XML Schema Structural Equivalence

Angela C. Duta, Ken Barker, Reda Alhajj
University of Calgary, Department of Computer Science
2500 University Drive N.W. Calgary, Alberta Canada, T2N 1N4
{*duta, barker, alhajj*}*@cpsc.ucalgary.ca*

November 2004

Abstract

The Xequiv algorithm determines when two XML schemas are equivalent based on their structural organization. It calculates the percentages of schema inclusion in another schema by considering the cardinality of each leaf node and its interconnection to other leaf nodes that are part of a sequence or *choice* structure. Xequiv is based on the Reduction Algorithm [8] that focuses on the leaf nodes and eliminates intermediate levels in the XML tree.

1 Introduction

1.1 Hypothesis and Methodology

Much work has been done in the XML schema equivalence area ([6], [7], [9], [10], [11]) that is applied optimally in only some situations. We propose an approach that finds equivalent XML schemas from the same domain (the same entities and attributes) that have different tree organizations. The difficulty of comparing and finding matchable schemas arises for two reasons: (1) there are three data storage units in XML: elements, attributes, and text content, and (2) the hierarchical features of the XML structure. XML schema equivalence must be evaluated from three perspectives: (1) hierarchical structure (structural equivalence), (2) elements and attributes data types (syntactic equivalence), and (3) elements and attributes names (semantic equivalence).

This paper focuses on determining the structural equivalence of XML schema by using reduced XML trees generated by the Reduction Algorithm (RA) [8]. In the reduced XML trees the three data storage units (element, attribute, and text content) are transformed into a single storage unit: the element node (also called the node). RA eliminates intermediate organizational nodes from each XML schema so that a comparison between them is efficient. The reduced XML schema contains only information about leaf nodes: data types, labels, number of occurrences, and interconnections between them. Our argument for using reduced XML trees is that leaf nodes are the important nodes as they store the data in XML files. Higher level nodes represent a subjective hierarchical organization that allows an intelligible reading of the information stored in leaves. From this perspective our approach is contrary to the assumption "elements at higher levels ... are more relevant than subelement deeply nested" [3] used by some methods [3].

1.2 Contribution

The purpose of this paper is to define a new method for optimizing the schema structure equivalence process that applies to schema trees of similar or different organization. A classification, from the structural perspective of XML schema trees, that can be identified as equivalent is the following:

- similar tree structures that use different data storage units;
- different tree structures that use different order, grouping and/or nesting of subelements within a parent element.

All approaches published to date focus on similar tree structures and do not address schema equivalence for different tree structures. The novelty of our method is to determine structural matching based on the equivalent *leaves content* rather than contexts and vicinities. A *leaf content* is defined by (1) data type and (2) number of minimum and maximum occurrences. Our approach finds equivalent XML schemas in all situations detailed above as long as the minimum information is provided to find a match (labels that are abreviations/synonyms and matching data types).

1.3 Paper Organization

Following the introduction Section 2 details two DTD examples that have different tree structures but refer to the same entities: employees, projects, and tasks. They are equivalent even if other approaches cannot determine it. Section 3 briefly discusses several developed methods for schema equivalence. Our approach is presented starting with Section 4 that summarizes the Reduction Algorithm. It continues with Sections 5, 6 and 7 that detail the Xequiv Algorithm and ends with an example in Section 8. This paper draws some conclusions in Section 9.

2 Motivating Examples

Figures 1 and 2 illustrate two simple examples of DTDs that store data about employees, projects, and tasks for a company. The element data type definitions have not been included. No mechanism has yet appeared in the literature to clearly compare these XML schemas and decide if they are equivalent. This paper presents the Xequiv algorithm that structurally compares differently organized XML trees from the same domain.

```
<!ELEMENT company1 (employee+, project*, task+)>
<!ELEMENT employee (eid | sin, name, (pid* | task_name)+) >
<!ATTLIST employee address CDATA #REQUIRED>
<!ELEMENT project (pid, description, budget | manager | location )>
<!ELEMENT task(task_name, date)>
keys: project.pid, task.task_name
references: employee.pid to project.pid, employee.task_name to task.task_name
```

