Reverse Engineering from an XML Document into an Extended DTD Graph

Herbert Shiu, City University of Hong Kong, Hong Kong
Joseph Fong, City University of Hong Kong, Hong Kong

ABSTRACT

Extensible markup language (XML) has become a standard for persistent storage and data interchange via the Internet due to its openness, self-descriptiveness, and flexibility. This article proposes a systematic approach to reverse engineer arbitrary XML documents to their conceptual schema—extended DTD graphs—which is a DTD graph with data semantics. The proposed approach not only determines the structure of the XML document, but also derives candidate data semantics from the XML element instances by treating each XML element instance as a record in a table of a relational database. One application of the determined data semantics is to verify the linkages among elements. Implicit and explicit referential linkages are among XML elements modeled by the parent-children structure and ID/IDREF(S) respectively. As a result, an arbitrary XML document can be reverse engineered into its conceptual schema in an extended DTD graph format. [Article copies are available for purchase from InfoSci-on-Demand.com]

Keywords: Data Semantics; Extended DTD Graph; Reverse Engineering; XML Document

INTRODUCTION

As extensible markup language (XML) (Bray, Paoli, Sperbeg-McQueen, Maler, & Yergeau, 2004) has become the standard document format, the chance that users have to deal with XML documents with different structures is increasing. If the schema of the XML documents in document type definition (DTD) (Bosak, Bray, Connolly, Maler, Nicol, Sperberg-McQueen, Wood, & Clark, 1998) is given or derived from the XML documents right away (Kay, 1999; Moh, Lim, & Ng, 2000), it is easier to study the contents of the XML documents. However, the formats of these schemas are hard to read, not to mention rather poor user-friendliness.

XML has been the common format for storing and transferring data between software applications and even business parties, as most software applications can generate or handle XML documents. For example, a common scenario is that XML documents are generated and based on the data stored in a relational data-
base—and there have been various approaches for doing so (Thiran, Estiévenart, Hainaut, & Houben, 2004; Fernandez, Morishima, & Suciu, 2001). The sizes of XML documents that are generated based on the data stored in databases can be very large. Most probably, these documents are stored in a persistent storage for backup purposes, as XML is the ideal format that can be processed by any software applications in the future.

In order to handle the above scenario, it is possible to treat XML element instances in an XML document as individual entities, and the relationships from the different XML element types can be determined by reverse engineering them for their conceptual models, such as extended DTD graphs with data semantics. As such, users can have a better understanding of the contents of the XML document and further operations with the XML document become possible, such as storing and querying (Florescu & Kossman, 1999; Deutsch, Fernandez, & Suciu, 1999; Kanne, 2000).

This article proposes several algorithms that analyze XML documents for their conceptual schema. Two main categories of XML documents exist — data-centric and narrative. As the contents of narrative XML documents, such as DocBook (Stayton, 2008) documents, are mainly unstructured and their vocabulary is 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 article will concentrate on data centric XML documents.

**Referential Integrity in XML Documents**

XML natively supports one referential integrity mechanism, which are ID/IDREF(S) types of attribute linkages. In every XML document, the value of an ID type attribute appears at most once and the value of the IDREF(S) attribute must refer to one ID type attribute value(s). An IDREF(S) type attribute can refer to any XML element in the same document, and each XML element can define at most one ID type attribute. Due to the nature of ID/IDREF(S) type attributes in XML documents, relationships among different XML element types can be realized and it is possible to use them to implement data semantics.

This article will discuss the various data semantics and the possible ways to implement them. The algorithms presented in the article are based on the observations of the common XML document structures.

1. Using the nested structure of an XML document (the relationship between a parent element and its child element[s]), in which the child elements implicitly refer to their parent element.
2. For an IDREF or IDREFS type attribute, the defining element is referred to the element(s) with an ID type attribute by the referred value. Such linkages are similar to the foreign keys in a relational database. The two associated element types are considered to be linked by an explicit linkage.
3. As an IDREFS type attribute can refer to more than one element, there is a one-to-many cardinality from the referring element type and the referred element type(s).

The schema of an XML document can restrict the order of the XML elements—and the order of the elements may be significant—which depends on the intentions of the original XML document designer. For example, two XML documents with their corresponding DTDs are shown in Table 1.

The two XML documents shown in Table 1 are storing the same data, which are the data of two couples. For the former one, its couple elements use the two IDREF type attributes to denote the corresponding husband and wife elements. However, the use of ID/IDREF cannot ensure a particular husband or wife element must be referred by one couple element only. For the latter XML document, the DTD restricts that the husband and wife elements must exist as a pair. Furthermore, the
use of ID type attributes hid and wid ensures any husband and wife element instance must exist in the document at most once.

