Term Disambiguation in Natural Language Query for XML

Yunyao Li[1,⋆], Huahai Yang[2], and H.V. Jagadish[1,⋆]

[1] University of Michigan, Ann Arbor, MI 48109, USA
{yunyaol, jag}@umich.edu
[2] University at Albany, SUNY, Albany, NY 12222, USA
hyang@albany.edu

Abstract. Converting a natural language query sentence into a formal database query is a major challenge. We have constructed NaLIX, a natural language interface for querying XML data. Through our experience with NaLIX, we find that failures in natural language query understanding can often be dealt with as ambiguities in term meanings. These failures are typically the result of either the user's poor knowledge of the database schema or the system's lack of linguistic coverage. With automatic term expansion techniques and appropriate interactive feedback, we are able to resolve these ambiguities. In this paper, we describe our approach and present results demonstrating its effectiveness.

1 Introduction

Supporting arbitrary natural language queries is regarded by many as the ultimate goal for a database query interface. Numerous attempts have been made towards this goal. However, two major obstacles lie in the way: first, automatically understanding natural language (both syntactically and semantically) is still an open research problem itself; second, even if we had a perfect parser that could fully understand any arbitrary natural language query, translating this parsed natural language query into a correct formal query remains an issue since this translation requires mapping the user's intent into a specific database schema.

In [17, 18], we proposed a framework for building a generic interactive natural language query interface for an XML database. Our focus is on the second challenge—given a parsed natural language query (NLQ), how to translate it into a correct structured query against the database. The translation is done through mapping grammatical proximity of parsed tokens in a NLQ to proximity of corresponding elements in the result XQuery statement. Our ideas have been incorporated into a working software system called NaLIX[1]. In this system, we leverage existing natural language processing techniques by using a state-of-art natural language parser to obtain the semantic relationships between words in a given NLQ. However, the first challenge of understanding arbitrary natural language still remains. One solution to address the challenge is to provide training and enforce a controlled vocabulary. However, this requirement defeats our original purpose of building a natural language query interface for naive users.

⋆ Supported in part by NSF IIS-0219513 and IIS-0438909, and NIH 1-U54-DA021519-01A1.

[1] NaLIX was demonstrated at SIGMOD 2005 and voted the Best Demo [17].

H. Legind Larsen et al. (Eds.): FQAS 2006, LNAI 4027, pp. 133–146, 2006.

Through our experience with NaLIX, we find that several factors contribute to failures in natural language query understanding, and these failures can often be dealt with as ambiguities in term meaning. In this paper, we describe how we resolve these ambiguities with automatic term expansion techniques and appropriate interactive feedback facilities. In particular, we depict how we engage users, who are naturally more capable of natural language processing than any software, to help deal with uncertainty in mapping the user's intent to the result queries. We discuss the details of our approach in Sec. 3. We then evaluate the effectiveness of our term disambiguation techniques through a user study. The experimental results are presented in Sec. 4. In most cases, for a user query that initially could not be understood by the system, no more than three iterations appears to be sufficient for the user to reformulate it into an acceptable query. Previous studies [4, 23] show that even casual users frequently revise queries to meet their information needs. As such, our system can be considered to be very useful in practice. Examples illustrating our interactive term disambiguation approach are described in Section 5.

Finally, we discuss related work in Sec. 6 and conclude in Sec. 7. We begin with some necessary background material on our framework for building a generic natural language query interface for XML in Sec. 2.

2 From Natural Language Query to XQuery

Translating queries from natural language queries into corresponding XQuery expressions involves three main steps. The first step is token classification (Sec. 2.1), where terms in a parse tree output of a natural language parser are identified and classified according to their possible mapping to XQuery components. Next, this classified parse tree is validated based on a context-free grammar defined corresponding to XQuery (Sec. 2.2). A valid parse tree is then translated into an XQuery expression (Sec. 2.3). In this section, we briefly describe each of the three steps to provide necessary background information. More detailed discussion of these three key steps can be found in [18]. The software architecture of NaLIX has been described in [17].

