DaNaLIX: a Domain-adaptive Natural Language Interface for Querying XML

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

Ishan Chaudhuri
University of Michigan
Ann Arbor, MI 48109
ishanrc@umich.edu

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

Satinder Singh
University of Michigan
Ann Arbor, MI 48109
baveja@umich.edu

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

ABSTRACT

We present DaNaLIX, a prototype domain-adaptive natural language interface for querying XML. Our system is an extension of NaLIX, a generic natural language interface for querying XML [4, 5]. While retaining the portability of a purely generic system like NaLIX, DaNaLIX can exploit domain knowledge, whenever available, to its advantage for query translation. More importantly, in DaNaLIX such domain knowledge does not have to be pre-defined; instead it can be automatically obtained from the interactions between a user and the system. In this demonstration, we describe the overall architecture of DaNaLIX. We also demonstrate how a generic system like DaNaLIX can take advantage of domain knowledge to improve its usability and query translation accuracy. In addition, we show DaNaLIX still possesses the portability of a generic system by using data collections from three different domains. Finally, we present how domain knowledge can be obtained through user interactions in an automatic fashion.

Categories and Subject Descriptors: H.3.3[**Information Systems**]: Information Search and Retrieval—*query formulations*; I.2.6[**Artificial Intelligence**]: Learning—*knowledge acquisition*

General Terms: Design, Human Factors

Keywords: Natural Language Interface for Databases, XML, Domain Adaptiveness, Domain Awareness

1. INTRODUCTION

XML is fast becoming the standard format to store, exchange and publish data for various domains. Not surprising, more and more non-database experts now rely on XML for their daily work. For example, a biologist may need to search a protein database on daily bases for her research. Unfortunately, the lack of an easy way to specify precise database queries often prevents these users from fully exploiting information stored in XML—formal database languages are precise yet too intimidating, whereas keyword-based search is too blunt to obtain precise answers.

To support easy yet precise access to XML databases for non-experts, we developed NaLIX (**Na**tural **L**anguage **I**nterface to **X**ML), a generic interactive natural language interface to an XML database [4, 5]. In NaLIX, an arbitrary English language sentence, which can be quite complex, is translated, potentially after reformulation, into an XQuery expression. This XQuery statement may include aggregation, nesting, and value joins, among other things, and 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 is performed to determine query nesting and grouping. To the best of our knowledge, NaLIX is the first generic interactive natural language interface to XML databases.

We have recently built DaNaLIX(**D**omain-**a**daptive **NaLIX**), an extension of NaLIX, which substantially improves the usability of the system by incorporating domain knowledge in query translation. First of all, the incorporation of domain knowledge helps to reduce the need for query reformulation for queries containing terms with domain semantics. Consider the query "*Return all the coauthors of Jim Gray.*" If no "*coauthor*" exists in the database, NaLIX will interactively guide the user to rephrase the query into one that our system can understand. However, DaNaLIX can utilize knowledge on the semantic meaning of "*coauthor*" in query translation and requires no query reformulation. In addition, the incorporation of domain knowledge may also help to improve the translation accuracy for such queries. For example, terms with domain semantics, such as "*most expensive*" in the query "*What is the most expensive book in year 2000*?", may simply be ignored by NaLIX, resulting loss in accuracy. In contrast, DaNaLIX can take advantage of existing knowledge that a "*most expensive*" book is one "*with the highest price*" in the translation.

DaNaLIX, unlike previous domain systems, does not require the existence of pre-defined domain knowledge, the acquisition of which requires expensive manual effort. Instead, our system starts with a generic framework and can incre-

*Supported in part by NSF grant IIS 0438909 and NIH grants R01 LM008106 and U54 DA021519.

SIGMOD '07, June 12-14, 2007, Beijing, China.
ACM 978-1-59593-686-8/07/0006.

