Automating Spoken Dialogue Systems

Mona Singh,[1,2⋆] James Barnett,[2] Munindar P. Singh[1⋆⋆]

[1] Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA

[2] Dragon Systems Inc.
320 Nevada Street
Newton, MA 02160, USA

mona_singh@ncsu.edu

Abstract. Spoken dialogue interfaces apply in a number of applications. Engaging in meaningful conversation presupposes the ability to recognize and generate different conversational moves, and to adaptively carry on a dialogue. Although the portability of dialogue interfaces is highly desirable, few current approaches address it seriously.
We describe a portable toolkit for constructing spoken dialogue interfaces. We present the representations and techniques used to customize an interface to a particular domain and application. Our approach relies on shallow knowledge of the domain, and interprets a rule-based model of the dialogue.

1 Introduction

Spoken dialogue interfaces have been widely recognized as being highly desirable for a number of applications [1, 6]. There are two important dimensions of a user interface: user-friendliness and developer-friendliness. The former is improved through *mixed-initiative* dialogue, and the latter through *portability*. Mixed-initiative dialogue means that the system dynamically shares control of the dialogue with the user. Portability refers to ease of applying the interface to a new domain or application (portability across languages has to do with lower-level aspects, and we ignore it here).

Current systems tend to be limited along both of the above dimensions. They are frequently system-centric in requiring rigid control of the dialogue, and frequently tied to a specific application [7], such as air travel information or train schedules [1]. Although components of these systems might be portable, there is clear need for *toolkits* to facilitate system construction [6].

⋆ Mona Singh and James Barnett are supported by the National Institute of Standards and Technology under grant NIST-70NANB5H1181.

⋆⋆ Munindar Singh is supported by the NCSU College of Engineering, the National Science Foundation under grants IRI-9529179 and IRI-9624425, and IBM corporation.

Before we come to the technical details, we outline our major design criteria. Our work is being performed in the research division of a speech recognition products vendor. The applicability of our research to products is an essential factor. Our goal is to market a commercial toolkit that can be used by programmers to build applications of interest to end-users (some of these applications are built in-house). These programmers would customize the toolkit as appropriate by acquiring knowledge, and developing application programs. Importantly, as a commercial reality, the programmers using the product are not expected to be specialists in AI or linguistics (Fraser & Dalsgaard make a similar observation [6]). Thus, the complex models used in classical knowledge-based systems are not feasible. This echoes a sentiment from recent work in information retrieval and text extraction [4]. Consequently, we have been forced to make particular design trade-offs in attempting to maximize user-friendliness (e.g., as reflected in mixed-initiative dialogue) without compromising developer-friendliness (e.g., as reflected in portability). We believe, however, that our considerations are not far different from toolkit approaches in general.

The design of a portable toolkit poses special challenges. We must ensure that the toolkit is generic, and can be readily customized to a wide variety of applications. On the other hand, full automation is not required—wizards that suggest good defaults to developers can help increase their productivity. Lastly, there is an incentive to use the most robust, rather than the fanciest, techniques in the toolkit.

This paper focuses on the adaptive management of conversational moves. Our contribution is in an approach that facilitates the developer's task

- without rigid control of the user's actions, and
- using a not-very-complex knowledge representation.

Section 2 describes our system architecture, and the models necessary to customize it. Section 3 presents the generic parts of our toolkit. It describes the conversational moves and dialogue controller, as well as an algorithm for proposing utterances for the conversational moves. Section 4 presents a sample dialogue. Section 5 presents the key features of our toolkit.

2 Architecture and Key Models

Our system is customized to a specific application and domain through the following main models. For concreteness, we consider the application of giving healthcare advice.

2.1 Domain Model

Since a dialogue system must support a conversation between a human and an information system, it is important to model the structure of the latter. The Domain Model (DM) does just that. It provides a static *conceptual* model of the domain, which includes the key concepts and their interrelationships.

A conceptual model of the data is easy to understand, and is a key step in developing an information system [3].

