# PROMOTING PORTABILITY IN DIALOGUE MANAGEMENT

*Joseph Polifroni*[*]

Spoken Language Systems Group
MIT Laboratory for Computer Science
Cambridge, MA 02139

*Grace Chung*[†]

Corporation for National
Research Initiatives
Reston, VA 20191

## ABSTRACT

We have recently begun an effort to develop a domain-independent dialogue manager that can be used as part of a mixed-initiative spoken dialogue system to construct more complex systems without modifying underlying code. Inside a *generic* dialogue manager, the developer would select from a suite of self-contained dialogue flow functions, and tailor them in accordance with the specifics of the application. We have also developed grammars catered to semantic concepts such as dates/times, and prices, along with a server that interprets and canonicalizes these concepts. These grammars can be used by developers in the SPEECHBUILDER framework [1] to obtain precompiled meaning representations for commonly used concepts, and embed them into their applications, drastically reducing the work required to configure a conversational system. The generic dialogue manager and functions supporting common semantic concepts have been applied to several new domains including a Hotel Information Domain.

## 1. MOTIVATION

Since 1989, our group at MIT has built over 15 conversational systems in multiple domains (e.g., weather [2] and air travel [3]). For the most part, these systems have been built by expert developers, with hand-crafting of domain-specific knowledge and functionality. As the underlying technologies matured, we became increasingly interested in making these systems portable and configurable by novice developers. One of our first steps towards portability was the introduction of the GALAXY architecture [4], which enables users to configure a number of specialized servers that communicate with one another via a central programmable hub. GALAXY was designed to promote plug-and-play modularity.

To encourage non-experts to use our core human language technology servers, we developed SPEECHBUILDER designed to facilitate the creation of grammars and language models for recognizers. In its first iteration, SPEECHBUILDER could automatically configure an interface to an SQL database represented by attributes and their corresponding properties, as well as access information from developer-specified sources of data. Developers could build interfaces to answer simple queries about those attributes. In an effort to expand SPEECHBUILDER's ability to answer more complicated queries in a variety of domains, we turned our attention towards portability issues in the dialogue manager.

Dialogue management has traditionally resisted the push towards portability and rapid configurability in our systems; its role in planning and response generation has been considered too domain-dependent. In the course of building systems, however, we have noticed that some basic functionalities are applicable throughout all domains. For instance, each system must gather information from a user and prompt the user for critical missing pieces, and each system must have a way of filtering responses from the database to match user-specified constraints. Furthermore, certain categories of information, such as dates and times, recur in multiple domains. Users can ask for flights on "Tuesday," or about the weather "the day after tomorrow," or for the estimated landing time of a flight scheduled for "late this afternoon."

In this paper, we describe a domain-independent or *generic* dialogue manager that can perform a set of essential dialogue flow operations, customizable to a specific domain through a simple, text-based interface. We also describe a new mechanism for interpreting and canonicalizing concepts such as date/time strings. Examples of how these functionalities have been utilized are given in a new hotel information domain. To illustrate the relative portability of the generic server, we also describe the recent development of four new domains that have adopted this dialogue manager.

## 2. DIALOGUE MANAGEMENT IN GALAXY

In our systems, the dialogue manager is just one of many servers configured around a GALAXY hub. The dialogue manager is tasked with interpreting a fully specified representation of each utterance in context as a set of attribute-value pairs, which we refer to as the *dialogue state*. It is responsible for generating a reply to the user, which can involve planning, querying a content provider, evaluating database tuples, and/or offering help. The output of dialogue management is a response frame that is sent to a generation component for generation and synthesis to the user.

In our initial spoken dialogue systems, the dialogue manager was completely implemented in code specific to each domain. To understand program flow, a system developer had to run the code in a debugger to follow the progress through a series of function calls. Beginning with our JUPITER system in 1997, we began using an externally specified *dialogue control table* to maintain control in the dialogue manager. Written in a high-level scripting language, this table specifies all the operations available to the dialogue manager, and the sequence of conditions that will determine when these operations are executed [3]. Once a function is called, it may modify or add to attributes in the dialogue state and, upon

[*]This research was supported by DARPA under contract N66001-99-1-8904 monitored through Naval Command, Control and Ocean Surveillance Center, as well as by an industrial consortium supporting the MIT Oxygen Alliance.

[†]This research was supported by DARPA under contract N66001-00-2-8922 monitored through SPAWAR Systems Center – San Diego.

```
:destination & :airline & !:source -> need_source
:source & :destination & :airline & :flight_number -> get_flight_info
:num_found > 0 & :arrive_time | :depart_time -> filter_for_time
```

**Table 1**. *This is a partial sequence of rules from the dialogue control file implemented for a flight information domain. The first rule checks for constraints specific to answering a flight query, calling a function that constructs a response soliciting the user for a source airport. The second rule sends the query to a specific content provider to obtain flight information, and the third rule takes output from the content provider and filters for the time constraints mentioned by the user.*

completion, return one of three possible moves: CONTINUE onto the next rule in the table, STOP processing, or RESTART at the first rule.

