Data Modeling using XML

Murali Mani, WPI
Antonio Badia, University of Louisville

Oct 13, 2003
Outline

- **Part I**: How to come up with good XML designs for real world database applications?

- **Part II**: Translation between Relational and XML models.
Part I:

How to come up with good XML designs for real world database applications?
What is XML?

<book>
  <author>J. E. Hopcroft</author>
  <author>J. D. Ullman</author>
  <publisher name="Addison-Wesley"/>
</book>
XML for information exchange
XML Publishing

Text applications

<reviewer>X</reviewer>
gave <rating>two thumbs up</rating> to <movie>Fugitive, The</movie>

Database applications

<person>
  <Name>A</Name>
  <Age>25</Age>
  <Salary>20000</Salary>
</person>

<person>
  <Name>B</Name>
  <Age>62</Age>
  <Salary>140000</Salary>
</person>
Publishing Relational Databases

- Users/Applications see a uniform XML view
- Exchange data with other applications
- Querying XML is easier?

Problem:
What is a good XML schema for a relational schema?
XML for Data Modeling

Location → location (@val, @time, GPS)
GPS → gps (@satellite)

Location → location (@val, @time, Bstation*)
Bstation → bstation (@id, @sigStrength)

Location → location (@val, @time, (GPS | Bstation*))
XML as a logical data model

Location → location (@val, @time, (GPS | Bstation*))

- Use data modeling features provided by XML
  - Union types
  - Recursive types
  - Ordered relationships
- Easier to Query?

Problems:
- What is a good XML schema for an application?
- How do we store the data in relational databases?
XML for data integration

XML Wrapper

Source1

Source2

SourceN

Browse
Query, Update
Database Design Stages

Application Requirements

Conceptual Design

Conceptual Schema

Logical Design

Logical Schema

Physical Design

Physical Schema
### Logical Data Model and Redundancy

#### Bad Design

**Student_Professor**

<table>
<thead>
<tr>
<th>sname</th>
<th>BS</th>
<th>advisor</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>CS</td>
<td>MXM</td>
<td>40</td>
</tr>
<tr>
<td>YC</td>
<td>EE</td>
<td>MXM</td>
<td>40</td>
</tr>
</tbody>
</table>

**Professor**

<table>
<thead>
<tr>
<th>pname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXM</td>
<td>40</td>
</tr>
</tbody>
</table>

**Student**

<table>
<thead>
<tr>
<th>sname</th>
<th>BS</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>CS</td>
<td>MXM</td>
</tr>
<tr>
<td>YC</td>
<td>EE</td>
<td>MXM</td>
</tr>
</tbody>
</table>

#### Good Design

**Person**

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV</td>
<td>A1</td>
<td>Los An</td>
<td>CA</td>
<td>90034</td>
</tr>
<tr>
<td>AN</td>
<td>A2</td>
<td>Los An</td>
<td>CA</td>
<td>90034</td>
</tr>
</tbody>
</table>
What is a Data Model?

- Structural Specification
- Constraint Specification
- Operations
Entity Relationship (ER) Model

- Structures: Entity Types, Relationship Types
- Constraints: Cardinality constraints
Relational Model

- Structures: Relations
- Constraints: Key, Foreign Key

<table>
<thead>
<tr>
<th>Professor</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
<td></td>
<td>age</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
<td></td>
<td>advisor</td>
</tr>
</tbody>
</table>

Key Constraints:
- Key (Professor) = <pname>
- Key (Student) = <sname>

Foreign Key Constraints:
- Student (advisor) references Professor (pname)
Specifying Structures for XML

\[ G = (N, T, P, S) \]
\[ N = \{\text{Book, Author, Publisher, } \#\text{PCDATA}\} \]
\[ T = \{\text{book, author, publisher, pCDATA}\} \]
\[ S = \{\text{Book}\} \]
Book \rightarrow \text{book (Author +, Publisher)}
Author \rightarrow \text{author (}\#\text{PCDATA})
Publisher \rightarrow \text{publisher (@name::String)}
\#\text{PCDATA} \rightarrow \text{pCDATA (}\varepsilon)\]

Regular Tree Grammar

Every production rule is of the form \( A \rightarrow a X \)
\( A \in N, a \in T, X \) is a regular expression over \( N \)

Oct 13, 2003  Murali Mani, Antonio Badia
XML Schema Language Proposals

- W3C DTD: local tree grammar
- W3C XML Schema: single type tree grammar
- ISO/OASIS RELAX NG: full-fledged regular tree grammar
Properties of different Regular Tree Grammar classes

- Expressiveness
  - Regular tree grammar *strictly more expressive* than single type tree grammar
  - Single type tree grammar *strictly more expressive* than local tree grammar

