Recovering data semantics from XML documents into DTD Graph with SAX

HERBERT SHIU
Department of Computer Science
City University of Hong Kong
83 Tat Chee Av., Kowloon
HONG KONG
http://www.cs.cityu.edu.hk

JOSEPH FONG
Department of Computer Science
City University of Hong Kong
83 Tat Chee Av., Kowloon
HONG KONG
http://www.cs.cityu.edu.hk

ROBERT P. BIUK-AGHAI
Department of Computer and Information Science
University of Macau
Av. Padre Tomás Pereira, S.J.
MACAU
http://www.fst.umac.mo

Abstract: We propose a systematic approach to reverse engineer arbitrary XML documents to their conceptual schema, DTD Graphs. The necessity for doing so is due to the fact that XML documents are frequently used for storing structured data and their schemas, such as in Document Type Definition (DTD) format, are missing, especially for those existing historical XML documents. As such, it is difficult for software developers or end users to make use of them. Even the schemas exist, they are difficult to read and undetermined of the underlying relationships among the elements in the documents. In view of this, it is necessary to determine the data semantics from the XML documents. If the DTDs of the XML documents exist with the identifications of the ID/IDREF(S) type attributes, then more data semantics can be derived. Another application of the determined data semantics is to verify the linkages implemented by ID/IDREF(S). If the element is referring to an incorrect XML element type, an extra data semantic will be determined as a result, and such findings can be used for verification purposes. Furthermore, the approaches proposed in this paper use Simple API for XML (SAX) so that the algorithms are applicable to small to huge sized XML documents.

Key-Words: XML document, DTD Graph, reverse engineering, data semantics, ID/IDREF(S), cardinality, SAX

1 Introduction
As Extensible Markup Language (XML) [1] has become the standard document format on the Internet, software developers have to deal with XML documents in different formats. According to the usages of the XML documents, their document sizes vary from several kilobytes to several gigabytes. For small XML documents, it is feasible to study their structures with either usual text editors or XML enabled viewers, such as a web browser like Microsoft Internet Explorer. However, for medium to huge sized XML documents, what people can do at best is to read the XML document contents just by scrolling up and scrolling down. If the schema of the XML documents, such as in DTD [2] or XSD format, are given or are derived from the XML documents right away, it is easier to study the contents of the XML documents but the formats of these schema are hard to read, not to mention their lack of user-friendliness.

In this paper, a methodology is proposed so that arbitrary data-centric XML document structure can be analyzed and reverse engineered to their conceptual schema, which are DTD Graphs, including cardinalities among entities implemented by parent-child relationship and ID/IDREF type attributes. There are mainly two categories of XML documents, which are data-centric and narrative. As the contents of narrative XML documents, such as DocBook [3] documents, are mainly unstructured and their vocabularies are basically static, the necessity of handling them as structured contents and reverse engineering them into conceptual models is far less than that of handling data-centric ones. Therefore, this paper will concentrate on data centric XML documents.

2 Related Work
Accompanying the widespread adoption of XML for representing many different kinds of information in organizations world-wide, there has been considerable interest in more fully integrating these documents into existing systems and organizational information infrastructures. Some XML documents may have been created in an ad-hoc fashion, but subsequently need to be integrated with other documents or databases. To address this need, these existing XML documents can be reverse engineered to recover their semantics, then re-engineered, before being forward engineered into the desired new structure. This process is illustrated in Fig. 1. Different approaches have been proposed for individual steps shown in this process: the recovery of data semantics from XML documents in the form...
of DTDs has been described in [12], while [9,13] describe the extraction of XML schemas. The subsequent step of recovering design semantics has been addressed by [14,15] for deriving UML class diagrams, by [16] for deriving EER models, and by [17] for deriving domain ontologies. However, the majority of research work to date has been concerned with the task of recovering design semantics, whereas little research exists that tackles the extraction of data semantics.

Although there is an approach that can reverse engineer data semantics from XML documents [7], the algorithm maps some predefined templates of document structures to data semantics, and the algorithm can only be implemented with DOM, which needs to read the entire XML document to the memory that is inapplicable to huge sized XML document. On the other hand, the methodology presented in this paper determines all candidate data semantics from arbitrary XML documents with SAX that is applicable to XML document of any size. As such, some of the determined data semantics may not be the intentions of the original writer and it therefore needs user supervision for verification.

Besides, some existing works concern the extraction of schema, such as DTD, from XML document [9] [10] whereas the algorithms proposed in this paper concern the determination of data semantics among the XML element instances rather than simply schema among XML elements. Besides, compared with the approach proposed by Goldman and Widom [11] that directly manipulates semi-structured databases, such as a XML documents, the algorithm proposed here enables the user to have a clear picture of the data semantics among the XML element instances before further manipulating them.

