

# Semistructured Data Store Mapping with XML and Its Reconstruction

Enhong CHEN<sup>1</sup> Gongqing WU<sup>1</sup> Gabriela Lindemann<sup>2</sup> Mirjam Minor<sup>2</sup>

<sup>1</sup>Department of Computer Science  
University of Science and Technology of China  
Hefei Anhui 230027 P.R.CHINA  
cheneh@ustc.edu.cn, wugq@mail.ustc.edu.cn

<sup>2</sup>Humboldt University of Berlin  
Department of Computer Science  
Berlin, Germany  
lindeman@informatik.hu-berlin.de

**Abstract.** XML has been quickly emerging as a dominant standard for data representation and exchange on the World Wide Web for its many good features such as well-formed structure or semantic support. Research on semistructured data over the last several years has focused on data models, query languages, and systems where the database is modeled in some form of a labeled, directed graph. Processing this as a sophisticated query on semistructured data is not very easy because of the complexity of the structure of the graph and the lack of corresponding schemata associated with it. To deal with such problems the paper proposes an approach to process semistructured data with XML. Although there are many similarities between semistructured data and XML there exist some differences. A key difference is that current XML DOM only supports tree structures and does not directly support graph structures. To deal with such differences two approaches in this paper are proposed to treat an XML document as a semantic graph and literal tree which are the foundation to transform semistructured data into XML documents for processing. For this purpose several algorithms are designed to transform semistructured data into XML documents and XML-Schema document based on the schema tree extracted from original semistructured data. To ensure that semistructured data can be reconstructed from XML documents this transformation must be lossless. Finally the paper also presents an algorithm for reconstructing semistructured data.

## 1 Introduction

An increasing number of semistructured data such as HTML documents and XML documents are available on World Wide Web. In most cases there is some structure in the data. However, it is too irregular to be easily modeled with a relational or an object-oriented approach. Data integrated from heterogenous data sources are often semistructured data. Research on semistructured data over the last several years has focused on data models, query languages, and systems where the database is modeled

in some form of a labeled, directed graph [1]. OEM (Object Exchange Model) often adopted for representing semistructured data is a simple self-describing object model with nesting and identity [2]. In OEM every object consists of an identifier and a value. The identifier uniquely identifies the object. The value of an object id is either an atomic quantity, such as an integer or a string, or a set of references denoted as  $\langle attribute, object \rangle$ . Here *attribute* denotes the relation between objects. This is flexible to store semistructured data for its often missing schema in advance, but it is difficult and inefficient to query semistructured data.

XML (eXtensible Markup Language) is a meta-markup language. It has become the dominant standard for representing and exchanging data in the World Wide Web [3]. Users can build their application with tools such as DOM (Document Object Model) or SAX (Simple API for XML) for XML.

OEM data model has the structure of a general graph, while the DOM data model for XML has the structure of a tree. However, semistructured data and XML have many similarities [4]. The database group of Stanford University had migrated the Lore data model and query language to support XML by creating OEM from XML documents and building the DateGuide from DTD [5]. On the other hand, there are some differences between the two data models, and it should be an important problem to study the method to represent and manage semistructured data with XML. Though DOM of XML documents is a node tree we can treat it as a semantic graph with XML applications. This observation is the foundation of our work. The paper tries to solve the problem how to use semantic information of XML tags to losslessly transform semistructured data of graph structures into XML and XML-Schema documents of a tree structure. In the following, an approach is proposed to transform semistructured data into XML and XML documents, and then the corresponding algorithms are designed for such lossless mapping. For the purpose that the changes made on the data of XML and XML-Schema documents can be reflected in the corresponding semistructured data the algorithm for reconstructing semistructured data from XML document and XML-Schema is also presented.

## 2 Approaches for store mapping

The data model of semistructured data in this paper is OEM. Data consists of a collection of objects, in which each object is either complex or an atomic one. A complex object is a set of  $\langle attribute, object \rangle$  pairs, and an atomic object is an atomic value of type int, string, video, etc. Hence, the structure of the data is a graph with edges labeled with attributes and some leaves labeled with atomic values.