- Closure properties
  - Regular tree grammar closed under union, intersection and difference
  - Single type tree grammar/local tree grammar closed only under intersection

- Type assignment
  - Type assignment can be ambiguous for regular tree grammar.
  - Type assignment is unambiguous for local tree grammar/single type tree grammar.
Ambiguous Type Assignment

\[ G = (N, T, P, S) \]
\[ N = \{ \text{Book, Author1, Author2, Publisher, } \#\text{PCDATA} \} \]
\[ T = \{ \text{book, author, publisher, pcdatan} \} \]
\[ S = \{ \text{Book} \} \]

\text{Book} \rightarrow \text{book (Author1*, Author2*, Publisher)}

\text{Author1} \rightarrow \text{author (#PCDATA)}

\text{Author2} \rightarrow \text{author (#PCDATA)}

\text{Publisher} \rightarrow \text{publisher (@name::String)}

\#\text{PCDATA} \rightarrow \text{pcdata (ε)}

- Book
- Author
- Publisher
- J. E. Hopcroft
- J. D. Ullman
- name= "Addison-Wesley"
Constraint Specification for XML – why?

If we represent all relationships only by hierarchies, then the logical model will have redundancy.

What constraint specification?

- Key, Foreign Key
- ID/IDREF
Specifying Constraints for XML: Example

library

- person
  - name = "RRM"
  - book
    - ISBN = "I1"
    - title = "T1"
    - BID = "B1"
    - review
      - article = "P1"
      - rating = "9"
  - book
    - ISBN = "I2"
    - title = "T2"
    - BID = "B2"
    - review
      - article = "B2"
      - rating = "9"
  - paper
    - title = "T1"
    - journal PID = "J1"
    - PID = "P1"
    - review
      - article = "B1"
      - rating = "9"
  - review
    - article = "B2"
    - rating = "10"

name = "CZ"
Specifying Constraints for XML

- Keys are specified using \((\text{rel, sel, field})\)
  - \text{rel} is relative axis
  - \text{sel} is selector axis
  - \text{field} is a set of path expressions
- For any element that “belongs to” \text{rel}, “\text{sel}” will give a set of elements. For this set of elements, \text{field} is the key.
- \text{rel} and \text{sel} can be types or path expressions
- Foreign keys are specified as \((\text{rel}_1, \text{sel}_1, \text{field}_1)\) references \((\text{rel}_2, \text{sel}_2, \text{field}_2)\)
Constraint Specification
Proposals

- W3C XML Schema
  - Relative axis = type
  - Selector axis = path expression
- Keys for XML – WWW10
  - Relative axis = path expression
  - Selector axis = path expression
- UCM – WWW10
  - No relative axis
  - Selector axis = type
Our proposal

- Relative axis = type
- Selector axis = type
- IDREF and IDREFS identify target types
(Library, Person, <@name>)
(Library, Book, <@ISBN>)
(Library, Paper, <@title>)
(Person, Review, <@article>)

@article::IDREF references
(Book | Paper)
Why use XML as logical data model?
Path Expressions vs Joins

<table>
<thead>
<tr>
<th>Professor</th>
</tr>
</thead>
<tbody>
<tr>
<td>pname</td>
</tr>
<tr>
<td>MXM</td>
</tr>
<tr>
<td>CZ</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>sname</td>
</tr>
<tr>
<td>SD</td>
</tr>
<tr>
<td>YC</td>
</tr>
<tr>
<td>FC</td>
</tr>
</tbody>
</table>

Student (advisor) references Professor (pname)

Query: Give names of students of professors of age 40

\[ \pi_{\text{name}}((\sigma_{\text{age}=40}(\text{Professor})) \bowtie \text{Student}) \]

professor [@age=40]/student/@sname
Union Types - attributes

Person

<table>
<thead>
<tr>
<th>name</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV</td>
<td>Los A</td>
<td>CA</td>
<td>null</td>
</tr>
<tr>
<td>AN</td>
<td>null</td>
<td>null</td>
<td>90034</td>
</tr>
</tbody>
</table>

Person → person (@name, ((@city, @state) | @zip))

Diagram:
- University
  - Person [name = "AV", city = "Los A", state = "CA"]
  - Person [name = "AN", zip = "90034"]
Union Types - Relationships

Person → person (@name, @zip, (Book* | Paper*))

<table>
<thead>
<tr>
<th>name</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>01609</td>
</tr>
<tr>
<td>CZ</td>
<td>90095</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ISBN</th>
<th>title</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>T1</td>
<td>RRM</td>
</tr>
<tr>
<td>I2</td>
<td>T2</td>
<td>RRM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>title</th>
<th>journal</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>J1</td>
<td>CZ</td>
</tr>
</tbody>
</table>