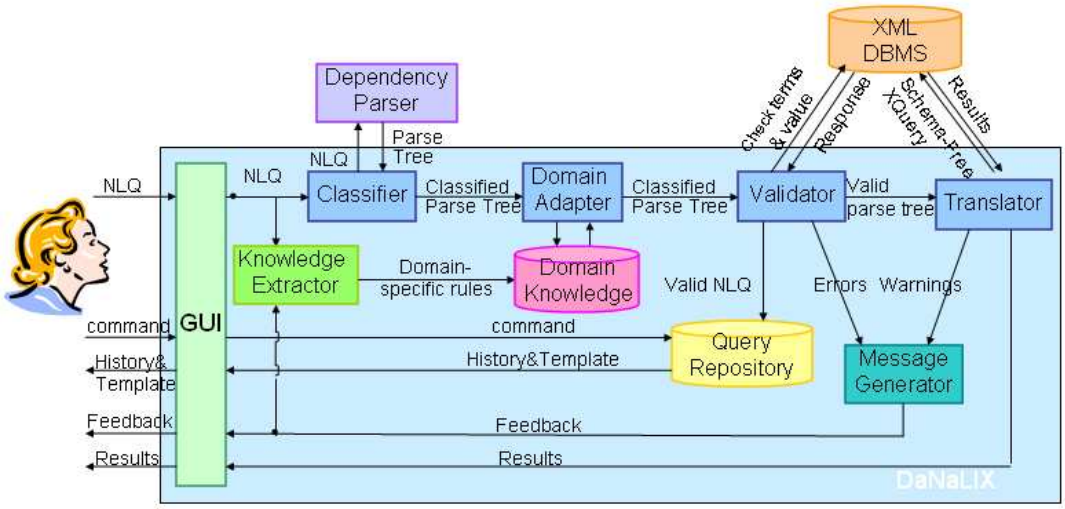


Figure 1: Architecture of DaNaLIX

mentally obtain domain knowledge in an automatic fashion from user interactions with the system. The resulting system retains the portability of a domain-independent system yet still benefits from the accuracy of a domain-dependent system.

The rest of the paper is organized as follows. Section 2 describes the architecture of DaNaLIX. Section 3 presents an overview of the features of DaNaLIX that will be demonstrated. For a more detailed description of the NaLIX system and the related work, the reader is referred to [4, 5]

2. OVERALL ARCHITECTURE

Figure 1 depicts the architecture of DaNaLIX. As can be seen, DaNaLIX consists of three parts: one part is responsible for query translation from natural language queries to XQuery expression, including *classifier*, *domain adapter*, *validator* and *translator*; another part provides support for interactive query formulation, including *query repository* and *message generator*; finally, the *knowledge extractor* supports learning from user interactions. A typical user interaction process in DaNaLIX engages all the three parts.

Query translation The translation from a English query into an XQuery expression involves four main steps.

- *Parse Tree Tokenization*: First, the *classifier* identifies words and phrases that can be directly mapped into XQuery components in the parse tree of a given sentence.
- *Domain Knowledge Incorporation*: Then, the *domain adapter* checks existing domain knowledge and transforms the parse tree by applying any relevant rules. Domain knowledge is represented as a set of rules representing the mapping between a partial parse tree containing terms with domain meanings with one expressed in terms understandable by a generic system like NaLIX. An example rule is illustrated in Figure 3.
- *Parse Tree Validation*: A parse tree may still contain terms that cannot be understood by the system, even after transformed using domain knowledge known to the system. The *validator* checks whether the parse tree (possibly after rule-based transformation) is one that we know how to map into XQuery. It also checks whether the element/attribute names and/or values contained in the user query can be found in the database. 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*: Finally, the *translator* will utilize the structure of the natural language constructions as reflected in the parse tree to produce an XQuery expression.

Interactive query formulation The *domain adapter* may find multiple applicable rules for the same query; the most relevant one (based on an internal probabilistic model) will be applied; the modified tree is then validated and translated. Meanwhile, the user is given the option of viewing the results without any rules being applied. Additionally the user can view all the applicable rules and select any of them for application. During parse tree validation, the parse tree of some queries may be found to invalid and cannot be properly mapped into XQuery expressions by the system. For such queries, the system will provide feedback messages[1] that identify problems with the query and suggests possible reformulations to the user.

Knowledge Extraction After a successful query, the *knowledge extractor* will search for similar queries in the history of failed queries since the last successful query to the present.[2] If one is found, the system will ask the user whether the two queries are actually semantically equivalent. If the user confirms that they do, the *knowledge extractor* will build a new rule based on these queries and add it to the repository of domain-knowledge used by the *domain adapter*.

Figure 2(a) illustrates an example iteration where a query with a domain term is reformulated into one that the system can understand. A rule has been successfully extracted from this process (a visualization of this rule is shown in Figure 3). This rule is then applied later on to transform another query that contains the same domain term; the resulting parse tree is successfully validated and translated into XQuery (Figure 2(b)).

[1]More detailed descriptions on the generation of feedback messages can be found in [5].
[2]Based on the similarity of their parse trees

3. DEMONSTRATION OVERVIEW

We implemented DaNaLIX as a stand-alone Java application to work with any XML database that supports Schema-Free XQuery. In this demonstration, we use TIMBER [3] as the XML DBMS, and MINIPAR [6] as our natural language parser. The main focus of this demo is to illustrate how domain knowledge learning and incorporation can be integrated into a domain-independent framework. In particular, our demonstration will showcase DaNaLIX using data collections from three different domains, namely *bibliography*, *geography* and *restaurant* [1, 2]. For each collection, we have pre-defined a small set of domain knowledge and queries.