Figure 1: Repeated employee, project, and task elements

```
<!ELEMENT company2 (employee+)>
<!ELEMENT employee (sin, name, address*, dateOfBirth?, projects?)>
<!ATTLIST employee eid CDATA #REQUIRED>
<!ELEMENT projects (project+)>
<!ELEMENT project (description?, manager | location, task+)>
<!ATTLIST project pid CDATA #REQUIRED>
<!ELEMENT task+ (#PCDATA)>
<!ATTLIST task date CDATA #REQUIRED>
keys: employee.eid, project.pid
references: project.manager to employee.eid
```

Figure 2: Nested structure of employee, project and task elements

3 Related Work

Our work is based on the XML normal form defined by Arenas and Libkin [2]. They define a generalized BCNF for nested XML structures named XNF (XML Normal Form). XNF optimizes two very common problems in XML: update anomalies and redundancy. The algorithm developed to transform any arbitrary DTD into a well-design DTD in XNF is proven to be lossless [2].

Following Salminen and Tompa's suggestion [12] that the canonical forms for XML recommended by W3C [4] must be further researched to solve the XML schema equivalence problem much work has been done in this area. A *generic* schema matching algorithm called Cupid is presented [10] that can be applied to XML, relational, or other schema types. Cupid focuses on the leaf nodes and is based on automatic linguistic matching (elements' name) and structural matching (schema structure, path matching, constraints, and element data types).

Lee *et al.* [9] propose a method for scalable integration of DTDs. The method is based on two steps (1) finding similar DTDs and (2) generating DTD clusters. The similarity between two DTDs is evaluated from three perspectives: (1) semantic similarity (similarity between node labels, constraints and path context (ascendants)), (2) immediate descendant similarity, and (3) leaf context similarity. Ontology similarity [9] (a component of semantic similarity) is based on finding similar labels or synonyms (using WordNet Java API [1], [5]) and the depth (number of similar subsequent characters) for abbreviations (such as emp for employee). Constraints such as +, *, ?, or none are given weights of similarity. The authors have conducted several experiments using more than 150 DTDs, based on them, their approach has better performance than others. This work is similar to ours in that it addresses some DTD transformation rules also adopted by us.

The problem of clustering XML documents based on their structural similarity is also addressed by Nierman and Jagadish [11]. A collection of documents with DTD's from the same domain is divided into smaller sets of very similar DTDs based on the edit distances. The edit distance is calculated using dynamic programming as the minimum cost to transform a tree A into B through operations such as relabel node, insert node, delete node, insert tree, and delete tree. Clusters are formed for DTDs where the edit distance is minimum. The major drawback of this approach is that its starting point is a cluster of similar DTDs generated by other methods. This method works for documents with DTDs having the same tree structure but it cannot be applied to trees that have a significant different structure even though they refer to the same domain.

COMA [7] is a system that combines several simple and hybrid matching algorithms. The simple algorithms refer to one aspect in DTD: labels, data types, or user input. Hybrid matchers focus either at the element level (i.e. Name and TypeName) or include the structural organization (i.e. NamePath, Children, and Leaves). A novelty is reusing the previous match results which proved to be efficient in some cases but missed some matchings in other cases. Our approach extends the structural matchers Children and Leaves by combining and generalizing them to any type of node (repeated, optional, alternative options, key, reference, *etc.*)

An evaluation of the most recent approaches in schema matching for relational or XML models is available [6]. The variety in the studied matching tools given by algorithms, experiments, quality metrics and pre-match effort make it difficult to decide how effective they are on complex schemas used in real life.

4 The Reduction Algorithm (RA)

RA addresses two of the XML schema features that generate difficulties in determining schema equivalence: (1) multiple data storage units and (2) hierarchical organization. An XML element stores data in a text unit, attributes, and/or subelements. Each data unit is represented by a node. Thus, we easily distinguish between an empty element and a text-only element (element types used accordingly with the W3C standard [13]) because the first has an element node and several attribute subnodes as data units, while the second uses an element node, a text subnode and several attribute subnodes (for more details refer to [8]). RA is based on six rules that convert the node types of the source structure (element, attribute, text) into a single node type (element) and eliminates intermediate tree levels.