2.1 Token Classification

To translate a natural language query into an XQuery expression, we first need to identify words/phrases in the original sentence that can be mapped into corresponding XQuery components. We call each such word/phrase a *token*, and one that does not match any XQuery component a *marker*. Tokens can further be divided into different

Table 1. Different Types of Tokens

Type of Token	Query Component	Description
Command Token(CMT)	Return Clause	Top main verb or wh-phrase [21] of parse tree, from an enum set of words and phrases
Order by Token(OBT)	Order By Clause	A phrase from an enum set of phrases
Function token(FT)	Function	A word or phrase from an enum set of adjectives and noun phrases
Operator Token(OT)	Operator	A phrase from an enum set of preposition phrases
Value Token(VT)	Value	A noun or noun phrase in quotation marks, a proper noun or noun phrase or a number
Name token(NT)	Basic Variable	A non-VT noun or noun phrase
Negation (NEG)	function not()	Adjective "not"
Quantifier Token(QT)	Quantifier	A word from an enum set of adjectives serving as determiners

Table 2. Different Types of Markers

Type of Marker	Semantic Contribution	Description
Connection Marker(CM)	Connect two related tokens	A preposition from an enumerated set, or non-token main verb
Modifier Marker(MM)	Distinguish two NTs	An adjectives as determiner or a numeral as predetermine or postdeterminer
Pronoun Marker(PM)	None due to parser's limitation	Pronouns
General Marker(GM)	None	Auxiliary verbs, articles

Table 3. Grammar Supported By NaLIX

1. Q → RETURN PREDICATE* ORDER_BY?
2. RETURN → CMT+(RNP|GVT|PREDICATE)
3. PREDICATE → QT?+((RNP$_1$ |GVT$_1$)+GOT+(RNP$_2$|GVT$_2$)
4. |(GOT?+RNP+GVT)
5. |(GOT?+GVT+RNP)
6. |(GOT?+[NT]+GVT)
7. |RNP
8. ORDER_BY → OBT+RNP
9. RNP → NT |(QT+RNP)|(FT+RNP)|(RNP∧RNP)
10. GOT → OT|(NEG+OT)|(GOT∧GOT)
11. GVT → VT |(GVT∧GVT)
12. CM → (CM+CM)

Symbol "+" represents attachment relation between two tokens; "[]" indicates implicit token, as defined in Def. 11 of [18]

types as shown in Table 1 according to the type of components they match[2]. Enumerated sets of phrases (enum sets) are the real-world "knowledge base" for the system. In NaLIX, we have kept these small—each set has about a dozen elements. Markers can be divided into different types depending on their semantic contribution to the translation.

2.2 Parse Tree Validation

The grammar for natural language corresponding to the XQuery grammar supported by NaLIX is shown in Table 3 (ignoring all markers). We call a parse tree that satisfies the above grammar a *valid* parse tree. As can be seen, the linguistic capability of our system is directly restricted by the expressiveness of XQuery, since a natural language query that may be understood and thus meaningfully mapped into XQuery by NaLIX is one whose semantics is expressible in XQuery. Furthermore, for the purpose of query translation, only the semantics that can be expressed by XQuery need to be extracted and mapped into XQuery.

2.3 Translation into XQuery

A valid parse tree, obtained as described above, can then be translated into XQuery. XML documents are designed to be "human-legible and reasonably clear" [27]. Therefore, any reasonably designed XML document should reflect certain semantic structure isomorphous to human conceptual structure and hence expressible by human natural language. The major challenge for the translation is to utilize the structure of the natural

[2] When a noun/noun phrase matches certain XQuery keywords, special handling is required. Such special cases are not listed in the table and will not be discussed in the paper due to space limitation.

language constructions, as reflected in the parse tree, to generate appropriate structure in the XQuery expression. We address these issues in [18] through the introduction of the notions of *token attachment* and *token relationship* in natural language parse trees. We also propose the concept of *core token* as an effective mechanism to perform semantic grouping and hence determine both query nesting and structural relationships between result elements when mapping tokens to queries. Our previous experimental results show that in NaLIX a correctly parsed query is almost always translated into a structured query that correctly retrieves the desired answer (average precision = 95.1%, average recall = 97.6%).

3 Term Disambiguation

The mapping process from a natural language query to XQuery sometimes fails at token classification or parse tree validation stage. We observe that failures in this mapping process can always be dealt with as term ambiguities. Disambiguation of terms are thus necessary for properly mapping a user's intent into XQuery. In this section, we outline different types of failures we identified, and how they present themselves as term ambiguities. We then describe how we disambiguate the terms via automatic term expansion and interactive feedback.