Fig. 1. Simplified Domain Model for Healthcare

The DM is expressed using a variant of the Entity-Relationship (ER) approach [3]. The ER approach diagrammatically captures a high-level view of the data, independent of its physical storage. Figure 1, which is loosely based on a very small part of a healthcare advisory model [2, pp. 70–232], illustrates the key ideas. The basic concepts of ER diagrams are *entities*, *attributes*, and *relationships*. An entity describes a class of real or abstract objects that are worth talking about—e.g., *patient*. An attribute gives some simple, direct properties of entities—e.g., *name*. A relationship defines a specific relation among entities—e.g., *suffer from*. We include information on the cardinality of a relationship, and whether participation in it is required or optional.

Entities can be subclasses of other entities; they inherit the attributes and relationships of their superclasses. Importantly, we attach *action* descriptions to the ER model. The actions describe what operations a user may carry out in the given application—e.g., *assess risk*.

Thus, the DM gives the main concepts of interest to an application, but

does not supply any of the background common-sense knowledge. Indeed, ER models are small—typically, proportional to the number of tables or columns in a relational database, for instance. Thus, we only require shallow knowledge, which limits the cost of porting the system to a new domain [4].

Identifiability. To generate utterances for the conversational moves, we must distinguish between the attributes that the users typically know, e.g., *temperature*, and those they don't, e.g., *problem or disease*. This knowledge is given in the DM. The system only queries about attributes and entities that are known to the user.

Independent existence. Some concepts may be *dependent* on other concepts. For example, *symptom*s are well-defined only for a given patient, not in themselves. Dependence is important, because it influences how we refer to a concept during a dialogue. Dependent concepts are shown as double boxes in Figure 1.

Relational logical model. We assume that the DM is in a cleaned up form in which all relationships are binary, with cardinality 1-1, 1-M, or M-1.

2.2 Transaction Model

The transaction model (TM) specifies the desired actions, which are the domain-specific operations that users are allowed to perform. The TM is essentially specified as a nested template or form, which specifies the slots that users may fill and the operations they may invoke. The slots and operations are jointly termed transaction nodes or Tnodes.

Actions are specified by giving their *procedure name*, their *input* slots, *outputs* slots, and a specification of which of the inputs are *vital*. The TM also includes a specification of the likelihood of (a) different slots being filled, and (b) different operations being invoked. These likelihoods are used in disambiguating the users' utterances when necessary. The TM representation maximizes the flexibility with which the slots are filled, which is crucial in enabling mixed-initiative dialogue.

In order to carry out an effective dialogue, it is also important to model the dynamic properties of the different actions. For this reason, we incorporate the abstractions of commit, rollback, and checkpoint as conversational moves in the toolkit. The TM specifies where and how to incorporate them, based on the application and the view being presented to the users.

2.3 Dialogue Model

The dialogue model describes the desired properties of the conversation with the user. These are given in terms of user and system rule-sets that specify the *dialogue control strategy*, which encodes when and how the different conversational moves are exercised in the dialogue. Confusable slots may be set up for paraphrasing instead of verification, since paraphrasing is likelier to resolve ambiguity more reliably [11]. Factors such as the level of confidence in the speech

input and the anticipated confusability of the subslots can influence the selection of moves [9].

The dialogue model carries the functionality to determine the meanings of the users utterances, and to deal appropriately with the user. Some of this functionality is simply derived from the specifications provided by the developer in terms of the DM, TM, and the grammars for the various slots. However, a significant component of the dialogue is conducted based on more flexible specifications of dialogue control strategies, and of the allowed conversational moves. These specifications are given as rule-sets.

3 Generic Components of the Toolkit

The above models are interpreted by the *dialogue controller*, which is the core of the toolkit. At run-time, the system constructs internal representations that involve dialogue nodes (Dnodes) linked to appropriate Tnodes—operations and slots. The Dnodes are automatically generated as needed for each Tnode in the TM. The system maintains a history of the ongoing conversation in terms of the Dnodes that are instantiated. The history is used to determine the appropriate rules to apply at a given stage in the conversation.

