query type 12 (sem **observe**) (sem **this**) (sem **story**)

Follow this story

Query type 13 (sem **abandon**) (sem **this**) (sem **story**)

Drop this story

Browsing *BrowseCommand*

Another one

Preference (sem **set**) (sem **preferred**) ((sem **media type**) [(sem **to**)] *NewsMediaName* | (sem **media**) [(sem **to**)] *MediaType*)

Set favorite video type to CNN

These query types were defined after studying typical user access patterns. The query type itself carries all the semantic information needed by the dialog model to take appropriate action when a query is received from the language model. Hence, if (or, more precisely, when) access patterns change and new query types are required, they only need to be defined in the LM, without necessitating any changes in the DM's implementation.

Before going further, it is important to address the "new word" problem, which occurs when a user's utterance (either spoken or written) contains words outside of the system's vocabulary. Dealing with this problem usually requires an open-vocabulary system that provides mechanisms for defining new words at run-time. A significant

amount of research has been done in this area ([Asadi91, Young93]). However, the problem falls outside the scope of this thesis and will not be dealt with here. Consequently, our system assumes a fixed vocabulary and any "new words" in user input are detected during parsing and result in LM-generated errors.

3.3 Dialog Model

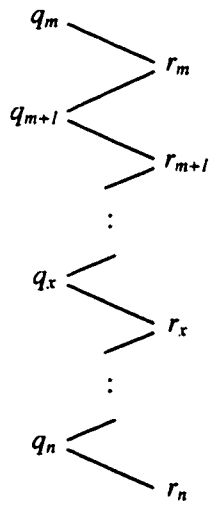
A dialog between the user and system is a sequence of exchanges (q, r), where q is a user query, and r is the system's response (in visual and/or auditory form). In the sequence above, $q \in Q$, where Q is a finite set of query types. A correctly parsed user query q yields a set of fields $F(q)$ (such as *NewsTopic*, *NewsCategory*, etc.) that are defined for each query in Q. The interpretation of q can thus be considered as a two-step process:

Step 1: query type is recognized (mapped to one of the types)

Step 2: $F(q)$ is determined

A system response may also yield a set of fields $R(q)$ that are specifically relevant to the result of the query.

Consider the following exchange between the user and the system:



Let q''_x denote a completely resolved query that can be used to retrieve results for a corresponding user query q_x. For the purposes of this thesis work, the two commonly used terms incompleteness and ambiguity are defined as follows:

Incompleteness: q_x is said to be incomplete if

$$\exists f \in F(q''_x)\ (f \notin F(q_x) \wedge \exists\, i \in [m..x\text{-}1]\ (f \in F(q_i) \vee f \in R(q_i)))$$

Ambiguity: q_x is said to be ambiguous if

$$\exists f \in F(q''_x)\ (f \notin F(q_x) \wedge f \in F(UM))$$

In other words, an incomplete query is one whose meaning cannot be determined without consulting the current dialog history. An example of an incomplete query is *"Anything from CBC on that?"*. Here, the meaning of the query cannot be determined without knowing the news subject selected in the previous query.

An ambiguous query is one whose meaning cannot be determined without consulting the user model (UM). The query *"Now, get me the latest sport results"* is ambiguous. The system knows that the user wants sports results, but does not know what media types it should retrieve, nor what date it should begin searching from. In this particular case, the user model is consulted to determine the user's preferred media type as well as the last date and time of invocation.

3.3.1 Dialog History

In case of incomplete queries, the system must rely on the dialog manager for obtaining the missing information. Therefore, all information extracted from the user queries must be kept in a dialog history. A logical approach for storing such information is a list. Consider the following dialog:

Dialog 7:

①	User:	*Give me the hockey results*
②	System:	(displays the results of the Stanley Cup Playoffs)

③	User:	*Show me the video from the Colorado Avalanche game*
④	System:	(shows the video clip from the hockey game)
⑤	User:	*Give me a summary*
⑥	System:	(displays the game summary)

Assuming that all ambiguities have been resolved by consulting the necessary models, the dialog history after user utterance ③ would look as illustrated in Figure 6.

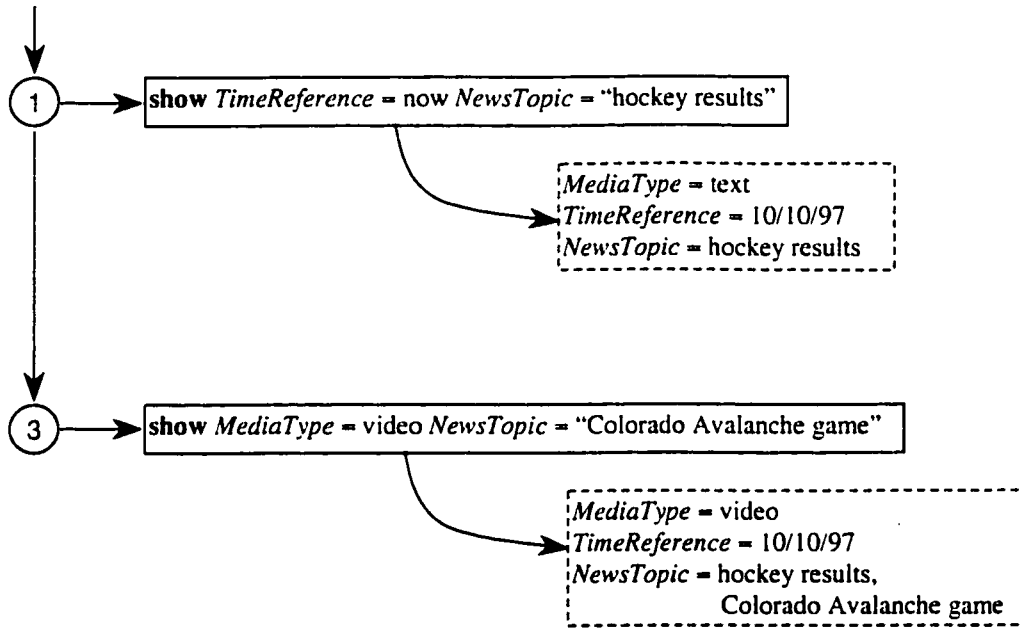


Figure 6: Sample dialog history

The solid boxes represent the original user query, while the dashed boxes represent the user query where incompleteness and ambiguity have been resolved. It is worth noting that in case of utterance ③, the keyword field and time reference field are inherited from the corresponding entry for utterance ①. Based on this dialog history, the corresponding entry for user utterance ⑤ is:

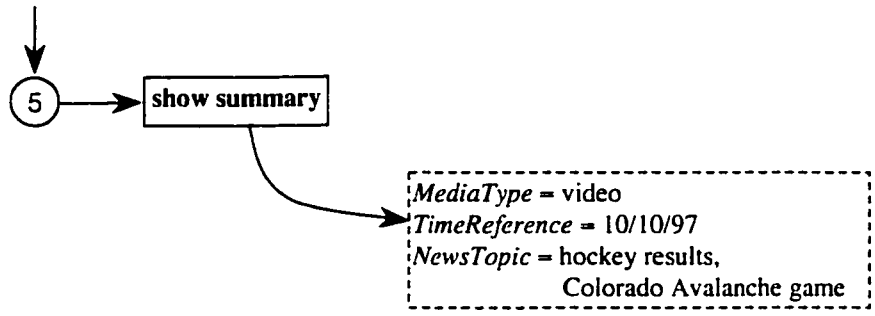


Figure 7: Sample dialog history (cont'd)

A problem arises here that is always present in the processing of natural language. A conversation usually consists of several dialog contexts that are independent of each other (as in Dialog 6). In order to distinguish between these, the system should be able to detect whenever the context of the conversation switches. We deal with this problem by means of the following heuristic:

```
q = Parse(user input)
if (q successfully parsed) {
        if (q is complete) and (Results(q) ≠ ∅) {
                NewContext()
        }
}
```

Whenever a new query is successfully parsed (for example, *Show me the Business section of the Gazette* from Dialog 6), it suffices to look at its structure to determine whether it is incomplete. If no incompleteness is present (i.e. the query does not rely on the dialog history) and if the query produces new results, a new conversation context should be created.

3.4 User Model

The user model is necessary to model the current state of the user, his preferences, and other relevant user-dependent parameters. As a result, the user model consists of both static and dynamic entities, and comprises a collection of data structures.

The dynamic entity specifies the physical state of the user, which can be any of the following:

(1) user sitting in front of the TV

(2) user lying in bed

(3) mobile – within the confines of a home

(4) mobile – outside, with hands and eyes busy

(5) mobile – outside, with hands and eyes partly free

(6) inhibited state to receive info

The static part specifies:

(1) receiving equipment

(2) profile (preferences)

(3) quality requirement

(4) granularity requirement

(5) modes of operation (regular, learning, browsing, etc.)

(6) task graph model

3.4.1 Task Set

The task set is derived from the command words of the user interaction. The set of possible tasks, derived from the analysis of user requirements in the context of providing electronic news, is as follows:

T1. Receive news (one session)

T2. Receive news (multiple sessions, stop & restart)

T3. Select parts & archive

T4. Mark advertisements (store in a file)

T5. Follow a news item (possibly backwards)

T6. Update user profile

T7. Query the system for a specific news item

The news item can be a full news story in its original format, or a set of highlights, headlines, or summaries.

Each task can further have a set of subtasks. For tasks outlined above, we can have:

For T1 and T2:	specify mode (audio, video, text)
	specify type (regular, headlines, summary)
For T3 and T4:	mark a block (or multiple blocks)
	specify archival type (file, remind, follow-up)
For T5:	choose an item
	issue command when ready (navigation)
For T6:	add, delete, modify
	sub-operations for each subtask
For T7:	form-filling type of operation from NL-query

The task name allows us to infer the user's goals at the current level of interaction, which in turn allows the resolution of some types of ambiguities without initiating a new sub-dialog.

The dynamic entity puts a restriction on the set of candidate output modalities, which can further help in ambiguity resolution. For the 6 physical user states outlined earlier, the possible output modalities are shown in Table 1.

Table 1: Possible output modalities

Physical state	Possible output modalities
user sitting in front of the TV	*video, audio, text*
user lying in bed	*video, audio, text*
mobile – within the confines of a home	*video, audio, text*
mobile – outside, with hands and eyes busy	*audio*
mobile – outside, with hands and eyes partly free	*audio*
inhibited state to receive info	*audio*

Thus, the system detects the current physical state of the user in order to determine the set of suitable output modalities before delivering the response[1].

