Using XMorph to Transform XML Data

Curtis E. Dyreson
Utah State University
Logan, Utah

+1 435 797-0742

Curtis.Dyreson@usu.edu

Sourav S. Bhowmick
Nanyang Technological University,
Singapore

+65 6790-4320/6929

assourav@ntu.edu.sg

Kirankanth Mallampalli
Utah State University
Logan, Utah

+1 435 797-0742

kirankanth.mallampalli@usu.edu

ABSTRACT

XMorph is a new, shape polymorphic, domain-specific XML query language. A query in a shape polymorphic language adapts to the shape of the input, freeing the user from having to know the input's shape and making the query applicable to a wide variety of differently shaped inputs. An XMorph query specifies the shape of the output. The XMorph query engine transforms the input to the desired shape by shredding an XML document to a graph of closest relationships, and performing a *closeness preserving* transformation. We plan to demonstrate XMorph using a Java applet, which can also be used by the audience during the demonstration, to evaluate various XMorph queries. The applet will show the output, the shapes generated by the query, and report on potential data loss in a transformation.

1. INTRODUCTION

The goal of this research presented in this demonstration is to make it easier for users to transform XML data. One factor that complicates XML query languages is that a query writer has to know the *shape* of the data to effectively query it. Long before the advent of XML E. F. Codd wrote about this problem in his foundational paper on the relational model [4]. Codd critiqued the hierarchical model because it relies on *path expressions* to locate data. A path expression is a specification of a path in a hierarchy. Codd presented five hierarchies for a simple database and demonstrated that, in general, a path expression formulated with respect to one hierarchy would fail on some other. For instance, suppose that `supplier/part` locates parts "below" suppliers. The same expression fails when the data is organized differently, say when `parts` are above `suppliers`.

In this demonstration, we will show how to use a new, shape polymorphic data transformation language, called XMorph, which we introduced previously [6]. The demonstration will use a Java applet to take the audience through a series of XMorph queries. The applet will show the output, the shapes generated by the query, and report on potential data loss in a transformation. We invite readers to visit the XMorph project website to experiment with XMorph in an on-line demo or download the Java

implementation.[1] In XMorph, a query writer only needs to specify the shape of the output, and this specification is simple. XMorph extracts and transforms the data into the specified shape. The transformation is *closeness-preserving*, that is, it preserves the neighborhood of data that is close to a node in the input, keeping it close in the output. XMorph offers the following features in a data transformation language.

Easy to specify and transform the data's shape. The primary component of XMorph is a *morph* in which the user declares the desired shape of the result. XMorph reorganizes the source data to match the specified shape.

Shape polymorphism. In XMorph, only the shape of the output needs to be given, the query adapts to the shape of the input. *Shape polymorphism* was first described for object-oriented languages by Jay and Crockett [8]. The notion applies to database query languages as follows: a language is *shape polymorphic* if a query evaluated on the *same* data in *different* structures yields (approximately) the *same* result[2].

XQuery support. XMorph can be evaluated to produce XML, but can also be translated to XQuery.

Ability to identify information loss. An XMorph query always transforms the input data to the desired shape, but not all transformations preserve the data. Prior to evaluating a query, the XMorph query engine can classify each transformation as *reversible* (no information loss), *non-additive* (the transformation can lose some information), *inclusive* (information might be added), or both additive and non-inclusive.

Ability to treat attributes as indistinct from sub-elements. Though data modelers often arbitrarily choose to use attributes rather than subelements, XMorph queries do not force users to differentiate between them.

Easy creation of groups. XQuery 1.0 has ad-hoc support for groups using a distinct-values function. XQuery 1.1 adds support for grouping in aggregation. XMorph supports both persistent and dynamic group creation for data transformation.

Vocabulary translation. To use XMorph, a user has to know the "vocabulary" (e.g., the names of the elements) in a data collection. But XMorph also supports vocabulary translation, so that users can change their terminology.

It is important to note that XMorph is a domain-specific language, that is, a *special purpose* language, lacking many features found in a general purpose, Turing-complete language like XQuery. XMorph makes it easier to specify data transformations.

