

A Java Based Parser Software for Converting XML Documents to the ER Model and Relational Databases

Sikha Bagui and Damien Walker
Department of Computer Science
University of West Florida
Pensacola, FL 32514
bagui@uwf.edu

Abstract

In this paper we present a conceptual design model that was used to create a tool for converting XML documents and their data to an ER model and then a relational database. This tool, a Java based parser software, parses and reads in XML documents, maps them to an ER model, and then maps the ER model to a relational database. The software also creates an Access database from the relational mapping. This tool will be very useful to avail of the benefits of relational database technology once the XML data is in relational database format.

Keywords

XML, Database Design, Entity Relationship Diagrams, Relational database.

1 Introduction

XML, a meta-markup language developed by the World Wide Web Consortium (W3C) in 1996, is universally recognized as a standard document for information interchange. Large amounts of data, both financial and business data, and even data obtained from satellites, that is, most of the data in today's web-driven world are being continuously converted to XML[5].

In this paper we present a conceptual design model (and tool) for converting XML documents and their data to the ER model[2, 3], which can then be mapped to a relational database. Our tool, a Java based parser software reads in XML documents, converts them to an ER model, maps the ER model to a relational database and then creates an Access database from the relational mapping. The main challenge in developing a conceptual model to map XML documents to the ER model and subsequently relational databases is that, XML files are hierarchical in nature and the ER model and relational tables are by no means hierarchical in nature.

In the next section we present the XML to ER mappings, the basis for the Java based parser.

2 XML to ER Model Mappings

XML documents generally have two types of files, an XML file and a Document Type Definition (DTD) file[4]. The DTD file is basically the schema for XML documents. Our Java based program first reads in the DTD file, followed by the XML files.

2.1 Reading the DTD file

The `readDTD` module of the program reads the DTD file. The DTD file provides several pieces of information: (i) the entities; (ii) the attributes, their data types and their relationships to the respective entities; (iii) the relationships between the entities, which may be partial or full.

2.1.1 Mapping entities

A DTD tag is of the form: `<!ELEMENT element-name (sub-element +)>`

The `element-name` would map to an entity in the ER model.

2.1.2 Mapping attributes

Attributes can be mapped by three different DTD tags formats:

(i) The DTD tag format: `<!ELEMENT Instructor (lastName, firstName)>`

Maps to: The `Instructor` entity has the `lastname` and `firstname` attributes associated with it. This tag format sets the dependency of the attributes to an entity.

(ii) The DTD tag format: `<!ELEMENT firstName (#PCDATA)>`

Maps to: `firstName` is the attribute. `#PCDATA` means that the attribute has no child elements.

(iii) The DTD tag format:

```
<!ATTLIST Student
      slastname CDATA #REQUIRED
      sfirstname CDATA #IMPLIED
      ssn ID #REQUIRED
>
```

is used to declare the data types and attribute types. `ssn`, `sfirstname` and `slastname` are attributes of the `Student` entity. `CDATA` and `ID` are datatypes. `#REQUIRED` and `#IMPLIED` are the attribute types. `CDATA` means that the attribute or field will contain character data. `#REQUIRED` means that the attribute is required, and `#IMPLIED` means that the attribute is optional. `ID` declares a field as a unique identifier (which we would call a primary key in the ER model or relational database).

2.1.3 Mapping relationships

To map relationships, we need to know the participation ratio between entities, and the cardinalities. The participation ratios can be determined by the operators (`*`, `+` or `?`) after the sub-elements in the DTD files.

Full participation is mapped by a tag like: `<!ELEMENT Student (Course+)>`

The + indicates that the sub-element may occur one or more times in relation to the element. It also indicates that the element must occur at least once. So, the ER translation of the + sign after the Course entity would be that there is a full participation relationship from the Course entity to the Student entity. This means that a course *must* have at least one student.

DTD tags have two ways of showing partial participation. ? or * after the sub-element name show a partial participation.