**Extended DTD Graph**

As XML element instances are treated as individual entities, the relationships from the element types are therefore related not only to the structure of the XML document but also to the linkages from the different types. As such, DTD cannot clearly indicate the relationships.

An extended DTD graph for XML is proposed to add data semantics into a DTD graph so that the data semantics can be clearly identified, which is an excellent way of presenting the structure of an XML document. As such, in order to visualize the data semantics determined based on the XML document with its optional schema, it will provide the notations to be used for presenting the various data semantics. This article uses the authors’ notations of the extended DTD graph for presenting the structure and the data semantics from the elements, as follows:

1. The vertexes as squares are drawn on the graph for elements, and vertexes as circles are drawn for occurrence operators (?, + and *) and selection operator (|).
2. Attributes and simple elements are omitted from the graph, as they specify a particular attribute of their defining and parent elements respectively.
3. Data semantics, other than one-to-one and one-to-many cardinality relations, are presented in the graph as arrows pointing from the referring element to the referred element with suitable descriptions as legends.

Based on the above criteria, it is possible to consider the ELEMENT declarations only for constructing the extended DTD graph. Three types of ELEMENT declarations can be identified as follows:

<table>
<thead>
<tr>
<th>DTD</th>
<th>XML Document</th>
</tr>
</thead>
</table>
| `<!ELEMENT couples (husband*,wife*,couple*)>` | `<?xml version="1.0"?>
<br/>&lt;couples&gt;
<br/>&lt;husband hid="A123456" name="Peter"/&gt;
<br/>&lt;husband hid="B234567" name="John"/&gt;
<br/>&lt;wife wid="X123456" name="Amy"/&gt;
<br/>&lt;wife wid="Y234567" name="Bonnie"/&gt;
<br/>&lt;couple hid="A123456" wid="X123456"/&gt;
<br/>&lt;couple hid="B234567" wid="Y234567"/&gt;
<br/>&lt;/couples&gt;` |
| `<!ELEMENT husband EMPTY>` | `<?xml version="1.0"?>
<br/>&lt;husband hid="A123456" name="Peter"/&gt;
<br/>&lt;husband hid="B234567" name="John"/&gt;
<br/>&lt;wife wid="X123456" name="Amy"/&gt;
<br/>&lt;wife wid="Y234567" name="Bonnie"/&gt;
<br/>&lt;/couples&gt;` |
| `<!ELEMENT wife EMPTY>` | `<?xml version="1.0"?>
<br/>&lt;husband hid="A123456" name="Peter"/&gt;
<br/>&lt;husband hid="B234567" name="John"/&gt;
<br/>&lt;wife wid="X123456" name="Amy"/&gt;
<br/>&lt;wife wid="Y234567" name="Bonnie"/&gt;
<br/>&lt;/couples&gt;` |
| `<!ATTLIST husband hid ID #REQUIRED
<br/>name CDATA #REQUIRED>` | `<?xml version="1.0"?>
<br/>&lt;husband hid="A123456" name="Peter"/&gt;
<br/>&lt;husband hid="B234567" name="John"/&gt;
<br/>&lt;wife wid="X123456" name="Amy"/&gt;
<br/>&lt;wife wid="Y234567" name="Bonnie"/&gt;
<br/>&lt;/couples&gt;` |
| `<!ATTLIST wife wid ID #REQUIRED
<br/>name CDATA #REQUIRED>` | `<?xml version="1.0"?>
<br/>&lt;husband hid="A123456" name="Peter"/&gt;
<br/>&lt;husband hid="B234567" name="John"/&gt;
<br/>&lt;wife wid="X123456" name="Amy"/&gt;
<br/>&lt;wife wid="Y234567" name="Bonnie"/&gt;
<br/>&lt;/couples&gt;` |
| `<!ATTLIST couple hid IDREF #REQUIRED
<br/>wid IDREF #REQUIRED>` | `<?xml version="1.0"?>
<br/>&lt;husband hid="A123456" name="Peter"/&gt;
<br/>&lt;husband hid="B234567" name="John"/&gt;
<br/>&lt;wife wid="X123456" name="Amy"/&gt;
<br/>&lt;wife wid="Y234567" name="Bonnie"/&gt;
<br/>&lt;/couples&gt;` |

Table 1. Two equivalent XML documents that can represent the same data
1. An ELEMENT declaration defines sub-elements only.
2. An ELEMENT declaration involves sub-elements and #PCDATA as its contents.
3. An ELEMENT declaration that defines #PCDATA as its contents only.

The above three types correspond to the following three examples:

```xml
<!ELEMENT PARENT (CHILD1+, CHILD2*)>
<!ELEMENT MIXED_ELEMENT (#PCDATA | CHILD1 | CHILD2)*)>
<!ELEMENT SIMPLE_ELEMENT (#PCDATA)>
```

For each ELEMENT declaration of the first type, the content model expression can be tokenized as individual elements and occurrence indicators and sequence separators (,), and represented as a tree structure with the element name as the root node. For example, the first example above can be visualized as the following tree diagram: in Figure 1, the sequence “,” is implied in the diagram.