Rule 1 A text node from a text-only element is transformed into an attribute node by borrowing the label of the parent element node. ■

Rule 2 An attribute node is transformed into an element node and a text node. ■

Rule 3 All text nodes are eliminated. ■

Rule 4 The reduction of a non-repeated element node with non-repeated subnodes requires its optional operator (if any) be transferred (**Rule 4.1**) individually to child nodes if all subnodes are optional,or (**Rule 4.2**) to the structure that gathers all its child nodes if at least one subnode is *required.* ■

Rule 5 The reduction of a non-repeated element with repeated subelements requires its optional operator (if any) be transferred (**Rule 5.1**) individually to child nodes if all subnodes are optional, or (**Rule 5.2**) to the structure that gathers all its child nodes if at least one subnode is *required.*

■

Rule 6 The reduction of a repeated element with non-repeated subelements requires its cardinality be transferred (**Rule 6.1**) individually to child nodes if all subnodes are optional, or (**Rule 6.2**) to the structure that gathers all its child nodes if at least one subnode is *required.* ■

Rule 7 The reduction of a repeated element with repeated subelements requires its cardinality be transferred (**Rule 7.1**) individually to child nodes if all of them are optional, or (**Rule 7.2**) to the sequence that gathers all its subnodes if at least one subnode is *required.* ■

The bottom-up Reduction Algorithm contains three parts (see Figure 3). The first part prepares the XML schema for reduction by normalizing it and modifying it to use only one node type: the element. The normalization process is based on the algorithm developed by Arenas and Libkin [2] that obtains an XNF normal form for XML following the "standard treatment" of BCNF. XNF refers to a well-designed schema with non-redundant information that prevents update anomalies in the XML data file. RA next eliminates text and attribute nodes from the structure by transforming them into element nodes through Rules 1, 2, and 3. The second part of the algorithm reduces nodes by using Rules 4, 5, 6, and 7 and preserves the initial constraints of the XML schema. The result of Part 2 is an XML reduced tree using sequences (XRTS). Part 3 transfers the outer expression operators (?, +, *) to inner elements. The resulted tree is an XML reduced tree to leaf nodes (XRTN). Part 3 generates some information loss regarding element occurrences restricted to occurrences of other elements but allows a fast first evaluation of schemas similarity.

1. `Part 1:  Prepare the XML schema for reduction`
   (a) `Convert the XML schema into an XNF using Arenas and Libkin's algorithm [2] (normalization).`
   (b) `Represent the XML schema using the DTD nested notation.`
   (c) `Apply Rule 1, 2, and 3 in this order to transform text and attribute nodes into element nodes.`
2. `Part 2:  Create the XRTS tree by applying Rules 4, 5, 6, and 7 according to each situation.`
3. `Part 3:  Create XRTN tree by transferring the sequence multiplicator (+, *) or optional indicator ?  to each node in the sequence.`

Figure 3: The Reduction Algorithm

5 The XML Schema Equivalence Xequiv Algorithm

The purpose of Xequiv is to find XML schemas that are similar in terms of leaf content (see Figure 4). To compare leaf contents the source schemas must be reduced using RA so that the intermediate nodes are eliminated. Xequiv focuses only on the nodes that store data and compares leaf nodes which are of only one type: element nodes. A *leaf content* shows how much data is stored in the XML data file based on the node definition.

$$leaf\ content = < data\ type, leaf\ cardinality(minOccurences..maxOccurences) >$$

For example, the structure $tasks(task\_name?)$ with $task\_name$ of type *string* has the leaf content $task\_name_{content} =< string, 0..1 >$. The node task_name is a perfect match with the node *job* that has the leaf content $job_{content} =< string, 0..1 >$ and only $\alpha\%$ similar (see Table 1) with the *job_name* node where $job\_name_{content} =< string, 1..1 >$.

1. Reduce schemas using RA and create XRTS1, and XRTS2 in Part 2, and XRTN1 and XRTN2 in Part 3 from the source schemas XSD1 and XSD2, respectively.

2. Determine the equivalence values $Sim_{XRTN1 \rightarrow XRTN2}$ and $Sim_{XRTN2 \rightarrow XRTN1}$. If they are greater than a predefined threshold, then they are somewhat equivalent and we proceed to Step 3 to determine the structural equivalence.