Person

Book

Paper
Union Types - Relationships

<table>
<thead>
<tr>
<th>Conference</th>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>venue</td>
</tr>
<tr>
<td>ER</td>
<td>Chicago</td>
</tr>
</tbody>
</table>

| Journal |
| name | publisher |
| TOIT | ACM |

<table>
<thead>
<tr>
<th>title</th>
<th>inst</th>
<th>conf</th>
<th>journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>WPI</td>
<td>ER</td>
<td>null</td>
</tr>
<tr>
<td>T2</td>
<td>UCLA</td>
<td>null</td>
<td>TOIT</td>
</tr>
</tbody>
</table>

Conference → conference (@name, @venue, Paper*)
Journal → journal (@name, @publisher, Paper*)
Recursive Types

### Assembly

<table>
<thead>
<tr>
<th>name</th>
<th>superPart</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>seat</td>
<td>frame</td>
<td>1</td>
</tr>
<tr>
<td>tire</td>
<td>wheel</td>
<td>1</td>
</tr>
<tr>
<td>frame</td>
<td>bike</td>
<td>1</td>
</tr>
<tr>
<td>wheel</td>
<td>bike</td>
<td>2</td>
</tr>
<tr>
<td>bike</td>
<td>null</td>
<td>0</td>
</tr>
</tbody>
</table>

WITH RECURSIVE SubPart (name) AS
  (SELECT name FROM Assembly
   WHERE superPart=bike)
UNION
  (SELECT R2.name FROM SubPart R1, Assembly R2
   WHERE R2.superPart = R1.name)
SELECT * FROM SubPart

Query: What are subparts of bike?

part[@name=bike]//part/@name
IDREF vs Foreign Keys

**Professor**

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>MXM</td>
<td>40</td>
</tr>
<tr>
<td>CZ</td>
<td>55</td>
</tr>
</tbody>
</table>

**Student**

<table>
<thead>
<tr>
<th>name</th>
<th>year</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>1998</td>
<td>MXM</td>
</tr>
<tr>
<td>YC</td>
<td>2000</td>
<td>MXM</td>
</tr>
<tr>
<td>FC</td>
<td>1999</td>
<td>CZ</td>
</tr>
</tbody>
</table>

Student (advisor) references Professor (name)

@PRef::IDREF references (Professor)

Query: Give names of students of professors of age 40

student[@PRef⇒professor/@age=40]/@name
IDREF as union of foreign keys

**Book**
- ISBN: I1, T1
- ISBN: I2, T2

**Person**
- name: RRM, zip: 01609
- name: CZ, zip: 90095

**Paper**
- title: T1
- journal: J1

**Review**
- name: RRM, book: null, paper: T1, rating: 9
- name: CZ, book: I1, paper: null, rating: 9
- name: CZ, book: I2, paper: null, rating: 10
IDREF as union of foreign keys

@article::IDREF references (Book | Paper)
Conceptual Model:

**ERex** (ER extended for XML)
Entity Relationship (ER) model

- Entity Types, Relationship Types and their attributes
Object Role Modeling (ORM)

- closer to natural language sentences
- attributes/relationships are expressed uniformly using roles
Unified Modeling Language (UML)

- Modeling software systems
- Class Diagrams, Association Classes

Diagram:

```
Professor
  pname
  age

1...1

Advisor
  years

0...*

Student
  sname
```
From ER model

Binary 1:n relationships

Professor
  pname
  age
  (0,*)

Advisor
  year
  (1,1)

Student
  sname

ReviewPerson Book (0,* ) (0,* )
  name
  zip
  ISBN
  title
  year
  rating
  (0,*)

Binary m:n relationships
From ER Model

N-ary relationships

- **Company** connects to **Supplies** with (1,*) relationship to **City** and (0,*) relationship to **Product**.
- **Product** connects to **Supplies** with (0,*) relationship to **City**.

Recursive relationships

- **Part** connects to **Contains** with (0,1) relationship to **Part** and (0,*) relationship to **Part**.

Oct 13, 2003

Murali Mani, Antonio Badia
Ordered Relationships

![Diagram of the ordered relationships between Person, Author, and Book entities with relationships and example data]

**Person**
- **name**
  - Ullman
  - RRM
- **zip**
  - 95123
  - 90095

**Book**
- **ISBN**
  - B1
  - B2
  - B3
- **title**
  - DB
  - Aut
  - MMSL
- **order**
  - 2
  - 1
  - 1
- **author**
  - Ullman
  - Ullman
  - RRM
Categories and Set Constraints

PersonCity \cap PersonZip = \emptyset

PersonCity \cup PersonZip = \text{Person}
Set constraints on Roles

\[ \text{personBook} \cap \text{personPaper} = \emptyset \]
Set Constraints on Roles

confPaper ∩ journalPaper = ∅
confPaper ∪ journalPaper = Paper
Translating ERex schemas to XML schemas
System Architecture