DTDs mostly contain more than one ELEMENT declaration but each element type can only appear once. Therefore, to construct the complete DTD graph for a DTD, the tree structures of all ELEMENT declarations in a DTD are constructed first and they are eventually merged by replacing each sub-element node in a tree by the tree structure of that element. Such merging is repeated until there is only one tree structure or all sub-elements have been replaced with their corresponding tree structures.

### Cardinality / Participation

Element types are visualized as rectangles in the graph and a cardinality relationship is presented as an arrow pointing from the referring element type to the referred element type, with double-line and single line for total participation and partial participation respectively. The cardinality types, including one-to-one (1/1), one-to-many (1/m), many-to-one (m/1) and many-to-many (m/m), are shown as legends of the arrows. If the cardinality relationship is implemented as explicit ID/IDREF(S) linkages, the name of the ID type attribute of the referring element is appended to the legend, such as 1/m (parent_id). To identify explicit linkages from implicit linkages, cardinality relationships due to ID/IDREF(S) type attributes are shown as arrows with a curved line. Table 2 presents the eight possible combinations of arrows and legends.

### N-ary relationship

An n-ary relationship is implemented as a particular element type involved in more than two binary relationships. To represent such a relationship, a diamond-shaped vertex is used.

---

**Figure 1. A sample extended DTD graph**

![Sample extended DTD graph](image-url)
for such element types. Figure 2 presents a sample diagram with an \( n \)-ary relationship.

### Aggregation Relationship

An aggregation relationship denotes that the involved element types must exist as a unity. In Figure 2, an aggregation exists as the defining characteristic of mandatory participation between parent and child elements. As such, a rectangle is to be drawn enclosing all involved element types.

### RELATED WORK

In order to have a complete picture of the reasons behind the algorithms for determining various data semantics, this article explains the existing approaches of constructing XML documents, especially those exported from relational databases.

#### The Determination of XML Schema

There is some existing work concerning the extraction of schema, such as DTD, from XML documents (Chidlovskii, 2001; Min, Ahn, & Chung, 2003). The outputs of these algorithms are the schemas that can validate the XML documents. However, the derived schemas provide no semantic interpretation other than the containment structures of the XML documents. The algorithms proposed in this article concern the determination of data semantics from the XML element instances rather than simply XML schema among XML elements. Compared with the approach proposed by Goldman and Widom (1997) that directly manipulates semi-structured databases, such as an

---

**Table 2. The arrows illustrating various cardinalities with participation types**

<table>
<thead>
<tr>
<th>Participation Cardinality</th>
<th>One-to-one</th>
<th>One-to-many</th>
<th>Many-to-one</th>
<th>Many-to-many</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/1</td>
<td>1/m</td>
<td>m/1</td>
<td>m/m</td>
</tr>
</tbody>
</table>

---

**Figure 2. A sample diagram with an n-ary relationship**

```
   element1
     ↓ 1/m
   element2
     ↓ 1/m
   relation
     ↑ m/1
   element3
```
XML document, the algorithm proposed here provides the user with a clear picture of the data semantics from the XML element instances before further manipulating them.

The Determination of Data Semantics from XML Documents

One approach exists that can reverse engineer data semantics from XML documents (Fong & Wong, 2004), but the algorithm maps some predefined templates of document structures to data semantics, and the algorithm can only be implemented with DOM (W3C, 2003), which needs to read the entire XML document to the memory—and that is inappropriate for huge XML documents. The methodology presented in this article, however, determines basic candidate data semantics from arbitrary XML documents with SAX (Saxproject, 2004), which is applicable to XML documents of any size. Some of the determined data semantics may not be the intentions of the original writer and needs user supervision for verification.

The Implementation of Inheritance Among XML Elements

Schema for object-oriented XML (SOX) (W3C, 2005) introduced the idea of element and attribute inheritance, which enables an element to extend another element so that the derived element can have all attributes defined by the base element with its own new attributes.

Due to the limitations and low extensibility of DTD (Sahuguet, 2000), XML schema definition (XSD) (Sperberg, 2000) is becoming the popular replacement schema of DTD. Unlike DTD, XSD is an XML document itself and it can define more restrictive constraints and clear definitions of the XML documents to be validated. In other words, the set of capabilities for defining the structures and data types of XSD are the superset of that of DTD. As such, there has been research and software for converting DTD to XSD (Mello, 2001; W3C, 2000).

There are other alternative schemas, such as RELAX NG (RELAX NG, 2003) and Schematron (Schematron, 2008); and Lee and Chu ( ) evaluated six common XML schemas, including DTD and XSD. As they are not as popular as DTD and XSD, they are not discussed in this article.