3. Compare XRTS1 and XRTS2 by finding a match for each structure (sequence, *all* or *choice*). A node part of a structure or a series of nested structures in XRTS1 is equivalent to a node in XRTS2 which is part of similar series of nested structures.

Figure 4: The Xequiv Algorithm

We assume that the schema definition used allows a variety of data types and cardinality values. For this reason we recommend XML Schema. In this paper we use DTD only to present schemas[1] in a compact way but our algorithm is based on XML Schema as it has a larger variety of data types and features (e.g. primary and foreign keys for attributes and elements).

Two nodes are selected for comparison based on their data types and labels. XML data types can be classified into several compatible classes such as: (1) string: string, normalizedString, token, (2) numerical: long, int, short, byte, float, decimal, *etc.*, (3) date: time, date, datetime, *etc.*

[1] As a result we take a few liberties with DTD nomenclature in our examples and we will indicate where they occur.

Equivalent nodes must have similar data types from a single class and matchable labels. Labels are matchable if they refer to the same concept either using the same words, abbreviations, or synonyms. The WordNet thesaurus offered by the Cognitive Science Laboratory [5] and acknowledged by MIT Press [1] retrieves efficiently the set of synonyms for any label. We consider that synonyms and abbreviations are 100% equivalent as we determine structural and not ontology equivalence. From this perspective our approach is contrary to XClust developed by Lee *et al.* [9] that determines levels of equivalence for abbreviations. We use the information provided by labels to select candidate nodes. The schema equivalence using our method identifies schemas that are from the same domain and similar in leaves content. Thus, finding similar node labels is just an instrument to reach our goal but not our purpose. This is accomplished by the function Θ that determines if two nodes have a similar label and their data types are from compatible classes. Consider node N_1 defined by $(type\ T_1, label\ L_1)$ from XRT1[2] and node $N_2(T_2, L_2)$ from XRT2.

$$\Theta(N_1, N_2) = \begin{cases} 1 & \text{, if } T_1 \equiv T_2 \text{ and } L_1 \equiv L_2 \\ 0 & \text{, otherwise} \end{cases} \tag{1}$$

6 Nodes Similarity

6.1 Similarity Metric for Simple Nodes

To determine if two schemas are structurally equivalent, Xequiv first evaluates their leaf nodes similarity. This provides a fast evaluation that separates schemas into different domains. Using the reduced schemas obtained at Part 3, Xequiv identifies for each node a matching node and determines the measure of inclusion between them. Consider the structures $str1 : (a)$ and $str2 : (a+)$. Structure $str1$ requires node a to appear exactly one time, while $str2$ requires node a to occur several times but at least once in the corresponding XML file. We consider that $str1$ is included in structure $str2$ because $a \subset a+?$, and, thus, $R \subset +$. Also, $R \subset ?$ because ? admits two statuses: present one time (like R) or non-present. The operator $? \subset *$ as $*$ allows in addition the node to occur multiple times. Similarly, inclusion hierarchies are determined between all operators (see Figure 5).

$$\boxed{\begin{array}{c} R \subset ? \subset * \\ R \subset + \subset * \end{array}}$$

Figure 5: Operators inclusion

The inclusion of a structure $str1$ into a structure $str2$ is based on the inclusion of each node

[2]XRT is a general term that refers to the XML reduced tree, either XRTS or XRTN

from $str1$ into a single node in $str2$. The inclusion $\varepsilon_{x \to y}$ of a node x from $str1$ into another node y from $str2$ if $\Theta(x, y) = 1$ is based on the inclusion of their expression operators (see Figure 5). For nodes with the same operator the inclusion measure $\varepsilon = 1$. If the operator of the x node is included in the operator of the node y then $\varepsilon_{x \to y} = 1$. Otherwise, the node x is included in y with a lower percentage (see Table 1). Values of α, β, γ, δ, ϵ, ρ, and σ represent the inclusion percentage, and $0 <= \alpha$, β, γ, δ, ϵ, ρ, and $\sigma < 1$. It is very important how these values are set as they are directly correlated with the minimum threshold set for schema equivalence. Note that Table 1 is asymmetrical as the node $a? \subset a*$ and, thus, $\varepsilon_{a? \to a*} = 1$ but $\varepsilon_{a* \to a?} < 1$. If a node x from $str1$ does not have a correspondent in $str2$ then the inclusion factor is 0.