ERex Schemas → Translator → XML Schemas → Final XML Schema

Schema Designer
1:n relationships

Professor

(0,*)

Advisor

(1,1)

Student

pname

age

year

sname
Representing 1:n Relationships - subelement

Professor

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>62</td>
</tr>
<tr>
<td>CZ</td>
<td>55</td>
</tr>
</tbody>
</table>

Student

<table>
<thead>
<tr>
<th>name</th>
<th>year</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>1998</td>
<td>RRM</td>
</tr>
<tr>
<td>YC</td>
<td>2000</td>
<td>RRM</td>
</tr>
<tr>
<td>FC</td>
<td>1999</td>
<td>CZ</td>
</tr>
</tbody>
</table>

Student (advisor) references Professor (name)

University → university (Professor*)
Professor → professor (@name, @age, Student*)
Student → student (@name, @year)

(University, Professor, <@name>)
(University, Student, <@name>)
Representing 1:n Relationships - IDREF

**Professor**

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>62</td>
</tr>
<tr>
<td>CZ</td>
<td>55</td>
</tr>
</tbody>
</table>

**Student**

<table>
<thead>
<tr>
<th>name</th>
<th>year</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>1998</td>
<td>RRM</td>
</tr>
<tr>
<td>YC</td>
<td>2000</td>
<td>RRM</td>
</tr>
<tr>
<td>FC</td>
<td>1999</td>
<td>CZ</td>
</tr>
</tbody>
</table>

Student (advisor) references Professor (name)

University → university (Professor*, Student*)
Professor → professor (@name, @age, @PID)
Student → student (@name, @year, @PRef)

(University, Professor, <@name>)
(University, Student, <@name>)
@PRef::IDREF references (Professor)
Representing 1:n Relationships – foreign keys

### Professor

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>62</td>
</tr>
<tr>
<td>CZ</td>
<td>55</td>
</tr>
</tbody>
</table>

### Student

<table>
<thead>
<tr>
<th>name</th>
<th>year</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>1998</td>
<td>RRM</td>
</tr>
<tr>
<td>YC</td>
<td>2000</td>
<td>RRM</td>
</tr>
<tr>
<td>FC</td>
<td>1999</td>
<td>CZ</td>
</tr>
</tbody>
</table>

University → university (Professor*, Student*)
Professor → professor (@name, @age)
Student → student (@name, @year, @advisor)

Student (advisor) references Professor (name)

(University, Professor, <@name>)
(University, Student, <@name>)
(University, Student, <@advisor>) references
(University, Professor, <@name>)
m:n relationships

Person

(0, *)

Review

(0, *)

Book

name

rating

ISBN

title
Representing m:n relationships

<table>
<thead>
<tr>
<th>Person</th>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>ISBN</td>
</tr>
<tr>
<td>RRM</td>
<td>I1</td>
</tr>
<tr>
<td>CZ</td>
<td>I2</td>
</tr>
<tr>
<td>CZ</td>
<td>I3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Review</th>
<th>pname</th>
<th>ISBN</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>I3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>RRM</td>
<td>I2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>CZ</td>
<td>I1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>CZ</td>
<td>I1</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Library \(\rightarrow\) library (Person*, Book*)
Person \(\rightarrow\) person (@name, Review*)
Book \(\rightarrow\) book (@ISBN, @title, @BID)
Review \(\rightarrow\) review (@article, @rating)

Review (pname) references
Person (name)
Review (ISBN) references
Book (ISBN)

(Library, Person, <@name>) (Library, Book, <@ISBN>)
(Person, Review, <@article>)
@article::IDREF references (Book)
Query: What is the rating given by RRM for book T1

\[ \pi_{\text{rating}} \left( (\sigma_{\text{title}=T1} (\text{Book})) \times (\sigma_{\text{pname}=RRM} (\text{Review})) \right) \]

person[@name=RRM]/review[@article=B1]/@rating
N-ary relationships

Root → root (Company*, Product*, City*)
Company → company (@name, Supply+)
Supply → supply (@ProdRef, @CityRef, @qty)
Product → product (@name, @ProdID)
City → city (@name, @CityID)

(Root, Company, <@name>)
(Root, Product, <@name>)
(Root, City, <@name>)
@ProdRef::IDREF references (Product)
@CityRef::IDREF references (City)
Recursive Relationships

Assembly

<table>
<thead>
<tr>
<th>name</th>
<th>superPart</th>
<th>qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>seat</td>
<td>frame</td>
<td>1</td>
</tr>
<tr>
<td>tire</td>
<td>wheel</td>
<td>1</td>
</tr>
<tr>
<td>frame</td>
<td>bike</td>
<td>1</td>
</tr>
<tr>
<td>wheel</td>
<td>bike</td>
<td>2</td>
</tr>
<tr>
<td>bike</td>
<td>null</td>
<td>0</td>
</tr>
</tbody>
</table>