---

[1] `http://www.cs.usu.edu/~cdyreson/pub/XMorph`

[2] The same result *modulo* duplicates, ordering, and attribute/sub-element swaps.

```
                      bibliography
              book                    book
    title author price publisher  title  author price publisher
      |    |     |     |            |      |     |     |
     DB  name  46.95 Addison    Automata name 9.99 Academic
          |          Wesley            |            Press
       E.F. Codd                    E.F. Codd
```

Figure 1 Authors listed by book

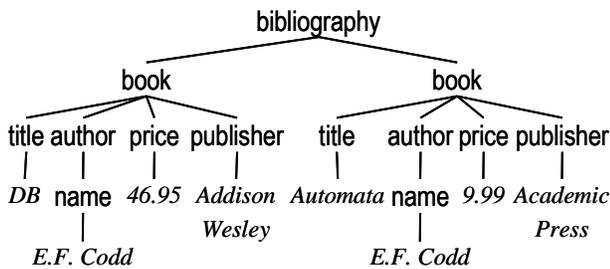**Figure 2 List titles by author, query and result shown**

Figure 3 List the authors for the book "DB"

2. DEMO OVERVIEW

This section gives a short tutorial on XMorph through a series of examples of increasing complexity. The examples will transform the data about books written by E. F. Codd shown in Figure 1. We will use a Java applet to demonstrate XMorph. The applet can be downloaded from the project's website. The website also gives the ANTLR syntax for XMorph, and allows users to download the code, which is a pure Java implementation. The demonstration applet is self-contained. It runs an "in-memory" version of XMorph, in which the XML data to transform is parsed and shredded to memory-resident tables; the full XMorph package shreds to disk-resident, BerkeleyDB tables.

The primary function in XMorph is a *morph*, which places children below a parent in the result. The parameter of the morph function is a *pattern*, which specifies the shape of the result. Figure 2 gives a simple example. The figure shows a screenshot of the applet. The "XMorph query" text area contains a query that is intended to list the titles written by each author extracted from a collection of book data. The pattern specifies that `<title>` and `<name>` elements are placed as children of `<author>`[3]. Only `<title>` and `<name>` elements that are *closest* to an `<author>` element are placed within that `<author>`. *Closeness* is intuitively the idea that authors are closely related to the titles of their own books and articles (and their own names), but not close to titles written by others (or the names of others). Figure 2 also shows the result of the query when evaluated on the data in Figure 1 in the "Output of query" text area. The "Data's shape" box shows the "shape" of the data (essentially a DataGuide [7]). An XMorph query transforms the data's shape into a new shape, which is shown in the "Query's shape" box in the figure. The query's shape dictates the shape of the output. Finally, the "Information loss" area turns on a radio button to indicate whether the transformation potentially loses information. The analysis of information loss considers the relationship between the shape of the data and that of the query. The transformation could be *reversible*, indicating that each title and name remains close to a specific author through the transformation (no relationships were lost or introduced). Less desirably, the query might create new closest relationships (be *inclusive*), or retain only some of the existing relationships (be *non-additive*), or be neither. We have recast the terminology of data transformations to a type system terminology as follows: *strongly typed* is reversible, *weakly typed with narrowing* is non-additive, *weakly typed with widening* is inclusive, while *weakly typed* is both additive and non-inclusive. The query might have a semantic *type error* if it uses a term not supported by the data's vocabulary.