3.1 Types of Failure

Parser failure. An obvious kind of failure in any natural language based system is one due to limited linguistic capability. In our system, a natural language query with correct grammar, though possible to be manually translated into XQuery, can still be found invalid at the validation step due to the incorrect parse tree generated by the parser. For example, for query "Display books published by addison-wesley after 1991," the parser we use generates a parse tree rooted by "published" as the main verb, and "display" as a noun underneath "books." This parse tree results in "published" being classified as unknown. A better parser may avoid such ambiguities, but such a solution is out of the scope of this paper.

Limited system vocabulary. A query sentence may contain terms that cannot be properly classified due to the restricted size of vocabulary that our system understands. For instance, it is impractical to exhaustively include all possible words for each type of token and marker in our system. As a result, there always exists the possibility that some words in a user query will not be properly classified. These words will be singled out in our system.

Inadvertent user error. Inadvertent user errors, such as typos, are unavoidable in any user interface. Such errors could cause failures in natural language query understanding, including unclassifiable terms and undesirable search results. Although some queries with typos can be successfully validated and translated, the results could be different from what the user desires, and are often found to be empty. For instance, the user may write "boks" instead of "books" in the query. Identifying such terms can help explain invalid queries and avoid frustrating users with unexpected results. Users may also write

queries in incorrect grammar. These grammatically incorrect query sentences may result in certain words being labeled as untranslatable as well.

Limited user knowledge. Users often do not possess good knowledge of database schema. Although the requirement for users to have perfect knowledge of a XML document structure can be eliminated by using Schema-Free XQuery as our target language [16], users still need to specify query terms exactly matching element or attribute names in the database. Consequently, query sentences containing unmatched terms could result in misses, unless users are given the opportunities to choose a matching term.

Invalid query semantics. Any natural language system has to deal with situations where users simply do not intend to request an allowable system service. In our system, a user may write a natural language sentence that cannot be semantically expressed in XQuery. For example, some users typed "Hello world!" in an attempt to just see how the system will respond. Such a natural language query will of course be found to be invalid based on the grammar in Table 3, resulting in words such as "Hello" being marked as untranslatable.

3.2 Interactive Term Disambiguation

All five types of failures in natural language query translation can be dealt with as problems of term ambiguity, where the system cannot understand a term or find more than one possible interpretation of a term during the transformation into XQuery. Clever natural language understanding systems attempt to apply reasoning to interpret these terms, with limited success. Our approach is complementary: get the user to rephrase the query into terms that we can understand. By doing so, we shift some burden of semantic disambiguation from the system to the user, for whom such task is usually trivial. In return, the user obtains better access to information via precise querying.

A straightforward solution to seek the user's assistance is to simply return a notification whenever a failure happens and ask the user to rephrase. However, to reformulate a failed query without help from the system can be frustrating to the user. First of all, it is difficult for a user to recognize the actual reason causing the failures. For example, it is almost impossible for a casual user to realize that certain failures result from the system's limited vocabulary. Similarly, given the fact that an empty result is returned for a query, it is unlikely for the user to discover that it is caused by a mismatch between element name(s) in the query and the actual element name(s) in the XML document. In both cases, the user may simply conclude the system is completely useless as queries in perfect English fail every time. Furthermore, even if the user knows exactly what has caused the failures, to correct most failures is nontrivial. For instance, considerable effort will be required from the user to study the document schema in order to rephrase a query with mismatched element names.

From the above discussion, we can see that the difficulties in query reformulation without system feedback are largely due to the user's lack of (perfect) knowledge of the system and the XML document. Intuitively, query reformulation will be easier for the user if the system can provide the needed knowledge without demanding formal training. With this in mind, we designed the following interactive term disambiguation mechanism. First, unknown terms (beyond the system and document vocabulary

Table 4. Error Messages in NaLIX. (The error messages listed below do not include the highlighting of offending parts in the user sentence and the full informative feedback offered; objects in "⟨⟩" will be instantiated with actual terms at feedback generation time.)