By constructing a graph by placing vertices for elements—and the elements that are involved in a parent-child relation, which is defined by ELEMENT declaration in DTD, are connected with edges—it is possible to derive a graphical representation of the DTD that is commonly known as a DTD graph. Up to now, there is no formal standard for DTD graphs and various researchers are using their own conventions, as in Klettke, Schneider, and Heuer (2002), Shanmugasundaram, Shekita, Kiernan, Krishnamurthy, Viglas, Naughton, and Shematron (2001), Lu, Sun, Atay, and Fotouhi (2003), and Böttcher and Steinmetz (2003), and the graph introduced in Funderburk, Kiernan, Shanmugasundaram, Shekita, and Wei (2002) is the first one that was denoted as a DTD graph.

There is a graphical representation of XSD (Fong & Cheung, 2005), which derives an XML conceptual schema of an XML tree model from an XML schema of XSD. Its approach is different from this paper’s approach by deriving an extended DTD graph from an XML document.

As the conventions of most graphs for presenting the structure of an XML document are applicable to different schema languages, the graph is also known as semantic graph (An, Borgida, & Mylopoulos, 2005). Some researchers proposed other graphical representations of XML schemas, such as the use of UML (Booch, Christerson, Fuchs, & Koistinen, 1999).

The Application of Extended DTD Graph

Data graph is a DTD in graph. Zhao and Ziau (2007) described that DTD can be a good common data model, when the majority of data sources are XML sources, for the interoperability between relational databases and XML databases. Reserve engineering XML docu-
ment into DTD graph is similar to data mining XML document into a data tree (Zhang, Lui, Ling, Bruckner, & Tija, 2006). The former is a database schema while the later is an internal data in tree structure. Trujillo and Luján-Mora (2004) demonstrated that a DTD can be used to define the correct structure and content of an XML document representing main conceptual multidimension model for data warehouses.

Compared with the approach proposed by Goldman and Widom (1997) that directly manipulates semi-structured databases such as an XML document, the algorithm proposed in this article enables the user to have a clear picture of the data semantics from the XML element instances before further manipulating them. Table 3 provides a comparison between the proposed algorithms and other existing approaches.

REVERSE ENGINEERING METHODOLOGY

There are basically two different definitions in a DTD, which are ELEMENT and ATTLIST. Each ATTLIST definition defines the attributes of a particular element, whereas ELEMENT defines its possible containments, and each ELEMENT definition can be represented in a tree structure with the element name as the root element with its child sub-elements as leaves, and there must be another ELEMENT definition for each of its child elements.

It is not mandatory to define the ELEMENT declaration prior to all its child elements, and it is actually uncertain which element is the root element of the corresponding XML documents. The root element of the XML document is defined by the DOCTYPE declaration before the root element start tag.

Implementations of Various Data Semantics in XML

The following subsections provide all possible implementations of various data semantics, some of which are consistent with those proposed by other researchers (Lee, Mani, & Chu, 2003; Lee & Chu, 2000).

Cardinalities

One-to-many cardinalities can be realized by both explicit and implicit referential linkages. 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 >
```

<table>
<thead>
<tr>
<th>Table 3. A comparison between the proposed and other existing approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Proposed approach</strong></td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>Input</td>
</tr>
<tr>
<td>Output</td>
</tr>
<tr>
<td>Completeness</td>
</tr>
<tr>
<td>User friendliness</td>
</tr>
<tr>
<td>Performance</td>
</tr>
</tbody>
</table>
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 Figure 3.

For explicit referential linkages, to determine if the cardinality is one-to-one or one-to-many, it is necessary to scan the entire XML document. An XML element type may be involved in more than one one-to-many relationship. In other words, all elements of such XML element types 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 define two linkages, one implicit linkage by the nested structure and one explicit linkage by the IDREF(S) type attribute. If the two linkages are both one-to-many relationships, the two referred element types by such a referring element type can be considered to be in a many-to-many relationship. For example, the XML document in Figure 4 illustrates a many-to-many relationship.

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.

The linkages from the XML elements in an XML document are identified by the referring element name, linkage name and the referred element name. The algorithm shown in Figure 6 is used to determine Table 4 of the linkages.

Figure 5 illustrates the meanings of the four attributes.

There are eight XML elements in the document and there is only one implicit linkage from them. The values of the above four linkage attributes for such implicit linkage are:

```
<!ELEMENT PURCHASE_ORDER ...>
<!ELEMENT PURCHASE_ORDER_LINE ...>
<!ATTLIST PURCHASE_ORDER
  PO_ID ID #REQUIRED
  ...>
<!ATTLIST PURCHASE_ORDER_LINE
  PO_ID IDREF #REQUIRED
  ...>
<PURCHASE_ORDER PO_ID="PO001" ... />
...<PURCHASE_ORDER_LINE PO_ID="PO001" />
...</PURCHASE_ORDER_LINE
... />
<PURCHASE_ORDER_LINE PO_ID="PO001"
... />
```
Figure 4. A many-to-many cardinality implemented by an element type with two IDREF type attributes

```xml
<!ELEMENT KEYWORD ...>
<!ELEMENT TOPIC ...>
<!ELEMENT MESSAGE ...>
<!ATTLIST KEYWORD
KEYWORD_ID ID #REQUIRED
...>
<!ATTLIST TOPIC
TOPIC_ID ID #REQUIRED
...>
<!ATTLIST MESSAGE
MSG_ID ID #REQUIRED
TOPIC_ID IDREF #REQUIRED
KEYWORD_ID IDREF #REQUIRED
...>
<KEYWORD KEYWORD_ID="KW001" NAME="XML"/>
<KEYWORD KEYWORD_ID="KW002" NAME="DATABASE"/>
...<TOPIC TOPIC_ID="TP001" NAME="Reverse Engineer an XML document"/>
<TOPIC TOPIC_ID="TP002" NAME="Exporting a database as an XML document"/>
...
<MESSAGE MSG_ID="MG001"
TOPIC_ID="TP001"
KEYWORD_ID="KW001">
...</MESSAGE>
<MESSAGE MSG_ID="MG002"
TOPIC_ID="TP002"
KEYWORD_ID="KW002">
...</MESSAGE>
```

Table 4. The attributes and their sources for determining data semantics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxReferring</td>
<td>The maximum number of referred elements referred by a single referring element</td>
<td>Get from Referring Info with key ((RGE, RDE, L))</td>
</tr>
<tr>
<td>MaxReferred</td>
<td>The maximum number of the referring elements that is referring to the same referred element with the same linkage type.</td>
<td>Get from Referred Info with key ((RGE, RDE, L))</td>
</tr>
<tr>
<td>SumReferring</td>
<td>The number of referring elements that possess the linkage.</td>
<td>Get from ReferringInfo with key ((RGE, RDE, L))</td>
</tr>
<tr>
<td>NumberElements</td>
<td>The number of referring elements in the document.</td>
<td>Get from ElementNameCount with key RGE</td>
</tr>
</tbody>
</table>
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 show in Table 6.

The algorithm is composed of 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 4.

The algorithm shown in Figure 6 can determine whether the XML document is valid, in particular whether a non-existing ID value is referred by an IDREF(S) type attribute. If the XML document is valid, three tables can be obtained — ReferringInfo, ReferredInfo, and ElementNameCount. The key for the former two tables is the composite key \((RGE, RDE, L)\), that is, the referring element name, the referred element name and the linkage name, whereas the key for the ElementNameCount is simply the element name. With three such tables, it
is possible to derive the linkage attributes as shown in Table 4.

The complete algorithm is presented in Figure 6 and the following is a list of definitions for the variables to be used:

Once the four attributes of a linkage are determined, the data semantics can be determined by using the matrix shown in Table 6. According to the determined one-to-one and one-to-many relationships, it is then possible to consolidate the related ones into many-to-many and n-ary relationships.

As mentioned above, if an XML element type defines two linkages that are determined to be many-to-one cardinalities, the two referred XML element types are considered to be in a many-to-many relationship. Similarly, if an XML element type defines more than two linkages that are determined to be many-to-one cardinalities, the referred XML element types are considered to be in an n-ary relationship. Therefore, based on the one-to-many cardinalities determined by the previous algorithm, the many-to-many and n-ary relationships can be determined, and the algorithm is shown in Figure 7.

The many-to-one relationship to be considered should be those implemented by explicit linkages; that is, those defined by ID/IDREF(S) linkages. Otherwise, an element type exhibits implicit a one-to-many relationship due to nested structure and defines a many-to-one relationship that will be considered to be a many-to-many relationship, but the two referred elements are actually not related at all.

### PARTICIPATION

Participation concerns whether all instances of a particular element type are involved in a relationship with the corresponding element type.

For implicit referential linkage by a parent-child relation, such as the following DTD `ELEMENT` declaration,