Let's explore the notion of a *closeness preserving transformation* in more detail. XMorph distills an XML document into a graph of *closest relationships*. Two nodes are related in the graph if the distance between the nodes is the minimal distance between any pair of nodes of that *type* [15]. Figure 4 depicts the closest graph for the data of Figure 1. For the data of Figure 1, we assume that the *type* of a node is its label. The closest graph also has group edges that represent (persistent) group relationships, e.g., there is a group edge between a pair of vertices if they have the same type, same value, and the value is not empty. In the figure, a solid line represents a closest edge, and a dashed line is a group edge.
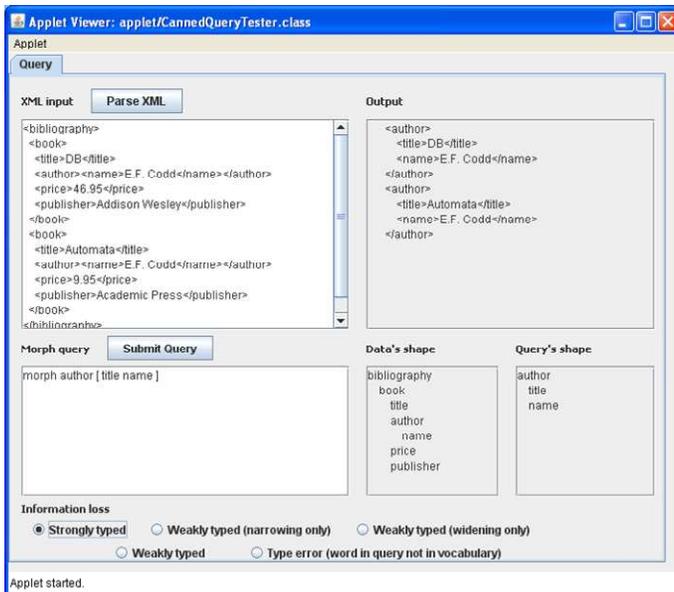
---
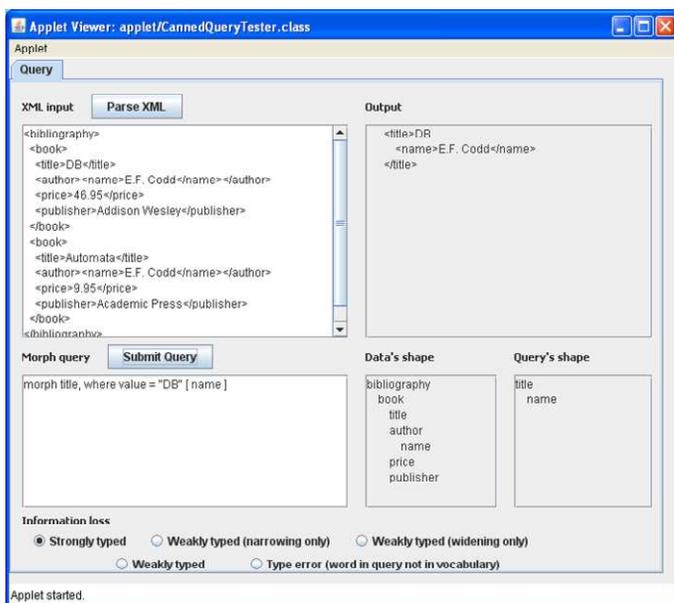
[3] In the explanation of this example, we've assumed elements rather than attributes, but, in general, "author", "name", or "title" could be either an attribute or element.
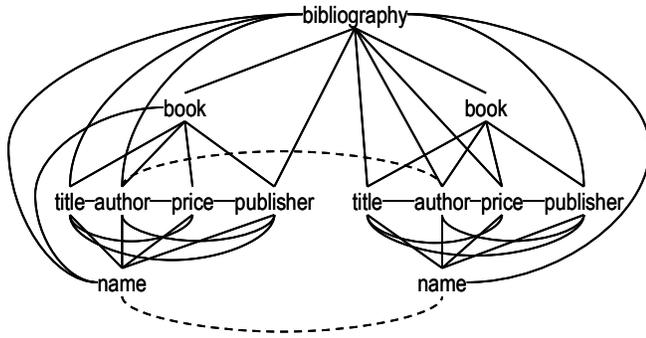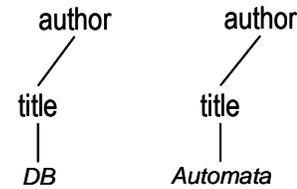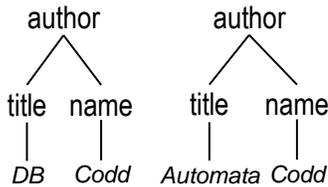
Figure 4 The closest graph for the data of Figure 1

a) Put title elements in place

b) Relate to a closest author

c) Put closest names in place

Figure 5 Transforming the data

A closeness preserving transformation changes the shape of the data, but preserves the closest relationships in the graph. So in the evaluation of the query of Figure 2 an `<author>` becomes a parent of only the closest `<title>` and `<name>` elements. The transformation is (logically) depicted in Figure 5. First, `<title>` elements are related to the closest `<author>`. Next the `<name>` elements are similarly related.

A morph can be restricted to select individual authors. Suppose we want only the titles "DB". Then we can use the query given in Figure 3.

There may be duplicate authors in the data, but authors can be *grouped* to eliminate the duplicates. Figure 6 shows an example that groups titles by an author's name using a 'group' modifier. Modifiers are listed after a label, separated by commas. The group modifier uses the default, persistent grouping for author (e.g., author is grouped by its 'key' as specified by the data's schema, or by the distinct-values function for a schema-less data collection). Authors could also be dynamically grouped during query evaluation, by specifying a grouping pattern.