Error 1	*The system cannot understand what ⟨UNKNOWN⟩ means. The closest term to ⟨UNKNOWN⟩ the system understands is ⟨KNOWN⟩. Please rewrite your query without using ⟨UNKNOWN⟩, or replace it with ⟨KNOWN⟩.*
Error 2	*The value "⟨VT⟩" cannot be found in the database.*
Error 3	*No element or attribute with the name "⟨NT⟩" can be found in the database.*
Error 4	*At least one noun phrase should be used.*
Error 5	*Please tell the system what you want to return from the database by using the following commands (list of ⟨CMT⟩).*
Error 6	*⟨FT⟩ must be followed by a common noun phrase.*
Error 7	*⟨CMT|OBT⟩ must be followed by a noun phrase (link to example usage of ⟨CMT|OBT⟩). Please specify what you want to return (if CMT) or order by (if OBT).*
Error 8	*CMT|OBT should not attach to a noun phrase. Please remove (RNP|GVT)$_1$.*
Error 9	*The system does not understand what ⟨non-GOT|non-CM⟩ means. Please replace it with one of the following operators (a list of typical OTs with closest OT first) or connectors (a list of typical CMs with closest CM first).*
Error 10	*OBT should not be attached by a proper noun such as GVT.*
Error 11	*The system does not understand what ⟨GVT + RNP⟩ means. Please specify the relationship between ⟨GVT⟩ and ⟨RNP⟩ by using one of the following operators (a list of typical OTs with the closest OT first) or connectors (a list of typical CMs with the closest CM first).*
Error 12	*⟨GOT|CM⟩ must be followed by a noun phrase (example usage of ⟨GOT|CM⟩).*

Table 5. Warning Messages in NaLIX

Warning 1	*System may not be able to understand pronouns correctly. If you find the returned results surprising, try express your query without pronouns.*
Warning 2	*There is no element/attribute with the exact name ⟨NT⟩. You may choose one or more from the list (of matching elements/attributes).*
Warning 3	*There are multiple elements/attributes with the value ⟨VT⟩. You may choose one from the list (of matching elements/attributes).*
Warning 4	*We assume that ⟨NT⟩ elements/attributes is related with ⟨$\mathtt{coretoken}$⟩. If this is not what you intended, you may choose one from the list (of matching elements/attributes).*

boundary) and the exact terms violating the grammar in the parse tree are identified and reported in the feedback messages. The types of ambiguities caused by these terms are also reported. In addition, for each ambiguous term, appropriate terms that can be understood by the system are suggested to the user as possible replacement for the term. Finally, example usage of each suggested term is shown to the user. A complete list of error messages generated by our system is shown in Table 4.

The above feedback generation techniques work as three defensive lines against uncertainty in term disambiguation. Identification of term ambiguities is the first essential defensive line. It not only helps a user to get a better understanding of what has caused the query failure, but also narrows down the scope of reformulation needed for the user. Certain failures, such as those caused by typos, can easily be fixed based on the ambiguous terms identified. For others, the user may need to have relevant knowledge about the system or document vocabulary for term disambiguation. For such cases, our system uses term suggestion as the second defense line. Relevant terms in the system and document vocabulary are suggested based on their string similarity and function similarity (in the XQuery translation) to each ambiguous term. Obviously, not every

term suggested can be used to replace the ambiguous term. The user is responsible for resolving the uncertainty issue associated with term suggestion by selecting suggested terms to replace the ambiguous term. Finally, when queries fail due to parser errors, incorrect grammar, or invalid query semantics, the exact terms causing the failures are difficult to pinpoint. The system is likely to wrongly identify term ambiguities and thus may generate less meaningful error messages and suggest irrelevant terms. For such cases, providing examples serves as the last line of defense. Examples supply helpful hints to the user with regard to the linguistic coverage of the system without specifying tedious rules. However, exactly how the information conveyed by the examples is used for term disambiguation is associated with greater uncertainty.

For some queries, the system successfully parses and translates the queries, yet may not be certain that it is able to correctly interpret the user's intent. These queries will be accepted by the system but with warnings. A complete list of warning messages is presented in Table 5.

3.3 Error Reporting

The failures of a natural language query may be attributed to multiple factors. For example, a query may contain multiple typos and mismatched element names. For such queries, multiple error messages will be generated and reported together, with the following exception. If an error message of category Error 1 is generated, then any error message of category Error 4 to 12 for the same query will not be reported to the user.

The above error reporting policy is based on the following observation: any parse tree containing unknown term(s) validates the grammar in Table 3. Therefore for the same query, error message(s) of both category Error 1 and category Error 4 to 12 are likely to be caused by the same unknown term(s). In such a case, an error message directly reporting the unknown term(s) provides more relevant information for query reformulation. Moreover, our study shows that users tend to deal with feedback message one at a time—withholding less meaningful error messages in the report is unlikely to have any negative impact over the reformulation.

