ShreX: Managing XML Documents in Relational Databases

Fang Du
OGI/OHSU
fangdu@cse.ogi.edu

Sihem Amer-Yahia
AT&T Labs – Research
sihem@research.att.com

Juliana Freire
OGI/OHSU
juliana@cse.ogi.edu

Abstract

We describe ShreX, a freely-available system for shredding, loading and querying XML documents in relational databases. ShreX supports all mapping strategies proposed in the literature as well as strategies available in commercial RDBMSs. It provides generic (mapping-independent) functions for loading shredded documents into relations and for translating XML queries into SQL. ShreX is portable and can be used with any relational database backend.

1 Introduction

As applications manipulate an increasing volume of XML data, there is a growing need for reliable systems to store and provide efficient access to these data. The use of relational database systems for this purpose has attracted considerable interest with a view to leveraging their powerful and reliable data management services.

In order to store an XML document in a relational database, the tree-structure of the XML document must first be mapped into an equivalent, flat, relational schema. XML documents are then shredded and loaded into the mapped tables. Finally, at runtime, XML queries are translated into SQL, submitted to the RDBMS, and the results are then translated into XML.

There is a rich literature addressing the issue of storing XML documents in relational backends. Several mapping strategies (e.g., [3, 5, 6, 10, 9]) and query translation algorithms (see [7] for a survey) have been proposed. In addition, support for XML storage is already available in most commercial RDBMSs. Unfortunately, existing XML-to-relational mapping solutions suffer from several drawbacks. None of these solutions addresses all the storage problems in a single framework. For example, works on mapping strategies often have little or no details about query translation [7]. Although it has been shown that the efficiency of a mapping depends on the data and on the requirements of the applications that use the data [3], many of the available mapping solutions hard-code mapping choices; and whereas some of the solutions proposed by relational vendors do provide flexible mechanisms to define mappings, they are proprietary, i.e., tied to one relational backend, making the shredding and query translation algorithms system- and mapping-dependent.

ShreX (Shredding XML) is a freely available system that addresses many of the limitations of existing mapping systems. To the best of our knowledge, ShreX is the first system to provide a comprehensive solution to the relational storage of XML data. In ShreX, an XML-to-relational mapping is specified through annotations over an XML Schema, making the mapping easy to define as well as validate. By combining different annotations, a wide range of mappings can be expressed, including all mapping strategies proposed in the literature as well as strategies supported by database vendors. ShreX also provides generic (mapping-independent) functions for document shredding and query translation. This is made possible by an API which provides access to the mapping information.

In what follows, we give an overview of the key features of ShreX. In Section 2, we describe the architecture of the system including the mapping interface, the shredder and the query translator. The demonstration is described in Section 3. We review related work in Section 4 and conclude in Section 5.

2 The System

The main components of the ShreX system are shown in Figure 1. In ShreX, a mapping is defined by adding annotations to an XML Schema which indicate how elements and attributes should be stored in a relational database (e.g., as columns, as tables). The annotation processor parses an annotated XML Schema, checks the validity of the mapping and creates the corresponding relational schema. In addition, the mapping information is made persistent in the mapping repository. The document shredder accepts as in-
put a document and uses the mapping API to access the information in the mapping repository to generate the tuples and populate the tables in the relational schema. The mapping repository is also accessed by the query translator, which generates SQL queries from XML queries.

Figure 1: ShreX Architecture

Below, we give a brief overview of the mapping definition framework, and of the various modules of ShreX. The reader is referred to [1] for more details.

2.1 Mapping Definition Framework

A ShreX mapping is expressed by annotating an XML Schema. This not only makes the mapping definition portable, i.e., independent from the underlying relational database, but also expressive and extensible. Mapping specifications also enable useful analyses, for example, to ensure that a mapping is valid (Section 2.2).