3.1 Conversational Moves

A variety of conversational moves and coherence relations have been proposed to explain the structure of discourse [10, 12]. While useful, these moves are not sufficient for transaction-oriented applications, especially with spoken interfaces [8]. While researchers acknowledge this fact, traditional moves may not even cover simple database access applications.

The moves should be powerful enough to represent the dynamic properties of the underlying actions. For example, making an appointment requires that the user commit to the agreed upon time; if the user is a no show, she may be liable for charges. Similarly, the interface should be able to guarantee that the appointment is in fact made. This limits the flexibility in other respects. For example, if a patient want to schedule two separate appointments in the same time-frame, she must be able to hold a spot without committing to it, or be able to cancel her commitment (so as to reschedule). Similarly, the user might want to checkpoint (or bookmark) a state of the dialogue to be able to return to it in case of error or confusion. The toolkit provides the above abstractions as conversational moves; the dialogue model specifies where and how to incorporate them, based on restrictions imposed by the TM.

We propose seven classes of conversational moves. For reasons of space, we do not describe all the moves in detail, but give examples of each class below.

Transactional. These moves deal with the backend system, and are inherently asymmetric between the user and system. For example, a user's request for information is treated as invoking an operation—a query. Transactional moves include

request for information, *information*, *offer to perform an operation*, *invoke an operation*, *undo an operation*, and *further development*.

Authorizational and Commissive. Many applications require the user to explicitly authorize the system to take some action on his behalf. This authorization might involve the user committing some transactions, with social or economic effects beyond the dialogue. Authorizational moves include *offer to commit* and *commit*.

Error Detection and Correction. These are moves through which the system and the user find and remove miscommunications and misunderstandings. Error detection and correction moves include *request for verification*, *paraphrasing*, *request for confirmation*, *verification*, *confirmation*, *correction*, and *restatement*. In principle, these could all be performed by the system or the user. Some of these, e.g., *paraphrasing*, if attempted by a user, might be difficult for the system to understand.

Interactional. These moves enable the system and user to maintain their conversation history. Interactional moves include *request for location*, *location*, *topicalization*, *request for options*, *options*, *return to prior context space*, *offer to checkpoint*, *checkpoint*, *rollback*, *interruption*, and *silence*.

Presentational. These are moves through which the system gives information to the user. The realization of presentational moves depends on the media available for interaction. In general, we assume that either the presentation can be of (a) some atomic entity, possibly large like a picture, or (b) some structured entity, such as a set of documents. Atomic entities can be viewed based on their physical components; structured entities can also be viewed based on their logical structure. A generic toolkit can only include a means to present the results serially (temporally or spatially), and a means to identify components. Presentational moves include *request a component*, *present logical component*, *request scrolling*, *present physical component*, *serialize*, *request attention*, and *grant attention*.

Assistance. These are moves through which the system helps the user in understanding the meaning of the different steps in the given dialogue. Assistance moves include *request for explanation*, *explanation*, and *feedback*.

Sessional. These are moves through which the user and system initiate, resume, and close their conversations. Sessional moves include *initiate*, *authenticate*, *request preferences*, *set preferences*, *close*, and *resume*.

3.2 Dialogue Controller

The dialogue controller is the heart of our system. This is the module that interprets the application-specific models, and orchestrates the conversation with the user in a way that respects those models.

To ensure mixed-initiative dialogue, the controller gives the user an opportunity to speak (and allows interruptions where possible). This is processed during the *user phase*. When the user relinquishes control of the dialogue, the *system phase* is initiated.

User phase. The user phase invokes the recognizer with a list of possible grammars for the nodes (slots or actions) in the TM. The recognizer produces a list of interpretations along with the matching grammar and (depending on the recognizer) the confidence in the match. The controller picks out the node yielding the best interpretation. The best interpretation is determined by a metric that prefers the best confidence and highest probability of occurrence—the latter is based on the probability of a particular node being the next node. From the selected slot, the controller applies the rules describing the user interactions (these are as encoded in the user-rules rule-set).