XMorph also supports *mutation* of a shape. A mutation is similar to a morph but unlike a morph the entire shape is implicitly
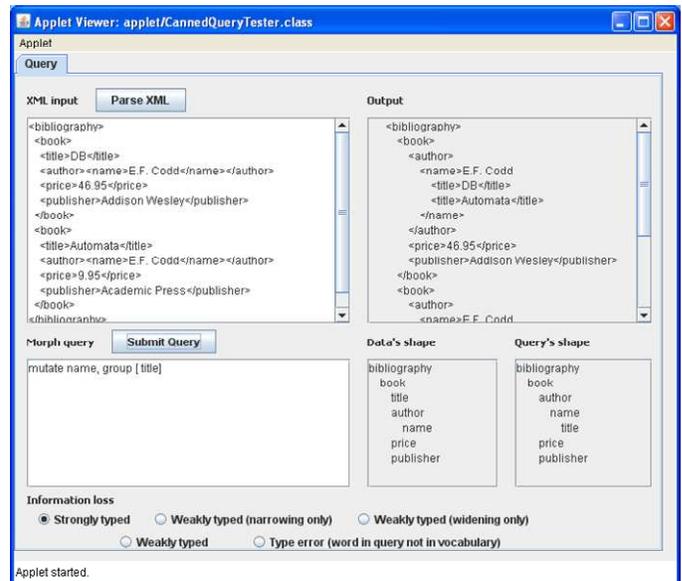

Figure 6 List titles grouped by author E. F. Codd

involved rather than just the portion explicitly given in the pattern. A mutation is given in Figure 7. The mutation explicitly lists only two types, but it outputs the entire shape of the data, with a mutation. It moves `<title>` elements to within the closest `<name>` elements.

XMorph transformations can also be composed as shown in Figure 8. The previous query is piped into an extra morph.

In addition to the simple transformations, the demonstration will also cover other features of XMorph such as the translation to XQuery, transformation information loss analysis, dynamic grouping, morph composition, and type analysis reporting.

3. RELATED WORK

Previous shape-related research on making it easier to query XML can be broadly classified into five categories.

1) **Query relaxation/approximation.** Techniques have been proposed to find data that inexactly or approximately matches a query [1],[9] by relaxing the notion that only crisp answers be returned by query evaluation [3]. These techniques implicitly generate and explore a space of shapes that are related to the shape of the input and/or query, usually all shapes within a given edit distance. But these techniques are orthogonal to XMorph (they could be employed to create approximate XMorph).

2) **Query correction/refinement**. In this approach, similar queries are automatically generated when a query is unable to be satisfied [3] or a query is refined [2]. The user guides the search for the query they want to execute by choosing among alternatives, and these alternatives implicitly involve trying different shapes for the input. These approaches are also orthogonal to XMorph, which does not require interactive user input.

3) **XML search engines**. XML search engines have simple, easy-to-use interfaces [5],[12]. Like XMorph, they de-couple queries from specific hierarchies. But unlike XMorph, XML search engine queries typically do not transform data. XMorph straddles the middle ground between XML search engines and path expression-dependent XML query languages by borrowing useful techniques from each end of the spectrum.
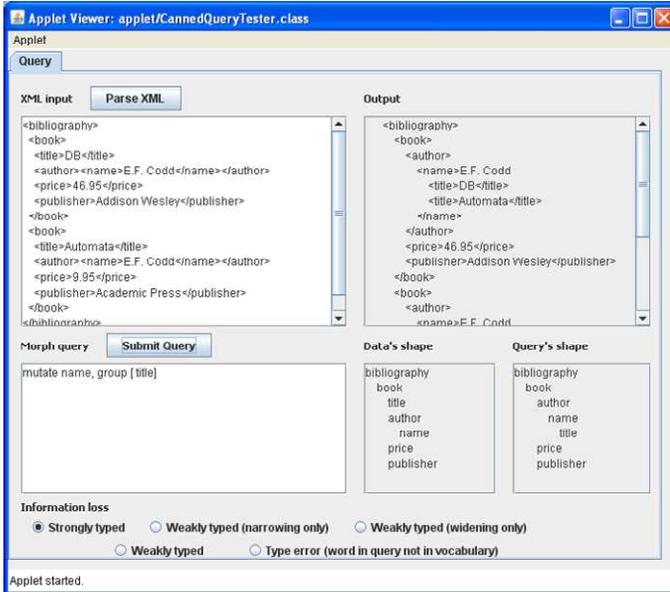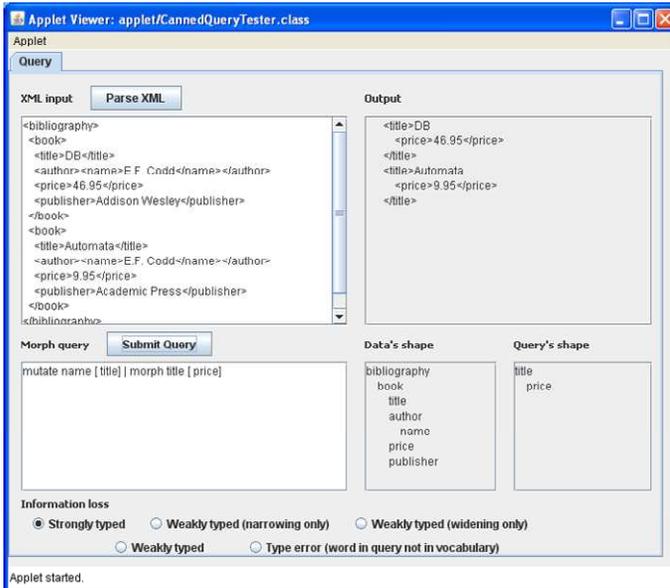
Figure 7 Mutating the data