3 Approaches of Implementing Various Data Semantics

3.1 Cardinalities – one-to-many/one-to-one
One-to-many cardinalities within an XML document can be realized by both explicit and implicit referential linkages [6][7]. By implicit referential linkages, a parent element can have child elements of the same type, such as:

```xml
<PURCHASE_ORDER>
    <PURCHASE_ORDER_LINE .../>
    <PURCHASE_ORDER_LINE .../>
</PURCHASE_ORDER>
```

The parent element `PURCHASE_ORDER` and the child elements `PURCHASE_ORDER_LINE` are implicitly in a one-to-many relationship. If the occurrences of child element `PURCHASE_ORDER_LINE` are at most one for all `PURCHASE_ORDER` elements, they are in a one-to-one relationship instead.

If the schema of the XML document is given, it can specify the `ID/IDREF(S)` type attributes. If an XML element defines an `IDREF` attribute and all such elements refer to the same element type, there is a one-to-many relationship between the referred and referring XML elements. For example, sample DTD and XML documents are shown in Fig. 2.

```xml
<!ATTLIST PURCHASE_ORDER
    PO_ID ID #REQUIRED
...>
<!ATTLIST PURCHASE_ORDER_LINE
    PO_ID IDREF #REQUIRED
...>
```
In Fig. 2, a PURCHASE_ORDER element is referred by one or more PURCHASE_ORDER_LINE elements, and then there is a one-to-many relationship between these two element types. If the attribute definition of the PO_ID attribute of PURCHASE_ORDER_LINE is #IMPLIED instead of #REQUIRED, it is optional for a PURCHASE_ORDER_LINE element to refer a PURCHASE_ORDER element or not, and they can be considered to be partial participation. In the above example, as the PO_ID attribute definition of the PURCHASE_ORDER_LINE is #REQUIRED, they are considered to be total participation.

Besides IDREF, element with IDREFS type attribute can be used to implement one-to-many cardinality. As IDREFS type attribute can refer more than one XML element in the document, if the referred elements are of the same type and each referred element is referred once, the referring element and the referred elements can be considered to be in a one-to-many relationship. For example, consider the sample DTD and XML documents shown in Fig. 3.

In Fig. 3, the PURCHASE_ORDER is referring to two PURCHASE_ORDER_LINE elements with its IDREFS type POL_IDS attribute. If each PURCHASE_ORDER_LINE element is referred by one PURCHASE_ORDER element only, the PURCHASE_ORDER and the PURCHASE_ORDER_LINE can be considered to be in a one-to-many relationship. For explicit referential linkages, to determine the cardinality is one-to-one or one-to-many, it is necessary to scan the entire XML document to determine the maximum count of referring elements referring of that type referring to the same referred XML element.

3.2 Cardinality – many-to-many
An XML element type may be involved in more than one one-to-many relationship. In other words, all elements of such XML element type define more than one linkage. For example, if an XML element type defines an IDREF(S) type attribute, all elements of such XML element type actually defines two linkages, one implicit linkage by the nested structure and one explicit linkage by IDREF(S) type attribute.

If the two linkages are both one-to-many relationships, the two referred element types by such referring element type can be considered to be in a many-to-many relationship. For example, the XML document in 3 illustrates a many-to-many relationship.

![Fig. 2 A many-to-one cardinality implemented by an IDREF type attribute](image)

![Fig. 3 A one-to-many cardinality implemented by an IDREFS type attribute](image)

For an XML element type that defines two linkages and hence two one-to-many relationships, the two referred XML element types can be considered to be in a many-to-many relationship that is consistent with existing approach of exporting XML elements for many-to-many relationships [4]. Take a step further. If the XML element type defines three or more linkages and it is therefore involved in more than two one-to-many relationships, the referred XML element types are considered to be in an n-ary relationship.

Many-to-many relationship can be implemented with IDREFS type attribute as well, since an IDREFS type attribute can refer to more than one instance of the same XML element types. For example, consider the DTD and XML documents as shown in Fig. 4.

Such co-existence relationship specified in the schema can be extended to more than one nested level. For example, if the existence of a course element must be accompanied by a lecturer element and a tutor element, that is:

```xml
<!ELEMENT course (lecturer, tutor)>
```
the elements, enrollment, student, course, lecturer and tutor, must exist as a whole. Then, we can consider all these elements are in an aggregation relationship.

4 Algorithms for Determining Cardinality Relationships

The data structure of the algorithms are:
1. **MNG**: The maximum number of elements of the same element type that are referred by a single referring element with the same linkage type. The value must be one for IDREF type attribute and implicit linkages, and can be greater than one for IDREFS type attribute.

2. **MND**: The maximum number of the referring elements of the same element type that are referring to the same referred element with the same linkage type.

3. **NL**: The number of referring elements that possess the linkage.

Besides the above information, it is necessary to obtain the counts of all referring elements (NE) in the XML document.

According to the combination of the values of the four attributes, it is possible to determine the cardinality data semantics for the involved elements. The rules are shown in Table 1.

The algorithm is composed of a two passes of parsing of the same XML document. The first pass assigns a synthetic element identity to each XML element in the document and determines all ID type attribute values and their corresponding element types. For the second pass, the XML document is traversed again and the linkages of each XML element are investigated and their attributes are stored. Finally, the stored linkage attributes are consolidated to give the four linkage attributes mentioned above and in Table 1. The complete algorithm is presented in Fig. 5.