The set of static entities is saved across multiple sessions, and consists of several entries in a table. Some of these entries (user profile, quality and granularity requirements) are directly modifiable by the user, while others (such as receiving equipment) are only accessible by the system. This table is consulted by the system whenever disambiguation is needed.

3.4.2 Profile Management

A user profile allows the user to customize the system environment. It consists of a static table with a set of values for such entries as preferred channels, media types, and output modalities (Table 2). This table initially contains default entries, but preferences can be modified any time the user is interacting with the system, and are saved across multiple sessions.

Table 2: Sample user profile

modality	*video*
video	*BBC*
audio	*CJFM*
news channel	*BBC*
sports channel	*TSN*

3.5 World Model

The world model is a source of knowledge for the entire system (Figure 8). The information stored within can be categorized as either domain-independent (common-sense) or domain-specific (relevant to the news domain). The common-sense knowledge is comprised of:

[1] The current implementation does not yet have the capability to distinguish between the possible user states.

- temporal information
- spatial information
- geographic taxonomy

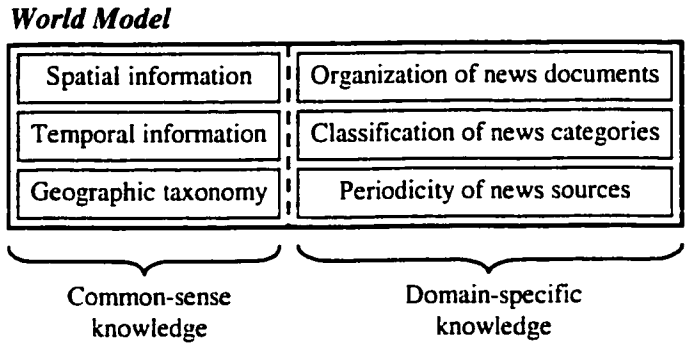


Figure 8: Organization of knowledge in the WM

The domain-specific knowledge consists of:

- organization of news documents
- classification of news categories
- periodicity information for the various news sources

As will be seen in Chapter 4, the system allows the user to use relative time references within queries (Figure 9). Since relative references that go back several weeks or months are inherently vague, the temporal knowledge is used to associate a certain level of uncertainty to each of the different granularities (days, weeks, and months). Those values allow the WM to convert a relative time reference to a closed interval whose endpoints are the specified distance away from the exact target date. The conversion mechanism is examined in detail in Chapter 4.

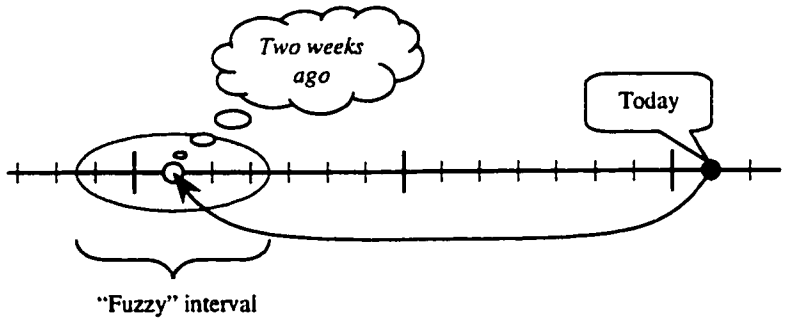


Figure 9: Relative time references

Geographic information is another type of common-sense knowledge stored in the WM. This information is needed to improve recall during geographic searches by looking at finer levels of granularity. Since geographic elements often cannot be uniquely partitioned into distinct classes (for example, mountain ranges and rivers may span several states or provinces), multiple inheritance should be used. An example of such structure is presented in Figure 10. The spatial information comes into play here, as it tracks the proximity relationships between the entries in the geographic taxonomy.

The domain-specific information is supplied by individual news providers. It consists of a forest of trees, where each tree can be

- an organization of news within a particular provider, which allows to break a news source into segments
- a classification of the different news categories, which can be used for expanding searches based on news category
- periodicity information, which allows the system to know which news sources are expected to be available on a particular date

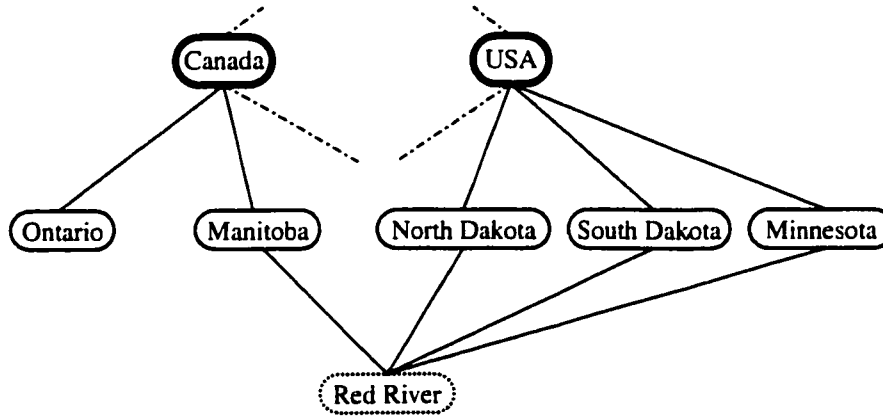


Figure 10: Storage of geographic information

3.6 Error Model

There are many sources of errors in a natural language interface, particularly when speech input is used. Referring to Figure 5, the EM may handle errors from either the language model, or the dialog model.

When the language model is unable to process a user query, it signifies that the query does not fit into any of the predefined types. As such, the EM has no other alternative than to request the user to restate his query. Such request may be either auditory (*I'm sorry?*), or visual (flashing icon).

When the dialog model passes control to the EM, it means that a clarification is required from the user. The kind of clarification required is determined by the DM, and the error model simply interprets the request on behalf of the DM. Once again, the request may be either auditory (*Do you mean CNN or BBC?*), or visual (multiple choices are shown on the screen).

When spoken language is used, another level of complexity is introduced into the EM by the speech recognizer. If we only consider a subset of recognizer errors, namely the rejection errors and substitution errors, one way of handling them is shown in the flowchart of Figure 11.

When there is a chance of substitution errors, the recognizer will return a set of utterance/probability pairs to the language model. If a correct parsing cannot be made, a verification/confirmation dialog could be used, multiple modalities, or a judicious application of verification sub-dialogs based on the grounding requirements.

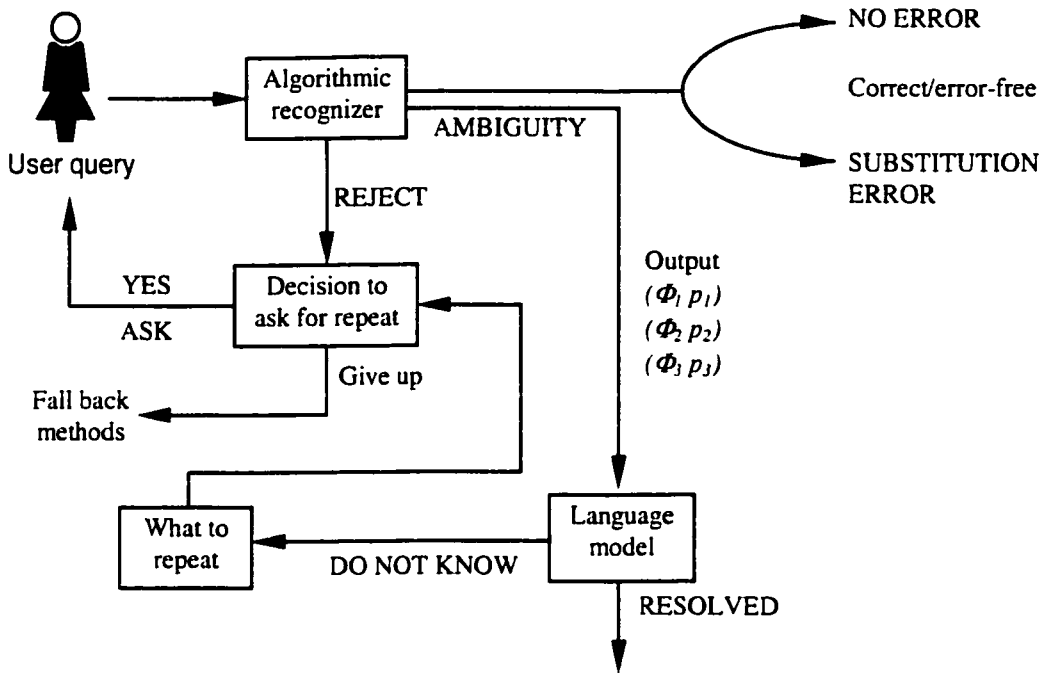


Figure 11: Error handling

4. Implementation and Testing

4.1 Implementation Environment

The system has been implemented using the Java programming language, with most development done on a Sun Ultra-1 running Solaris 2.5. A number of Sun packages has been used during the development:

- JDK 1.1.7 (Java Development Kit)
- Swing1.1beta3 (Java Foundation Classes for the GUI)
- JavaCC-8pre2 (Java Compiler Compiler)
- JMF1.0 (Java Media Framework)

Due to the lack of robust Java integrated development environments (IDEs) for Solaris, an Emacs package for Java has been used instead. The Java Development Environment (JDE) provides a highly configurable Emacs wrapper for command-line Java development tools. The JDE supports syntax coloring, auto indentation, source-level debugging, and automatic code generation. Unfortunately, there is no built-in GUI editor available, which was quite an inconvenience during the design of the visual interface with Swing.

4.2 Software Architecture

The basic architecture of our system can be subdivided into three major components: discourse management, database access, and result presentation. The most challenging part, of course, is the discourse management, and we will concentrate more on the implementation details of this particular component. The database component consists

of a video database backend, providing the retrieval of video clips by matching the user's query against the indexed data. Finally, the presentation component provides the necessary facilities for displaying and navigating through the results of a query. The overall software architecture is shown in Figure 12.