```xml
<!ELEMENT PARENT (CHILD*)>
```

and there are no other `ELEMENT` declarations that define `CHILD` as their child elements, all `CHILD` element instances must appear as the child element of a `PARENT` element, and hence the participation can be considered to be total, as all instances of `CHILD` must be involved in the one-to-many cardinality relation with `PARENT`. If no schema is provided, and if all instances of an element type always appear as the child

---

Table 6. Matrix for determining cardinality & participation based on the determined linkage attributes

<table>
<thead>
<tr>
<th>Participation</th>
<th>Total</th>
<th>Partial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MaxReferring = 1</td>
<td>MaxReferring = 1</td>
</tr>
<tr>
<td>One-to-one</td>
<td>MaxReferring = 1</td>
<td>MaxReferring &lt; NumberElements</td>
</tr>
<tr>
<td></td>
<td>MaxReferred = 1</td>
<td>MaxReferred = 1</td>
</tr>
<tr>
<td></td>
<td>SumReferring = NumberElements</td>
<td>SumReferring &lt; NumberElements</td>
</tr>
<tr>
<td>One-to-many</td>
<td>MaxReferring = 1</td>
<td>MaxReferring = 1</td>
</tr>
<tr>
<td></td>
<td>MaxReferring &gt; 1</td>
<td>MaxReferring &gt; 1</td>
</tr>
<tr>
<td></td>
<td>MaxReferred = 1</td>
<td>MaxReferred = 1</td>
</tr>
<tr>
<td></td>
<td>SumReferring = NumberElements</td>
<td>SumReferring &lt; NumberElements</td>
</tr>
<tr>
<td>Many-to-one</td>
<td>MaxReferring &gt; 1</td>
<td>MaxReferring &gt; 1</td>
</tr>
<tr>
<td></td>
<td>MaxReferring = 1</td>
<td>MaxReferring = 1</td>
</tr>
<tr>
<td></td>
<td>MaxReferred = 1</td>
<td>MaxReferred &lt; NumberElements</td>
</tr>
<tr>
<td></td>
<td>SumReferring = NumberElements</td>
<td>SumReferring &lt; NumberElements</td>
</tr>
<tr>
<td>Many-to-many</td>
<td>MaxReferring &gt; 1</td>
<td>MaxReferring &gt; 1</td>
</tr>
<tr>
<td></td>
<td>MaxReferring &gt; 1</td>
<td>MaxReferring &lt; NumberElements</td>
</tr>
<tr>
<td></td>
<td>MaxReferred &gt; 1</td>
<td>MaxReferred &lt; NumberElements</td>
</tr>
<tr>
<td></td>
<td>SumReferring = NumberElements</td>
<td>SumReferring &lt; NumberElements</td>
</tr>
</tbody>
</table>
Pass One:
Let EID = 1;
Repeat until all XML document elements are read
Let E be the current element to be processed
If ∃ record in TableElementNameCount where ElementName = element name of E
    Get record (ElementName, NumberElement) from TableElementNameCount
    Increment NumberElement by 1;
    Update (ElementName, NumberElement) into TableElementNameCount
Else
    Add (ElementName, 1) into SetElementNameCount
End If
Add (EID, ElementName) into SetElementIDName;
If there exists ID type attribute A of element E with attribute value AV
    Add (AV, ElementName) into SetElementIDName
End If
Increment EID by 1;
Navigate to the next element E in the XML document

Pass Two:
Repeat until all XML document elements are read
Let RGE is the current element to be handled
For each linkage, L, of RGE
    For each linkage value, Lvalue of linkage L of RGE
        Get record (EID, ElementName) from TableElementIDName
        where primary key value is Lvalue
        If no such record exist in TableElementIDName
            XML document is invalid
        Else
            Let RDE = ElementName of the record obtained from TableElementIDName
        End If
    End For
Get record (RGE, RDE, L, Lvalue, ND) from TableRawReferredInfo
for primary key (RGE, RDE, L, Lvalue);
If record exists

Figure 6. The algorithm for determining linkage information by traversing the XML document

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EID</td>
<td>The current element ID. While processing the XML document sequentially, the EID determines the ID to be assigned to individual element encountered.</td>
</tr>
<tr>
<td>E</td>
<td>The current element to be handled.</td>
</tr>
<tr>
<td>A</td>
<td>An attribute of the current element to be handled.</td>
</tr>
<tr>
<td>AV</td>
<td>The attribute value of attribute A.</td>
</tr>
<tr>
<td>L</td>
<td>A linkage of the current element. It can be an implicit linkage with its parent element or an explicit linkage with an IDREF(S) type attribute. For a non-root element without IDREF(S) attribute, the element has only one implicit linkage to its parent element. Otherwise, the element can have more than one linkage, one implicit linkage and at least one explicit linkage.</td>
</tr>
<tr>
<td>Lvalue</td>
<td>The element ID of the linkage L for the current element E. For example, if L is an implicit linkage, Lvalue is the element ID of the parent element of E. Otherwise, Lvalue is the attribute value of IDREF value and the value should be an ID type attribute of an element in the same document.</td>
</tr>
<tr>
<td>NG</td>
<td>The number of referring element of the same element name is referring to the same referred element with the same link.</td>
</tr>
<tr>
<td>RGE</td>
<td>The referring element of a link.</td>
</tr>
<tr>
<td>RDE</td>
<td>The referred element by a link.</td>
</tr>
</tbody>
</table>

Copyright © 2009, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
elements of the same parent element type, the participation is also considered to be total.

For explicit referential linkage by ID/IDREF(S) attributes, if all instances of an element type use the same attribute with values referring instances of the same element type, the relationship is considered to be total participation. Otherwise, the relation is considered to be partial. The DTD of the XML document can only identify the ID/IDREF(S) type attributes but it cannot restrict the referring and referred element types. As such, actually parsing the XML document is required to determine the type of participation.
Aggregation

An aggregation means that the creation of a whole part of an element depends on the existences of its component sub elements. An aggregation relation is signified by the scenario that elements of different types are considered to be a single entity and all constituting elements must exist altogether. An XML document by itself does not provide any facility to enforce such a constraint. At best, the schema can hint at the correlations of the existence of the elements in the corresponding XML document.

For implicit referential linkage by an aggregation relationship, such as the following DTD ELEMENT declaration,