Assembly → assembly (Part*)
Part → part (@name, @qty, Part*)

(assembly, part, <@name>)
Ordered Relationships

Person

<table>
<thead>
<tr>
<th>name</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ullman</td>
<td>95123</td>
</tr>
<tr>
<td>RRM</td>
<td>90095</td>
</tr>
</tbody>
</table>

Book

<table>
<thead>
<tr>
<th>ISBN</th>
<th>title</th>
<th>order</th>
<th>author</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>DB</td>
<td>2</td>
<td>Ullman</td>
</tr>
<tr>
<td>B2</td>
<td>Aut</td>
<td>1</td>
<td>Ullman</td>
</tr>
<tr>
<td>B3</td>
<td>MMS</td>
<td>1</td>
<td>RRM</td>
</tr>
</tbody>
</table>

Library → library (Person*)
Person → person (@name, @zip, Book*)
Book → book (@ISBN, @title)

(Library, Person, <@name>)
(Library, Book, <@ISBN>)
Categories and set constraints

PersonCity \cap PersonZip = \emptyset
PersonCity \cup PersonZip = \text{Person}

Person \rightarrow \text{person (@name, ((@city, @state) | @zip))}
Categories and Set Constraints

Root → root (Person*, Book*, Paper*)
Person → person (@name, Review*)
Review → review (@article, @rating)

(Root, Person, <@name>)
(Person, Review, <@article>)
@article::IDREF references (Book | Paper)
Set constraints on Roles

\[ \text{personBook} \cap \text{personPaper} = \emptyset \]

Person \rightarrow \text{person} (\text{\@name}, (\text{Book}\ast | \text{Paper}\ast))
Set Constraints on Roles

confPaper \cap \text{journalPaper} = \emptyset
confPaper \cup \text{journalPaper} = \text{Paper}

Conference \rightarrow \text{conference} (@\text{name}, @\text{venue}, \text{Paper}*)
Journal \rightarrow \text{journal} (@\text{name}, @\text{publisher}, \text{Paper}*)
Converting ERex → XML

- Goals
  - Maximize relationships represented using subelement.
  - Others try to represent using IDREF
Algorithm: ERex → XML

- A non-terminal symbol for each
  - entity type with key
  - m:n relationship
  - n-ary relationship
  - Root non-terminal symbol
- Represent attributes
- Represent relationships and identify top nodes
  - 1:1 and 1:n relationships
  - m:n relationships
  - n-ary relationships
- Identify key and IDREF constraints.
personBook ∩ personPaper = ∅
personBook ∪ personPaper = Person

PersonCity ∩ PersonZip = ∅
PersonCity ∪ PersonZip = Person
N = {Root, Person, Book, Paper, Review}

Book → book (@ISBN, @btitle, @year)
Paper → paper (@ptitle, @year, @journal)
Person → person (@name, ((@city, @state) | @zip))

Root → root (Person*)

(Root, Person, <@name>)
(Root, Book, <@ISBN>)
(Root, Paper, <@ptitle>)
(Person, Review, <@article>)
@article::IDREF references (Book | Paper)

Oct 13, 2003
Murali Mani, Antonio Badia
Conclusions

- Obtained good XML Schema from ERex schemas
Part II: Translation between Relational and XML models.
Why publish relational databases as XML?

- Provide an XML view for our legacy data
  - Users/Applications can query our data over the web using standards.
  - Easier to query XML than legacy (relational) data?
- Convert our legacy data to XML
  - We can exchange data with applications.
  - Store data in XML databases?
  - Easier to query?
Application Scenarios

● Schema Matching Problem
  - Given a relational schema and an XML schema by a standards body, how do we map this relational schema to XML?
  - Tools such as XML Extender from IBM, Clio (University of Toronto and IBM), MS SQL Server

● Schema Mapping Problem
  - Given a relational schema, how do we come up with a good XML schema?
Schema Matching: MS SQL Server Architecture

XML Views - BizTalk

Uses same schema format as BizTalk framework and server

BizTalk Schema → Copy → BizTalk Schema

+ Annotations

= Mapping Schema

+ XPath

/Customers

→ Query → SQL Server → Results

XML in BizTalk Grammar

COPYRIGHT NOTICE. Copyright © 2003 Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 U.S.A. All rights reserved.
Schema Mapping Problem

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
<td>20000</td>
</tr>
<tr>
<td>B</td>
<td>62</td>
<td>140000</td>
</tr>
</tbody>
</table>

- Users/Applications see a uniform XML view
- Exchange data with other applications
- Store XML in native XML databases
- Querying XML is easier?