Figure 8 Composing transformations

transformations of XML data [10],[13]. These languages hide from users many of specification details that would be needed in a language such as XQuery or XSLT. However, these techniques, unlike XMorph, are not shape polymorphic. A transformation query might have to be rewritten for a different hierarchy.

4. SUMMARY
The demonstration will guide the audience through examples in a new language called XMorph. The audience can download our applet from the project's website and try their own queries during the demonstration.

5. REFERENCES

[1] S. Amer-Yahia, S. Cho, and D. Srivastava, "Tree Pattern Relaxation," in *EDBT*, 2002, pp. 89-102.

[2] A. Balmin, L. Colby, E. Curtmola, Q. Li, and F. Ozcan, "Search Driven Analysis of Heterogeneous XML Data," in *CIDR*, 2009.

[3] T. Brodianskiy and S. Cohen, "Self-Correcting Queries in XML," in *CIKM*, 2007, pp. 11-20.

[4] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. CACM 13(6): 377-387 (1970).

[5] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv, "XSEarch: A Semantic Search Engine for XML," in *VLDB*, Berlin, Germany, 2003, pp. 45-56.

[6] C. Dyreson, S. Bhowmick, K. Mallampalli, A. Jannu, and S. Zhang, "XMorph: A Shape-Polymorphic, Domain-Specific Data Transformation Language," in *ICDE*, 2010, pp. 1000-1003.

[7] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," in *VLDB*, Aug. 1997, pp. 436-445.

[8] C. B. Jay and J. R. B. Crockett, "Shapely types and shape polymorphism," in *European Sym. on Programming*, 1994, pp. 302-316.

[9] Y. Kanza, W. Nutt, Y. Sagiv, "Flexible Queries over Semistructured Data," in *PODS*, June 2001, pp. 40-51.

[10] S. Krishnamurthi, K. Gray, and P. Graunke, "Transformtion-by-example for XML," in *Workshop of Practical Aspects of Declarative Languages*, LNCS 1753, 2000, pp. 249-262.

[11] Y. Li, C. Yu, and H. V. Jagadish. "Schema-Free XQuery," in *VLDB*, Sep. 2004, Toronto, CA, pp. 72-83.

[12] Z. Liu, J. Walker, and Y. Chen. "XSeek: a semantic XML search engine using keywords," in *VLDB*, 2007, pp. 1330-1333.

[13] T. Pankowski. "A High-Level Language for Specifying XML Data Transformations," in *ADBIS*, LNCS 3255, 2004.

[14] Y. Xu and Y. Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases," in *SIGMOD*, 2005, pp. 527-538.

[15] S. Zhang and C. Dyreson, "Symmetrically Exploiting XML," in *WWW*, May 2006, Edinburgh, Scotland, pp. 103-111.

4) **Structure-independent querying**. The final category of research is more clearly applicable to XMorph. The idea of exploiting a least common ancestor (LCA) when querying data has been explored. Schema-free XQuery uses the meaningful LCA [11], XSeek exploits node interconnections [5], and others use the smallest LCA [12],[14]. Similarly, we proposed a closest XPath axis [15] based on the LCA. In contrast to all of the above research, XMorph focuses on the use of the LCA in data transformation, characterizing the potential information loss, and explicitly specifying and mutating the shape of data.

5) **Declarative transformations**. Similar to XMorph there are other proposals for declarative languages for specifying