NaLIX: an Interactive Natural Language Interface for Querying XML

Yunyao Li*
University of Michigan
1309 Beal Ave.
Ann Arbor, MI 48109
yunyaol@eecs.umich.edu

Huahai Yang
University at Albany, SUNY
135 Western Avenue
Albany, NY 12222
hyang@albany.edu

H. V. Jagadish*
University of Michigan
1309 Beal Ave.
Ann Arbor, MI 48109
jag@eecs.umich.edu

ABSTRACT

Database query languages can be intimidating to the non-expert, leading to the immense recent popularity for keyword based search in spite of its significant limitations. The holy grail has been the development of a natural language query interface. We present NaLIX, a generic interactive natural language query interface to an XML database. Our system can accept an arbitrary English language sentence as query input, which can include aggregation, nesting, and value joins, among other things. This query is translated, potentially after reformulation, into an XQuery expression that can be evaluated against an XML database. The translation is done through mapping grammatical proximity of natural language parsed tokens to proximity of corresponding elements in the result XML. In this demonstration, we show that NaLIX, while far from being able to pass the Turing test, is perfectly usable in practice, and able to handle even quite complex queries in a variety of application domains. In addition, we also demonstrate how carefully designed features in NaLIX facilitate the interactive query process and improve the usability of the interface.

1. INTRODUCTION

In the real world we usually obtain information by asking questions in a natural language, such as English. It follows that supporting arbitrary natural language queries is regarded by many as the ultimate goal for a database query interface. Not surprisingly, there have been numerous attempts to provide a natural language interface to a database [4]. However, two major obstacles lie in the way of reaching the ultimate goal of support for arbitrary natural language queries. First, automatically understanding natural language (both syntactically and semantically) is itself still an open research problem. Second, even if we had a perfect parser that could fully understand any arbitrary natural language query, translating the parsed natural language query into a correct formal query still remains an issue since this translation requires mapping the understanding of intent into a specific database schema. For example, for the query "who is the director of "Gone with the Wind"", we do not know if "Gone with the Wind" is a movie or a book. Moreover, even if we know "Gone with the Wind" is a movie, we still need information on actual structural relationship between *director* and *movie* in the XML document in order to formulate a correct XQuery.

We describe NaLIX (Natural Language Interface to XML), a generic interactive natural language interface to XML database systems. Our focus is on the second challenge: given a parsed natural language query, how to translate it into a correctly structured query against the database. In NaLIX, an arbitrary English language sentence, which can be quite complex and include aggregation, nesting, and value joins, among other things, is translated, potentially after reformulation, into an XQuery expression that can be evaluated against an XML database. For example, it is possible for the user to write "find the title of publications with more than 5 authors" without having to worry about the actual structural relationships among the elements/attributes or the actual element/attribute names used in the XML documents for *title*, *publication*, or *author*. In addition, automatic semantic grouping based on the user query will be performed to determine query nesting and grouping.

Of course, the first challenge of understanding arbitrary natural language remains. In NaLIX, instead of attempting to address this widely regarded "AI Complete" problem, we use an off-the-shelf natural language parser, and interactively guide the user to pose queries that our system can understand by providing meaningful feedback and helpful rephrasing suggestions. In most cases one or two iterations appear to suffice for the user to submit a natural language query that the system can parse. A correctly parsed query in turn almost always is translated into a structured query that correctly retrieves the desired answer.

To the best of our knowledge, NaLIX is the first generic interactive natural language interface to XML databases. NaLIX, while far from being able to pass the Turing test, is perfectly usable in practice, and able to handle even quite complex queries, e.g. involving nesting and aggregation, in a variety of application domains.

The rest of the paper is organized as follows. Section 2 describes the architecture of NaLIX. Section 3 is a list of features of NaLIX included in the demonstration.