Problem:
What is a good XML schema for a relational schema?
Goals

- XML Schema should maintain constraints.
- Resulting XML should not introduce redundancies.
- Most relationships can be navigated using path expressions, rather than joins.
- Minimal user interaction: Our translator should suggest good XML schemas to the database designer.
System Architecture

RDB → NeT → CoT → XML Schemas → Final XML Schema

Schema Designer
Related Work

- **XML-DBMS**
  - Template driven mapping language

- **SilkRoute**
  - Declarative Query Language (RXL) for viewing relational data as XML

- **Xperanto**
  - User specifies query in XML Query Language
Algorithms

- Naïve
  - FT (Flat Translation)
- Consider relational data
  - NeT (Nesting-based Translation)
- Consider relational schema
  - CoT (Constraint-based Translation)
FT: Flat Translation

- 1:1 mapping from relational to XML
- Idea
  - A type (non-terminal) corresponding to every relation
  - Attributes of a relation form attributes of the type
  - Keys and foreign keys are preserved
FT: Flat Translation - Example

**Professor**

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>62</td>
</tr>
<tr>
<td>CZ</td>
<td>55</td>
</tr>
</tbody>
</table>

**Student**

<table>
<thead>
<tr>
<th>name</th>
<th>year</th>
<th>advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>1998</td>
<td>RRM</td>
</tr>
<tr>
<td>YC</td>
<td>2000</td>
<td>RRM</td>
</tr>
<tr>
<td>FC</td>
<td>1999</td>
<td>CZ</td>
</tr>
</tbody>
</table>

Student (advisor) references Professor (name)

University → university (Professor*, Student*)
Professor → professor (@name, @age)
Student → student (@name, @year, @advisor)

(University, Professor, <@name>)
(University, Student, <@name>)
(University, Student, <@advisor>) references
(University, Professor, <@name>)

Oct 13, 2003

Murali Mani, Antonio Badia
Net: Nesting-based Translation

- Idea:
  Make use of non-flat features provided by XML: represent repeating groups using *, +
### NeT: Example

#### Course (cname, prof, text)

<table>
<thead>
<tr>
<th>cname</th>
<th>prof</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>Gafni</td>
<td>Udi Manber</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Gafni</td>
<td>CLR</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Majid</td>
<td>Udi Manber</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Majid</td>
<td>CLR</td>
</tr>
</tbody>
</table>
# NeT: Example

**Course** \( (\text{cname}, \text{prof}^\pm, \text{text}) \)

<table>
<thead>
<tr>
<th>cname</th>
<th>prof</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>{Gafni, Majid}</td>
<td>Udi Manber</td>
</tr>
<tr>
<td>Algorithms</td>
<td>{Gafni, Majid}</td>
<td>CLR</td>
</tr>
</tbody>
</table>

**Course** \( (\text{cname}, \text{prof}^\pm, \text{text}^\pm) \)

<table>
<thead>
<tr>
<th>Cname</th>
<th>prof</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>{Gafni, Majid}</td>
<td>{Udi Manber, CLR}</td>
</tr>
</tbody>
</table>
NeT: Example

University → university (Course*)
Course → course (@cname, Prof+, Text+)
Prof → prof (#PCDATA)
Text → text (#PCDATA)
#PCDATA → pcdata (ε)

ValueRatio=12/5

Oct 13, 2003  Murali Mani, Antonio Badia
NeT: Example

Person (*name*, *city*, *state*, *zip*)

<table>
<thead>
<tr>
<th>name</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90034</td>
</tr>
<tr>
<td>AN</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90034</td>
</tr>
</tbody>
</table>

Person (*name*, *city*, *state*, *zip*)

<table>
<thead>
<tr>
<th>{MM, AN}</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Los Angeles</td>
<td>CA</td>
<td>90034</td>
</tr>
</tbody>
</table>
NeT: Example

Directory → directory (Person*)
Person → person (@name, @city, @state, @zip)

(directory, person, <@name>)

ValueRatio=8/5
NeT: Summary

- Consider Table $t$ with column set $C$. Nesting on column $X$ is defined as:
  Any two tuples with the same values for $(C - X)$ will be combined to one tuple

- **Observation:** We need to nest only on key columns

- **Advantages of NeT**
  - NeT removes redundancy if relation is not in 4NF
  - NeT provides more intuitive XML schemas with less redundancy
# NeT: Experimentation