The dialogue control table provides a useful mechanism for understanding and modifying the response planning process. An excerpt of this table from the flight domain is given in Table 1. The example given shows the inherently domain-specific nature of these files as we originally conceived them. In our previous systems, we exploited a common methodology for designing dialogue control tables and dialogue flow functions, but until now, these commonalities have remained implicit in the code and the tables. The next section will describe how these operations are incorporated into a generic dialogue manager.

## 3. GENERIC DIALOGUE STRATEGIES

### 3.1. Approach

Our dialogue management strategy can be viewed as a process of form-filling. In a flight reservation system such as the one whose dialogue control table is shown in Table 1, the user needs to specify, or the system needs to elicit, at a minimum, a source, a destination, and a date, before any answers can be provided. The underlying "form" for such a domain would contain required slots for these attributes, as well as other, optional attributes, such as airline, time of day, and price. The dialogue manager must coordinate the acquisition of these attributes, query a content provider, and construct a response, either as a paraphrase of the information it received from the content provider or as a system-initiated response, based on its understanding of the state of the dialogue.

The process of acquiring information from a user and constructing a response can consist of a sequence of four phases, and any individual dialogue turn may visit one or more of the phases. In the first phase, *pre-retrieval*, the system must verify the input, check confidence scores, if available, and interpret fragmentary responses in context. Here the system also determines whether sufficient constraints have been elicited from the user to obtain data. The second phase is *retrieval*, where a query is dispatched to the content provider. After retrieving the data, the next phase involves *filtering* the response based on the constraints from the user (e.g., finding the cheapest). Finally, in the *response planning* phase, the dialogue manager, armed with data from the database, determines an appropriate reply to the user. Responses may include follow-up queries to help guide the user. This phase of dialogue management is also responsible for providing help messages, which are usually conditioned on the dialogue state.

### 3.2. Generic Dialogue Functions

To perform the four phases of dialogue management, our approach has been to identify and modularize a suite of generic functions, to be offered by the dialogue manager and invoked in the dialogue control table. Domain-specific information is stored in an external table in a standardized format, whose interface is uniform across all domains. This information is configurable by a developer, and is uploaded in the generic dialogue manager at run-time. We are currently developing a text-based interface for eliciting this information from the developer. The individual dialogue actions that have domain-specific attributes are specified in a template file. Developers edit this file to include the values for these attributes that are relevant in their domains. The file is then parsed into a standard dialogue control table used to drive the dialogue manager.

In the following sections, we will detail some of the operations that are implemented as generic functions. We focus on four features: checking for sufficient constraints, filtering operations, response planning, and user selection.

### 3.3. Checking for Sufficient Constraints

Whenever critical pieces of information required to complete a task are missing, the system should prompt the user for each missing piece. Often, this needs to be in a prioritized order, to make sense pragmatically to a user. For example, when a user requests information about hotels, the logical first question a system would ask is what city the user is interested in. The system might then ask the user to suggest a preferred hotel brand and a date for checking in. Other constraints such as a check-out date and specific amenities might be asked for, as well. We have implemented a single generic function that is called repeatedly, with a list of attributes whose values need to be filled for the particular request. If they are not filled, the function will direct the dialogue to respond to the user with a prompt to demand the missing information. The prompt name is passed into the function via the dialogue control table. The server subsequently constructs a response frame for this prompt. The constraints, the required order and the corresponding prompts are all entered in a template and converted into the appropriate dialogue control table entries at compile time. (Note that the developer is simply specifying an order for eliciting missing information. The compiled system will still recognize and understand this information in any order, as part of larger queries, or as modified in a follow-up query.)

### 3.4. Filtering Database Results

In our system, a database retrieval operation will return a list of items which often require further processing prior to presentation to the user. Filtering involves selecting a subset of the returned list based on conditions specified by the developer or verbally by the user. Following are some of our filtering capabilities:

- In some instances, a database query may return a much larger dataset than required by the user request. Our general filtering function can be configured to look for matches on particular values in the attribute-value pairs returned from the content provider. For example, the user may request a hotel with a swimming pool. Our content provider does not include this particular amenity as a specifiable constraint in a query. We must issue a query for all hotels in the area of interest and then filter out items without "swimming pool" in the attribute-value pairs returned.

- A second filtering feature supplies the ability to reject database items according to some numerical threshold by searching for a given attribute and comparing its value to a threshold. The attribute and threshold may be provided *a priori* by the developer. For example, hotels located more than ten miles from downtown could be rejected when a user has requested a hotel in the city. Alternatively, filtering by a threshold can be triggered verbally by the user. For instance, if a user wants a hotel for less than a certain price per night, our filter can select those that satisfy this requirement on the attribute for price.