2. SYSTEM ARCHITECTURE

Figure 1 depicts the architecture of NaLIX. The entire NaLIX system consists of two parts: one part is responsible for query translation from natural langauge queries to XQuery, including *parse tree classifier, validator* and *translator*; the other provides supports for interactive query formulation, including *query repository* and *message generator*. A typical user interaction process in NaLIX involves both query translation and interactive query formulation.

*Supported in part by NSF under grants IIS-0219513 and IIS-0438909, and by NIH under grant LM-0810601.

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA.

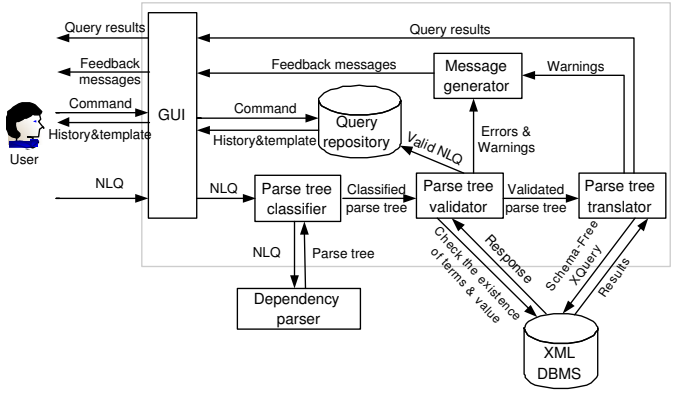

Figure 1: Architecture of NaLIX

2.1 Query Translation

Three main steps are involved in the translation of natural language queries into corresponding XQuery expressions.

Parse Tree Classification: The parse tree classifier identifies the word/phrase in the parse tree of the original sentence that can be mapped into corresponding components of XQuery, and classifies it into a token (if it matches a XQuery component) or a marker (if it does not). Due to the vocabulary restriction of the system, some words/phrases may be unknown to the system, and cannot be properly classified. They will be reported later during parse tree validation, when their relationship with other tokens/markers can be better identified.

Parse Tree Validation: The parse tree validator checks the classified parse tree and see whether the parse tree obtained is one that we know how to map into XQuery. It also checks whether the element/attribute names or values contained in the user query can be found the database, if a database is available. If a parse tree is found to be invalid, information about the errors will be sent to the message generator to generate appropriate error messages.

Parse Tree Translation The challenge of translating a valid parse tree into XQuery is to utilize the structure of the natural language constructions, as reflected in the parse tree to generate appropriate structure in the XQuery expression. We address this issue through the introduction of the notion *token attachment* and *token relationship*. 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 into queries.[1] Our system as it stands supports comparison predicates, simple negation, quantification, nesting, aggregation, value joins, and sorting.

2.2 Schema Free XQuery

Generating correct XQuery expressions involves knowledge of the database schema, and bringing this knowledge to bear on the patterns of token attachment and token relationship can be quite challenging. We solve this problems through the use of Schema-Free XQuery [7]. The central idea in Schema-Free XQuery is that of a *meaningful lowest common ancestor structure* (MLCAS) of a set of nodes. Given a collection of keywords, each of which identifies a candidate XML element to relate to, the MLCAS of these elements, if one exists, automatically finds relationships between these elements, if any, including additional related elements as appropriate. Schema-Free XQuery greatly eases our burden in

[1]Due to space limitation, details of these notions are presented elsewhere [6].