Annotations can be associated to attributes, elements and groups in the input XML Schema. Their syntax corresponds to adding attributes from a namespace called shrex to a given XML Schema. The attributes supported by ShreX are shown in Table 1. Figure 2 illustrates the use of some of the annotations (shown in boldface). The sample schema describes information about shows, where a SHOW element can also be mapped into a CLOB, using the annotation maptoclob.

![XML Document Structure](image)

Figure 2: Annotated movie schema

<table>
<thead>
<tr>
<th>TABLE SHOW( ID VARCHAR(128), Showyear NUMBER(4))</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE Showtitle( ID VARCHAR(128), ParentID VARCHAR(128), TITLE VARCHAR(512))</td>
</tr>
<tr>
<td>TABLE REVIEW( ParentID VARCHAR(128), source VARCHAR(128), ordinal VARCHAR(128), attrname VARCHAR(128), flag VARCHAR(128), value VARCHAR(128))</td>
</tr>
<tr>
<td>TABLE AKA( ID VARCHAR(128), ParentID VARCHAR(128), AKA VARCHAR(512))</td>
</tr>
</tbody>
</table>

Figure 3: Relational configuration for movie schema

Mapping schemaless documents and mixing strategies.

The use of annotated schemata in ShreX does not preclude the system from expressing generic (schemaless) mappings. For example, in Figure 2, the annotation edgemapping="true" in the element REVIEW indicates that REVIEW and its descendants are mapped using Edge mapping [6], i.e., a single table to store all the REVIEW elements and contents. This functionality is specially useful to map elements whose structures are not known in advance, such as for example, elements of type ANYTYPE.

Annotations naturally allow the definition of mappings that combine different mapping strategies. Note that in this example, part of the document is mapped using Edge, and part is mapped using KFO.

2http://www.oclc.org/dewey/about/about_the_dlc.htm.
Transformation-based mappings. Additional mapping strategies are supported by combining ShreX annotations with the schema transformations proposed in [3]. For example, if repetition split is applied to AKA in the original schema, i.e., AKA* → AKA?, AKA*, the first occurrence of AKA could be inlined in the table SHOW:

```
TABLE SHOW( ID VARCHAR(128), Showyear NUMBER(4), AKA VARCHAR(512))
```

2.2 Annotation Processor

This module is in charge of parsing an annotated XML Schema, checking the validity of a mapping, generating a mapping repository, and producing the CREATE TABLE statements necessary to construct the relational schema. In order to check the validity of a mapping, the annotation processor validates the input (annotated) schema against an XML Schema for annotations [1]. Validity checks include verifying whether annotations are attached to the appropriate elements, and if table names are unique in the mapping definition. Additional checks, such as verifying whether a mapping is lossless, are also possible.

Writing an annotation for every element and attribute definition in an XML Schema can be tedious, especially for large schemata. ShreX provides a set of default rules that is used to complete mapping specifications. In fact, using these default rules, the system is able to automatically map an XML Schema without any user input. It is worthy of note that users can add to or override these rules.

2.3 Mapping Repository and API

Mapping information is processed and stored in a database. By making this information persistent, ShreX avoids the need to re-parse a mapping specification each time a document is loaded into the target database or that a query needs to be translated into SQL. ShreX provides an API to the mapping repository that allows access to information such as, how elements and attributes are mapped (isInlined(ElemName, AttName)), which mapping is used to capture document structure (getStructureScheme()), and which tables are available in the relational schema (getTableInfo( TableName)) (see [1] for details). This API allows users to write mapping-independent code which is not tied to specific features of a particular mapping.

The API also contains functions that expose information about the schema being mapped. These functions are useful both during shredding and query translation. For example, in order to translate a descendant step //t, the query translator needs to determine all paths from the root to tag t – in ShreX, this can be done through a call to pathToTag.