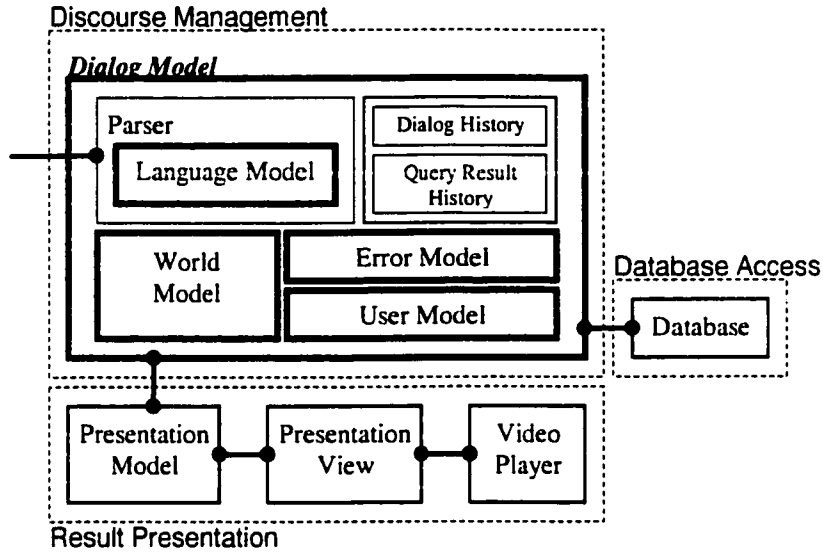


Figure 12: Software architecture

4.3 Module-Level Description

4.3.1 Discourse Management

The discourse management component is responsible for keeping track of the current dialog state between the user and system, resolving ambiguities and incompleteness, and handling errors. In order to achieve this, the discourse management component contains the five models described earlier. Since efficient discourse management requires a tightly-coupled system, the dialog model takes the responsibility of coordinating the interactions between the remaining four models.

4.3.1.1 Parser and Language Model

The queries allowed in our system have a restricted structure. Thus, parsing user input is a fairly straightforward task. If the parser returns a valid production as a result of the parse, the DM proceeds with the query; otherwise, the error model takes over. The JavaCC specification of allowed query types is shown in Figure 13.

```
Query Q() :
{
    Show() (
              // Query type 1
              NewsMediaReference() [ story() ]

            | // Query type 2
              NewsTopic() [ LOOKAHEAD( in() GeographicLocation() )
                            In() GeographicLocation() ]

            | // Query type 3
              headlines() [ in() NewsMediaReference() ]

            | // Query type 4
              MediaType() [ about() NewsTopic() ]

            | // Query type 5
              NewsCategory() segment() [ NewsMediaReference() ]

            | //Query type 6
              about() NewsTopic() [ in() ( MediaType()
                                         | NewsMediaReference() ) ]

            | // Query type 7
              new() in() NewsTopic()

            | // Query type 8
              developments()

            | // Query type 9
              in() GeographicLocation()

            | // Query type 10
              in() ( NewsMediaReference() | MediaType() ) about() this()

            )

  | // Query type 11
    "what else is" new()

  | // Query type 12
    observe() this() story()

  | // Query type 13
    abandon() this() story()

  )
}
```

Figure 13: Allowed query types

For every query successfully parsed, a corresponding Java object ***QueryXX***[1] is created (where XX indicates the query type). This object contains all the information supplied in the query, and can consist of a subset of the following fields:

- *timeReference* – the absolute or relative date referenced in the query
- *newsMediaName* – the name of the news media
- *mediaType* – the type of media desired (audio, video, etc.)
- *event* – the news topic
- *geographicLocation* – the location of the event
- *person* – the person mentioned in the news item
- *newsCategory* – the category of the news item (politics, sports, etc.)

For example, for the query *Is there anything about the plane crash in Ohio?*, the following object is returned:

Query2	
event:	plane crash
geographicLocation:	Ohio

It is then the Dialog Manager's job to resolve the ambiguity and incompleteness present in this query and do further processing.

The `NewsMediaReference()` production used in Figure 13 consists of a reference to a news media name and an optional time reference. This production's expansion is shown in Figure 14. A valid news media reference could thus be *"CNN from two weeks ago"*.

```
void NewsMediaReference() :
{}
{
    TimeReference() NewsMediaName()
  | NewsMediaName() [ Of() TimeReference() ]
}
```

Figure 14: News media references

The optional time reference can be of two types – either absolute or relative. This `TimeReference()` production is shown in Figure 15. In case of absolute references,

---

[1] Every ***QueryXX*** inherits from a generic object ***Query***, which mainly provides accessor methods used by its children.

this consists of a name of the month followed by a number (*October 15*, for example). With relative references, this consists of nouns *today* or *yesterday*, or references of the form ***number*** *{days, weeks, months} ago* (such as *two weeks ago*).

```
void TimeReference() :
{ int lapse, granularity;
  int m, d;
}
{
  (   <today>
      { timeReference = new RelativeDate(0, NewDate.DAY); }

    | <yesterday>
      { timeReference = new RelativeDate(1, NewDate.DAY); }

    | ( (   <number>
            { lapse = Integer.parseInt(token.image); }

          | lapse = NumberLiteral()
        )

        (   <day>
            { granularity = NewDate.DAY; }

          | <week>
            { granularity = NewDate.WEEK; }

          | <month>
            { granularity = NewDate.MONTH; }

        ) "ago"
      )
      { timeReference = new RelativeDate(lapse, granularity); }

    | ( (   "january"   { m = 1;  }
          | "february"  { m = 2;  }
          | "march"     { m = 3;  }
          | "april"     { m = 4;  }
          | "may"       { m = 5;  }
          | "june"      { m = 6;  }
          | "july"      { m = 7;  }
          | "august"    { m = 8;  }
          | "septemer"  { m = 9;  }
          | "october"   { m = 10; }
          | "november"  { m = 11; }
          | "december"  { m = 12; }
        )
        (   <number> { d = Integer.parseInt(token.image); }
          | d = NumberLiteral()
        )
      )
      {  GregorianCalendar c = new GregorianCalendar(1997, m, d);
         timeReference = new AbsoluteDate(c);
      }
  )
}
```

Figure 15: Time references

4.3.1.2 User Model

The user model stores user-specific information (such as preferred media types and news sources, as well as some internal properties). The user can directly modify his preferences through the use of queries illustrated in Figure 16:

```
Query Preference() :
{}
{ set() preferred() (   MediaType() [ "to" ] NewsMediaName()
                      | media() [ "to" ] MediaType() )
  { p = new Preference();
    return p;
  }
}
```

Figure 16: User preferences

Typical user preference settings are of the form

Set favorite video to BBC

or

Set preferred media to audio

These properties are used for the final sorting and presentation of results, and do not influence the actual database search – the presence of a user preference is simply a hint to the presentation component indicating which results should be displayed first.

In addition to user-modifiable properties, there are also UM properties that the user does not directly control. The first of such properties is the date of last invocation, which is used to track the boundary between old news and recent news. This information is useful – and often used by the DM – when answering a query in which the user is interested in recent events.

The UM also keeps a list of news topics being currently followed by the user, where each news topic is stored as a pointer to the corresponding news item in the database. This list, in combination with the date of last invocation, is used by the DM to retrieve new developments in topics the user is interested in.

4.3.1.3 World Model

The world model is a source of domain-independent and domain-specific knowledge for the rest of the system. It also maintains a global clock, which is used for keeping track of the current date in the context of the system. This clock allows all components of the system to have a common time reference.

The WM's common-sense knowledge consists of temporal and geographic information. The temporal knowledge is used to resolve relative time references in user queries (Figure 15). As mentioned in Chapter 3, relative time references that go back several weeks or months tend to be imprecise. For that reason, the WM associates an uncertainty interval to each of the three different granularities (days, weeks, and months). This interval allows it to convert a relative time reference to a closed interval whose endpoints are the specified distance away from the target date (refer to Figure 9 in Chapter 3). In our work, those values are defined as follows:

Table 3: Uncertainty in temporal references

Granularity	Uncertainty
day	± 0 days
week	± 2 days
month	± 7 days

References to *today* and *yesterday* are there for the user's convenience, and are internally treated as *0 days ago* and *1 day ago*, respectively. For example, consider the diagram in Figure 9. A relative time reference *Two weeks ago* can be converted to an interval [2 days before two weeks ago, 2 days after two weeks ago], which spans 5 days. Similarly, the reference *yesterday* can be converted to [0 days before 1 day ago, 0 days after 1 day ago], spanning only one day.

The geographic information is organized in a tree structure, broken down by location granularity (World, Region, Country, Territory, City). All the geographic information was extracted from the indexing data for the video clips, and is therefore representative

only in the context of the project. A small excerpt from the world model is shown in Figure 17.

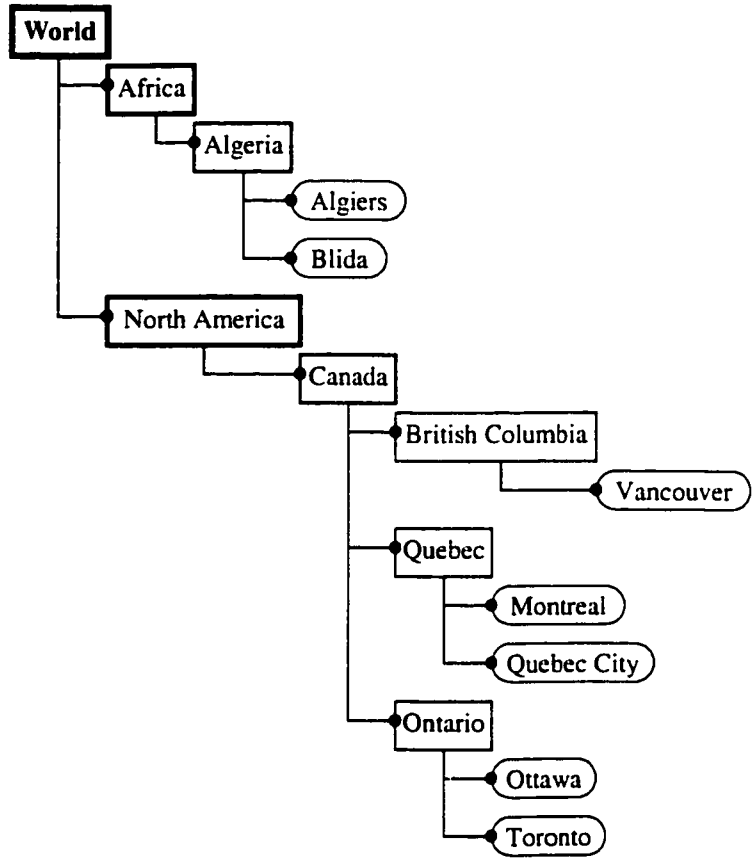


Figure 17: Organization of geographic knowledge

Given the name of a geographic location in this hierarchy, the world model can provide a list of all its immediate children; the ancestor(s) of a node are also available. This capability of the world model is used by the DM for query expansion during geographic searches.