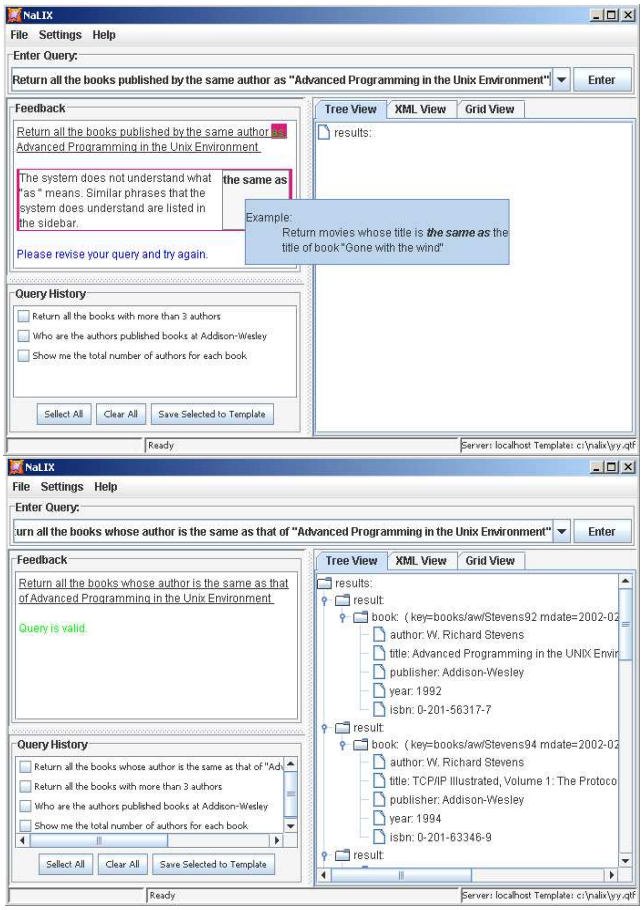

Figure 2: An example iteration.

translating natural language queries - it is no longer necessary to map the query to the precise underlying schema. For example, for our previously mentioned query "who is the director of "Gone with the Wind"", there may be both the *title* of a *movie* or the *title* of a *book* with the value "Gone with the Wind" in the database. However, we do not need advanced semantic reasoning capability to know that only movies can have a director and hence "Gone with the Wind" should be the *title* of a *movie*. Rather, the computation of **mlcas**(*director*, *title*) will automatically choose only the *title* of *movie*, as it is the only *title* structurally meaningfully related to *director*. Furthermore, it does not matter whether the schema has *director* under *movie* or vice versa (for example, movies could have been organized according to their directors). In either case, the correct structural relationships will be found, with the correct *director* elements be returned.

2.3 Interactive Query Formulation

The mapping process from natural language to XQuery requires our system to be able to map words to query components based on token classification. Due to the limited vocabulary understood by the system, certain terms cannot be properly classified. 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.

Feedback Messages If the parse tree validation fails, message generator will issue error messages and suggestions based on the information provided by parse tree validator.

Each error message is dynamically generated, tailored to the actual query producing the errors. Inside each message, possible ways to revise the query are also suggested. Figure 2 shows an example of such iteration in NaLIX. By providing such meaningful feedbacks tailored to each particular query instance, we eliminate the need to require users to study and remember tedious instructions on the system's linguistic coverage. In addition, by suggesting users using the given terms whenever possible, we effectively restrict the linguistic scope of the user query, and thus improve the chances that the user query will be successfully served.

Our system also issues warnings in a few situations, along with the query results to let the user be aware of issues that may have negative impact on the query translation. For example, whenever there exists a pronoun in a user query, warnings will be issued to remind the user of a possible occurrence of the "anaphora" resolution problem that may lead to the loss of search quality. For another example, if there exist multiple matches in database while checking the existence of an element/attribute name or value, we give the user an opportunity to choose one or more matches of interest; if the user decides to proceed, we process the query and the user has now better anticipation of retrieved results.

Query Repository The query repository automatically stores natural language queries that have been successfully validated during a search session. The user can access and manage query history at any time from the GUI. The user can also choose to save the whole query history, or selected part of it into a *template* file. The template file can be later imported into NaLIX, with each query in it becoming a query template. The user can choose a query template, and change part of it to compose a new query. Facilities that allow the user to organize and edit existing templates are also provided. The query history is similar to the history of search words used in the famous SuperBook project [10]. The idea behind it is that searches often are not one-shot; the user needs to look at prior queries and their results to modify search accordingly. Keeping search history and providing query template preserves the user's prior search efforts, and provides the user a quick starting point when he/she needs to create new queries. More importantly, in our system, query history and template help the user to reinforce his/her knowledge on the linguistic coverage of the system without extra burden, since only validated queries may be kept in the history, and be used as templates.