? indicates that the element *may* occur zero or exactly one time. Both the “may” and the “zero” help us determine that this is a partial participation relationship. Suppose we had the following DTD tag: `<!ELEMENT Course (Book?)>`

This implies that there is a partial participation relationship between the Book entity and Course entity. This means that a book *may* belong to a Course, and if it (the book) does belong to a Course, it can only belong to one Course.

* indicates that the element *may* occur zero or more times. Once again, both the “may” and the “zero” help us determine that this is a partial participation relationship. For example, given the DTD tag: `<!ELEMENT Course (Instructor*)>`

The ER Conversion would be: There is a partial participation relationship between the Instructor entity and the Course entity. This means that an instructor *may* teach a course, but can also teach more than one course.

Recursive entities would be mapped by a DTD tag like:
`<!ELEMENT elementA (elementA*)>`

2.2 Reading the XML file

The readXML module of our program is used parse the XML file and determine the cardinality ratios - whether the relationships are 1:1, 1:M, or M:N. Data will be extracted from this portion of the program since the values to be stored within the database are stored in this file. No changes from the basic XML format are needed for parsing. Given the following XML file:

```
<StudentInfo>
  <Student slastname = "Doe" sfirstname="John" ssn="123-45-6789">
    <Course cname="Intro to XML" cno="101">
      <Instructor>
        <lastName>Doe</lastName>
        <firstName>Jane</firstName>
      </Instructor>
    </Course>
    <Course cname="Intro to Programming" cno = "210">
      <Instructor>
        <lastName>Doe</lastName>
        <firstName>Jane</firstName>
      </Instructor>
    </Course>
  </Student>
</StudentInfo>
```

Figure 1: XML file

The Conversion would be: One student can take many courses, so there is a 1:M relationship between Student and Course. One course has one instructor, so there is a 1:1 relationship between Course and Instructor, and an Instructor can teach more than one Course, so there is a 1:M relationship between Instructor and Course. The XML files do not give us any information on the participation ratios. So, to get a complete picture for an ER diagram, we will have to read both the DTD and XML files.

3 Conclusion

In this paper we have shown (conceptually) how xml files can be read in and mapped to the ER model. Our Java based software is based on these mapping rules. The software first asks for the name a location of a DTD file, reads it in, and creates the basic schema of the ER diagram (based on the rules presented above), then it asks for the name and location of the XML file[s]. From the XML files it determines the cardinality ratios and, combined with the information from the DTD files, it is able to produce a Chen-like ER diagram (showing the entities, attributes and keys and relationships) of the DTD file and XML file[s]. The user then has the option of selecting to create relational mappings of the ER diagram. The rules for converting the ER diagram to the relational mappings have been adopted from [1]. The user then also has the option of selecting to create an Access database from the relational mappings. Once this option is selected, the Access database is created using CREATE TABLE scripts and using the table and attribute names from the relational mappings. The data (read in from the XML file) is also loaded into the Access database using INSERT statements.

The tool will be very useful to people who want to read in XML data and convert them to relational databases – a very necessary step in order to be able to available of the advances in relational database technology. In our future work we plan see if we can capture any other relationships within the data and see if we can extend this to the EER modeling.

References

- [1] Bagui, S. and Earp, R., *Database Design Using Entity-Relationship Diagrams*, Auerbach Publications, A CRC Press Company, New York, 2003.
- [2] Chen, P., “The Entity Relationship Model – Towards a Unified View of Data”, *ACM Trans. Database Systems*, Vol. 9, No. 36, 1976.
- [3] Elmasri R. and Navathe, S. B. *Fundamentals of Database Systems*, Fifth Edition, Pearson Education, Boston, MA, 2007.
- [4] Hughes, C., *The Web Wizard’s Guide To XML*, Pearson Education, 2003.
- [5] SenGupta A. and Mohan, S., “Formal and conceptual models for XML Structures – the past, present and future”, *Technical Report no. 137-1*, Information Series Department Working Paper Series. Indiana University, April 2003.