Since the test database is limited to video content only, the only domain-specific knowledge available in the WM is the classification of news categories. It is similar in organization and storage to geographic information, and allows for query expansion in searches based on category.

When a query is successfully parsed, the dialog model proceeds to resolve the incompleteness and ambiguity that may be present within. In order to deal with query incompleteness (where the system must rely on what has been previously said), the DM maintains a dialog history that keeps all previous user queries on a stack. There is also a result history that tracks the results associated with each user query.

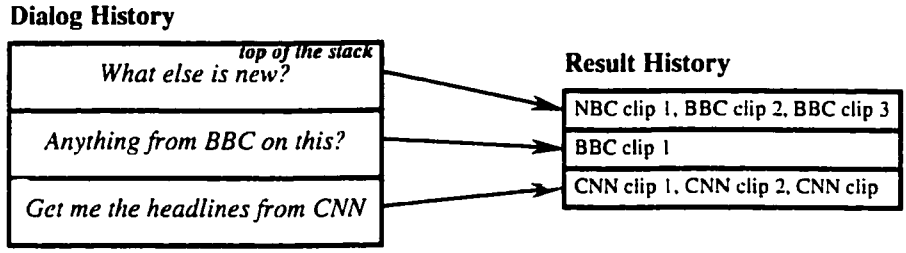


Figure 18: Dialog History and Result History

The notion of dialog context emerges during the discussion of dialog history. A sequence of user queries $q_1\ q_2\ \ldots\ q_n$ belongs to the same context if each query relies on something that appeared earlier in the sequence. Thus, the dialog history is used for the sole reason of tracking the current dialog context. Whenever there is a query that is complete (i.e. does not rely on dialog history), it marks the beginning of a new dialog context. When this happens, the old dialog history is purged and a new one is started.

An important issue arises here – how to determine if a query is complete? Since all user queries have a restricted structure, the type of the query itself determines whether the query is complete or not. Take, for example, the query

Is there anything from CNN on this?

This is a query coming from the production

(sem **show**) (sem **in**) (*NewsMediaReference* | *MediaType*) (sem **about**) (sem **this**)

The DM knows that this is an incomplete query and, in order to determine what the pronoun *this* refers to, it has to look at the dialog history to find out and what news topic (stored in the data field *event*) has been specified previously. If several news

stories have been returned as a result, the DM also has to look at the result history to see which one is currently selected.

When resolving such incompleteness in a user query, the DM traverses the dialog history stack from top to bottom in attempt to find the missing data field. By using a stack, the last queries are accessed first, allowing for a retrieval of the most recent information. If the DM is unable to find the missing field, the error model is allowed to take over the interaction.

Ambiguity is different from incompleteness in that it is often impossible to know in advance whether a query is ambiguous by looking at its type. In order to deal with an ambiguity, the DM consults the user model for determining user preferences and other user-specific parameters. The query

Is there anything from China?

is ambiguous in the sense that the DM does not know which news media sources it should retrieve the information from, nor what date it should begin searching with. The information needed for disambiguation can be found in the UM. Rather than using this information for constraining the query, it is used to modify the presentation of information – relevant news items from the user's preferred news source (say, BBC) would be presented before similarly-ranking stories (from CNN, NBC, and others).

The date of last invocation obtained from the UM is used differently. It is computationally expensive to search an entire database for a particular news item, and we assume that users are normally interested in recent events. For this reason, when no time reference is given in the user query, the DM uses the date of the last invocation provided by the UM to set a starting point for the database search.

After a query has its incompleteness and ambiguity resolved, it is almost ready to be sent to the database component for searching. The only detail that remains is the case of a missing *timeReference* field. If none is given in the user query, the DM assumes that the user wants to retrieve news items for the current day. It should be noted that this field

may be ignored by the database component when no matches are found for the specified date.

4.3.1.5 Error Model

Ideally, the error model should be able to handle errors arising in the LM as well as the DM. Language model errors are mostly due to parsing difficulties, usually when user queries do not fit into one of the predefined types. Dialog model errors, on the other hand, tend to concentrate around queries whose incompleteness cannot be positively resolved (one example of such error is the query *Anything from CNN on that?* with empty dialog and result histories). There is an additional source of errors arising from the DM when trying to remove or asking for follow-up on a story that has not been defined in the UM. It should be noted here that the system provides visual feedback to the user, with the current news story clearly highlighted in the results window. In that respect, the EM does not have to deal with as many problems as it would have in a speech-only environment (as required by SpeechActs [Yankelovich95]). In particular, the problem of having the user's perceived state of the system different from the actual system state is eliminated with visual feedback.

Dealing with errors originating from the dialog model usually requires the use of clarification dialogs (such as *Did you mean BBC or NBC?*). However, the amount of visual feedback present and the presence of typed user input makes this approach unnecessary. As a result, all errors from the LM and DM are dealt with in the same manner: a visual cue is used to indicate that an error was encountered, and the user is asked to restate the query.

4.3.2 Database Access

The system uses its own database engine due to the lack of a JDBC-compliant Solaris database at the time of development. If required, however, the system can be migrated to a JDBC-compliant engine with minimal effort. This section expands on the type of

indexing of media items used, as well as some specific functionality provided by the engine.

4.3.2.1 Indexing of news media items

In order to perform searches on the database, a set of metadata fields was associated with each news item. These fields were as follows:

- *filename* – file name of the clip
- *size* – size of the clip (bytes)
- *duration* – duration of the clip (seconds)
- *source* – source of the news item
- *date* – broadcast date of the news item
- *keywords* – set of 5-10 keywords best describing the clip
- *summary* – one- or two-sentence summary of the contents of the clip
- *category* – classification categories of the news clip (politics, sports, etc.)
- *reporter* – name of the reporter
- *location* – geographic location of the event in the clip

For example, a news clip about a jet-powered car that broke the sound barrier might have the following metadata fields:

filename	bbc/13oct97-3.mpg
size	3,864,831 bytes
duration	211 seconds
source	BBC
date	October 13, 1997
keywords	British jet car speed of sound desert sound barrier Andy Green
summary	A British car has become the first to go faster than the speed of sound in the Nevada desert
category	sports current events
reporter	Jonathan Miller
location	Nevada desert

4.3.2.2 Search engine functionality

The metadata fields allow for a content-based search of the news media. As a result, the search engine has the ability to accept requests for retrievals based on any of the fields found in a user query (*timeReference*, *newsMediaName*, *geographicLocation*, etc.)

When the search is based on *timeReference*, the engine first checks whether the time reference is absolute or relative. In the former case, the search consists of simply checking for matching date from the database. In case of a relative date, however, the engine first constructs a closed time interval, and then searches for dates that fall within it.

The keyword and summary fields of a news item are sometimes redundant in terms of information they carry, but, more often than not, complement each other. For this reason, when looking for *person* or *event* field of a user query, the engine searches in both keywords and summaries.

For *geographicLocation*, the database engine uses the WM's geographic knowledge to obtain all the children of a location being searched. This hierarchy is then flattened, and the resulting set is used for the search. When there are results for both the original location and one of its children, results matching the former are returned first; this is due to the fact that a news story from the original location answers a user query better than a news story that is not.

Often, the DM requires the retrieval of clips similar to a particular news item (as, for example, in the query *Follow this story*). In order to support this functionality, the database engine implements a similarity search. Given a news story, it looks at the keywords of other news items; if there is an overlap of two or more keywords, the item is assumed to be similar, and is returned.

4.3.3 Result Presentation

The presentation module is responsible for managing typed input, displaying information returned as a result of a query, and playing back video clips. The user interface consists of four windows, as shown in Figure 19.

The results window (top left in Figure 19) displays a numbered sequence of news items that match the query, with the current item highlighted. For every item, a thumbnail of the video clip is shown, together with date, source, summary, and location. The user can play back the currently selected item, and perform a number of operations on a playing clip, which include stopping, resuming, fast-forwarding, and rewinding.