$\varepsilon_{x \to y}$	R	?	+	*	ϕ
R	1	1	1	1	0
?	α	1	β	1	0
+	γ	δ	1	1	0
*	ϵ	ρ	σ	1	0

Table 1: Operators inclusion percentages $\varepsilon_{x \to y}$ (ϕ = non-existent node)

We define the similarity function for an XML reduced schema XRTN1 with $n1$ nodes to another schema XRTN2 with $n2$ elements based on the nodes inclusion. We assume that for each node x from XRTN1 there exists at most one node y in XRTN2 such that $\Theta(x, y) = 1$.

$$Sim_{XRTN1 \to XRTN2} = \frac{\sum_{i=1}^{n1} \varepsilon_{x_i \to y_j}}{n1} * 100\%, 1 \leq j \leq n2 \qquad (2)$$

The similarity Sim is an asymmetrical function. $Sim_{XRTN1 \to XRTN2}$ expresses how much of the structure of XRTN1 is included in XRTN2. Note that $Sim_{XRTN1 \to XRTN2}$ is different from $Sim_{XRTN2 \to XRTN1}$ if (1) there are nodes in one structure that do not have a match in the other structure, and (2) different operators are used for nodes with $\Theta = 1$.

6.2 Similarity Metric for Choice Nodes

The Sim metric considers each node and its match. The values provided by Table 1 work for simple nodes but not for nodes formed by several alternatives (the *choice* nodes or structures). The *choice* structure is formed by several mutually exclusive nodes. The metric $SimChoice$ must evaluate the similarity of two *choice* nodes by considering the number of alternatives and also the similarity between a *choice* node and a simple node. The equivalence metric must be no more than 1 (like nodes inclusion) and differentiate between different number of alternatives. We consider each situation below.

6.2.1 Similarity between two *choice* structures

Consider the node x formed by several alternative nodes $x = (x_1|..|x_m)$ in XRTN1. To evaluate how similar is node x from XRTN1 to a *choice* node y in XRTN2 $y = (y_1|..|y_n)$ we assume that alternatives are ordered in both nodes such that $\Theta(x_1, y_1) = \Theta(x_2, y_2) = .. = 1$.

$$SimChoice_{x \to y} = \frac{\sum_{i=1}^{m} \varepsilon_{x_i \to y_i}}{m} \tag{3}$$

If $\exists k$ such that $\Theta(x_k, y_k) = 0$, then $\varepsilon_{x_k \to y_k} = 0$. If $n < m$, then there are alternatives in x with no correspondent in y and $SimChoice_{x \to y} < 1$. $SimChoice_{x \to y} = 1$ if each alternative from x has a correspondent in y with the same or a more general expression operator.

6.2.2 Similarity between a *choice* and multiple simple nodes

Consider the *choice* node $(x|y)$ in XRTN1 and the sequence (x, y) in XRTN2. $(x|y)$ represents a single node so it must be similar to one node only from XRTN2. But as both alternatives x and y from XRTN1 have a correspondent in XRTN2 the one that maximizes the similarity function based on cardinality matching must be chosen.

$$SimChoice_{(x|y) \to (x,y)} = Max(SimChoice_{(x|y) \to x}, SimChoice_{(x|y) \to y}) \tag{4}$$

Thus, $SimChoice_{(x|y) \to (x,y)} = Max(\frac{\varepsilon_x}{2}, \frac{\varepsilon_y}{2})$.

Conversely, $SimChoice_{(x,y) \to (x|y)}$ must be evaluated using the similarity metric for each simple node $SimChoice_{x \to (x|y)}$ and $SimChoice_{y \to (x|y)}$.

$$SimChoice_{(x,y) \to (x|y)} = Max(SimChoice_{x \to (x|y)}, SimChoice_{y \to (x|y)}) \tag{5}$$

Thus, $SimChoice_{(x,y) \to (x|y)} = Max(\varepsilon_x, \varepsilon_y)$.

