A DECLARATIVE FRAMEWORK FOR BUILDING COMPOSITIONAL DIALOG MODULES

William Thompson, Harry Bliss
Motorola Labs
Schaumburg, IL 60196 USA
{thompson,bliss}@labs.mot.com

ABSTRACT
Rapid development of spoken language dialog applications requires a domain-portable dialog framework and the ability to re-use dialogs which have already been created. Satisfying this goal should not come at the expense of dialog flexibility and naturalness. Current frameworks which enable dialog re-use typically do so by linking dialogs sequentially in a finite-state graph, producing a rigid dialog flow. This paper describes a framework for specifying dialogs that both enables re-use and maintains flexibility and naturalness. This is accomplished through nesting frame descriptions. The structure of a dialog is compositionally determined both by its own top-level frame description and by the frame description of the dialogs which are nested within it.

1. INTRODUCTION
In recent years, spoken dialog systems have moved beyond the laboratory and into the real world. An ever increasing number of systems are being deployed as cost-effective means for conducting transactions or accessing information [4]. These systems are used in a wide range of domains, including call routing, stock quotes, movies, weather, restaurants, traffic, and airline information [5, 9, 14, 16]. The number of deployed systems will undergo continued growth due to the creation of the wireless internet and corresponding demand for voice access to web content [18]. The widening range of applications and application domains brings into sharp focus the need for tools and methodologies enabling the rapid development of spoken language dialog applications. This need will become particularly acute as dialogs increase in sophistication, and application domains increase in complexity.

Rapid development of spoken language dialog applications can be addressed with a domain-portable framework. One approach to portability is to take advantage of the fact that certain sub-dialogs are shared across multiple application domains. For example, time, date, and city sub-dialogs potentially appear in many different applications, including (among others) flight reservation and movie showtime information systems. It is possible to factor out these sub-dialogs into independent dialog modules, which can then be used as building blocks in multiple dialog applications.

At Motorola a framework is under development in which such dialog modules can be nested hierarchically, reflecting the hierarchical structure of an application domain. In this approach the structure of a dialog is compositionally determined both by its top-level declarative frame description and by the frame descriptions of the dialogs which are nested within it. Combined with a domain-independent dialog manager (DM), this approach enables both a measure of domain-portability as well as flexible, mixed-initiative dialog flow.

2. APPROACHES TO DIALOG MANAGEMENT
Various groups have adopted the approach of packaging coherent units of dialog into independent dialog modules, which can be used as building blocks for constructing larger spoken language dialog applications [8, 10, 14]. As one example, Nuance Communications has created an open source framework for developing re-usable dialog modules, termed “SpeechObjects” [10]. A SpeechObject is a Java software component which implements a dialog state and packages together a set of grammars, prompts, and semantic slots.

In the SpeechObject framework, dialog re-use is accomplished by linking SpeechObjects together sequentially in a finite-state graph (see Figure 1). This graph takes over the role of a DM, since the states and their transitions determine the flow of the dialog [1].

![Figure 1: Dialog modules linked together into a finite-state graph. Note that Source and Destination are two instances of the same sub-dialog.](image-url)
While this approach successfully tackles the problem of dialog re-use, it has drawbacks which are common to all graph-based dialog systems [12]. Dialog interaction in a graph-based system is analogous to that of a menu-based IVR system, where navigation of the state space is rigidly determined by the pre-existing state transitions. Additionally, since each state transition must be manually specified, graph-based systems are difficult to maintain as the number of dialog states grows.

Frame-based systems are a common alternative to graph-based systems. A frame consists of a set of feature-value pairs, where a value may either be atomic or complex (i.e., another set of feature-value pairs). A frame-based DM relies upon the current state of its frame in order to decide what to do next. A frame-based system is therefore driven by the information state of the dialog it is currently engaged in [17]. This leads to greater flexibility and domain-portability, since the DM exists independently of the states of the dialog.