DOM for XML is a node tree what does not directly support the graph structure. To solve the key difference we try to represent the graph structure at a semantic level. We notice that the parsing structure of an XML document is a tree, but its semantic structure is a graph. A parser for XML can accomplish the task of parsing and an XML application can recognize its corresponding semantic graph.

To represent a semantic graph with XML we can use attributes of XML tags, such as “id”, “idref”, “idrefs”. Furthermore, we can also define “ID”, “IDREF” or “IDREFS” elements of XML. Each value of an “id” attribute or an “ID” element

## Semistructured Data Store Mapping with XML and Its Reconstruction

provides a unique identifier of the object, so that other objects can refer to it. Given an instance of semistructured data we can generate an XML document and an XML-Schema document. Here we adopt XML-Schema instead of DTD because an XML-Schema is also an XML document convenient to be processed. XML-Schema designed as a substitute for DTD has many good features. The following two examples illustrate our store mapping approaches.

### Example 1:

The approach proposed in this example is to represent semistructured data with XML. "OBJECTS", "OBJECT", "ID", "IDREF" and "DATA" are five XML elements designed in the example. An "OBJECTS" element is designed for representing an object set. An "OBJECT" element is designed for representing an object, and an "ID" element is designed for representing an object identifier. Edges of OEM can be described with "IDREF" element which contains an "edgename" attribute for denoting the attribute between objects in OEM. The values of objects in OEM are recorded by "DATA" elements. An OEM graph is shown in Fig.1. Fig.2 shows its corresponding textual representation of the semistructured data. The OEM data graph of Fig.1 is represented with XML and the XML-Schema documents in Fig.3 and 4.

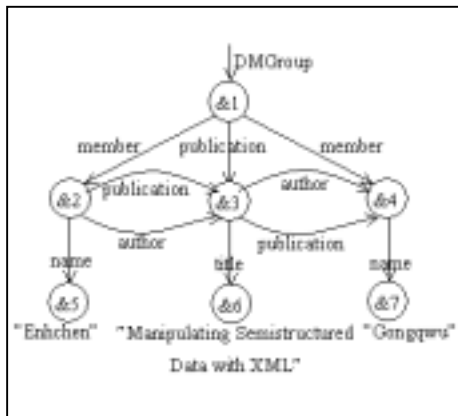


Fig.1: Semistructured data represented with OEM

```

DMGroup: &1
{member: &2
 {name: &5 "Enhchen",
  publication: &3
 }
publication: &3
 {title: &6 "Manipulating
  Semistructured Data with XML",
  author: &2,
  author: &4}
member: &4
 {name: &7 "Gongqwu"
  publication: &3}
}
    
```

Fig.2: Textual representation of data

```

<OBJECTS>
<OBJECT>//root object
  <ID/>
  <IDREF edgename="DMGroup">&1</IDREF>
</OBJECT>//root object
.....//content at here represents for object &1
<OBJECT>//object &2
  <ID>&2</ID>
    
```

```

<IDREF edgename="name">&5</IDREF>
<IDREF edgename="publication">&3</IDREF>
</OBJECT>//object &2
<OBJECT>//object &5
  <ID>&5</ID>
  <DATA>Enhchen</DATA>
</OBJECT>//object &5
.....// content at here represents for object &1&3,&4,&6,&7
</OBJECTS>
    
```