### Table 1 Matrix for determining cardinality & participation based on the determined linkage attributes

<table>
<thead>
<tr>
<th>Cardinality</th>
<th>Participation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
</tr>
<tr>
<td>One-to-one</td>
<td>MNG = 1</td>
</tr>
<tr>
<td></td>
<td>MND = 1</td>
</tr>
<tr>
<td></td>
<td>NL = NE</td>
</tr>
<tr>
<td>One-to-many</td>
<td>MNG = 1</td>
</tr>
<tr>
<td></td>
<td>MND &gt; 1</td>
</tr>
<tr>
<td></td>
<td>NL = NE</td>
</tr>
<tr>
<td>Many-to-one</td>
<td>MNG &gt; 1</td>
</tr>
<tr>
<td></td>
<td>MND = 1</td>
</tr>
<tr>
<td></td>
<td>NL = NE</td>
</tr>
<tr>
<td>Many-to-many</td>
<td>MNG &gt; 1</td>
</tr>
<tr>
<td></td>
<td>MND &gt; 1</td>
</tr>
<tr>
<td></td>
<td>NL = NE</td>
</tr>
</tbody>
</table>

### Given Relation

<table>
<thead>
<tr>
<th>Relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElementIDName (ID, RDE)</td>
<td></td>
</tr>
<tr>
<td>ElementNameCount (RGE, NE)</td>
<td></td>
</tr>
<tr>
<td>RawReferedInfo (RGE, RDE, LINK_NAME, LINK_VALUE, ND)</td>
<td></td>
</tr>
<tr>
<td>ReferringInfo (RGE, RDE, LINK_NAME, MNG, NL)</td>
<td></td>
</tr>
<tr>
<td>ReferredInfo (RGE, RDE, LINK_NAME, MND)</td>
<td></td>
</tr>
</tbody>
</table>

### Pass One:

Let element ID (EID) = 1

Traverse the XML document with SAX.

Whenever the start element of E is encountered:

- Select the record from `ElementNameCount` for the element name of E.
- If the record exists:
  - Increment NE by 1 and update the record to the `ElementNameCount` table.
- Else:
  - Insert a new record (element name, 1) to the `ElementNameCount` table.
  - Insert a new record (ID, element name) to the `ElementIDName` table.
  - If E defines an ID type attribute A:
    - Insert a new record (Value of A, element name of E) to the `ElementIDName` table.
- End If

Increase the value of EID by 1.

### Pass Two:

Traverse the XML document with SAX.

Whenever the start element (the referring element, RGE) is encountered:

- For each linkage, L, of RGE:
  - Get referred element (RDE) from `ElementIDName` table by attribute value of L, Lvalue.
  - Select record from the `RawReferedInfo` table for primary key (RGE, RDE, L, Lvalue).
If the record exists
Increase ND by 1 and update the record to the table
Else
Insert a record (RGE, RDE, L, \(L_{\text{value}}\), 1) to the table RawReferredInfo
For each referred element type, RDE
Let NG be the number of RDE referred by this linkage, L
Select the record from the table ReferringInfo for (RGE, RDE, L)
If the record exists
Update MNG with maximum of (MNG, NG) and increment NL by 1
Update the record to the table ReferringInfo
Else
Insert a new record (RGE, RDE, L, NG, 1) to the table ReferringInfo
Upon the completion of traversing the XML:
Consolidate the records with the same combination of (RGE, RDE, L) in the table RawReferredInfo
let MND to be the maximum of the ND values of all records
insert a record (RGE, RDE, L, MND) to the table ReferredInfo

Fig. 5 The table structures and algorithm for determining linkage information by traversing the XML document with SAX

5 Case Study and Prototype
To illustrate the applicability and correctness of the algorithms mentioned in this paper, a prototype was built that implements the algorithms that are proposed in this paper. With such prototype, a sample XML document with DTD file as shown in Fig. 6 are provided to the prototype and the data semantics are determined as shown in Fig. 4-Error! Reference source not found.

6 Conclusion
This paper provides algorithms to help the users to understand the relationships among the elements by reverse engineering data semantics from the document. Furthermore, the algorithms apply SAX for processing the XML documents so that even huge XML documents can be processed without reading the documents entirely into the computer memory. Moreover, the data structures to be used can be supported by most programming language, or tables in a relational database, and it is therefore feasible to apply the algorithms to XML documents of any size.

Acknowledgement: This paper is funded by Strategic Research Grant 7001839-540 of City University of Hong Kong

References:
[9] Boris Chidlovskii, Schema Extraction from XML Data: A Grammatical Inference

Fig. 6 test.xml and test.dtd

The sample XML and DTD file, test.xml and test.dtd, are supplied to the prototype software and the determined findings are shown in Fig. 7-Fig. 9.
Approach, KRDB’01 Workshop (Knowledge Representation and Databases)


Fig. 7 The determined linkage information

Fig. 8 The determined data semantics

Fig. 9 DTD Graph based on DTD with two one-to-many cardinalities (one many-to-many cardinality)