System phase. In this phase, the controller selects the best target action based on the current probabilities. Unlike in the user phase, no slot can be selected. Once the target action is determined, the controller checks if enough is known to execute it. Specifically, if the vital inputs for the action are not known (at a high enough confidence level), the controller attempts to obtain those values from the user. When the values are obtained, the system executes the appropriate procedure and presents the results. The execution can involve checkpointing and committing, which as explained in section 3.1, gives the user an opportunity to cause backend system actions in a controlled manner.

History matching. The apply-strategy module takes a node, a history, and a rule-set to determine what to do. The rule-set declaratively captures the intended behavior of the system. The rules are if-then rules with a pattern in the antecedent and an action in the consequent. We allow rules with antecedents that have a predicate-variable structure, and provide restricted constructs for the temporal aspects.

The rule-set is matched against the history to find the best (i.e., most specific) match. The action of this rule is executed on the given node. We allow actions for each of the conversational moves. The details of matching are beyond the scope of this paper, but we present a partial set of rules for verification, simplified and sanitized for ease of exposition.

If you are at a node and the confidence level is <0.5 and the node is verifiable, then verify the node. This rule can match any node whose name is bound to ?node. ?value is its likeliest value and ?conf is the corresponding confidence in the recognized speech. This is the most basic rule for verifiable Tnodes.

```
if (Tnode ?node (?value ?conf))
    AND (?conf < 0.5)
    AND (verifiable?  ?node)
then VERIFY ?node
```

Thus rules that match the dialogue history are executed and determine what the next conversational move should be.

3.3 Move Generation

Based on the action desired by the user, the dialogue control strategy determines a specific system move. This invokes the move generation module to produce the appropriate utterance. The generation module uses knowledge encoded in the DM and TM. In practice, a developer would design—and fine tune—the DM and TM concurrently with testing the generation. We discuss certain aspects of the DM that are especially important to generation, following which we describe our strategic and tactical algorithms.

Handle. An effective dialogue requires that there be something that the system and the user can both refer to, and assume the other party will know. We call this default starting point the *handle* of a dialogue model. As a last resort, the handle can be taken to be the user himself—the meaning of *you*. We should then be able to ask who are you and what do you want. Good DMs would have an entity for the user, or for something equally salient, such as the *patient*. In selecting pronouns, it matters whether the conversant and the *patient* are the same person.

Relative significance. Knowledge of the relative significance of entities is used in choosing utterances appropriately. In principle, this knowledge is part of the dialogue model, although it is more practical to put in the DM, where the entities are defined. In general, identifiable entities are more significant, as are entities that exist independently (as explained in section 2.1).

3.3.1 Strategic Algorithm The strategic algorithm attempts to generate utterances necessary to have the dialogue proceed in a manner that respects the TM, while allowing the user to share control with the system. This algorithm is essentially heuristic graph search. For informational prompts, we assume we know a relevant action. If all its vital inputs are known, we can proceed with it. Otherwise, we make a list of all missing inputs (even the nonvital ones). We find the entities these attributes occur in. These entities become the terminal points. We use a number of heuristics to identify likely starting points, and to search for a set of paths from the starting points to the terminal points. We lack the space to include these heuristics here.

Given a path, we proceed along it by querying for entities along it. To move from an entity to the next, we use the intervening relationship to generate the relationship prompts, e.g., *What is the patient's nurse's name?* When we arrive at a terminal entity, we query the user for the necessary attributes of that entity. For this purpose, we use the tactical algorithm.

3.3.2 Tactical Algorithm The new entity's identifier is first introduced by referring to it by a linguistic construction that relates both the known with the unknown entity. When the new entity has been identified, the tactical algorithm is invoked to query the remaining attributes.

Consider informational prompts. Such prompts need to be generated for all slots. In order to generate natural language utterances from the DM, we need heuristics to generate queries to obtain information about specific attributes and whole entities via their identifiers. Assuming our DM states that patients take medications, we can generate utterances such as *What medication is the patient taking?* The other moves are similar.