Fig.3: XML representation of data with the approach in example 1

```

<?xml version="1.0"?>
<Schema name="Apch1Schema" xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <AttributeType name="edgename" dt:type="string"/>
  <ElementType name="IDREF" content="textonly">
    <attribute type="edgename"/>
  </ElementType>
  <ElementType name="ID" content="textonly"/>
  <ElementType name="DATA" content="textonly"/>
  <ElementType name="OBJECT" content="eltonly">
    <element type="ID" minOccurs="0" maxOccurs="*" />
    <element type="DATA" minOccurs="0" maxOccurs="*" />
    <element type="IDREF" minOccurs="0" maxOccurs="*" />
  </ElementType>
  <ElementType name="OBJECTS" content="eltonly">
    <element type="OBJECT" minOccurs="0" maxOccurs="*" />
  </ElementType>
</Schema>
    
```

Fig.4: XML Schema for the XML document shown in Fig.3

### Example 2:

Each name of an edge in the breadth first spanning tree (BFSTree) of the OEM is defined as an XML element tag. An object in OEM is represented with the value of an element of XML. Here the elements come from the corresponding name(attribute) of an edge of OEM. The identifier of each object is represented by its "id" attribute. The value of an atomic object is recorded with the value of its corresponding element. Subobjects are described with the subelements of its parent object in the BFSTree, so the structure of the BFSTree is reflected in the XML document. Reference edges which are not in the BFSTree can be represented by "IDREF" elements mentioned in Example 1. Fig.5 and Fig.6 are the new representation of the XML document and its XML-Schema document.

```

<DMGroup ID="1">
  <member ID="2">
    <name ID="5">Enhchen</name>
  </member>
</DMGroup>
    
```

## Semistructured Data Store Mapping with XML and Its Reconstruction

```
<IDREF edgename="publication">&3</IDREF>
</member>
<publication ID="&3">
  <title ID="&6">Manipulating semistructured data with XML</title>
  <IDREF edgename="author">&2</IDREF>
  <IDREF edgename="author">&4</IDREF>
</publication>
<member ID="&4">
  <name ID="&7">Gongqwu</name>
  <IDREF edgename="publication">&3</IDREF>
</member>
```

Fig.5: XML representation of data with the approach in example 2

```
<?xml version="1.0"?>
<Schema name="Apch2Schema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <AttributeType name="ID" dt:type="id"/>
  <AttributeType name="edgename" dt:type="string"/>
  <ElementType name="IDREF" content="textonly">
    <attribute type="edgename"/>
  </ElementType>
  <ElementType name="name" content="textonly">
    <attribute type="ID"/>
  </ElementType>
  .....//ElementType "title" is defined at here
  <ElementType name="member" content="elonly">
    <element type="name" minOccurs="0" maxOccurs="*" />
    <element type="IDREF" minOccurs="0" maxOccurs="*" />
    <attribute type="ID"/>
  </ElementType>
  .....//ElementType "publication" and "DMGroup" are defined at here
</Schema>
```

Fig.6: XML Schema for XML document shown in Fig.5

Semistructured data can be described with XML by the approaches mentioned in Example 1 and 2. Attributes "idref" and "idrefs" provide other ways to solve the problem. "IDREF" elements are just introduced to describe our algorithms more easily.

Though the approach introduced in Example 1 can express semistructured data with XML directly and simply, it can not completely reflect the tree structure of the original data that could easily be dealt with an XML application. The approach introduced in Example 2 can express the tree structure convenient for XML application. Thus in the following the paper chose the approach proposed in Example 2. Certainly, there are many things to be done for such transformation. The corresponding algorithms for mapping and data reconstructing are given in the next two sections.

When performing the transformation above, we should notice that:

1. Not all edges in the OEM are labelled with attributes, but suitable labels can be added when necessary.
2. Some objects are ordered and others are not, “group” and “order” attributes in XML can represent the sequence of the selected objects.

### 3 Store mapping

Given the textual representation of semistructured data store mapping rewrite it into an XML document and an XML-Schema document. First OEM graphs are created from the textual data. Then a node is added to be the parent of all original roots (top level objects) of the OEM graphs, and an OEM graph with unique root will be obtained. The edges which are not in the breadth first spanning tree of OEM graph are given additional flags which will later be used for XML and XML-Schema document generation. Then the OEM graph with added flags is used to create an XML document whose semantic meaning corresponds to the structure of the original graph. Through deleting edges having an added flag a tree called T will be obtained, and a schema tree used to create an XML-Schema document will be obtained by compressing the tree T. Algorithms for mapping OEM data to XML documents are illustrated in Fig.7 and 8. Fig.7, 9, and 10 describe the algorithm for mapping an OEM data to an XML-Schema document.