2.4 Document Shredder

The shredder is in charge of generating tuples, field values and CLOBs from an input document. It was designed to be generic and independent from the mapping specification: it uses the mapping API to retrieve information about how a particular element or attribute is mapped. Since mapping annotations are specified using attributes from a different namespace, the document shredder can validate the input XML document against the annotated Schema. Tuples are generated while the document is parsed, using a standard XML parser. In our implementation, we use the SAX interface of Xerces [11], which is both efficient and scalable. For example, ShreX is able to shred and load a 1GB document into DB2 in less than 30 minutes. It is worth pointing out that even significantly smaller files cannot be loaded in commercial RDBMSs using their XML extensions. Consistently with what has been reported in [12], we were not able to load documents larger than 10MB using DB2’s XML Extender.

Users can set various parameters for the shredder, e.g., target database system, login information, bulk loading option. These parameters can be set either through the command line or through a configuration file.

2.5 Query Translator

In the current implementation, the query translator supports a subset of XPath that includes child and descendant axes; position-based predicates [position()=n]; and simple path predicates, to SQL. Similar to the document shredder, the query translator does not hard-code mapping choices, instead it uses the information provided by the mapping API to dynamically decide how to perform the translation.

3 Demonstration Overview

We will demonstrate the various features of ShreX and its utility for building applications that need to store and query XML data in relational databases.

Specifying mappings. We will show how different storage mapping strategies can be represented using our mapping specification. Users will be able to define mappings by choosing from a variety of XML schemata or creating their own schema. Using ShreX, they will annotate a schema, validate the corresponding mapping, and create the corresponding relational schemata. We will show how this process is simplified by the ShreX graphical user interface (GUI), which allows users to browse and select tables and fields in the relational schema and visually see the corresponding XML elements and attributes, and vice versa.

Shredding and loading. After a relational schema is created, users will be able to select a target RDBMS and instruct the system to shred and load a document into the target RDBMS. Shredding and loading can be done through the command-line, or from the GUI. Users can load the XML documents directly into the relational tables (i.e., they can be bulk-loaded), or generate loading commands and tuples.

Querying. Users will be able to input XPath queries and see the corresponding (translated) SQL queries as well as have these queries executed against the relational backend.

3 An XQuery translator is currently under development, and will be available in the next version of ShreX.
Table 1: Annotation Attributes. Each row in the table contains an annotation attribute, its target (i.e., element, attribute, and group to which it applies), its possible values and its action depending on its value.

4 Related Work

Bourret et al [4] developed XML-DBMS, a generic tool for loading XML documents into relational tables. Although similar to ShreX in motivation, the mappings supported by this tool are limited to the basic, shared, and hybrid techniques described in [10]. In addition, XML-DBMS has no support for query translation.

MXM [2] has been proposed as a declarative mechanism to express XML-to-relational mappings. Our mapping specification shares the flexibility of MXM while having the advantage of using an XML Schema syntax and providing a comprehensive set of tools.

Although XML support in commercial relational engines is improving rapidly, there is a wide variation in the supported features. Some practical problems include proprietary solutions, lack of flexibility and scalability. To define a storage strategy, the IBM DB2 XML Extender requires users to write a Document Access Definition specification; consequently, developers must learn a new language in order to use DB2 (and only DB2) as a backend. The mapping facilities provided by Oracle 9iR2 are not flexible enough to specify many useful strategies, for example, it is not possible to specify that part of the data is to be stored using a generic mapping such as Edge [6]. SQLServer’s OpenXML requires that documents be compiled into an internal DOM representation, which greatly limits its scalability.

5 Conclusion

To the best of our knowledge, ShreX is the first comprehensive system for mapping, loading and querying XML documents. ShreX has many novel features including the ability to mix mapping strategies and to specify a document structure scheme. We designed ShreX to be modular and extensible. And by making the source code available, we hope ShreX will serve as a platform to develop and evaluate new mapping strategies, query translation and optimization algorithms.

Acknowledgments. The National Science Foundation partially supports Juliana Freire under grant EIA-0323604.

References