- In the same vein, this basic manipulation of data can extend towards selecting a single desired item. We have implemented a generic function that chooses the item with either the maximum or minimum numerical value, corresponding with a given attribute. A developer can use this function to implement an action that allows a user to select the cheapest item. In this case, the developer would specify the price as a attribute to filter on, and the minimum as the condition for selection.

- Finally, the dataset can be re-ordered, prior to presentation to the user. A ranking function is provided so that the developer can select a attribute such as price and an order, such as ascending/descending, in order to present the list beginning with the least/most expensive.

### 3.5. Response Planning

After database retrieval and filtering, the next phase is constructing a system response. Here our goal is to incorporate as much flexibility for the developer as possible. That is, we merely provide a template for the developer to enter the responses an application would produce, given different scenarios. The actual sentences are determined by the developer. During response planning, one of several generic functions is invoked, and a response frame is assembled. The frame is then passed to the generation component to resolve into a text string. The contents of the response frame depends on which generic function was called, and this in turn depends on the dialogue state and data retrieved or filtered. The generic functions are designed to delineate among the following cases: (1) no data is found from the request, (2) exactly one item is found, (3) several items are found, (4) too many items are found. If exactly one item is found, then the response may provide a detailed summary. If several items are retrieved, the system may speak the list of items. And if the number of items exceeds a pre-determined number, the system may suggest the user to provide more specific criteria to prune down the dataset. In addition, as already existing in the current SPEECHBUILDER, the developer can tailor a response for the result of each action, such as answering yes/no to a question about a property of the item. When the system fails to understand the user request, a generic reply is given.

```
{c date :pred {month_date
        :topic {date :ntime 2
                :modifier "weeks later"
                :day "thursday" } } }
```

**Fig. 1**. Example semantic frame for the string "two weeks from Thursday."

### 3.6. User Selection of Post-Retrieval Data

Finally, following presentation of the requested data, the user may select one of the listed items. We have provided the mechanisms for an action that is triggered when a user verbally selects the $n^{th}$ item. This involves tagging the selected item, and updating the context in history to reflect that a selection has been made.

## 4. CANONICALIZING GENERIC CONCEPTS

In developing conversational systems, we have accumulated a rich array of parse rules and interpretation functions for handling the many ways people express commonplace concepts, such as dates/-times and prices. Generally, given an expression (e.g. "costing no more than four hundred dollars,") our systems would derive meaning representations from a hierarchical parse structure. In order to make these more powerful parse rules available to a developer, we have developed sub-grammars for certain concepts that are applicable across many domains. Furthermore, a *Canonicalization* server now operates in conjunction with the generic dialogue manager to interpret strings and substrings from user utterances. Developers can employ these sub-grammars and functionalities to interpret concepts associated with dates/times and prices.

Parse rules for sub-domains (e.g., dates/times, prices) are organized into sub-grammars that are easily embedded into any application. A developer selects the sub-grammar to include, just as libraries of code are included. During run-time, parsing an input sentence involves two stages. A first pass returns a set of attribute-value pairs. Phrases inside a sentence that correspond with dates/times appear as string values to attributes (e.g. *:date*, *:time*.) In the generic server, these attributes trigger a second pass, in which the string value to the *:date* or *:time* attribute is parsed by the richer set of date/time specific rules, outputting a detailed frame-like representation. An example is depicted in Figure 1. This meaning representation is then passed onto the *Canonicalization* server, which returns standardized values.

Our current capabilities can handle a range of phrases in dates and times and price constraints, effectively saving the need for developers to configure these concepts themselves. Dates and times are automatically transformed from their English surface forms to canonical representations such as "FEB 07, 2002" or "begin_time: 1300 end_time: 1600". In prices, a phrase such as "costing less than two hundred dollars" is converted to "costing: less_than amount: 200 currency: dollars."

## 5. AN EXAMPLE FROM THE HOTELS DOMAIN

We have been evaluating the effectiveness of our new generic dialogue manager in the context of a Hotel Information and Reservation Domain. This domain answers queries about hotels by retrieving information such as hotel locations, room rates and amenities available, drawn from the Internet.

We have developed a text-based interface for specifying domain-specific parameters for processing by the generic dialogue man-

```
<dialogue>
<find_extrema :maxmin min :filter_key price>
<filter_db_tlist :filter_key brand
                 :match_condition strstr>
<filter_for_threshold :filter_key miles_distant
                 :threshold 10>
<rank_by_key :rank_key price>
</dialogue>
```

**Table 2**. *This is a partial sequence of parameters elicited for functions called in the Hotels Domain. The italicized values are elicited from the developer and compiled into a dialogue control table.*

ager. An example of this interface can be seen in Table 2. The function calls are specified first, followed by specific attributes in the dialogue state. The values for these attributes are entered by the developer (in the example, the developer input is represented in italics). This template is compiled into a standard dialogue control table, such as in Table 1.