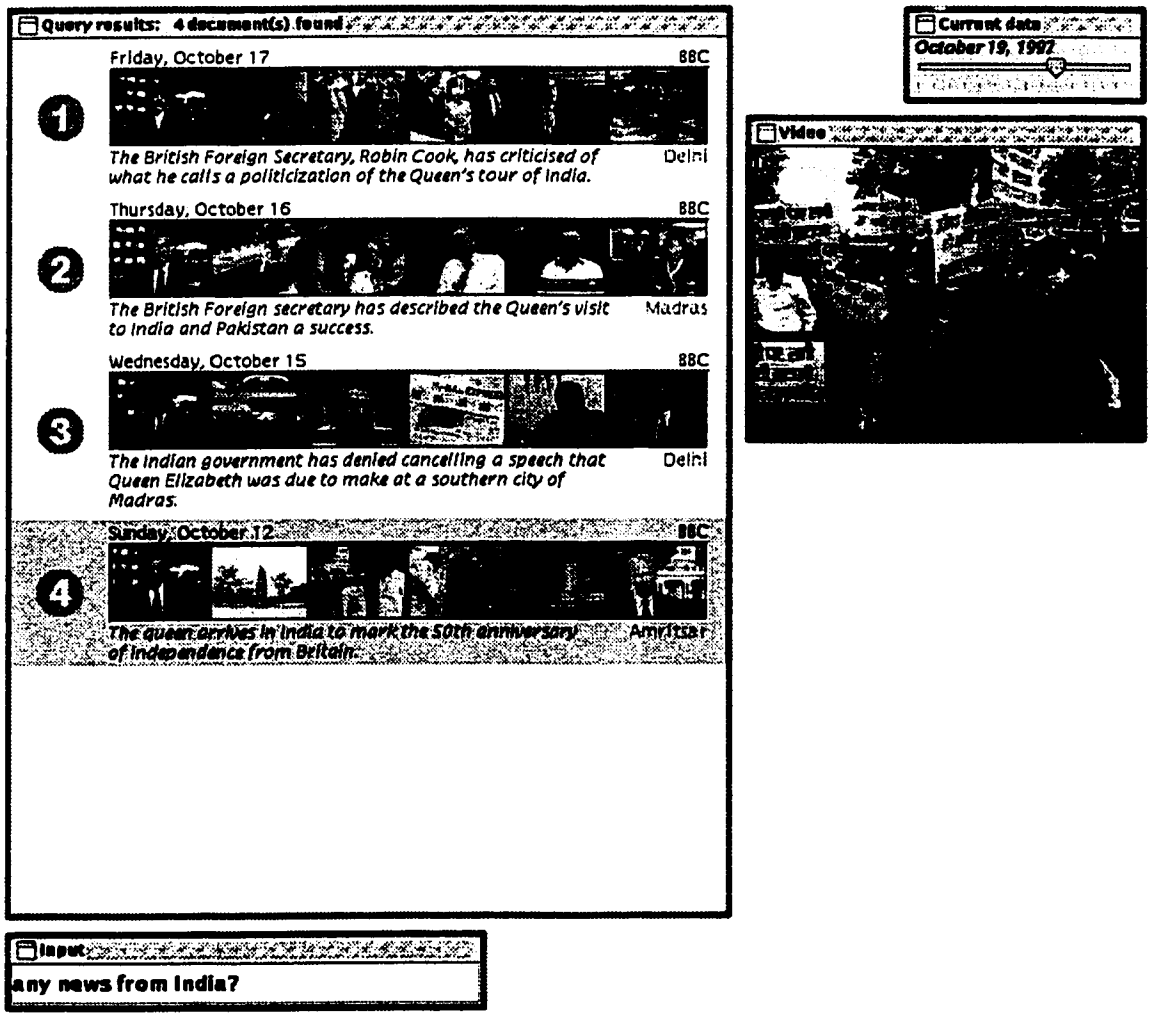


Figure 19: Graphical user interface

The presentation module uses the MVC architecture [MageLang98] to manage the interaction between the presentation model and presentation view (Figure 12). An MVC UI contains three communicating objects: the model, view, and controller. The model is the underlying data representation (in our case, presentation model), the view is the visual representation (presentation view), and the controller handles input (dialog model). When a model changes, it posts a notification to all views that depend on it. This way, whenever the presentation model changes (as a result of a user query), the view is automatically updated to reflect the latest changes.

There are times when it is desirable to simulate passage of time during an interaction. This proves useful for testing purposes, and is especially convenient for following-up on a story. For that reason, a fourth window was added (top right in Figure 19), which allows on-the-fly modification of the world model's internal clock by means of a slider. This allows us to select a story to follow (*Follow this story*), advance the timeline by several days, and get an immediate follow-up (*Any developments?*).

4.4 Test Database and Indexing

The multimedia test database contains over 120 media clips from four different news sources (BBC, CNN, NBC, TSN). The clips were collected over a period of 9 days (October 12 through October 20, 1997), and range from half-minute to several minutes in duration. The entire collection of media clips is stored in MPEG format, occupies nearly 2GB of disk space, and has a cumulative running time of over 4.5 hours. The collection contains over three dozen distinct news stories; each story often has clips from different news sources.

The indexing of data was done manually, using a tool written in Java (Figure 20). Since database searches are only as accurate as the indexing data in it, care has been taken to insure the accuracy of information during the indexing process.

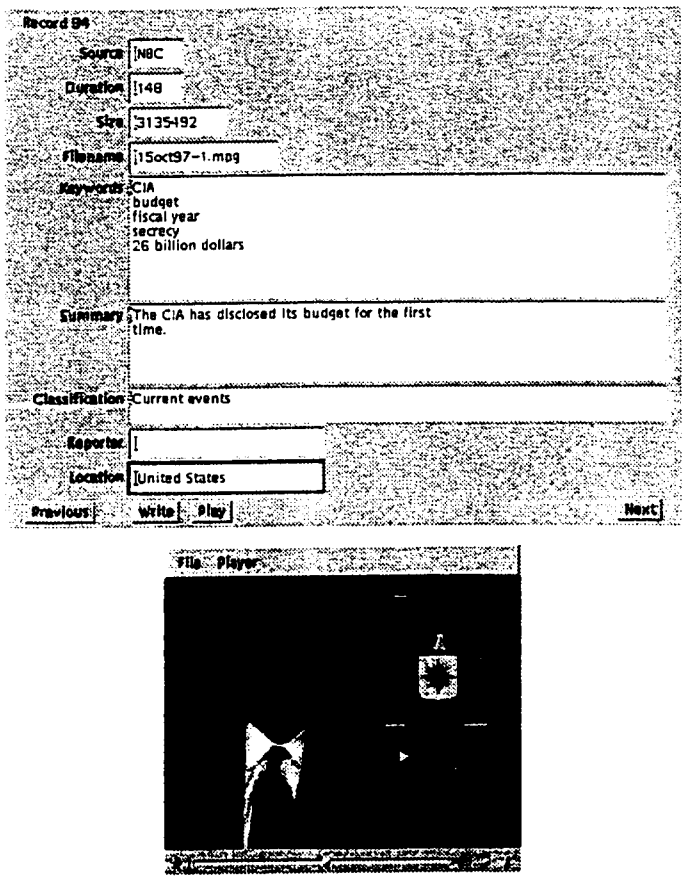


Figure 20: Indexing of news items

4.5 Test Queries – Coverage and Recall/Precision

Although the 13 different user query types defined for this project are by no means exhaustive (in the sense of covering arbitrary user inquiries), they provide sufficient support for typical user tasks, which we studied in the early phases of the project. In order to test the system performance and to gather corresponding metrics, we defined an interaction scenario for a hypothetical user, which is shown in Table 4 (the corresponding query types were defined in Figure 13). This scenario exhausts the set of allowable user queries (in terms of the 13 types), and is thus suitable for coverage testing.

Table 4: Dialog scenario

User query	Type	Date	Result
Show me the headlines	3	12/10/97	Headlines from all news sources are shown, beginning with CNN – the default preference
Get the CNN story	1		The first available CNN video clip is played
fast forward			Playback rate is increased by a factor of 2
resume			Normal playback rate is restored
Follow this story	12		The news topic is added to the list of stories to follow in the UM
What else is new?	11		The remainder of new stories is listed
Anything from BBC on that?	10		Stories related to the currently selected news item are retrieved from BBC
Simulate break in interaction – advance clock by 5 days		19/10/97	date slider is advanced by 5 ticks
Any developments?	8		All recent (less than 5 days old) news stories relating to the follow-up story from the UM are retrieved
play it			The selected clip is played
Drop this story	13		The story is removed from the UM
Is there anything about the flood in China?	2		List of stories is returned
play it			The selected clip is played
another one			The next clip is played
Get me the video about the Clinton scandal	4		Relevant stories are returned
Simulate break in interaction – advance clock by 2 days		21/10/97	date slider is advanced by 2 ticks
Anything new about the flood in China?	7		All recent (less than 2 days old) stories are returned
Get me the current events from yesterday's CNN	5		News items from one day ago are returned
Anything from India?	9		News items from India are shown
Get me the hockey results from TSN of 3 days ago	6		Results are retrieved from TSN

4.5.1 Coverage

Using the dialog scenario defined earlier, we can perform a coverage test of the system. This testing ensures that every allowed query type is used at least once. One may argue that such testing may not mean much, as most valid forms of any given query type will go untested. Though a valid point, it is important to realize that we are interested in testing the functional aspects of our system (such as resolution of incompleteness and ambiguity, use of UM, valid presentation of results), and this coverage test allows us to

do exactly that. Using the scenario in its original form, we are able to see the correct handling of user queries, proper disambiguation and completeness resolution, and a correct presentation of results.

4.5.2 Recall/Precision

Recall is the proportion of all relevant documents that are actually retrieved as a result of a query. Precision is the proportion of a retrieved set of documents that is actually relevant [Belkin92]. Having said that, searching in our system is tied to the underlying indexing of news data – better indexing would result in improved numbers for both recall and precision.

Most searches are performed on the keyword and summary fields of a news item. With care taken in producing accurate indexing, the recall is in the range of 80%-85%, depending on the type of search, and precision fluctuates between 85%-90%. The precision is less than perfect due to a small number of unrelated stories that use similar terms in their indexing. The justification for a lower value of recall is due to an elective indexing – only what is relevant to the main story is used in the index. As a result, sideline references to secondary topics usually go unnoticed. Improving the indexing scope to include everything said in a news story would adversely affect precision. Hence, due to a large size of the database, it is preferable to sacrifice some level of recall in favor of better precision.

5. Summary and Conclusions

The work described in this thesis originally started as a project under an industrially-oriented research grant from Nortel, whose aim was to integrate a news delivery system with a spoken language interface. Unfortunately, commercially available speech recognition products (at that time (1996), DragonDictate from Dragon Systems and VoiceType from IBM) fell short of our expectations. Both products were lacking in terms of speech APIs, and had unacceptable recognition rates for moderately-sized vocabularies (approximately 500 words). As a result, we were forced to concentrate on the discourse management aspect of our system, and had to resort to typed input as the only modality for user interaction. However, dealing with problems arising from using speech in a news delivery system is large enough to be a thesis of its own, and would not have allowed us to do significant amounts of work in the area of model integration.

5.1 Conclusions

Our system uses a model-based architecture for providing a conversational interface for electronic news delivery. The models allow the system to support pseudo-natural queries (language model), take advantage of human discourse (dialog model), make use of domain-independent and domain-dependent knowledge (world model), make inferences about user-specific needs (user model), and deal with errors arising from the interaction (error model).

The three major problems facing natural language interfaces are naturalness, habitability, and portability. Naturalness is used to refer to the "intuitive feel" of an interface, and is used to measure how well an interface supports natural human communication. The human language being very informal, designing an intuitive and reliable natural language interface requires considerable effort; very often, users rely on

what has been previously said, and make use of anaphoric references (*Anything else on this?*). Requiring users to employ fully-specified queries adversely affects the naturalness of the interface; for that reason, we allow queries that can be incomplete and ambiguous. In order to successfully deal with both ambiguity and incompleteness, our system uses a dialog-centered approach to discourse management. The dialog model (DM) is the controlling entity in our architecture; it keeps track of the dialog history and context, and also manages the interactions between the remaining four models. The language model contains the definition of query types, and specifies how each individual type is to be handled by the dialog model. When needed, the DM's dialog history is used to resolve incompleteness in user queries, while the user model provides the information needed for disambiguation.