```xml
<!ELEMENT AGGREGATION (COMPONENT1, COMPONENT2, ..., COMPONENTN)+>
```

For example, the following ELEMENT declaration can restrict the existence of the elements enrollment, student, and course.

```xml
<!ELEMENT enrollment (student, course)+>
```

Besides, no student or course elements exist in the document that are not the sub-elements of an enrollment element. For example, if there is another ELEMENT declaration in the same DTD, such as,

```xml
<!ELEMENT student_list (student*)>
```

student elements can exist in the document as the sub-elements of a student_list element. As such, the co-existence relationship of enrollment, student and course elements no longer holds.

Such a 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, all these elements are considered as an aggregation relationship. From another perspective, an aggregation relationship is actually composed of two one-to-one cardinality relations (course – lecturer and course – tutor), which are both total participation.

An exceptional case is that if the sub-elements are actually the attribute of the parent element, such as in example one, it is inappropriate to consider that the involved elements are in an aggregation relationship. As a result, user supervision is needed in the process.

Based on the DTD of the XML document, it is possible to determine the aggregation relationships from the elements. As the requirements of an aggregation relationship is the co-existence of the involved elements and the order of the sub-elements for a parent

---

**Figure 7. The algorithm for determining many-to-many and n-ary relationships**

Get referring XML element types from one-to-many cardinalities;
For each referring XML element T<sub>referring</sub> type
- Get referred XML element types, S<sub>referred</sub> referred by T<sub>referring</sub> via explicit linkages;
  - If the size of the set S<sub>referred</sub> = 2
    - XML element types in S<sub>referred</sub> = many-to-many relationship with T<sub>referring</sub>.
  - Else
    - If size of S<sub>referred</sub> > 2
      - XML element types in S<sub>referred</sub> = n-ary relationship with T<sub>referring</sub>.
Let Set\_temporary = empty;
For each ELEMENT declaration for element E\_parent
    For each child element, element\_child
        If element\_child = mandatory and non-repeatable
            Add an aggregation relation (E\_parent, E\_child) to Set\_temporary;
Let Set\_aggregation and Set\_root = empty;
For each relation R (E\_parent, E\_child) in Set\_temporary
    If (∃ tree, T, in Set\_aggregation) ∧ (E\_parent is a node in T) ∧ (E\_child is not a node in T)
        Add a path E\_parent to E\_child to T;
    Else
        (∃ tree, T, in Set\_aggregation) ∧ (E\_child is a node of T) ∧ (E\_parent is not a node)
        If (E\_child = root node) ∧ (E\_child not in Set\_root of T)
            Add the path E\_parent to E\_child to T;
        Else
            Add E\_child to Set\_root
            Remove the sub-tree starting with E\_child from T;
            If ∃ sub-tree starting with E\_child in multiple nodes
                Add sub-tree to Set\_aggregation;
            Else
                ∃ tree T\_i with a node for E\_parent and T\_j with E\_child as root node;
            Merge trees T\_i and T\_j with a path from node for E\_parent in T\_i to root of T\_j
        Else
            ¬∃ subtree in Set\_aggregation with node for either E\_parent and E\_child;
            Add a new tree with a path E\_parent to E\_child to Set\_aggregation;

Figure 8. The algorithm for determining aggregation relationships

The determination of aggregation relationships is separated into two parts. The first part first discovers the pair of parent and child elements that must co-exist. Once the pairs are determined, the second part of the algorithm treats each pair as a path from parent element to the child element in a sub-tree, and these sub-trees are merged to form a bigger tree. Eventually, the nodes in each tree must co-exist, and they are in aggregation relationship. The second part is straightforward except there is a tricky point that if a child element is found to be a non-root node of a particular sub-tree, it implies that such an element can have more than one parent element, and the aggregation relation that includes such element must start with the parent element.

For example, for a list of ELEMENT declaration in the DTD,