Query Results Queries generated by the parse tree translator are sent to a database system supporting Schema-Free XQuery. The XML output is returned to the user via the GUI. NaLIX supports displaying search results in different views to serve different user needs: text view is good for small and simple results; tree view allows the user to navigate and analyze large and complex results, while grid view imposes structure on the query results and provides spreadsheet-like interface, and makes it easier to understand the search results. The XML output can also be saved as a file at a location given by the user.

Ontology-Based Term Expansions Typically the user is unfamiliar with the specific element/attribute names in the database. For example, in query "find the title of all publications", if the document being queried upon contains *book* instead of *publication*, the direct translation of the user query will not be able to generate the correct results, since no *publication* can be found. Term expansion has been studied extensively in the information retrieval literature [2, 5, 9] as a means to address such name-mismatch problems. We borrow techniques from these works, and use them in our system: term expansion based on WordNet [2] is used by default; ontology-based term expansion is performed whenever the ontology for a given XML document is available.

3. DEMONSTRATION OVERVIEW

We have implemented NaLIX as a stand-alone Java application to work with any XML database that supports Schema-Free XQuery and term expansion. In this demonstration, we use Timber [1] as the XML database, and MINIPAR [8] as our natural language parser. We will demonstrate the following features of NaLIX:

Interactive Query Formulation In NaLIX, users can specify natural language queries in two ways. First, users can load query template files, and choose from a variety of preloaded sample natural language queries, including the "XMP" queries in the W3C XQuery Use Cases [3]. Each template query can be altered to create a new query. Second, they can write their own natural language queries. Due to the limited vocabulary "understandable" by the system, certain terms cannot be properly classified. The user will be asked to rephrase the query into terms that we can understand. Error messages will be dynamically generated, tailored to the actual query causing the error and sent back to the user. Inside each message, possible ways to revise the query are also suggested, whenever possible. Figure 2 shows a snapshot of such an iteration in NaLIX.

Query History and Template Users can access and manipulate query history at any time from the GUI. Users can also choose to save the whole query history, or selected queries in it into a new *template* file. Users can also import existing template files, with each query in it becoming a query template. Users can also organize and edit the template files using NaLIX.

Visualizing the Translation While a query is being evaluated by Timber, users can see the visualized validated classified parse tree used in the translation process, as well as the Schema-Free XQuery expression generated from the original natural language query. Users can also turn on/off term expansion from the GUI to see how the translation process and the results of their queries are affected.

Visualizing the Results Users may select any of three result display views for returned results: text view, tree view, and grid view. The size of display window can be easily adjusted to better display the results, whenever needed. Warning messages, if any, will be displayed along with the results.

4. REFERENCES

[1] Timber: http://www.eecs.umich.edu/db/timber/.

[2] WordNet: http://www.cogsci.princeton.edu/∼wn/.

[3] XML Query Use Cases: http://www.w3.org/TR/xquery-use-cases/.

[4] I. Androutsopoulos, G. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *Journal of Language Engineering*, 1(1):29–81, 1995.

[5] A. Burton-Jones, V. Storey, V. Sugumaran, and S. Purao. A heuristic-based methodology for semantic augmentation of user queries on the Web. In *ER2003*, 2003.

[6] Y. Li, H. Yang, and H. Jagadish. Constructing a Generic Natural Language Interface for an XML Database. Submitted.

[7] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *VLDB*, 2004.

[8] D. Lin. Dependency-based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*, 1998.

[9] P. V. R. Navigli. An analysis of ontology-based query expansion strategies. In *IJCAI*, 2003.

[10] J. R. Remde, L. M. Gomez, and T. K. Landauer. Superbook: an automatic tool for information exploration - hypertext? In *Hypertext*, 1987.