The habitability of an interface refers to the users' ease of expressing everything required for their tasks by using only queries allowed by the system. We used a user-centered approach to habitability, where the query types and query language were determined after analyzing the user needs and the capabilities of the news delivery system. Rather than attempting to exhaust the set of arbitrary input queries, we arrived at a small collection of query types that permitted users to use the full capabilities of the system; still, the query definitions were sufficiently broad to allow for variations in syntax. This set of queries was shown to be adequate for carrying out meaningful interactions with the system.

The portability of an interface can be looked into at three independent levels: task domains, users, and platforms. Task domain portability is used to measure the degree of ease in migrating a user interface from one domain to another. In our architecture, several models store domain-specific information: the language model contains the query type definitions for the news domain, the user model maintains user preferences, and a part of the world model stores knowledge specific to the news domain. As a result, those models need to be redefined when migrating them to a new domain; on the other hand, the dialog model, error model, and common-sense part of the world model would remain intact.

The portability across users refers to the ease of adapting the interface to different users. The user model, with its user profile, partially handles this aspect of portability by allowing each user to individually customize his media preferences. Finally, having been implemented in Java, the system is portable across different hardware platforms running a JVM (Java Virtual Machine). As it relies on the JMF (Java Media Framework) for video playback, the system is limited to hardware architectures for which a JMF implementation currently exists. At the time of this writing, the list includes Solaris running on Sun, Windows 95/NT 4.0 on Intel, and the currently pending Irix implementation on SGI.

From the onset, the language model was designed to be independent of the dialog model. The LM contains the definitions of query types, as well as the information required by the DM to handle each individual type. When a query is successfully parsed, the information passed from the language model is used by the dialog model to determine the presence of ambiguity and incompleteness. In addition, the query type definition provides sufficient hints to the DM that allow the determination of proper steps to be taken to resolve the user query. This distinction between query definition (LM) and query functionality (DM) allows for new query types to be defined without affecting the DM's implementation. It also simplifies system design and eases the migration of the interface to a different domain.

As mentioned earlier, the world model makes a clear separation between domain-independent and domain-specific knowledge. The domain-independent (common-sense) knowledge base stores temporal, spatial, and geographic information that is casually referred to in human conversations. This knowledge allows the resolution of relative time references, determination of references to geographic locations, and inference of spatial and hierarchical relationships between them. The domain-dependent knowledge is supplied by news providers, and consists of information describing the organization of news documents, the classification used by each provider, and expected time constraints for availability of different news documents. When migrating the world model to a new domain, the clear distinction between the

two knowledge bases makes it easier to define a new domain-specific part, without impacting the common-sense part of the model.

An operational prototype for the news delivery system has been implemented using a sample news database, and a set of coverage tests was performed using dialog scenarios that utilized all of the query types defined in the LM. The system was shown to properly resolve incompleteness and ambiguity, and the interaction resulted in a meaningful dialog with the user, with acceptable levels of recall and precision in retrieval.

A few words need to be said about the test database. When work started on the implementation of the system, we looked for an existing database that could be used for our purposes. None was freely available to us, which left us no choice but to build our own. This proved to be a major undertaking, as it required the determination of what should be in the database, followed by the daily recording of over 4 hours of news broadcasts from various networks over a period of nine days. The 30+ hours of news footage were then digitized, edited for content, and shortened whenever possible. This produced over 4.5 hours of quality news video footage, which was then compressed to MPEG format. Finally, indexing data was added to each of the video clips. As a result, we now have nearly 2GB of video data; some news stories in this database are spread out over several days and have follow-ups from different broadcast networks. This will be useful for future research in this domain.

5.2 Future Work

Several issues have emerged from our implementation of the prototype for electronic news access. The interface requires extensive usability testing in order to determine its usefulness in accessing news in a real-world application. Although the set of predefined query types was shown to be sufficient to use the capabilities of the prototype, whether the query types are adequate for real user interaction remains to be seen.

Speech input is an important direction for the future. Since the interface currently uses typed text as the only input modality, adding speech would definitely improve the naturalness of the interface. However, speech brings in a new category of problems that must be dealt with. Spontaneous speech with false starts, filled pauses, and user hesitations needs to be properly modeled. Run-on sentences combined with changes in user intonation must be detected. Finally, recognizer errors (rejection, substitution, and insertion errors) have to be properly handled for the speech interface to be natural.

To further enhance the habitability of the interface, a much larger set of query types (combined with a larger vocabulary) is desirable. In order to collect more information on typical user access patterns, in-depth user studies and Wizard-of-Oz simulations are needed. It would be interesting to see whether it would prove beneficial to allow users to combine several goals in their queries (for example, *Get me latest on the San Fancisco earthquake and follow that story*). Equally important is the new word problem, and exploring mechanisms for dealing with this issue would be a challenging task.

6. References

[Asadi91] A. Asadi, R. Schwartz, and J. Makhoul, "Automatic modelling for adding new words to a large-vocabulary continuous speech recognition system", *Proceedings of ICASSP '91*, Toronto, 1991

[Belkin92] N. Belkin and W.B. Croft, "Filtering and information retrieval: two sides of the same coin?", *Communications of the ACM*, vol. 35, no. 2, December 1992

[Brennan95] S. E. Brennan and E. A. Hulteen, "Interaction and feedback in a spoken language system: a theoretical framework", *Knowledge-Based Systems*, vol. 8, no. 2-3, April-June 1995

[Cole95] R. Cole *et al.*, "The Challenge of Spoken Language Systems: Research Directions for the Nineties", *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, January 1995

[MageLang98] "Swing Short Course, Part II – MVC Architecture", MageLang Institute, March 1998. Online, available at `http://developer.javasoft.com/developer/onlineTraining/swing2/swing.html`

[Mayhew92] D. Mayhew, "Principles and Guidelines in Software User Interface Design", Prentice-Hall, 1992

[Oviatt96] S. Oviatt, "User-Centered Modeling for Spoken Language and Multimodal Interfaces", *IEEE Multimedia*, January 1996

[Prince95] V. Prince and D. Pernel, "Several knowledge models and a blackboard memory for human-machine robust dialogues", *Natural Language Engineering*, vol. 1, no. 2, 1995

[Smith98] M. Smith and T. Kanade, "Video Skimming and Characterization through the Combination of Image and Language Understanding", *IEEE International Workshop on Content-Based Access of Image and Video Databases*, ICCV98, Bombay, India, 1998.

[Wyard96] P. J. Wyard *et al.*, "Spoken language systems – beyond prompt and response", *BT Technology Journal*, Vol. 14, no. 1, January 1996

[Yankelovich95] N. Yankelovich, G.-A. Levow, M. Marx, "Designing SpeechActs: Issues in Speech User Interfaces", *ACM CHI '95 Conference Proceedings*, Denver, Colorado, May 1995

[Young93] S.R. Young,W. Ward, "Learning New Words from Spontaneous Speech", *Proceedings of ICASSP '93*, Vol. 2, 1993

Appendix I User Survey

To examine the needs of users in the domain of electronic news delivery, the following questionnaire has been circulated among students, faculty, and staff in the department of Computer Science.

We are conducting a survey in order to study the information access patterns of a general user population. We would appreciate if you could spend a few minutes in completing this survey.

1. Your occupation:_____

2. How, when and where do you get your news?

	Weekdays			*Weekends*		
	Morning	Afternoon	Evening	Morning	Afternoon	Evening
Newspaper	☐	☐	☐	☐	☐	☐
Television	☐	☐	☐	☐	☐	☐
Radio	☐	☐	☐	☐	☐	☐
Other (please specify)	☐	☐	☐	☐	☐	☐

3. What newspaper(s) do you read regularly, if any?

_____

4. Do you occasionally read any other daily, weekly, or monthly newspapers?

_____

5. How much time, on average, do you spend daily on reading newspapers?

   - ☐ Less than ½ hour
   - ☐ ½ hour to 1 hour
   - ☐ 1 to 2 hours
   - ☐ More than 2 hours

6. What sections/parts of the newspaper do you read?

	Always	*Occasionally*	*Never*
Front page	☐	☐	☐
Lead Stories	☐	☐	☐
National	☐	☐	☐
International	☐	☐	☐
Regional	☐	☐	☐
Business	☐	☐	☐
Sports	☐	☐	☐
Entertainment	☐	☐	☐
Science/Technology	☐	☐	☐
Health	☐	☐	☐
Opinion	☐	☐	☐
Weather	☐	☐	☐
Miscellaneous	☐	☐	☐
Classifieds	☐	☐	☐

7. Some people browse through a newspaper without looking for anything specific. If you do, what kind of information do you look for?

   ☐ Headlines/News Topics
   ☐ Photographs
   ☐ Advertisements
   ☐ Promotions
   ☐ Classifieds
   ☐ Other (please specify):_____

8. If a personalized news delivery system were made available to you, which of the following features would make you use it instead of conventional news sources?

   ☐ Round-the-clock availability
   ☐ Ability to search for news of interest
   ☐ Easy access to older news
   ☐ Possibility of selecting between condensed and elaborate news versions
   ☐ Absence of ads/commercials
   ☐ Other (please specify):_____

Results

We received a total of 15 completed questionnaires at the end of the survey. Our findings are presented below.

How, when and where do you get your news?

	Weekdays			*Weekends*		
	Morning	Afternoon	Evening	Morning	Afternoon	Evening
Newspaper	18.2%	9.1%	0%	18.2%	0%	27.3%
Television	0%	9.1%	72.7%	0%	18.2%	63.6%
Radio	18.2%	36.4%	27.3%	18.2%	36.4%	27.3%
Other (please specify) *Internet*	18.2%	18.2%	18.2%	18.2%	18.2%	18.2%

What newspaper(s) do you read regularly, if any?

Montreal Gazette	63.5%
La Presse	27.3%
The Globe and Mail	18.2%
Financial Post	9.1%
Le Devoir	9.1%

How much time, on average, do you spend daily on reading newspapers?

Less than ½ hour	54.5%
½ hour to 1 hour	27.3%
1 to 2 hours	18.2%
More than 2 hours	0%

What sections/parts of the newspaper do you read?

