Enhancing Existing Form-Based Dialogue Managers with Reasoning Capabilities

Dirk Bührer

University of Ulm
Department of Information Technology
Ulm – Germany
dirk.buehler@e-technik.uni-ulm.de

Abstract

In this paper we present an approach to complement speech dialogue applications written in VoiceXML with reasoning capabilities simplifying the integration of domain knowledge into the traditionally coded dialogue flow. One of the characteristic enhancements of our approach is the use of a dedicated reasoning component for processing the knowledge items acquired during the dialogue. The approach is also novel in that, instead of building on top of a specialized research dialogue architecture, an industry-standard dialogue description language has been chosen as the basis. This enables enhancing existing voice application rather than starting from scratch. The paper describes a first implementation of the proposed interface between the traditional form-based dialogue manager and the reasoning module and includes the analysis of a sample dialogue conducted with a prototype version of the system.

1. Introduction

In many task-oriented speech dialogue systems, a frame or form-based dialogue processing algorithm (e.g. [1]) is responsible for acquiring the relevant pieces of information for solving a certain task, i.e. it collects values for the individual task parameters represented as form items. This form interpretation algorithm (FIA) is in particular responsible for choosing an appropriate item to query, playing prompts to the user, activating the necessary speech recognition and interpretation grammars, and handling any results or errors.

Typically, information items are grouped together in structures such as forms or frames, because they play certain related roles w.r.t. the task and their values are logically related, i.e. they have to satisfy certain logical rules or integrity constraints. Consider, for instance, a scheduling task where the end point of a meeting interval is always later than the starting point. In such a scenario, it does not make sense to allow the user to specify time values that violate this rule. Instead, the user should be notified about the violation, if any, and be given a possibility to repair the situation. Redundancy in the information items is also common, e.g. there may be individual items for the start, the end and the duration of a meeting. While two of these items would be sufficient to represent the information, the redundant third item is added in order to allow more flexible user interaction.

However, industry-standard human-machine dialogue description languages such as VoiceXML do not provide adequate support for automatically taking advantage of such relations. Instead, the current solution is to manually code procedures in a scripting sub-language such as ECMAScript for ensuring that in any situation the constraints are met, i.e. that values are propagated and conflicts reported. However, the complexity of these scripts for non-trivial logical interconnections quickly becomes a problem. In particular, the scripts have to handle a large number of situations (which is in the worst case exponential in the number of items involved), because they depend on the order in which the items are provided in a mixed-initiative interaction. As a result, maintaining consistency of such scripts soon becomes unmanageable.

Our approach is different from this ad-hoc scripting approach: We use a dedicated reasoning module (RM), acting as an inference engine that applies the relevant rules contained in an application-specific rule base to the current information state of the dialogue [2]. On the basis of the information provided by the user, the RM infers additional assertions, aiming at a cooperative deductive behavior. Consequently, the handcrafted procedures of the scripting approach may be replaced by significantly simpler and more maintainable code that invokes the RM.

The remainder of this paper is structured as follows: in section 2, we outline the interface between the form-based dialogue management and the reasoning module which is the core of the solution for the problems outlined above. In section 3, we present an overview of the technical realization of the forms/reasoner interface. An example dialogue illustrates its essential functionalities. Finally, after reviewing related projects in section 4, we conclude with an outlook on further research on top of the proposed approach.

2. The Forms/Reasoner Interface

In our approach, the form interpretation algorithm is to synchronize its information state with the reasoning process, i.e. it notifies the reasoning module whenever information is asserted or retracted (i.e. form items filled or cleared). Con-
versely, it is notified by the reasoning module when a deviation from the standard course of form interpretation might be necessary, e.g., in case of a detected conflict. Concerning redundant form items as in the example illustrated above, the reasoner is able to infer the value of the missing item using a single consistency rule that is applied independently of the order in which the user fills the items.

In order to describe the interface between the reasoning module and the form interpretation algorithm, we first review some of the internal workings of the RM.