3.4 Ontology-Based Term Expansion

A user may not be familiar with the specific attribute or element names contained in the XML document. For example, the document being queried uses *author*, while the user query says *writer*. In such a case, the Schema-Free XQuery translation of the query will not be able to generate correct results. We borrow term expansion techniques from information retrieval literature [2, 5, 22] to address such name-mismatch problems. When using term expansion techniques, one must deal with uncertainty associated with the added terms in the query representation. Such uncertainty is traditionally handled by introducing approximate scoring functions. However, terms added to the query may be weighted in a way that their importance in the query is different from the original concept expressed by the user. We avoid this issue by granting the user full control of the term expansion: warning messages are generated to alert the user whenever term expansion is employed; then the user may choose to use one or more of the terms added by the expansion in new queries. Uncertainty introduced by term expansion is thus explicitly

revealed to the user and clarified by the user's response. Such an interactive term expansion process not only avoids the drawback of scoring functions, but also allows the user to gradually learn more about the XML database by revealing a part of the database each time. This approach is especially suitable for our system, since term expansion is limited to the terms used as attribute and element names in the XML document.

4 Experiment

We implemented NaLIX as a stand-alone interface to the Timber native XML database [1] that supports Schema-Free XQuery. We used Minipar [19] as our natural language parser. To evaluate our system, we conducted a user study with 18 untrained participants recruited from a university campus. Each participant worked on ten different search tasks adapted from the "XMP" set in the XQuery use cases [26]. Detailed discussion on the experimental set up and results on ease of use and retrieval performance of our system can be found in [18]. In this paper, we analyze the transcripts of system interaction with users to assess the effectiveness of our interactive term disambiguation approach.

Measurement. For each query initially rejected by the system, we recorded the number of iterations it took for a participant to reformulate the query into a system acceptable one or until time out (5 minutes for each search task). We also recorded the actual user input for each iteration in a query log. We then manually coded the query logs to determine how a user may have utilized the feedback messages in each reformulation process. The coding schema below were used:

- If the user changed a term that appears in the body of a feedback message in the next query input, we count the reformulation as being helped by the feedback message.
- If the user used a suggested term in the next query input, we count the term suggestion as being helpful.
- If the user rewrote the query in the same format as that of an example query contained in the feedback, then the example is regarded as useful.
- If we cannot attribute the user change(s) to any specific component of a feedback message, then the feedback is counted having failed to contribute to term disambiguation.

For example, one user initially wrote "List tiltes and editors." An error message "*tiltes* cannot be found in the database" was returned. The user then fixed the typo by changing "tiltes" into "titles" in the new query "List titles and editors," which was then accepted by the system. In this case, the disambiguation of term "tiltes" was considered as made with the assistance of the error message. Feedback message, suggested terms, and term usage examples are not exclusive from each other in assisting query reformulation. One or more of them may be utilized in a single iteration.

We consider the entire reformulation process as a whole when determining the types of helpful feedback. The reason is two-fold. First, we cannot determine the effectiveness of term disambiguation for a query until the reformulation is complete. Furthermore, users were often found to revise the query based on only one feedback at a time, even when multiple feedback messages were generated. Thus multiple iterations are needed

Table 6. Types of Aids Used in Query Reformulation

Body of Feedback Message	Suggested Term	Example	None
75.1%	22.4%	10.9%	7.3%

to disambiguate all the terms in one failed query, and they should be considered as parts of a whole reformulation process.

Results. We coded 166 query reformulation processes in total. The average number of iterations needed for participants to revise an invalid natural language query into a valid one was 2.55; the median number of iterations was 2.

The statistics on different types of aids utilized in the reformulations is listed in Table 6. As can be seen, the body of feedback messages provided helpful information for majority of the reformulation processes. Suggested terms and their example usage were found helpful for less than one quarter of the reformulations. Nevertheless, these two techniques are not thus less important. Both suggested terms and their example usage are most likely to be utilized when failures were caused by parser failure, grammar error, or incorrect query semantics. For such cases, there is no easy means to determine the actual factors resulting in the failure, because we depend on an outside parser to obtain dependency relation between words as approximation of their semantic relationships. In another word, it is unlikely for our system to generate insightful feedback messages for such failures. Even if we do have full access to the internals of a parser, we would still not be able to distinguish a parser failure from an incorrect query semantics. A user, however, has little difficulty in taking helpful hints from the identified problematic words, the suggested terms and their example usage to reformulate queries successfully. This is true even when the content of the error message is not entirely insightful.