The demonstration proceeds in the following steps:

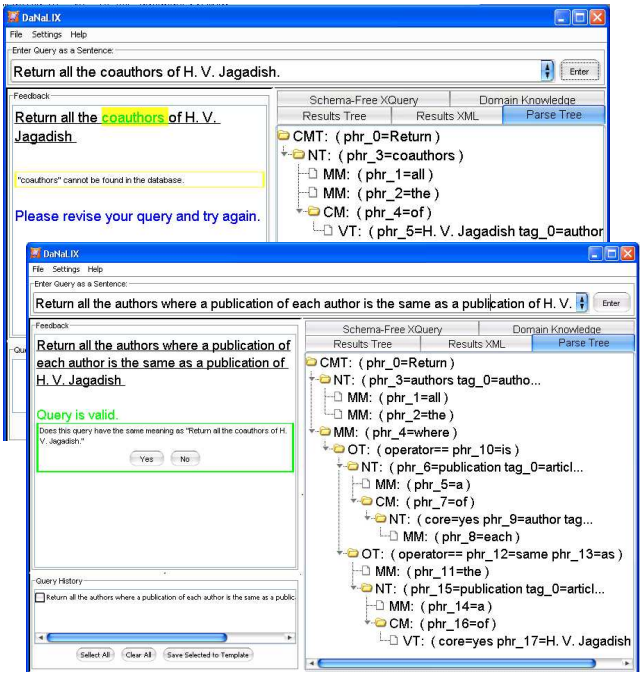
Choice of domain A user may start with choosing one database from the given three different domains. This selection determines both the XML documents to be queried upon as well as the pre-defined domain knowledge assertions initially available to the *domain adapter* in DaNaLIX.

Domain-knowledge application The user can then run example queries for the domain of his/her choice. The system also provides the user the option to run a query with domain knowledge applied. The user can turn this option on/off for the same query to see how the use of domain-knowledge affects the accuracy of query translation of DaNaLIX. In addition, when multiple rules are found to be applicable to the same query, the user is given the opportunity to choose one to be used in the translation.

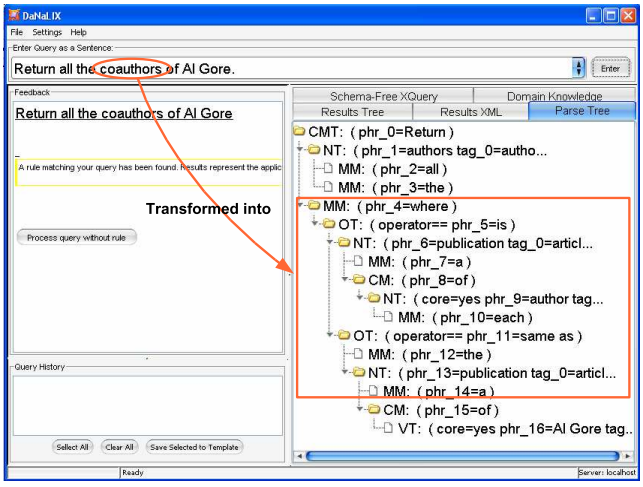
Visualization of the Translation While a query is being evaluated by Timber, the user can also view the intermediate results at each step of the query translation, including both the classified parse tree and the Schema-Free XQuery expression generated from the original natural language query. If a rule is used during the query translation, the user can also view the transformed parse tree after applying the rule as well as the specific rule in domain-knowledge entry that has been used. After the results are returned from Timber, the user may choose to view the results in text or tree view.

Domain-knowledge acquisition The user can learn about the acquisition of domain-knowledge via an example iteration of query formulation. The user can see the rule generated by the *knowledge extractor*. She can also see how the newly learned rule helps query translation for a new query.

User interaction Finally, the user may interact with the system by issuing her own queries. New domain-knowledge may be learned on-the-fly from the user interactions and be used by system for any new queries issued by the user. The user can also see a visualization of the rules, if any, learned from the interactions.



(a) Example iteration on knowledge extraction



(b) Knowledge incorporation

Figure 2: Example interations

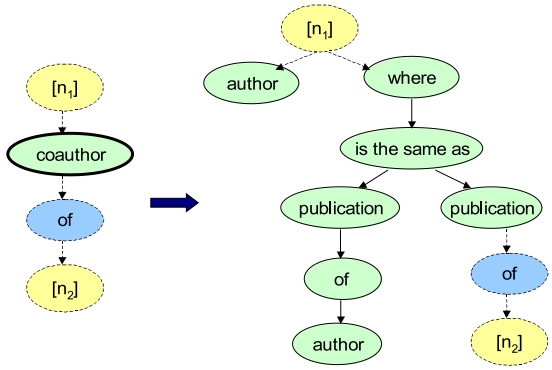


Figure 3: Sample Rule Extracted by DaNaLIX from the iteration in Figure 2(a) (Terms in dashed cycles provide context for rule-based transformation only.)

4. REFERENCES

[1] DBLP: `http://dblp.uni-trier.de/xml/`.

[2] Geoquery & Restaurant Data: `http://www.cs.utexas.edu/users/ml/nldata.html`.

[3] TIMBER: `http://www.eecs.umich.edu/db/timber/`.

[4] Y. Li et al. NaLIX: an Interactive Natural Language Interface for Querying XML. In *SIGMOD*, 2005.

[5] Y. Li et al. Constructing a Generic Natural Language Interface for an XML Database. In *EDBT*, 2006.

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