2.1. Reasoning Procedure

The rule base in the RM consists of a set of range-restricted first-order logic (FOL) clauses. Each clause is a disjunction of literals, the negated ones of which may be regarded as the rule’s preconditions (antecedents) and the positive ones as possible postconditions (consequents). We employ a finite model generation technique [3] to derive for any set of initial assumptions, saturated sets of assertions that conform to the rule base or a proof of inconsistency. The inference algorithm incrementally constructs a tree-like tableau structure that represents alternative partial solutions (called contexts). Each inference step is recorded in proof structures, so that all proofs may be traced back. This is important for conflict analysis, for instance. When a conflict occurs, the respective assumptions are recorded as a emph-no-good (a set of assertive solutions proven to be inconsistent).

Figure 1 shows a very simple rule base consisting of two clauses. Antecedents of the rules are marked with a leading minus sign. Variable names start with a question mark. The rule base captures the relation between the start, end, and duration of a time interval, e.g., for a scheduling application.

The proof structures generated by the RM contains two types of proof steps: The first one is forward inferences (modus ponens) where a certain rule from the rule base is applied to a set of matching facts, resulting in a set of alternatives (possibly empty in the case of a conflict). The other type is elimination inferences where a no-good is applied to remove a possible alternative consequent.

In contrast to other approaches [4, 5] that view domain reasoning as a constraint satisfaction problem (CSP), we think that model generation may lend itself even to more complex problems, such as planning (e.g., by inferring the existence of action events in the generated models).

2.2. Synchronization

The form interpretation algorithm’s and the RM’s information state have to be synchronized. The FIA’s state consists of the values of the individual form items. Upon changes in the form, the item values are mapped to logical assertions (in this case, ground atomic formulas) in the RM’s FOL language using the item name as a predicate.

The RM’s state, consisting of complete proofs and possibly alternative solutions, is much richer than the FIA’s. Thus only a subset can be made available to the FIA. Currently, the following situations are signaled by the RM:

1. The most important event is the detection of a conflict: A conflict is a situation where the values provided by the form imply a logical inconsistency when combined with the rule base.

2. A conflict is in fact only a special case of the more general event when an assertion (such as a value constraint) was inferred from previous assertions and inferences. When mapped to the FIA’s state, an assertion may fill a certain form item with a value.

3. If the initial assertions (or assumptions) provided by the FIA are under-specified, the RM generates alternative solutions. In that case assertions provided by the RM represent suggestions (e.g., possible values) rather than strict inferences.

Other classes of inferences such as disjunctions of values or no-goods are currently not communicated to the FIA, since in VoiceXML there is no standard way of dealing with sets of values for items or negated values.

2.3. Relevance Decision

A crucial question for the interface between the RM and the form interpreter is how to map inferences to system utterances: This requires determining which inferences are relevant, i.e., which inferences convey significant information. It may not be possible to define relevance in a domain-independent manner for every situation, but some guidelines seem to exist: A conflict notice always has the highest priority for presentation since no further reasoning makes sense if an inconsistency has already been proven. On the other hand, discussing inferences the user himself has already become aware of without the system would be annoying. The proof traces produced in the reasoning process may be used to determine which and how many rules and facts were used in the derivation. So, certain inferences may be classified as trivial, i.e., if the user may be expected to have inferred the same results as well. Conversely, some inferences may be classified as “interesting”, e.g., if facts from a time table database are involved, which the user is assumed not to know in advance.

1 Although, in principle, the FIA might decide to delay confronting the user with the conflict.
Relevance may also be influenced by the dialogue strategy that is implemented in the dialogue application. In particular, the application may be interested in suggestions in certain situations, while only in strict inferences in others. Therefore, the RM may be instructed to operate in different modes. In the first mode, the RM communicates all suggestions and inferences obtained from possible solutions to be negotiated with the user by the application. Suggestions may be obtained by the RM in a depth-first or in a breadth-first manner. In the first case, the system tries to navigate to a consistent solution as quickly as possible, while in the latter one, the system would try to communicate the earlier choice points first. The second operation mode is similar, however, only strict inferences by the RM are allowed. Finally, the third mode handles the explanations of proofs: In this mode, the RM provides more information about inferences, e.g., when the user has asked about the reasons for some inference of the system.