	Always	*Occasionally*	*Never*
Front page	54.5%	36.4%	0%
Lead Stories	18.2%	45.5%	9.1%
National	27.3%	45.5%	0%
International	81.8%	0%	0%
Regional	18.2%	54.5%	9.1%
Business	18.2%	45.5%	18.2%
Sports	0%	54.5%	18.2%
Entertainment	36.4%	45.5%	9.1%
Science/Technology	9.1%	54.5%	27.3%
Health	9.1%	36.4%	27.3%

	Always	*Occasionally*	*Never*
Opinion	27.3%	18.2%	36.4%
Weather	18.2%	36.4%	27.3%
Miscellaneous	0%	45.5%	18.2%
Classifieds	0%	45.5%	27.3%

Some people browse through a newspaper without looking for anything specific. If you do, what kind of information do you look for?

Headlines/News Topics	81.8%
Photographs	27.3%
Advertisements	18.2%
Promotions	0%
Classifieds	9.1%
Other (please specify): Comics	9.1%

If a personalized news delivery system were made available to you, which of the following features would make you use it instead of conventional news sources?

Round-the-clock availability	27.3%
Ability to search for news of interest	72.7%
Easy access to older news	54.5%
Possibility of selecting between condensed and elaborate news versions	54.5%
Absence of ads/commercials	45.5%
Other (please specify)	0%

Appendix II Selected Screen Captures of the News Delivery System

The 5 screen captures that follow were taken while interacting with the system. The query being answered can be clearly seen in the input window.

