Contextual Revision in Information Seeking Conversation Systems

Keith Houck
IBM Research
19 Skyline Drive
Hawthorne, NY 10532 USA
khouck@us.ibm.com

1. Introduction

Contextual revision is a powerful feature of human-human inquiry conversations, whereby users make incremental revisions to the context created by previous turns of the dialog. This greatly reduces the word length of spoken inquiries.

Example 1. Contextual revision in the real estate domain

In man-machine dialogs shortening the word length of users' input not only has obvious benefits for the user, but can also simplify the machines job of understanding the user's utterance. Shorter word lengths lead to less chance of speech recognition errors and simpler grammatical constructs with fewer attachment ambiguities. The above example shows a clearly identifiable linguistic form ("What about...") that suggests the second turn is a continuation of the first. Example 2, however, shows a case where linguistic cues alone are insufficient. Problems like this arise not only in determining whether an utterance is intended as a new query or a contextual revision, but also in determining the exact revision implied by the utterance.

Example 2. A semantically implied revision

A number of other systems described in the literature support some level of contextual revision (e.g., [2], [3], [4], and [6]). All are quick to explain the straightforward cases, however we have found that the subtle cases are more difficult to handle, requiring further heuristics and algorithms in input processing and in some cases, system level cooperation with the output generation components in order to effectively support the construct. Significant additional constraints are imposed by the limitations of speech recognition technology, the need to accommodate a wide variety of linguistic styles from both native and non-native English speakers, and by our desire to minimize the amount of customization work required to develop applications on the RIA framework, which embeds the work described here.

We begin with a brief overview of the RIA framework and RealHunter™, the residential real estate application built on the framework. We then describe in more detail our work on contextual revision and follow this with discussions of related work and our conclusions.

2. System Overview

The functionality described in this paper is being implemented in the Responsive Information Architect (RIA), an emerging domain independent framework that can be used for developing multimedia conversational applications to access information. From a purely end-user point of view, the goal of the framework is to enable a class of applications that makes information more easily accessible in a form that is more directly suitable to the users needs. Users can explore an information space without being bound by any predetermined navigational paths or assumptions about tasks. To achieve this goal, the framework allows applications to accept input from a combination of modalities including standard GUI controls, natural language (spoken or entered using the keyboard), and deictic gestures. Instead of presenting responses through statically pre-designed templates (views), multiple framework components cooperate to dynamically design a suitable combination of spoken language and 3D graphics to directly answer the user's inquiry, in the conversational and visual contexts in which it was asked. The framework maintains a hierarchical history of the conversation (both input and output) so that all components can utilize the full conversational context when performing their respective functions[5]. In addition, the framework maintains a domain ontology in order to allow its components to exploit domain dependant knowledge without having any programming dependencies on a specific application domain. Declarative definitions (e.g., lexicon, corpus) adapt the components to the application domain. Non-inquiry transactions are also supported, but require procedural implementation in the application embedding the framework.

When a new input is received from the user interface, it is first processed by the interpreter component which utilizes the conversation history and domain ontology together with its own lexicon definitions to generate a complete disambiguated, self-contained semantic form of the request,
which it inserts into the conversation history[5]. Subsequently invoked components determine the data content of the reply and dynamically design the form of the presentation. The final output design, after being recorded in the conversation history, is sent to the rendering components to generate the corresponding 3D graphic sequence and synthesized audio output (via text-to-speech). The output design is constrained not only by the nature of the system's response to the user's request, but also by the data definitions and relationships in the domain ontology, the prior visual and speech context (conversation history), and the specific data values retrieved.

RealHunter™ is a real-estate application that we use as a testbed for the RIA framework (see Figure 1). It provides access to a database of houses for sale and about a dozen other related object types (e.g., towns, schools, golf courses).

![Figure 1. Output from the RealHunter™ RIA application.](image)

3. Contextual Revision

3.1. RIA Interpreter

The interpreter is the RIA component primarily responsible for integrating the input request with the prior conversation context and thus is our focus here. The natural language portion of the interpreter utilizes a semantic parser to convert the language from the input request (spoken or typed) into an internal form, which is then further processed to resolve contextual references. Both explicit (e.g., pronominal reference) and implicit (e.g., comparative ellipses) references are resolved at this stage. Deictic resolutions, when available, are preferred over conversational ones. In addition, attachment ambiguities and object relationships are resolved based on a combination of lexical ordering in the input and semantic constraints extracted from the domain ontology. At this point the input is ready to be processed for contextual revision (as described in the next section) and then incorporated into the conversation history.

One of the important design goals of the RIA interpreter is to minimize the effort required to support new domains. Our basic approach is to drive the semantic parser primarily from a lexicon and, where possible, to reuse the lexicon entries across domains. In addition, all semantic constraints used by the interpreter are derived directly from the ontology, thus avoiding the burden of additional semantic definitions. Additional pragmatics can be implemented procedurally if needed.