5 Analysis and Discussion

In this section, three examples of iterations taken from our user study are presented. These illustrate how interactive term disambiguation helps to resolve different types of failures in natural language understanding.

Example 1 (Out of Vocabulary Boundary)

```
User Input 0: Return list of all books with title and author
      Status: Rejected
Error Message: No element or attribute with the name ''list''
               can be found in the database.
User Input 1: List all books by title and author
      Status: Accepted
```

In NaLIX, words/phrases in a given natural language query are classified based on small enumerate sets of phrases (Sec. 2.1) corresponding to XQuery components. A user may use terms outside of the system vocabulary. Some terms cannot be classified and are reported as unknown; others may be wrongly classified and result in error messages later on.

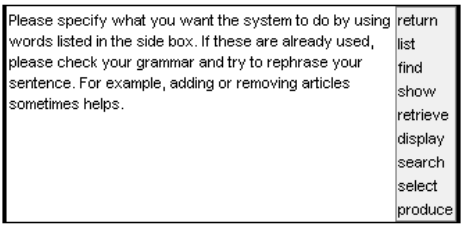

Fig. 1. Example Error Message with Suggested Terms

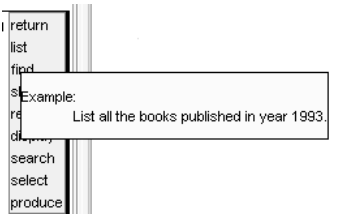

Fig. 2. Example Usage of a Suggested Terms

In this example, "list of" is beyond the boundary of system vocabulary. As a result, the token classification process wrongly identified "list" as a name token and "of" as a connection marker. NaLIX then reported failure when it tried to map "list" into a variable in XQuery, as no element or attribute with name "list" exists in the XML document of interest. The user recognized the term ambiguity from the feedback message generated and successfully disambiguated the term "list" by making minor changes to the original query. Specifically, the user removed "list of," implying that this phrase does not contribute to the query semantics, and replaced verb "return" with verb "list," implying that they are synonyms within the context of the query.

Example 2 (Parser Failure)

```
User Input 0: Show titles and publishers
      Status: Rejected
Error Message: As shown in Figure 1
User Input 1: Display titles and publishers
      Status: Accepted
```

NaLIX relies on a natural language parser to obtain dependency relation between words. The parser we currently use is Minipar [19]. Like any other generic natural language parser, Minipar may produce an incorrect parse tree[3] for an arbitrary natural language sentence.

In this example, verb "show" was parsed as a noun and wrongly regarded as depending on "titles," rather than the other way around. It was thus identified as a name token, instead of a command token (corresponding to RETURN clause of XQuery). As a result, the arse tree validation process determined that a command token was lacking and reported so.

There is no easy way to resolve such parsing problems without building a better parser. Even generating a meaningful feedback message is extremely difficult: for instance, the feedback message for the above example is confusing, as the feedback message is generated based on a wrong parse tree. Therefore, we alert users the possible confusion, and include related terms and meaningful examples in the feedback to provide more information about the system's linguistic capability without requiring any formal training. As we can see from the above example, this feature is very helpful in

[3] Minipar achieves about 88% precision and 80% recall with respect to dependency relations with the SUSANNE Corpus[19].

practice. In this example, the feedback message generated based on the incorrect parse tree does not make much sense, as the user has already specified what to be returned in input 0. However, by replacing "show" with a suggested term "display," the user successfully revised the failed query into one that can be accepted by the system without changing the query semantics. We also observed that users sometimes took hints from the given examples by writing part of the query in the way shown in the examples, and thus successfully obtained a system-understandable query.

Like translation failures that result from parser failure, those caused by incorrect grammar (e.g., "What books have at least one author?") or invalid query semantics (e.g., "Return love.") also pose significant challenges to natural language query translation. It is often impossible to determine the actual reason for a failure solely on the basis of a parse tree. Therefore, the content of a generated error message usually is not helpful in term of pinpointing the exact problem. Nevertheless, users were found to be able to reformulate queries successfully in these cases with the help of identified problematic words, suggested terms, and examples.