<p>ALGORITHM 1: Mapping Semistructured Data to XML and XML Schema Document INPUT: Textual Representation of Semistructured Data OUTPUT: XML Document and XML Schema Document</p> <p>Step 1: construct OEM graph with unique root node from textual data; Step 2: perform breadth first traversing from root node of OEM, record the level of all nodes and add flag to edges which point to an upper or the same level node; Step 3: create an empty XML document, write some routine information and start tag of root element into the document; Step 4: for all children C of root node do{Generate XML data by applying algorithm 2 from C} Step 5: write end tag of root element into XML document; Step 6: delete the edges that have been added flag in step 2 and get a tree T; Step 7: transform T into a schema tree T' by merging the subtrees of T from root node with algorithm 3; Step 8: create an empty XML Schema document, write some routine information and start tag of “Schema” element into the document; Step 9: write definitions of “ID” attribute, “edgename” attribute and “IDREF” element into XML Schema document; Step 10: transform T' into XML Schema with algorithm 4 starting from root node of T'; Step 11: write end tag of “Schema” element.</p>
--

Fig.7: Mapping Semistructured Data to XML and XML-Schema Documents

## Semistructured Data Store Mapping with XML and Its Reconstruction

```
ALGORITHM 2: Generate XML data by Depth First Traversing the OEM graph added
              flag at step 2 in Algorithm 1
INPUT:       Node N of OEM graph
OUTPUT:     XML data

Step 1: if node N is a atomic object then{
Step 2:   Label      := name of edge which connect node N and its parent node;
Step 3:   IdValue    := identifier of node N;
Step 4:   AtomValue  := string format value of node N;
Step 5:   String := '<'+Label+" ID="+IdValue+''+>'+ Atom-Value
              +'</'+Label+'>';
Step 6:   write String into XML document; }
Step 7: if node N is a complex object then{
Step 8:   Label      := name of edge which connect node N and its parent node;
Step 9:   IdValue    := identifier of node N;
Step 10:  String     := '<'+Label+" ID="+IdValue+''+>';
Step 11:  write String into XML document;
Step 12:  for all child node C of node N do{
Step 13:    if edge E which connect node N and C has been added flag then{
Step 14:      SubLabel := name of edge E;
Step 15:      SubIdValue := identifier of node C;
Step 16:      String := "<IDREF edgename="+SubLabel+''+>'+ SubIdValue
                    +'</IDREF>";
Step 17:      write String into XML Document; }
Step 18:      else generate XML data by recursively applying algorithm 2; }
Step 19:  String := "</"+Label+'>';
Step 20:  write String into XML document; }
```

Fig.8: Generating XML Data by Depth First Traversing the OEM graph

```
ALGORITHM 3: Generate schema tree
INPUT:       Node N of the tree T' constructed at Step 7 in Algorithm 1
OUTPUT:     Schema tree

Step 1: if N is a leaf node then return;
Step 2: for all child node C of node N do
Step 3:   if name of edge E which connect C and N existed at the same level then{
Step 4:     find node C' and corresponding edge E' holding same name with E, which
            connects C' and N;
Step 5:     all subtrees of C is moved to be subtrees of C';
Step 6:     delete node C and edge E; }
Step 7: for all child node C of node N do
Step 8:   recursively applying algorithm 3 from node C;
```

Fig.9: Generate schema tree algorithm

In algorithm 1 an OEM graph with unique root for processing conveniently is obtained at Step 1. Edges not on the breadth first spanning tree get additional flags at Step 2. This step is necessary to prepare the generation of the XML and XML-Schema document reflecting the tree structure of the original data. An XML document is obtained by Steps 3, 4, and 5. The schema tree for generating the XML-

Schema document is obtained by Steps 6 and 7, and an XML-Schema document is obtained by Steps 9, 10, and 11. Here, the generated XML Schema document losslessly express the structure of original data for schema information of edges deleted at Step 6 is added in Algorithm 4.