3.2. User Queries in RIA

When considering contextual revision, it is important to understand what is to be revised. The RIA framework currently allows queries against a structured information base with a known schema containing typed objects and attributes, as well as relationships among the objects. Both objects and attribute values may be queried and objects may be selected either thru direct reference or thru conjunctions of constraints on their attributes. In addition, queries can contain joins along ontological relationships (e.g., "show me colonial houses in school districts with over 95% of seniors going to college"), and be nested (e.g., "show me houses cheaper than this one"). The later capability is extended to allow spatial queries (e.g., "show me houses within 5 miles of the White Plains train station"). The language accepted ranges from essential keywords only (e.g., "colonials 3 bedroom") to full English sentences. Ungrammatical input is generally well tolerated, although proper semantic identification can sometimes be dependent upon a correct lexical context. This flexibility is important both in tolerating errors, introduced by the speaker or by speech recognition, and in accommodating the wide variety of linguistic styles we have encountered with users.

A RIA query can be modeled as a set of interconnected trees, where the nodes of a tree represent both object variables and constraints on those variables. A branch from an object variable (node) to a constraint node constrains the instances that satisfy the object variable. A branch from one object variable to another further constrains the instances of the object node closer to the root in that those instances must have a specified relationship with the instances represented by the child object node. An object variable with no parents is called the root object of the query and determines the base type of the result (although the actual output presentation may require depiction of many of the constraint object types and instances as well.) Currently, RIA queries are restricted to a single root. The above representation covers simple and joined queries. A nested query requires an additional relationship from a constraint node to an object variable node (the later is considered to be the root of the nested sub-query.) As it is theoretically possible for more that one constraint to reference the same nested query, the nested relationships could form a graph. For the purposes of this paper, we ignore this case and think of a query as being represented by a tree.

3.3. Definition

With the basic understanding of RIA queries from the previous section, we can now talk about the specific types of contextual revisions that can be supported against those queries and the resulting issues that arise. There are a number of ways that context created by the previous query can be combined into the current input (query). For the purposes of this paper, we use the term contextual revision to refer to the process where the entire context of the previous query is taken as the base for revision by the current input (see Example 1). This is in contrast to other forms of contextual reference such as pronoun resolution where the context from the prior query is inserted at some known point into the current input. We note that some input can require both types of processing (e.g., "colonials cheaper than the tudor one"), but most involve one or the other. With this definition in mind, we next consider the allowable revisions based on the query model presented in section 3.2.
3.4. Allowable Revisions

Our system allows users to request revisions that add or replace the root node and/or leaf nodes in the previous query tree. Subtrees may also be deleted. We have not found any need to allow the insertion of a node between two existing nodes, nor have we found any natural way to express this without using explicit reference. Further discussion of the revision operations can be found in Section 3.7, Algorithm.

3.5. Issues in Handling Revisions

There are three basic issues in handling revisions:

1) Determining whether an input request is intended as a revision,
2) Aligning the revision with the original query, and
3) Determining which if any parts of the original query are superseded by the revision.

We consider each of these problems in more detail below.

3.5.1. Context Boundary Ambiguity

The first problem encountered with contextual revision is to determine if the input request represents the start of a new query or a revision to the previous one. There are two observations that guide our expectations in this process. First, it is entirely possible for a user to create an unsolvable ambiguity using natural language. Second, users are unlikely to deliberately do this as they have been conditioned through experience to communicate in a single stream.

3.5.2. Alignment Ambiguity

In cases where the input request is to be treated as a revision, we must determine the position of the tree representing the input against the tree representing the prior query. For example, should “show colonial houses in towns with under 5,000 people” followed by “along the Hudson River” be interpreted as “show colonial houses in towns along the Hudson River with under 5,000 people” or “show towns along the Hudson River that contain the colonial houses in towns with under 5,000 people.”

3.5.3. Add/replace ambiguity

Once the alignment has been decided, the revision must be merged with prior query. The key decision here is what objects and constraints from the prior query are effectively replaced by the constraints from the revision. Again, the direct matches are straightforward. More difficult cases occur when new objects are introduced by the revision. In the case discussed above, we must decide if the new root replaces the old root or is qualified by it.

3.6. Cues for disambiguation

In this section we list the various cues we use to resolve the ambiguities identified in the previous section.

3.6.1. Linguistic cues

Grammatical constructs that take a direct object are used to indicate that the root of the revision is the subject of the query (e.g., “show houses with…” implies a subject for “houses”). Words like “just” and “only”, in proper context, indicate a desire to narrow the original query. Conversely, quantifiers like “all” or “any” suggest a displacement of one or more prior constraints. Prepositions are also useful, but often difficult to reliably recover through the speech channel. Finally, we allow explicit separation from the prior context with phrases like “new query”.

3.6.2. Semantic cues

The semantic ontology contains the relationships between object types (concepts) and identifies which attributes of a given concept serve to fully qualify an instance of that type. The former is useful to assess whether an object (type) in the revision can be joined to a given object type in the prior query. The later is used to evaluate the fully qualified heuristic (“Objects that are fully qualified tend not to receive further qualifications”) and its corollary (“Objects that are unqualified are more likely to be qualified by another object than they are to qualify it.”)