Example 3 (Mixed Failures)

```
User Input 0: List author's name and titles of all bkkos by
              that author.
      Status: Rejected
Error Message: 1. No element or attribute with the name
              ``name'' can be found in the database.
              2. No element or attribute with the name
              ``bkkos'' can be found in the database.
User Input 1: List author's name and titles of all books by
              that author.
      Status: Rejected
Error Message: No element or attribute with the name ``name''
              can be found in the database.
User Input 2: List author and titles of all books by that
              author.
      Status: Rejected
Error Message: As shown in Figure 1, 2
User Input 3: List all the authors and titles of all books
              by each author
      Status: Accepted
```

Example 1 and 2 present cases where the failure of natural language query understanding can be contributed to a single factor. In practice, cases such as Example 3, where multiple factors result in a natural language query failure, are also common [4]. In the above example, the initial user input contains one typo "bkkos" and one mismatched name token "name." Our feedback messages reported both. Note, however, although the corresponding parse tree for this query was invalid, no related error message was reported following the policy described in Sec. 3.3.

[4] In our user study, nearly half of the iterations were caused by user error alone, about 10% by parser failure alone, and around 20% by a mixture of the two, with the remaining iterations caused by other factors.

The last iteration in the above example illustrates how a user can be helped for parser failures by example usage included in the feedback message. The user successfully reformulated input 2 into input 3 by adding "all the" before "authors" in similar way as the example usage shown for "list" (Figure 2).

From the above examples, we can see that in many cases, it is very difficult, if not impossible, to determine the exact reason for failures of natural language query understanding and to generate specific feedback messages. The same error "*list* is not found in the database" may be produced for failures caused by limited system vocabulary, where phrases such as "list of" cannot be properly translated into XQuery semantics, or by the user's poor knowledge of the document schema, where no object with the name "list" exists in the XML database. However, by handing the problem over to an interactive term disambiguation process, users can successfully reformulate queries with minimal efforts. In return, they can express richer query semantics, have more control over search results, and obtain results with better quality.

6 Related Work

In the information retrieval field, research efforts have long been made on natural language interfaces that take keyword search query as the target language [6, 8]. In recent years, keyword search interfaces to databases have begun to receive increasing attention [7, 10, 16, 12, 13], and have been considered a first step towards addressing the challenge of natural language querying. Our work builds upon this stream of research. However, our system is not a simple imitation of those in the information retrieval field in that it supports a richer query mechanism that allow us to convey much more complex semantic meanings than pure keyword search.

Extensive research has been done on developing natural language interfaces to databases (NLIDB), especially during the 1980's [3]. The architecture of our system bears most similarity to syntax-based NLIDBs, where the resulting parse tree of a user query is directly mapped into a database query. However, previous syntax-based NLIDBs, such as LUNAR [30], interface to application-specific database systems and depend on database query languages specially designed to facilitate the mapping from the parse tree to the database query [3]. Our system, in contrast, uses a generic query language, XQuery, as our target language. In addition, unlike previous systems such as the one reported in [25], our system does not rely on extensive domain-specific knowledge.

The idea of interactive NLIDBs has been discussed in some early NLIDB literature [3, 15]. Majority of these focus on generating cooperative responses using query results obtained from a database with respect to the user's task(s). In contrast, the focus of the interactive process in our system is purely query formulation—only one query is actually evaluated against the database. Several interactive query interfaces have been built to facilitate query formulation [14, 28]. These depend on domain-specific knowledge. Also, they assist the construction of structured queries rather than natural language queries.

Human computation refers to a paradigm of using human to assist computer in solving problems. The idea has been applied in areas such as imagine analysis, speech

recognition, and natural language processing [9, 24, 29]. All these problems share one common characteristic—they appear to be difficult to computers to solve effectively, but are easy for humans. The natural language understanding problem in our system belongs to the same category. Our solution to this problem is interactive term disambiguation, where human assists to solve term ambiguities identified by the system. It can thus be considered as following a human computation approach as well.

NaLIX explicitly relies on query iterations to elicit user feedback. In the field of information retrieval, an alternative to manual feedback is to automatically infer feedback based on the user's interaction with the system (e.g. document browse pattern); such feedback can then be used to determine document relevance and to expand the original query to obtain more relevant results [11, 20, 31]. However, such an automatic feedback approach does not apply to NaLIX. First of all, explicit feedback requires much less user effort in our system - a user only need to read short feedback messages instead of long documents. More importantly, our system depends on explicit user feedback to resolve ambiguities in natural language understanding to generate precise database query. Unlike information retrieval queries, an incorrect database query will likely fail to produce any results that could be useful for enhancing the original query. Moreover, if we can infer proper information from query results and rewrite a structured query to get correct answer, we may have already solved the difficult natural language understanding problem.