```
<!ELEMENT A (B, C)>
<!ELEMENT B (D)>
<!ELEMENT C (D)>
<!ELEMENT D (E, F)>
```
The determined pairs of raw aggregation relations are (A, B), (A, C), (B, D), (C, D), (D, E) and (D, F). While merging the raw aggregation relations,

![Diagram of raw aggregation relations](image)

(c, D), (D, E) and (D, F)

While adding the path (C, D) to the sub-tree, as D is not a root node, D should be removed from the sub-tree and it is considered to be an individual sub-tree with D as the single node.

![Diagram of merged sub-trees](image)

After the path (D, E) and (D, F) is added to the sub-tree with node D as the root node, two sub-trees are obtained,

![Diagram of final sub-trees](image)

As such, the elements A, B and C, and the elements D, E and F, are considered as being two individual aggregation relationships.

**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 proposed in this paper. For actually drawing the DTD graph, the algorithm proposed by (Shiren, 2001) is used to define the layout of the vertexes on the graph. With such a prototype, a sample XML document with DTD file as shown in Figures 9 is provided to the prototype.

For this case study, both ID/IDREF type attributes are considered and the minimum number of common attributes is one. All elements with at least one attribute are sorted in ascending order of the lengths of their attribute lists. Therefore, the order of the elements to be processed is:

element1, element2, element3

According to the DTD of the XML document, only one ELEMENT declaration is used for constructing the extended DTD graph, as the contents of other element types are EMPTY.

```xml
<!ELEMENT test (element1*,element2*,element3*)>
</test>
```

Therefore, only those explicit one-to-many relationships are to be added to the graph, and the graph will become the one shown in Figure 10 and 11. The detailed derivation of the reverse engineering can be referred to in Shiu (2006).

![XML and DTD](image)

**Figure 9. test.xml and test.dtd**
In order to make use of the XML document, software developers and end users must have a thorough understanding of the contents in the XML document, especially those historical and huge XML documents. Sometimes the schemas of XML documents are missing and the XML documents cannot be opened to be inspected on the screen due to their huge size. Therefore, it is necessary to determine as much information as possible regarding the relationships from the elements in the document.

By reverse engineering the XML document with DTD, all explicit linkages can be determined and the resultant DTD graph can be used to verify the correctness of ID/IDREF(S) linkages, as any incorrect IDREF(S) linkage will be indicated as an extra cardinality and shown in the extended DTD graph. This article provides algorithms to help the users to understand the relationships from the elements by reverse engineering data semantics from the XML document, including:

1. Cardinality relationships
2. Participation relationships
3. n-ary relationships
4. Aggregation relationships
5. Many-to-many relationships (a special case of cardinality relationships)

In summary, to visualize the determined data semantics, a new extended DTD graph is
proposed. XML documents natively support one-to-one, one-to-many and participation, data semantics. With a corresponding schema, such as DTD, the ID and IDREFS attributes of the elements can be identified, and many-to-many, n-ary and aggregation relationships can also be determined.

ACKNOWLEDGMENT

This paper is funded by Strategic Research Grant 7002325 of City University of Hong Kong.

REFERENCES


Sahuguet, A. (2000). Everything you ever wanted to know about DTDs, but were afraid to ask. In Selected Papers of the 3rd International Workshop WebDB-2000, Dallas, TX, USA (LNCS 1997, pp. 171-183).


Semistructured Databases, Very Large Data Bases, Proceedings of the 23rd International Conference on Very Large Data Bases).

Herbert Shiu was the project manager of a funded research project under the Department of Computer Science at City University of Hong Kong. He graduated with a BSc in computer science from the University of Hong Kong in 1992, a MSc, and MPhil in computer science from City University of Hong Kong in 1998 and 2006 respectively. He is now a PhD student at City University of Hong Kong. His current research interests are in database, XML and object-oriented software design and development.

Joseph Fong is currently an associate professor in the Department of Computer Science at City University of Hong Kong. He graduated at University of Sunderland in Computing in 1993. He is a fellow of Hong Kong Computer Society and an editorial board member of International Journal of Web Information Systems. He was the founder chairman of the Hong Kong Computer Society Database Special Interest Group, the annual International Database Workshop, the Sybase Hong Kong User Group, the Hong Kong Web Society, the International Conference on Web-based Learning, and the International Conference on Hybrid Learning. His research interests are in database, data warehousing, data mining XML and hybrid learning. His above 100 publications include SCI Journals, Conferences, Patent (U.S. and Hong Kong), edited books, and text books on Information Systems Reengineering and Integration by Springer Verlag. He teaches data warehousing, data mining, data engineering, database systems and xml technologies. His expertise are in database reengineering and interoperability. He supervised and graduated three PhD students and eight MPhil students.