<table>
<thead>
<tr>
<th>Test Set</th>
<th>#attr</th>
<th>#tuples</th>
<th>ValueRatio</th>
<th>#nested attributes</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balloons1</td>
<td>5</td>
<td>16</td>
<td>3.64</td>
<td>3</td>
<td>1.08</td>
</tr>
<tr>
<td>Hayes</td>
<td>6</td>
<td>132</td>
<td>1.52</td>
<td>1</td>
<td>1.01</td>
</tr>
<tr>
<td>Bupa</td>
<td>7</td>
<td>345</td>
<td>1</td>
<td>0</td>
<td>4.40</td>
</tr>
<tr>
<td>Balance</td>
<td>5</td>
<td>625</td>
<td>2.79</td>
<td>4</td>
<td>21.48</td>
</tr>
<tr>
<td>TA_Eval</td>
<td>6</td>
<td>110</td>
<td>1.24</td>
<td>5</td>
<td>24.83</td>
</tr>
<tr>
<td>Car</td>
<td>7</td>
<td>1728</td>
<td>15.53</td>
<td>6</td>
<td>469.47</td>
</tr>
<tr>
<td>Flare</td>
<td>13</td>
<td>365</td>
<td>1.67</td>
<td>4</td>
<td>6693.41</td>
</tr>
</tbody>
</table>
CoT: Constraint-based Translation

- Translating relational schema
- Idea:
  Use foreign key constraints and our knowledge of how to represent relationships to come up with “better” XML models.
CoT: Step 1

Professor

<table>
<thead>
<tr>
<th>pname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>62</td>
</tr>
</tbody>
</table>

Student

<table>
<thead>
<tr>
<th>sname</th>
<th>advisor</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>RRM</td>
<td>DBs</td>
</tr>
<tr>
<td>YC</td>
<td>RRM</td>
<td>DBs</td>
</tr>
<tr>
<td>YC</td>
<td>RRM</td>
<td>QSs</td>
</tr>
</tbody>
</table>

Student (advisor) references Professor (pname)

ValueRatio=11/8

University → university (Professor*)
Professor → professor(@pname, @age, Student*)
Student → student (@sname, @cname)

(Student, University, Professor, <@pname>)
(Professor, Student, <@sname, @cname>)
CoT: Step 2

Professor

<table>
<thead>
<tr>
<th>pname</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRM</td>
<td>62</td>
</tr>
</tbody>
</table>

Course

<table>
<thead>
<tr>
<th>cname</th>
<th>since</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBs</td>
<td>1979</td>
</tr>
<tr>
<td>QSs</td>
<td>1962</td>
</tr>
</tbody>
</table>

Student

<table>
<thead>
<tr>
<th>sname</th>
<th>advisor</th>
<th>cname</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM</td>
<td>RRM</td>
<td>DBs</td>
</tr>
<tr>
<td>YC</td>
<td>RRM</td>
<td>DBs</td>
</tr>
<tr>
<td>YC</td>
<td>RRM</td>
<td>QSs</td>
</tr>
</tbody>
</table>

Student (advisor) references Professor (pname)
Student (cname) references Course (cname)

Student (sname, @CRef) references Professor (pname), University
(University, Professor, <@pname>)
(University, Course, <@cname>)
(Professor, Student, <@sname, @Cref>)
CRef::IDREF references (Course)
CoT: Example

Student (SID, name, advisor)
Emp (EID, name, projName)
Prof (EID, name, teach)
Course (CID, title, room)
Dept (dno, mgr)
Proj (pname, pmgr)

Student (advisor) references Prof (EID)
Emp (projName) references Proj (pname)
Prof (teach) references Course (CID)
Prof (EID, name) references Emp (EID, name)
Dept (mgr) references Emp (EID)
Proj (pmgr) references Emp (EID)
Top nodes

(University, Course, @CID)
(University, Emp, @EID)
(University, Prof, @EmpRef)
(University, Student, @SID)
(University, Proj, @pname)
(University, Dept, @dno)

@EmpRef::IDREF references (Emp)
@ProjRef::IDREF references (Proj)

University → university (Course*, Emp*)
Course → course (@CID, @title, @room, Prof*)
Prof → prof (@EmpRef, Student*)
Student → student (@SID, @name)
Emp → emp (@EID, @name, @ProjRef, @EmpID, Dept*, Proj*)
Proj → proj (@pname, @ProjID)
Dept → dept (@dno)
CoT Experimentation

- Ran on TPC-H data
- Value ratio > 100/88 (size decreased by more than 12%)
We obtained “good” XML schemas from relational schemas
- Constraints are maintained
- Redundancies are decreased
- Most relationships can be navigated using path expressions.
- Minimum user interaction
Storing XML data in relational databases
Options for storing XML data

- Store in relational databases
  - Relational databases are robust and efficient (IBM XML Extender, Oracle, MS SQL Server)
- Store in native XML databases
  - More efficient for XML than relational databases (Natix, eXist, Tamino)
- Store in a combination of both
  - Structured portion of XML data in relational database, and unstructured portion in native XML store.
Related Work