<p>ALGORITHM 4: Generate XML Schema Data from Schema Tree</p> <p>INPUT: Node N of Schema Tree</p> <p>OUTPUT: XML Schema Data</p> <p>Step 1: if N is a atomic object then{</p> <p>Step 2: Label := name of edge which connect node N and its parent node;</p> <p>Step 3: String1 := "&lt;ElementType name=" + Label + " content=" + "textOnly" + "&gt;";</p> <p>Step 4: String2 := "&lt;attribute type=" + "ID" + "&gt;";</p> <p>Step 5: String3 := "&lt;/ElementType&gt;";</p> <p>Step 6: write String1, String2 and String3 into XML Schema document;}</p> <p>Step 7: if N is a complex object then{</p> <p>Step 8: for all child node C of N do</p> <p>Step 9: recursively applying algorithm 4 from node C;</p> <p>Step 10: Label := name of edge which connect node N and its parent node;</p> <p>Step 11: String := "&lt;ElementType name=" + Label + " content=" + "eltOnly" + "&gt;";</p> <p>Step 12: write String into XML Schema document;</p> <p>Step 13: for all child node C of N do{</p> <p>Step 14: Label := name of edge which connect node C and N;</p> <p>Step 15: String := "&lt;element type=" + Label + " minOccurs=" + "0" + " maxOccurs=" + "*" + "&gt;";</p> <p>Step 16: write String into XML Schema document;}</p> <p>Step 17: String1 := "&lt;element type=" + "IDREF" + "&gt;";</p> <p>Step 18: String2 := "&lt;attribute type=" + "ID" + "&gt;";</p> <p>Step 19: String3 := "&lt;/ElementType&gt;";</p> <p>Step 20: write String1, String2, String3 into XML Schema document;}</p>
--

Fig.10. Algorithm for Generating XML Schema Data from schema tree

XML data are generated by depth first traversing the OEM graph in algorithm 2 where the structure of the XML document is a nested hierarchical structure. An XML data segment which is uniquely identified by its object identifier will be generated for each object of the OEM graph. Here XML element tag is the name of a corresponding edge. XML data for atomic object are generated by Step 1-6, and XML data for complex objects are generated by Step 7-20 in algorithm 2. All reference edges are represented by "IDREF" element tags as shown by Step 13-17. It can be seen from the mapping process that the XML document generated from semistructured data represents the structure of the original data and the mapping is lossless for reference edges that have been transformed into corresponding XML data.

XML-Schema data are generated from the schema by applying algorithm 4. Schema data of atomic objects are obtained by Step 1-6, and schema data of complex data are obtained by Steps 7-20 in algorithm 4. XML-Schema data generated by algorithm 4 represent the nested hierarchical structure of the schema tree. Schema information of A breadth first spanning tree has been obtained by deleting edges with

## Semistructured Data Store Mapping with XML and Its Reconstruction

added flag at Step 6 of algorithm 1. A schema tree for generating XML-Schema documents is obtained by applying algorithm 3 in which subtrees with a same edge name at the same level are merged.

Reference edges of the OEM graph are added at Step 16 of algorithm 4, so that the mapping for transforming the OEM graph into an XML-Schema document is lossless.

The algorithms demonstrated in Fig.7-10 have accomplished schema discovery for semistructured data and have implemented a lossless mapping for transforming semistructured data into XML and XML-Schema documents.