3.6.3. Conversational cues

If the tree representing the revision (see section 3.2) is a sub-tree of the previous query’s tree, there is no possibility that applying the revision to the prior query will change the result. Therefore, that interpretation is most likely incorrect. In this case we say that the previous query subsumes the revision and the later should be treated as a new query.

In cases where linguistic cues have predetermined the subject of the revision, we can detect a change in subject from the previous query. Queries involving both a change of subject and joined or nested object constraints likely indicate a new query.

3.6.4. Data cues

Sometimes knowing how much data was retrieved by the previous query can be helpful in cases that remain ambiguous after applying other cues. A query that returns no data is unlikely to be narrowed by a subsequent query, and thus such an interpretation should be disfavoured. Similarly, a query that returns a singleton (regardless of its expression), may be treated is if the base object was fully qualified.

3.7. Algorithm

The issues identified in section 3.5 are addressed together in a two-stage algorithm. Stage one evaluates the heuristic cues from section 3.6 and then applies a rule tree to select a merge operation, which is applied in stage two. Despite the apparent
separation of decision and action, we note that “new query” situations are not always identified in stage one, but are handled correctly when the selected merge operation deletes all branches of the prior query tree, in stage two.

Due to space limitations, we present stage one only in summary form.

The algorithm first checks for explicitly stated “new query”, subsumption by the previous query, and complex queries with an indicated new subject and selects new_query if any of these are true. Next, if the current query root is not linguistically constrained to be the subject, it searches for a proposed alignment point with the object nodes in the previous query tree (matching object type). If found, the merge_root operation is selected. Otherwise, the ontology is used to determine if there is a potential qualifying relationship between the root objects of the current and previous queries. If not, the replace_root operation is selected, otherwise the fully qualified heuristic, its corollary, and the data heuristics are used to determine if the current root should qualify the previous query root object or vice versa (insert_qualifier or insert_root).

The operations used in stage two are summarized below:

**New_query** Treat the current query as complete, and do not attempt to merge it with the previous one.

**Merge_root** Merge the current query into the previous one starting at the point of alignment and replace any of the prior constraints that are superseded by the current query (e.g., “show ranch houses under 1 million dollars”, “colonials in Rye City” \(\rightarrow\) “show colonial houses under 1 million dollars in Rye City”)

**Replace_root** Replace the root object in the previous query with the root object of the new query, preserving all of the constraints in the current query and only those (including joined objects) in the previous query that are compatible with the new root object type (e.g., “show parks near this house”, “what about hospitals” \(\rightarrow\) “show hospitals near this house”).

**Insert_root** Add the previous query as a joined object qualifier to the current query, if it is not superseded (e.g., “show colonial houses in Rye”, “show school district” \(\rightarrow\) “show the school district of the colonial houses in Rye”) in stage one.

**Insert_qualifier** Add the current query as a joined object qualifier to the root of the previous query (“show colonial houses”, “just Rye” \(\rightarrow\) “show colonial houses in Rye”)

A joined object constraint in the previous query is normally replaced by any joined constraint with the same relationship (e.g., spatial-contains) regardless of target object type, unless linguistic cues indicate narrowing (e.g., “only in Rye City”).

Finally we note here that in cases where there is any possible confusion, the RIA output components confirm the constraints in effect, both audibly and visually, to the user.

4. Results

The algorithm has been evaluated against a corpus collected earlier in the RIA project using the Wizard-of-OZ technique. In the Wizard-of-OZ study, 16 subjects were each asked to perform two of four house hunting tasks using spoken natural language. This generated 261 valid data queries, of which 89 were correctly identified as revisions, 171 were correctly identified as new queries, while 1 incorrectly ended up as a new query when a previous constraint was replaced by one that the user had intended to add, but did not accompany with the requisite linguistic cues.

5. Related Work

Contextual revision is an important tool in human conversation and even some of the earliest natural language systems, using keyboard input, supported it to some extent [2], [3], [4]. As spoken language input has become more feasible over the years, language understanding systems have had to adapt in order to accommodate the lower quality of language received thru the speech channel [6]. As far as we know, our system is the first to systematically handle all the subtleties of contextual revision discussed here in a domain independent way. It can be used to process even broken English across a variety of heterogeneous object types in reasonably complex queries. As our user studies have shown us, this is required to successfully handle the wide variety of linguistic styles observed with both native and non-native speakers.

Interesting approaches to the problem can also be found in more theoretical work, such as [7], however, we have not yet found it practical to incorporate the requisite user task models on which they depend, nor have we been able to recover some of their more subtle linguistic cues via the speech channel.

6. Conclusions and Future Work

We have shown how contextual revision can be supported in a domain independent way, even in challenging spoken language environments. Our approach augments the traditional linguistic cues, with semantic, conversational, and data cues to help overcome many of the omissions and errors inherent in informal spoken and typed natural language input.

Our future work will focus on expanding the range of request types the system can support, as well as further testing our existing work in new domains and in commercial environments. Support for languages other than English is also important area of opportunity.

7. References