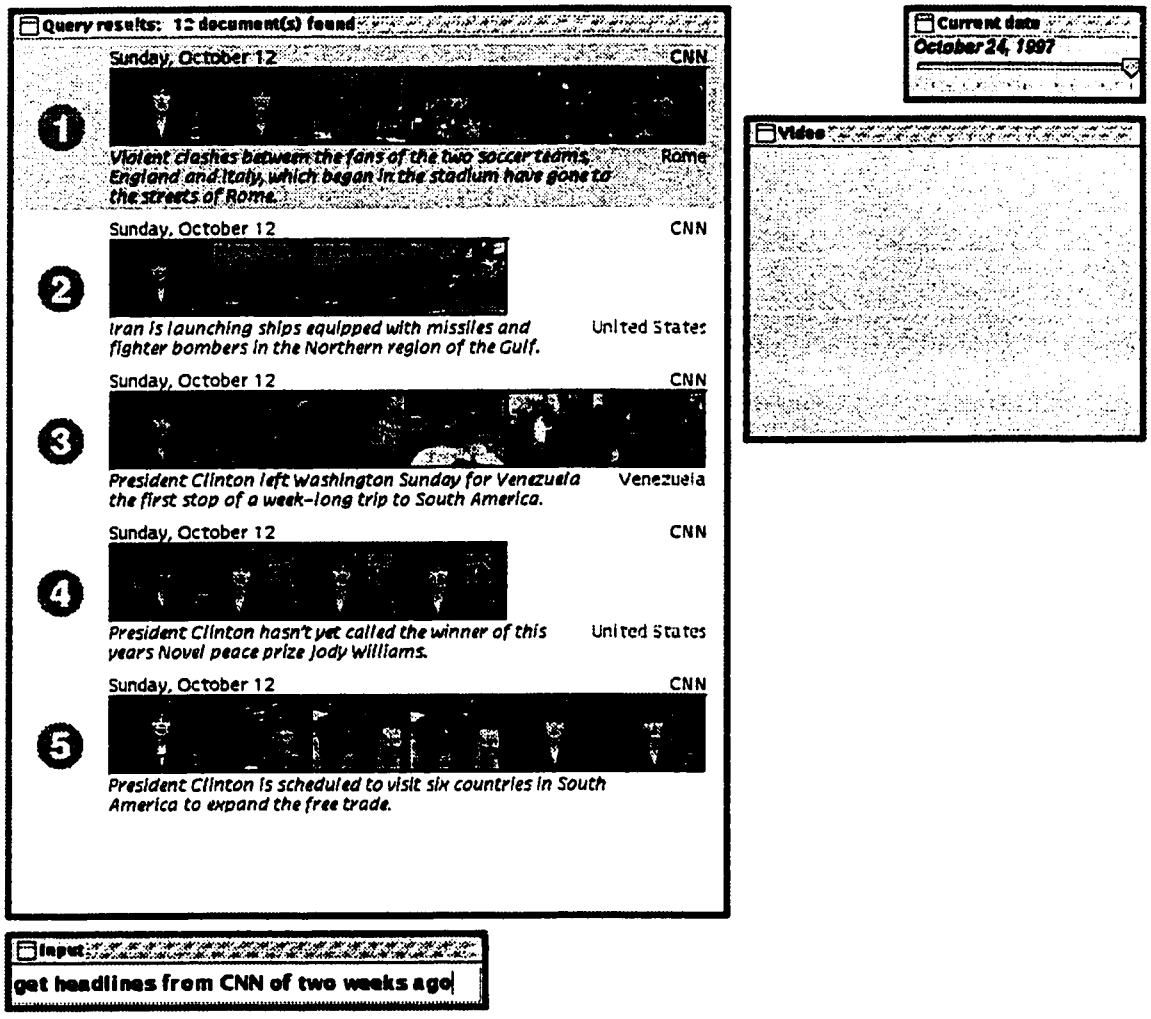


Figure 21: Using relative time references

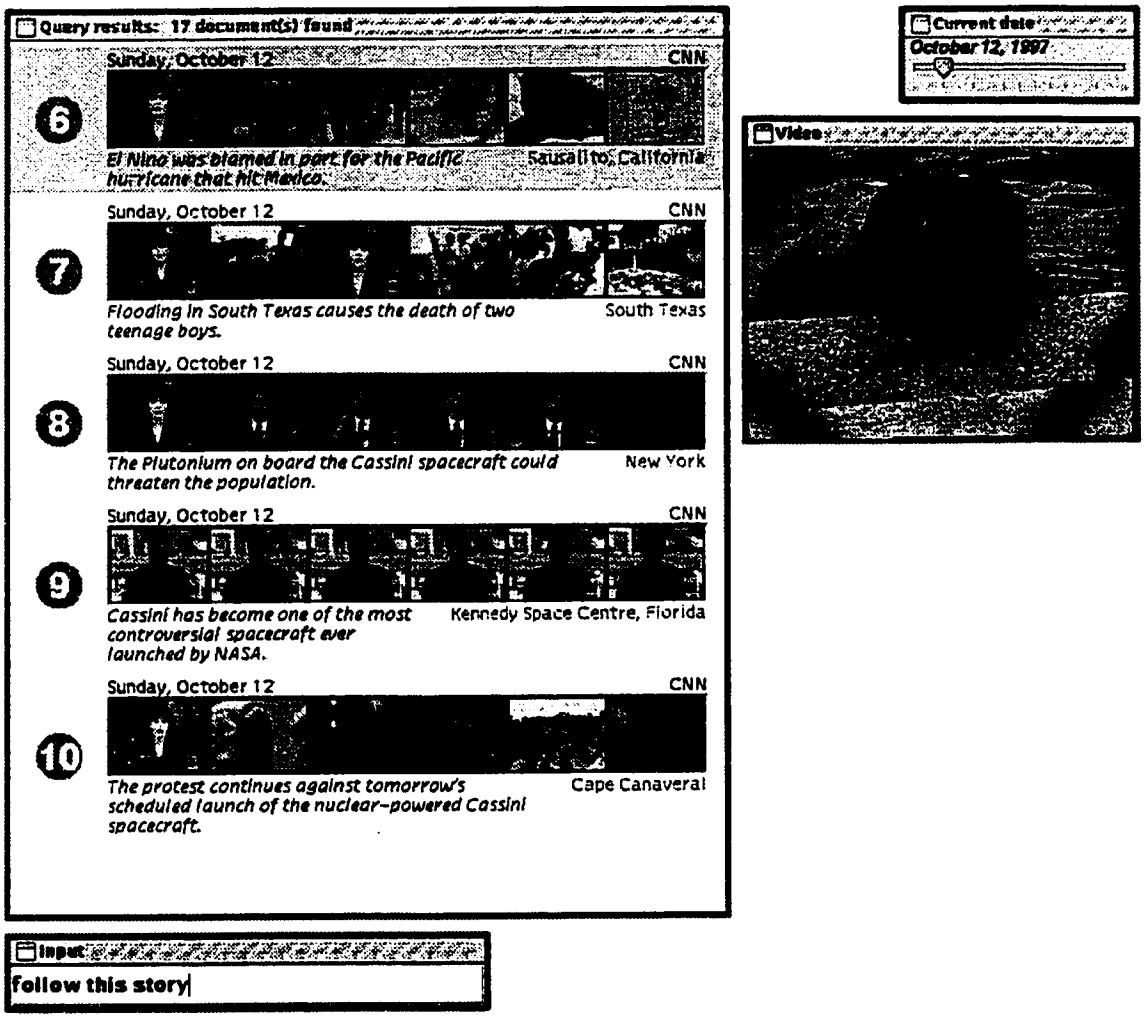


Figure 22: Selecting a story to follow

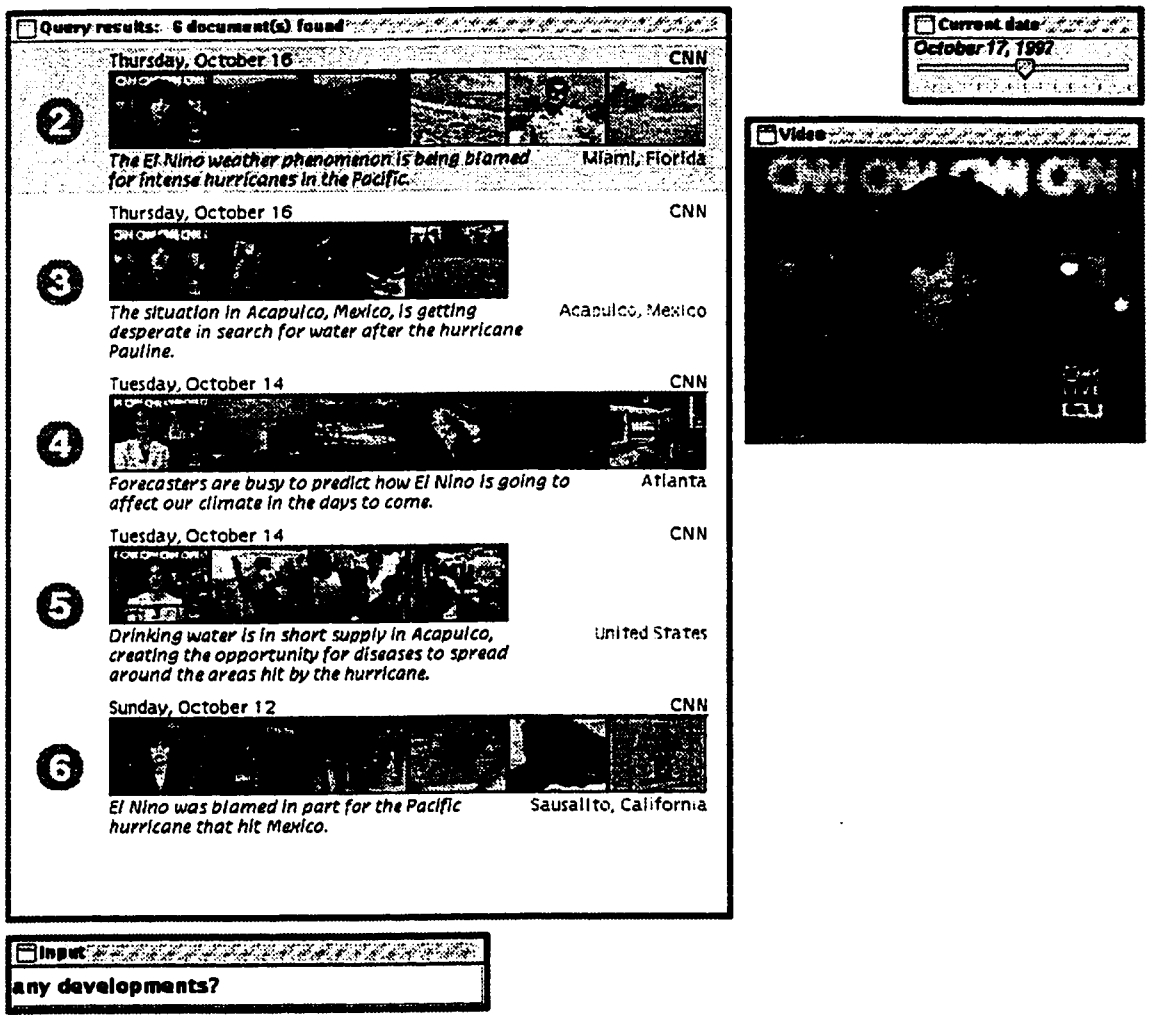


Figure 23: Getting follow-ups

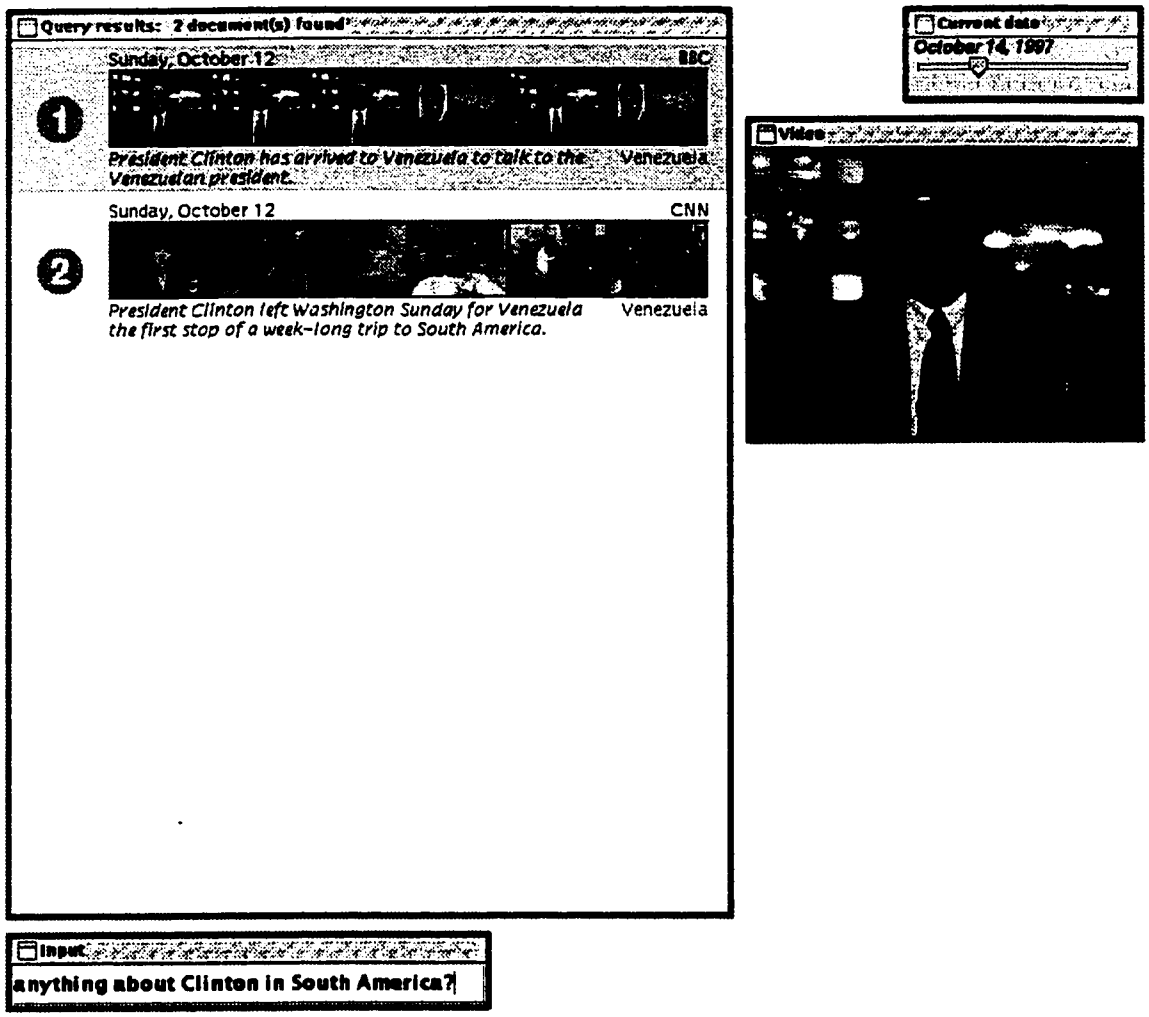


Figure 24: Using a topical search

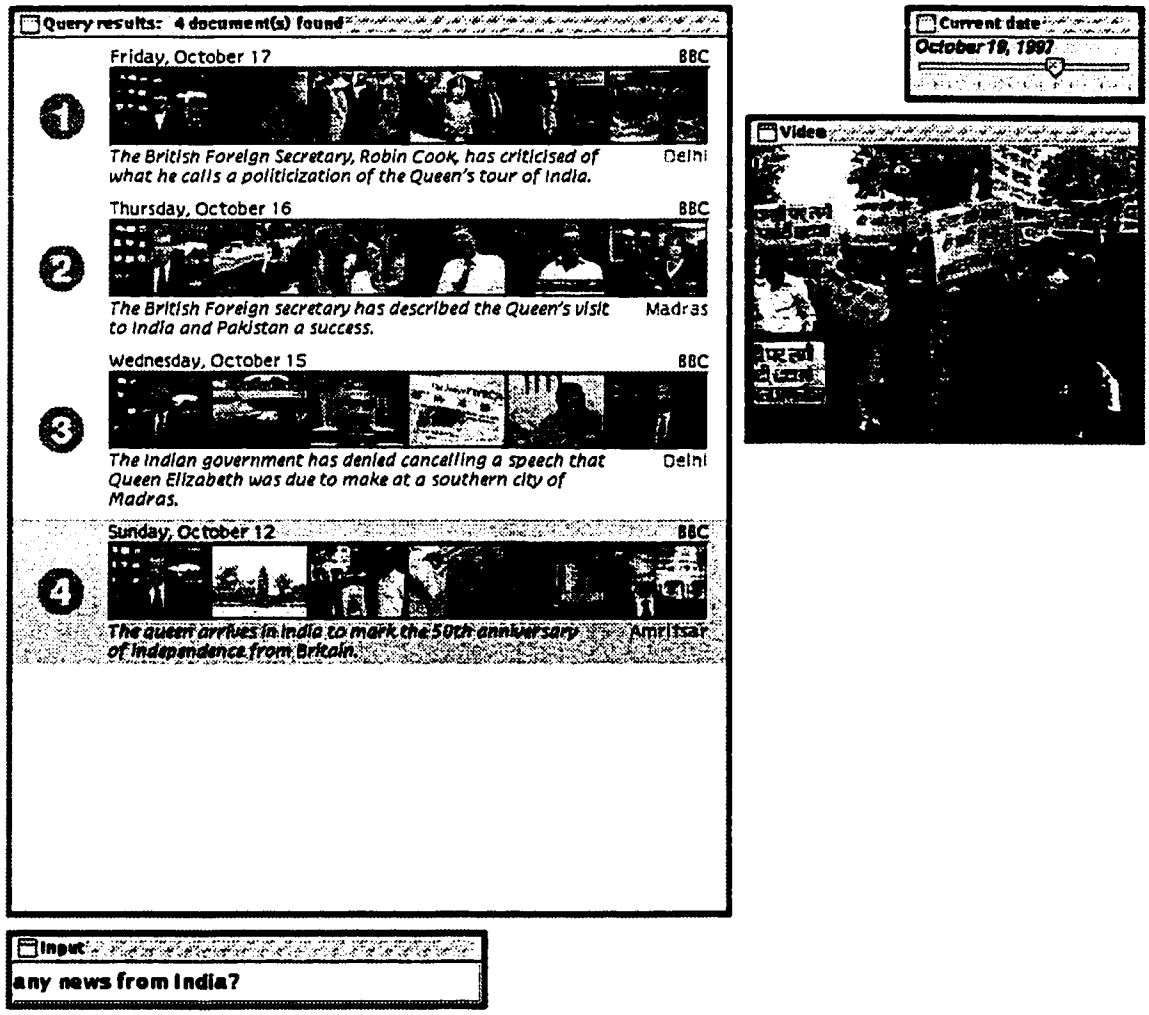


Figure 25: Using a geographic search