

XML to Relational Conversion using Theory of Regular Tree Grammars

Murali Mani
EEXTT 2002



Outline

- XML Application scenarios we envision
- Regular Tree Grammar framework for XML schema structural specification
- XML to Relational translation – state of the art and our approach
- NF2 - a normal form representation for regular tree grammars
- Our translation steps
- Assistance from Schema Language for translation.
- NF1 representation for regular tree grammars



XML Application scenario

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>61</Age>  
  <Salary>140000</Salary>  
</person>
```



Database applications and XML

- Our world consists of
 - Entities
 - Relationships
 - binary - 1:1, 1:many, many:many
 - n-ary
 - recursive
 - Attributes for entities
 - Attributes for relationships



Database applications and XML

- Parent-child represent attributes and 1:n relationships

- prof → (PName, student*)

```
<prof>
```

```
  <PName>Muntz</PName>
```

```
  <student SName="MM" since="1998"/>
```

```
  <student SName="YC" since="2000"/>
```

```
</prof>
```



Database applications and XML

- ID-IDREF represent 1:n relationships

- prof → (@PName, @ID_prof)

- student → (@SName, @Ref_prof)

```
<prof PName="Muntz" ID_prof="p1"/>
```

```
<student Sname="MM" Ref_prof="p1"/>
```

```
<student Sname="YC" Ref_prof="p1"/>
```



Database applications and XML

- ID-IDREFS represent m:n relationships – attributes for relationships cannot be specified

- person → (@Name, ID_person)

- book → (@Title, Refs_person)

```
<person Name="Ullman" ID_person="p1"/>
```

```
<person Name="Aho" ID_person="p2"/>
```

```
<book Title="Compilers" Refs_person="p2 p1"/>
```

```
<book Title="Formal Lgs" Refs_person="p1"/>
```



Database applications and XML

- XML can represent recursion very nicely – using ? or *

- person → (@Name, @Year, person?)

```
<person Name="A" Year="2000">
```

```
  <person Name="B" Year="1970">
```

```
    <person Name="C" Year="1940"/>
```

```
  </person>
```

```
</person>
```

Name	Year	Mother
A	2000	B
B	1970	C
C	1940	null



Database applications and XML

- person → (@Name, @Year, person*)

```
<person Name="X" Year="1970">  
  <person Name="Y" Year="2000"/>  
  <person Name="Z" Year="2002"/>  
</person>
```

Name	Year	Mother
X	1970	Null
Y	2000	X
Z	2002	X



Database applications and XML

- Union types

paper → (@Title, @journal | @conference)

```
<paper Title="X" conference="VLDB"/>  
<paper Title="Y" conference="TOIT"/>
```

- Set/List valued attributes

book → (@Title, year*)