A frame-based system can be implemented in combination with a graph-based approach. Each frame, of arbitrary size and complexity, can be associated with a node in a graph. The larger the size of the frames corresponding to the states, the fewer states and state transitions need to be specified. However, there is a trade-off between flexibility and re-usability. Larger states are more flexible, but they are also less re-usable, since they are typically more specific to a given application domain. There is also a trade-off between the size of a state and recognition performance, since larger states lead to less constrained user input.

3. NESTING FRAME DESCRIPTIONS

An alternative form of combining frames becomes possible if frames are hierarchically structured, rather than flat. In this situation, frames are composed via nesting, instead of being linked sequentially.1 In the nesting approach, frame structure reflects the structure of the application domain, which is generally also hierarchically structured. A frame can therefore be seen as a kind of domain model [13].

Systems which employ hierarchical frames combine the advantages of frame-based systems (flexibility and domain-independent DM) along with a natural ordering on the dialog flow, reflecting the hierarchical nature of the application task. Since the frame structure is essentially a tree, a default ordering of dialog topics can be induced by a (depth-first) tree traversal. This obviates the need for manually specifying state transitions. The system still retains flexibility, however, since the default ordering can be over-ridden by user initiated statements and requests.

A system built on frames, even hierarchical ones, still has limitations. Frames are static data structures, making them inadequate for applications in which the desired output is a dynamic product resulting from user collaboration. For example, booking a flight reservation involves an unspecified number of flight legs, each of which must be individually negotiated with the user [12, 19].

The approach described in this paper deals with this limitation by replacing frames proper with frame descriptions, which can be partial (i.e., underspecified). There is a one-to-many mapping from frame descriptions to frames, and the content of a frame description can be modified during an interaction with a user. Thus, the product of a dialog can vary across interactions. Figures 2 and 3 provide examples of frame descriptions, depicted in standard AVM notation [11].

![Figure 2: AVM representation of airport frame description.](image)

![Figure 3: AVM representation of flight frame description.](image)

Formally, a frame description is a typed feature structure [2], containing both domain information and dialog information (such as grammars, prompts, and goals). Specifying frame descriptions as typed feature structures makes it possible to compose them using unification. For example, the airport_frame in Figure 2 can be unified with either or both of the values of the SOURCE and DEST features in Figure 3, since these are underspecified objects of type airport_frame.

The value of the CONTENT feature is a feature structure representing an application domain object such as a flight or an airport. This feature structure can embed other feature structures representing other frame descriptions. For example, the SOURCE and DEST features in Figure 3 take airport_frame feature structures as values. The remaining features of the frame description, PROMPT, GRAMMAR, and GOALS, are used...
by the DM to guide an interaction with a user.

The PROMPT feature is used by the DM to prompt the user to supply a value (or range of values) for the CONTENT feature contained in the same feature structure. PROMPT values consist of expressions written in a Prompt Specification Language (PSL), which contains conditional logic, operators, and variables referring to portions of the feature structure. Evaluation of a PSL expression results in context dependent natural language output. For example, evaluating the PSL expression in Figure 2 will produce an output in which \[ \text{list} \] is replaced with a comma-separated list of the range of values currently specified in the NAME feature of the \textit{airport} feature structure.

The GRAMMAR feature contains the location of a grammar file to be used for speech recognition and language understanding at the stage in the dialog in which the current domain object is under discussion. In the current framework, these grammars are written as context-free semantic grammars, specific to the domain of the given feature structure.

The slashes around both the PROMPT and GRAMMAR feature values in Figure 2 indicate that these are default values, which may be over-ridden through unification. For example, the SOURCE and DEST features in Figure 3 each specify their own PSL expressions for the airport values they are to be unified with. These expressions will replace the default PSL expression specified for the PROMPT feature in Figure 2.

The value of the GOALS features is a disjunction of conjunctions, in which each conjunct consists of a reference to an embedded frame description. A conjunct is defined as true if the feature structure it refers to has exactly one value specified for its CONTENT feature, otherwise it is defined to be false. A frame description is said to be satisfied iff one of the disjuncts in its GOALS feature is true. The notion of frame satisfaction is recursive, since a frame description may contain nested frame descriptions with their own sets of goals.