- STORED
  - No Schema
  - Use data mining techniques to find structured and frequent patterns
  - These are stored in relational DB, others in semi-structured overflow store
  - Drawback: Requires integration of relational DB and semi-structured store

- Storing paths
  - One relation for storing nodes, one for storing edges.
  - Drawback: Type information is lost.
Type-based relational storage

- Jayavel Shanmugasundaram
  - Several key ideas such as schema simplification, inlining, handling recursion
- LegoDB
  - Use the query workload to come up with an “efficient” relational schema.
Main features

- The entire XML document is shredded and stored in a relational database.
- All semantic constraints in XML schema are not captured in relational schema.
- We do not discuss how operations on XML are translated to SQL.
Why not capture all constraints?

\[ A \rightarrow a (@b, ((@c, D)^* \mid (@e, F)^*)) \]

(Root, A, <@b>)

\[ A (@b, @c, @e) \]
\[ D (@\text{aRef}) \]
\[ F (@\text{aRef}) \]

Semantic constraints lost
- if D refers to an A, then the corresponding \(@c\) should be non-null
- if F refers to an A, then the corresponding \(@e\) should be non-null
NF2 representation of regular tree grammars

Eliminate union, “|”

\[
\text{paper} \rightarrow (@\text{ptitle}, @\text{journal} | @\text{conference})
\]

\[
\text{paper} \rightarrow (@\text{ptitle}, @\text{journal})
\]

\[
\text{paper} \rightarrow (@\text{ptitle}, @\text{conference})
\]

\[
\text{paper} \rightarrow (@\text{ptitle}, @\text{journal}, @\text{conference})
\]
Schema Simplification

\[ A \rightarrow a (@d, (B, C, B)^*) \]

\[ A \rightarrow a (@d [1, 1], (B [2, 2], C [1, 1]) [0, *]) \]
\[ A \rightarrow a (@d [1, 1], B [2, 2] [0, *], C [1, 1] [0, *]) \]
\[ A \rightarrow a (@d [1, 1], B [0, *], C [0, *]) \]

\[ A \rightarrow a (@d, B^*, C^*) \]

Semantic information lost
- The number of B’s is two times the number of C’s
- for every C, there is a B that occurs before it, and one that occurs after it.
Inlining

Conf → conf (@ctitle, @date, Venue)
Venue → venue (@city, @country)

Conf → conf (@ctitle, @date, @city, @country)

Why inlining?
• Lesser joins, hence more efficient
Mapping Collection Types

Conf → conf (@ctitle, @date, Paper*)
Paper → paper (@ptitle, @author)

Conf (@ctitle, @date)
Paper (@ptitle, @author, @confRef)

• Separate relation with foreign key for every collection type
IDREF attribute

Person → person (@name, @zip, Review*)
Book → book (@ISBN, @btitle, @BID)
Paper → paper (@ptitle, @journal, @PID)
Review → review (@article, @rating)

@article::IDREF references (Book | Paper)

Person (@name, @zip)
Book (@ISBN, @btitle, @BID)
Paper (@ptitle, @journal, @PID)
Review (@personRef, @bookRef, @paperRef, @rating)
Recursion using ?

\[ A \rightarrow a \ (@d, A?) \]

\[ A \ (@d, ARef) \]
ARef refers to the child of A, and can be null

\[ A \rightarrow a \ (@d, ARef) \]
ARef refers to the parent of A and can be null.
Recursion using *

A → a (@d, A*)

A (@d, ARef)
ARef refers to the parent of A, and can be null
Recursion – General technique

- For every cycle, we must have a separate relation

**Algorithm**

- For every strongly connected component, define a separate relation for one of the types.
- In a strongly connected component, if there is a type which can be children of multiple types, then define a separate relation for that type.
Capturing Order in the Document

- Through order attributes
- Corresponding to each type in XML schema, say A, we have an associated order attribute, say aOrder

Conf → conf (@cTitle, @date, Venue)
Venue → venue (@city, @country)

Conf → conf (@cTitle, @date, @confOrder, @city, @country, @venueOrder)
Conclusions

- XML schema with no recursion can be translated to relational schema with no nulls.
- XML schema with recursion cannot be translated to relational schema with no nulls.
- If recursion, separate relation needed for every cycle.
- All semantic constraints in XML cannot be captured in relational schema.
- XML resulting from CoT can be translated to the original relational schema; all semantic constraints are maintained.
Open Problems

- Standards specification: What structural and constraint specification schemes for XML are needed for database applications?
- XML used for text/document publishing: Keyword Search in XML documents
- Storing data consisting of structured and unstructured portions: integrating relational and XML stores.
Open Problem (contd…)

- Translating operations in XML model to underlying sources (relational)
  - Use annotated schema (MS SQL Server)
  - Use implicit annotations (LegoDB)
- Query minimization: When we do automatic translation, we might perform unnecessary joins?