7 Conclusion and Future Work

In this paper, we described term disambiguation in a natural language query interface for an XML database via automatic term expansion and interactive feedback. Starting from a failed natural language query, users reformulate a system understandable query with the help of feedback messages from the system. Uncertainty associated with term expansion, ambiguous term identification, term suggestion and term usage examples are explicitly revealed to the user in the feedback messages. The user then addressed such issues directly by query reformulation. Our user study demonstrates the effectiveness of our approach in handling failures in natural language query. The system as we have, although far from being able to pass Turing test, is already usable in practice.

In the future, we plan to investigate machine learning techniques to improve system linguistic coverage. We are also interested in integrating grammar checking techniques to deal with incorrect grammars. Additionally, we intend to redesign the user interface of our system for better usability. These techniques will all improve our interactive term disambiguation facility, and help users to formulate natural language queries that are a true expression of their information needs and are understandable by database systems.

References

1. Timber: http://www.eecs.umich.edu/db/timber/
2. WordNet: http://www.cogsci.princeton.edu/∼wn/
3. I. Androutsopoulos et al. Natural language interfaces to databases - an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.

4. M. J. Bates. The design of browsing and berrypicking techniques for the on-line search interface. *Online Review*, 13(5):407–431, 1989.
5. A. Burton-Jones et al. A heuristic-based methodology for semantic augmentation of user queries on the Web. In *ICCM*, 2003.
6. J. Chu-carroll et al. A hybrid approach to natural language Web search. In *EMNLP*, 2002.
7. S. Cohen et al. XSEarch: A semantic search engine for XML. In *VLDB*, 2003.
8. S. V. Delden and F. Gomez. Retrieving NASA problem reports: a case study in natural language information retrieval. *Data & Knowledge Engineering*, 48(2):231–246, 2004.
9. J. A. Fails and D. R. Olsen. A design tool for camera-based interaction. In *CHI*, 2003.
10. L. Guo et al. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.
11. W. Hill et al. Read wear and edit wear. In *CHI*, 1992.
12. V. Hristidis et al. Keyword proximity search on XML graphs. In *ICDE*, 2003.
13. A. Hulgeri et al. Keyword search in databases. *IEEE Data Engineering Bulletin*, 24:22–32, 2001.
14. E. Kapetanios and P. Groenewoud. Query construction through meaningful suggestions of terms. In *FQAS*, 2002.
15. D. Kupper et al. NAUDA: A cooperative natural language interface to relational databases. *SIGMOD Record*, 22(2):529–533, 1993.
16. Y. Li et al. Schema-Free XQuery. In *VLDB*, 2004.
17. Y. Li et al. Nalix: an interactive natural language interface for querying XML. In *SIGMOD*, 2005.
18. Y. Li et al. Constructing a generic natural language interface for an XML database. In *EDBT*, 2006.
19. D. Lin. Dependency-based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*, 1998.
20. M. Morita and Y. Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *SIGIR*, 1994.
21. R. Quirk et al. *A Comprehensive Grammar of the English Language*. Longman, London, 1985.
22. P. V. R. Navigli. An analysis of ontology-based query expansion strategies. In *Workshop on Adaptive Text Extraction and Mining*, 2003.
23. J. R. Remde et al. Superbook: an automatic tool for information exploration - hypertext? In *Hypertext*, pages 175–188. ACM Press, 1987.
24. B. C. Russell et al. Labelme: a database and web-based tool for image annotation. *MIT AI Lab Memo*, 2005.
25. D. Stallard. A terminological transformation for natural language question-answering systems. In *ANLP*, 1986.
26. The World Wide Web Consortium. XML Query Use Cases. W3C Working Draft. Available at http://www.w3.org/TR/xquery-use-cases/, 2003.
27. The World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation. Available at http://www.w3.org/TR/REC-xml/, 2004.
28. A. Trigoni. Interactive query formulation in semistructured databases. In *FQAS*, 2002.
29. L. von Ahn and L. Dabbish. Labeling images with a computer game. In *CHI*, 2004.
30. W. Woods et al. *The Lunar Sciences Natural Language Information System: Final Report*. Bolt Beranek and Newman Inc., Cambridge, MA, 1972.
31. J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR*, 1996.