Frame satisfaction is used by the DM to determine if an interaction with a user has or has not reached a successful conclusion. In this framework, the goal of a dialog interaction is to obtain sufficient information from a user for the application back-end to perform some operation, such as making a database query or accessing information from the web. The GOALS feature lets the system know when the frame description can be considered “complete” in order for further processing to take place.

Feature values may consist of lists of feature structures. List-valued features make it possible to model the dynamic flight leg scenario mentioned previously. Figure 4 shows a fragment of a frame description for a flight reservation system. The \textit{flight_legs} feature takes a list value, with members of the list being of type \textit{flight_frame}. Since the list value is underspecified for the number of elements it contains, it maps onto frames with an arbitrary number of actual flight legs. The number of flight legs and their content are determined during the course of a dialog interaction with a user.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{fig4.png}
\caption{Fragment of AVM for travel frame description.}
\end{figure}

\section{4. DIALOG SYSTEM IMPLEMENTATION}

One major design requirement of Motorola’s dialog system architecture is to enforce a clean break between domain-specific information and domain-general processes. To this end, domain-specific information is declaratively specified in separate files. These files include frame specifications, grammars, and databases containing sets of application domain objects (e.g., a set of cities or flights).

The DM contains a Natural Language Understanding (NLU) module, which processes user input with the aid of a robust parser, producing a frame description as output. After processing a user input, the DM attempts to update its own frame representation by unifying it with the output of the NLU module. Next, the DM attempts to automatically complete its frame description by inferring the values of features based on the domain knowledge it contains in its associated domain database. For example, if a user specifies “portland” as the name of a city, and the associated city domain database contains two instances of cities with this name (e.g., in Maine and in Oregon), the DM will specify that the value of the STATE feature is the disjunction of these two values.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{fig5.png}
\caption{Dialog System Architecture. Dotted lines indicated modules under development.}
\end{figure}

During a dialog with a user, the DM traverses the frame description, recursively checking that each of the specified goals is satisfied. When the DM encounters an unsatisfied goal, it prompts the user to supply the missing information, using the PSL expressions which are associated with the frame description features. The interaction ends when the DM has determined that its frame description has been satisfied, or if the user terminates the conversation.
5. FUTURE PLANS

The dialog system described here is in its early stages, and a number of modules are still under development. Future versions of the DM will have the capacity to perform a limited form of plan recognition, by matching user input with a set of compatible goals. If this set contains more than one member, the system will negotiate with the user to determine which of these goals is the correct one.

A natural language generation (NLG) module is currently being developed, which will provide a general-purpose alternative to the functionality currently handled by the PSL. This module will be specifically designed to take domain model representations as input.

A reference resolution module will provide a more sophisticated linguistic interface for dealing with pronominal anaphora, definite descriptions, and other presuppositional phenomena [7]. The ability to handle these phenomena will result in more natural and user-friendly dialog.

There are also plans to add a user model to the system, in order to enable behavior which is customized to an individual’s preferences and history of interaction with the system. Individually customized behavior should produce dialogs which are more efficient on average, since the system and the user will not need to negotiate information which is shared across multiple interactions (e.g., the user’s credit card number or home address).

6. CONCLUSION

In this paper, a framework is described for specifying dialogs which both enables dialog re-use and maintains flexible dialog flow. This is accomplished through nesting frame descriptions. The structure of a dialog is compositionally determined both by its own top-level frame description and by the frame descriptions of the dialogs which are nested within it.

The frame description approach is more flexible than traditional frame-filling approaches. Whereas frames are static structures with a predetermined list of slots, the approach described here permits frame descriptions which are underspecified. Given an underspecified description, a full specification of the frame structure is negotiated with the user of the system, in accordance with declarative constraints on possible frame structures.

7. REFERENCES