### 4 Reconstructing data

XML and XML-Schema documents generated by algorithms 1-4 may be changed in the future, therefore it is necessary to design reconstruction algorithms to reflect such update information of XML in semistructured data. It is not difficult for us to do that with the help of the DOM. Algorithm 5 shown in Fig.11 performs the transformation of XML and XML-Schema documents into semistructured data. A tree is generated in Steps 1-12 in Fig.11 firstly, then reference edges are added in Steps 13-16. Main operations of algorithm 5 are search and traversal of tree, and the efficiency of algorithm 5 depends on the implementation of the DOM

ALGORITHM 5: Mapping XML and XML Schema Document into Semistructured Data

INPUT: XML Document, XML Schema Document

OUTPUT: Textual Representation of Semistructured Data

Step 1: parse XML document into a node tree by applying DOM;  
Step 2: find node R in DOM corresponding to root element in XML document;  
Step 3: create and initialize node R' of OEM graph with "ID" attribute of R;  
Step 4: initialize Queue Q, Q' and put all child node for IDREF element into Q;  
Step 5: while not empty Q do{ get a node F from front of Q;  
Step 6: create and initialize node F' of OEM graph with "ID" attribute of F;  
Step 7: find node P which is parent node F, and find node P' of OEM graph corresponding to node P;  
Step 8: create and initialize edge E with element tag of F, start node of it is P', end node of it is F';  
Step 9: if T is a text child node of F then F' is evaluated by value of T;  
Step 10: else for all element child node C of F do{  
Step 11: if C is not an "IDREF" element node then put C into Q;  
Step 12: if C is an "IDREF" element node then put C into Q';}}  
Step 13: while not empty Q' do{ get a node F from front of Q';  
Step 14: find node P which is parent of F, and find node P' of OEM graph corresponding to P;  
Step 15: find node F' of OEM graph "ID" attribute's value of which is value of text child node of F;  
Step 16: create and initialize edge E with value of "edgename" attribute of F, start node of it is P', end node of it is F';}  
Step 17: write OEM generated through steps above with textual format;

Fig.11. Algorithm for Mapping XML to Semistructured Data

## 5 Conclusion

The paper has studied how to represent semistructured data with XML and has presented algorithms for such transformation. Semistructured data can be reconstructed from XML and XML-Schema and our transformation is lossless. Lore has been migrated from semistructured data to XML by transforming XML into an OEM model and build DataGuide with DTD. Our goal is just the opposite where the advantages are embodied in that we can directly apply existing tools such as DOM and SAX for XML to manipulate semistructured data. Processing steps such as inserting, deleting, modifying data and querying on XML is easier than performing such processing directly on semistructured data. Generating query statements on XML is also easier than performing the task directly on semistructured data.

Storing semistructured data in relations is an ambitious goal, because the two models are apparently incompatible. An approach has been proposed for storing semistructured data with relations by Deutsch [6]. Though it is a lossless transformation, it requires an overflow graph to store data that cannot be completely transformed. We will study how to implement such transformation with an overflow graph and query semistructured data with XML efficiently.

## Acknowledgement

The research work of the paper was supported by National Natural Science Foundation of China with Grant No. 60005004.

## References

- [1] S. Abiteboul. Querying semistructured data. In Proceedings of the International Conference on Database Theory, pages 1--18, Delphi, Greece, January 1997.
- [2] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In IEEE International Conference on Data Engineering, pages 251--260, March 1995.
- [3] Editors: T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible markup language (XML) 1.0, February 1998. W3C Recommendation available at <http://www.w3.org/TR/1998/REC-xml-19980210>
- [4] D. Suciu. Semistructured Data and XML. In Proc. of the 5th Int. Conf. of Foundations of Data Organization (FODO'98), November 1998
- [5] Roy Goldman , Jason McHugh, Jennifer Widom : From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. WebDB (Informal Proceedings) 1999 : 25-30
- [6] Alin Deutsch, Mary Fernandez, Dan Suciu, Storing Semi-structured Data Using STORED. In SIGMOD, 1999