When handling the situations signaled by the RM, a strategy for conflict resolution seems clearly necessary for the FIA [6]. Several options exist which may make sense in different situations. The most straightforward strategy would be for the system to reject the information that was added last. Alternatively, the system may retract (forget) the first assertion involved in the conflict. This may be helpful if the user wants to correct an earlier choice. Our prototype system described in the next section uses both strategies.

3. Implemented Prototype

In order to illustrate our approach, we have implemented several VoiceXML dialogue scripts that make use of the RM through the forms/reasoner interface. The interface has been implemented by means of a user-defined ECMAScript host object, called RM. This host object contains generic methods that can be used to notify the reasoning module about value changes in form items (cf. Figure 3). The methods are to be called from scripts in <filled> elements of a VoiceXML form, and in any other place when the value of a form item is changed.

The notification of the FIA by the RM takes place through various call-back methods in the RM object that may be implemented depending on the task of the dialogue. Currently, these methods are only invoked after some assertion was added or retracted. In fact, the domains explored for the prototypes so far are limited enough for guaranteeing that it is acceptable to wait after a FIA update until the reasoning process finishes with a definite result.

The interleaving of the reasoning process and dialogue management is illustrated in the example dialogue fragment shown in Figure 2. The dialogue was conducted with the prototype implementation of a reasoning-enabled form illustrated in Figure 3. Initially, the dialogue continues as it would without the reasoning module involved: after the form is initialized the FIA begins to query the individual items. The user’s utterances in U-2 and U-4 fill the items “start” and “end”. In S-5, the system selects the “confirm” item, instead of the “duration” item, because the reasoning module has inferred the value of that item and the FIA consequently does not consider it. The item was filled by the call-back function RM.onInference that was created when the dialogue was initialized (cf. Figure 3). In line U-6 the user overrules the value of the “duration” item, which causes a conflict in the reasoning module, since from the start and end time, a different duration was inferred. The conflict is handled (silently) by function RM.onConflict. The chronologically first culprit for the conflict is determined by the RM, and its value is simply cleared by the script. Thus, the inference procedure can restart and fill the “start” item’s value similarly to S-5.

4. Related Work

In the area of cooperative dialogue systems, separating the domain-related problem-solving knowledge from the dialogue management has been a common goal [7, 6, 8]. The TRIPS dialogue systems, for instance, contains a so-called Problem Solving Manager which manages different domain-specific planning and scheduling components. In the Pacifica evacuation domain, TRIPS allows the user to specify time constraints for the operations to be planned. Also, certain types of conflicts may occur, e.g., when a bridge used in an operation becomes unavailable. However, even though conflicts in certain actions are detected, it seems the user is somewhat left alone with the decision how to resolve the conflict, because no proof trace is available.

Qu et al. [4] present a dialogue generation model that uses a constraint-based problem solver to detect early and handle overconstrained and underconstrained information requests and to generate cooperative responses in these cases. They are able to show that generated cooperative responses improve dialogue efficiency and task success in overconstrained situations. However, in contrast to our approach, Qu et al. only consider queries to a relational data base. In addition, one focus of our work is to extend an existing speech dialogue description language, instead of implementing a specialized graphical interface.

Finally, Faltings et al. [5] in a similar spirit present interesting approaches to mixed-initiative information seeking as
5. Conclusions and Outlook

We have presented a practical approach to enhance applications created using an industry-standard speech dialogue description language with extended reasoning capabilities that aim at a more uniform treatment of domain knowledge. Our approach consists of a separate reasoning module that may be controlled from the dialogue application through a generic forms/reasoner interface. Thus, it is possible to avoid manually created code for an ad-hoc propagation of constraints between form items whose complexity presents a significant problem for the maintenance of the application code.

The presented work provides the basis for a number of future research directions: First of all, a tighter integration of the presented work with VoiceXML and the RM seems promising, e.g., by automat- tic adaptation of the RM mode seems interesting from the dialogue management perspective, in order to avoid presenting options or explanations when it is not desired.

6. References