The tactical rules apply when the entity's identifier slot has been filled and the entity is the active referent. For entities that are not current, but were previously identified, we use the strategic algorithm to "revive" them in the dialogue.

Our algorithm essentially derives simple templates from the DM. It proposes a set of utterances that the developer can choose from. Further, we use heuristics to choose among the various templates during a dialogue. The idea is to use varying templates even for successive instances of the same conversational move. This helps reduce the monotony of the dialogue.

4 An Example Dialogue

We present a brief example of a dialogue. Here **U** refers to the user and **S** to the system. Comments are in the typewriter font.

`User calls in.`
U: *I have a problem and I was wondering if I could get some help.*
S: *What kind of a problem do you have?*
U: *Well, I have pain in the chest.*
S: *Do you have chest pain?*
`verifying a vital slot.`
U: *Yes.*
S: *What kind of pain is it? Is it dull and persistent?* `Enumerating the possible values of the attribute` *kind*. `All possible values are prompted for till the first positive response.`
U: *No.*
S: *Is it stabbing?*
U: *No.*
S: *Is it crushing?*
U: *Yes, it's sort of crushing.*
S: *When did the crushing pain begin?*
`Confirmation of the kind of pain`
U: *Yesterday afternoon.*
...

5 Conclusions

Our toolkit can conduct mixed-initiative dialogue and is portable with relatively little effort. Our approach has the following distinct features. It

- models the dialogue control strategy as rules instead of finite state automata, which is typical, e.g., [5].
- incorporates reasoning about the confidence levels of the recognizer, if such information is available.
- dynamically determines desired actions based on user input.
- incorporates the ability to commit and rollback.
- helps generate utterances for moves.

Well-designed information systems almost always have ER models associated with them already. Several commercial products exist to help construct good ER models for database applications. We are considering enhancements to a product to include actions, TMs, as well as features to display the possible utterances for different moves. We are presently experimenting with ERWin^{TM}, which is a leading product. However, our approach is independent of any such tool.

References

1. James Allen, Bradford Miller, Eric Ringger, and Teresa Sikorski. A robust system for natural spoken dialogue. In *Proc. 34th Meeting of the ACL*, 1996.
2. AMA. *The American Medical Association Family Medical Guide.* Random House, 1987.
3. Carlo Batini, Stefano Ceri, and Sham Navathe. *Conceptual Database Design: An Entity-Relationship Approach.* Benjamin Cummings, 1992.
4. Jim Cowie and Wendy Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, January 1996.
5. N. Dählback and A. Jönsson. An empirically based computationally tractable dialogue model. In *Proc. 14th Conference of the Cognitive Science Society*, 1992.
6. Norman Fraser and Paul Dalsgaard. Spoken dialogue systems: A European perspective. In *Proc. International Symp. Spoken Dialogue*, 25–37, 1996.
7. Masato Ishizaki, Yasuharu Den, Syun Tutiya, Masafumi Tamota, and Shu Nakazato. Classifying dialogue tasks: Task oriented dialogue reconsidered. In *Proc. International Symp. Spoken Dialogue*, 37–40, 1996.
8. Arne Jönsson. Dialogue actions for natural language interfaces. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1405–1411, 1995.
9. C. Proctor and S. Young. Dialogue control in conversational speech interfaces. In M. Taylor, F. Neel, and D. Bouwhuis, editors, *The Structure of Multimodal Dialogue*, 385–399. Elsevier, 1991.
10. R. Reichman. *Getting Computers to Talk Like You and Me.* MIT Press, 1986.
11. S. Tanaka, S. Nakazato, K. Hoashi, and K. Shirai. Spoken dialogue interface in dual task situation. In *Proc. International Symp. Spoken Dialogue*, 153–156, 1996.
12. David Traum and Elizabeth Hinkelman. Conversation acts in task-oriented spoken dialogue. *Computational Intelligence*, 8:575–599, 1992.

This article was processed using the LaTeX macro package with LLNCS style