```
<book Title="Algorithms">  
  <year>1991</year>  
  <year>2000</year>  
</book>
```



Regular Tree Grammars for XML Schema Structural Specification

- $G = (N, T, S, P)$
 - N is a set of non-terminals
 - T is set of terminals
 - S is set of start symbols
 - P is a set of production rules of the form
 $A \rightarrow x RE$
- Attributes are considered equivalent to elements
- We add constraint specification on top of it to get a good XML schema language for database applications



XML to relational – State of the art

- Why? – Data from database applications is better managed by relational databases
- Given any regular tree grammar try to map it into a set of relations and constraints.
 - Regular tree grammar does not distinguish between parent-child representing 1:n relationships and attributes
 - Set/List valued attributes are considered as relationships and separate relations are created
 - Order is captured using attributes in the data.



XML to relational – State of the art

- LegoDB – Considers a space of possible relational models and selects one based on performance from user specified query statistics
- Our approach
 - Space of possible relational models.
 - Nulls In the relational model.
 - Maintain semantic constraints



NF2 representation for regular tree grammars

- Eliminate union “|”
paper → (@Title, @journal | @conference)

paper → (@Title, @journal)
paper → (@Title, @conference) (or)

paper → (@Title, @journal, @conference)



Inlining

paper → (@Title, journal | conference)

journal → (@JName)

conference → (@CName)

paper → (@Title, @JName, @CName) (or)

paper → (@Title, @JName)

paper → (@Title, @CName)



Mapping collection types

■ book → (@Title, @ISBN, author*)

■ author → (@Name)

book → (@Title, @ISBN)

author → (@Name, @book_Title)



Mapping collection types

- book → (@Title, @Publisher?, author*)

book1 → (@Title)

book2 → (@Title, @Publisher)

author1 → (@Name, @book1_Title)

author2 → (@Name, @book2_Title) (or)

book1 → (@Title)

book2 → (@Title, @Publisher)

author1 → (@Name, @book1_Title, @book2_Title) (or)

book → (@Title, @Publisher)

author → (@Name, @book_Title)



Shared elements

- book → (@BTitle, author*)
- paper → (@PTitle, author*)

book → (@BTitle)

paper → (@PTitle)

author1 → (@Name, @book_BTitle)

author2 → (@Name, @paper_PTitle) (or)

book → (@BTitle)

paper → (@PTitle)

author1 → (@Name, @book_Btitle, @paper_PTitle)



ID-IDREF attributes

- $\text{prof} \rightarrow (@PName, @Age?, @ID_prof)$
- $\text{student} \rightarrow (@SName, @Ref_prof)$

$\text{prof} \rightarrow (@PName, @Age)$
 $\text{student} \rightarrow (@SName, @prof_PName)$ (or)
 $\text{prof1} \rightarrow (@PName)$
 $\text{prof2} \rightarrow (@PName, @Age)$
 $\text{student1} \rightarrow (@SName, @prof1_PName)$
 $\text{student2} \rightarrow (@SName, @prof2_PName)$ (or)
 $\text{prof1} \rightarrow (@PName)$
 $\text{prof2} \rightarrow (@PName, @Age)$
 $\text{student} \rightarrow (@SName, @prof1_PName, @prof2_PName)$



ID-IDREFS attributes

- $\text{prof} \rightarrow (@PName, @Age?, @ID_prof)$
- $\text{student} \rightarrow (@SName, @Refs_prof)$

$\text{prof1} \rightarrow (@PName)$
 $\text{prof2} \rightarrow (@PName, @Age)$
 $\text{student} \rightarrow (@SName)$
 $\text{prof1student} (@prof1_PName, @student_SName)$
 $\text{prof2student} (@prof2_PName, @student_SName)$



Handling recursion

- Recursion will introduce nulls
- For every cycle, we **must** define at least one relation
 - For every strongly connected component, at least one of the elements must be defined as a separate relation
 - In a strongly connected component, if there exists an element which can be children of more than one element in the strongly connected component, we define a separate relation for that element.



Other possible approaches

- Consider the process of designing a relational schema from a conceptual ER schema. We create relations for all entity types and for some relationship types. We can use XML schema language as an intermediary step in the conversion, and use it to represent faithfully the ER schema.
 - Schema language can support distinguishing bt. attributes vs 1:n relationships for parent-child.
 - RELAX NG prof = prof { @PName {string}, student *}
student = student { @SName {string} }
- User can annotate the XML Schema



NF1 representation for regular tree grammars

- RELAX NG has a full syntax and a simple syntax. User writes schemas in the full syntax, and the validator works on the simple syntax.
- Simple syntax has minimal set of operators, and has only one production rule for any non-terminal
- NF1 combines all production rules for any non-terminal that has the same terminal symbol on the right hand side
 - Easier and more efficient for document validation
 - Schema validation requires this operation.



References

- RELAX NG – <http://www.relaxng.org>
- Jay et al, *Relational Databases for Querying XML Documents*, VLDB 99
- P. Bohannon et al, *From XML Schema to Relations: A Cost-Based Approach to XML Storage*, ICDE 2002
- *Semantic Data Modeling using XML Schemas*, ER 2001
- *Taxonomy of XML Schema Languages using Formal Language Theory*, Extreme Markup Languages, 2001