Because the data in the Hotels Domain are gathered from the Web using simple, generic queries whose only constraints are the metropolitan area and the hotel group, the dialogue manager is used extensively for filtering tuples from the content provider. For example, a user may request "a Sheraton hotel," and the content provider will return all hotels in the Sheraton corporate family of hotels. By specifying that the attribute *:brand* be used by the function *filter_db_tlist* to filter tuples, the dialogue manager will match each response from the content provider to the matching attribute in the user input query, i.e., "Sheraton." A loose matching condition (i.e., the user constraint must appear as a substring in the "brand" attribute) is used. The template for eliciting this information from a developer is shown as the second action in Table 2.

The hotel server also uses the threshold filtering function of the dialogue manager to restrict answers to hotels within a user-specified distance from a particular landmark or to hotels below or above a particular price. The line associated with the action *filter_for_threshold* in Table 2 shows an example of an automatically generated rule for restricting larger sets (i.e., over 5) of hotels to ones that are within 10 miles of the designated landmark (the distance itself is computed by the content provider). The values for *:filter_key* and *:threshold* were elicited from the developer and compiled into the application.

One final example of generic functionality is in reordering lists for reading back to the user. The fourth action in Table 2 (*rank_by_key*) is expanded into a dialogue control rule that orders database tuples by the attribute *price*. The firing of this rule causes a new database list to be generated, and furthermore, creates a reply frame that will speak the specified attribute for each entry.

## 6. OTHER DOMAIN APPLICATIONS

We have used the generic dialogue manager to answer queries in four other domains, described briefly below. Each domain uses the following functionality: checking for sufficient constraints, setting up discourse updates, processing system initiatives, and generating replies based on what was returned from the database.

The *Financial Planning Questionnaire* domain serves to elicit information from a user for input to a financial planning software. It follows a scripted dialogue flow, determined by a strict set of constraints that the system needs for developing a financial plan. The *TV Programming* domain knows about television schedules and descriptions of shows, and has relatively complex constraints based on the particular type of information being sought. The *LCSinfo* domain provides access to contact information for the approximately 500 faculty, staff, and students working at the MIT Laboratory for Computer Science and Artificial Intelligence Laboratory. The *Sports* domain can answer queries about games and sports teams, and is currently being developed by a student in the group. It is in the initial stages of development, in which answers responding to user queries can be provided with minimal domain-specific tailoring of a dialogue control file. We anticipate that this will change as the domain becomes more sophisticated.

## 7. FUTURE WORK

We plan on integrating the generic dialogue manager with SPEECHBUILDER, encouraging system developers to use it for their applications. In doing so, we expect to discover many areas for extension. For instance, whenever ambiguity has arisen during conversation, either in misunderstanding of the user's input query or in retrieval of the database response, a function in the dialogue manager should initiate a clarification subdialogue. Similarly, in cases of repeated recognition failures, the dialogue manager could automatically back off to more directed dialogues or entry at the keyboard or telephone keypad. We will also expand the breadth of the Canonicalization server, both to encompass more concepts and cover more ways for specifying those already implemented. These features will enable developers to quickly build applications that can handle commonly encountered semantic concepts, without any need to write grammars or even provide any examples.

## 8. ACKNOWLEDGEMENTS

The authors wish to thank Jim Glass and Stephanie Seneff for critical suggestions and advice in the design and implementation of this work, Stephanie Seneff for writing the initial parse rules that are used by the Canonicalization server for interpreting dates, times, and numbers, and Sterling Crockett for implementing the module that converts the rules template file into a dialogue control table.

## 9. REFERENCES

[1] J. Glass and E. Weinstein, "SPEECHBUILDER: Facilitating spoken dialogue system development," in *Proc. Eurospeech, 2001*, Aalborg, Sept. 2001, pp. 1335–1338.

[2] V. Zue, S. Seneff, J. Glass, L. Hetherington, E. Hurley, H. Meng, C. Pao, J. Polifroni, R. Schloming, and P. Schmid, "From interface to content: Translingual access and delivery of on-line information," in *Proc. Eurospeech, 1997*, Rhodes, Sept. 1997, pp. 2227–2230.

[3] S. Seneff and J. Polifroni, "Dialogue management in the MERCURY flight reservation system," in *Proc. ANLP-NAACL 2000 Satellite Workshop*, Seattle, May 2000, pp. 1–6.

[4] S. Seneff, R. Lau, and J. Polifroni, "Organization, communication, and control in the GALAXY-II conversational system," in *Proc. Eurospeech, 1999*, Budapest, Sept. 1999, pp. 1271–1274.