6.2.3 Similarity between one *choice* structure and multiple *choice* structures

Consider the alternative structure $(x|y|z|t)$ in XRTN1 and two alternative structures $(x|y)$ and $(z|t)$ in XRTN2. In XRTN1 there is a single node with four alternatives and it has two corresponding nodes in XRTN2 each with two alternatives. A single node from XRTN2 must correspond to a node from XRTN1 so the one that maximizes the similarity function must be chosen.

$$SimChoice_{(x|y|z|t)\to((x|y),(z|t))} = Max(SimChoice_{(x|y|z|t)\to(x|y)}, SimChoice_{(x|y|z|t)\to(z|t)}) \tag{6}$$

Thus, $SimChoice_{(x|y|z|t)\to((x|y),(z|t))} = Max(\frac{\varepsilon_x+\varepsilon_y}{4}, \frac{\varepsilon_z+\varepsilon_t}{4})$

In summary, a *choice* node $x(x_1,...x_m)$ from XRTN1 is equivalent to at most one node in XRTN2. If the alternatives from x exist in XRTN2 either as simple nodes $y_1,..y_k$ or as alternatives that are grouped in several *choice* nodes $y_{k+1},..y_n$ the similarity measure chooses the node y that is the "most" equivalent to x.

$$SimChoice_{x\to(y_1,..y_n)} = Max(SimChoice_{x\to y_1}, ..SimChoice_{x\to y_n}) = \frac{Max_{i=1}^{n}(\varepsilon_{y_i})}{m} \tag{7}$$

Thus, the similarity value between a schema XRTN1 with $n1$ nodes and a schema XRTN2 formed by $n2$ nodes is:

$$Sim_{XRTN1\to XRTN2} = \frac{\sum_{i=1}^{n1} Sim_{x_i\to y_j}}{n1} * 100\%, 1 \le j \le n2 \tag{8}$$

The equivalence value Sim for a node x for which there exists at least a node y in XRTN2 such that $\Theta(x,y)=1$ is defined as follows:

$$Sim_{x\to y} = \begin{cases} \varepsilon_{x\to y} & \text{, if } x \text{ and } y \text{ are simple nodes} \\ SimChoice_{x\to y} * \varepsilon_{x\to y} & \text{, if } x \text{ or } y \text{ is a } \textit{choice} \text{ node} \end{cases} \tag{9}$$

The value $\varepsilon_{x\to y}$ for *choice* nodes determines the equivalence between the operators applied to the *choice* structures making the difference, for example, between $(x_1|x_2)$ and $(y_1|y_2)+$. Note that $\varepsilon_{x\to y}$ for XRTN schemas is always 1 as there is no outer operator for *choice* structures. The *choice* operator is combined with alternative nodes' operators in Part 3 of RA as described in Section 5.

7 Structural Similarity

7.1 Structural Similarity Metric

The similarity Sim is calculated based on the reduced structures obtain in Part 3 of the RA. However it is important how nodes are grouped in sequences in the reduced schema. A more exact way to determine structural equivalence must consider the cardinality of each node and its correlation to other nodes as part of a sequence. Thus, based on reduced schemas obtained in Part 2 of RA we compute similarity $SimStr$ of each structure from XRTS1 with a structure from XRTS2 that contains the corresponding nodes. $SimStr$ determines how similar a structure $str1$ (sequence or *choice*) is to a structure $str2$ considering the nodes' cardinality, structure cardinality, and number of nodes in the structure $str1$.

$$SimStr_{str1\rightarrow str2} = \begin{cases} \frac{\sum_{i=1}^{m1} SimStr_{str1i\rightarrow str2j}}{m1} * \varepsilon_{str1\rightarrow str2} * 100\% & \text{, if } str1 \text{ or } str2 \text{ are sequences} \\ Sim_{str1\rightarrow str2} & \text{, otherwise} \end{cases} \tag{10}$$

The value $m1$ represents the number of inner structures (sequences, *choices*, or simple nodes) $str1i$ in structure $str1$ such that $str1i \bigcap str1j = \emptyset$ for any $1 \leq i, j \leq m1$, $i \neq j$. If a node from $str1$ has a correspondent in $str2$, then: first the equivalence of nodes is evaluated based on their cardinality, and second it is multiplied by the equivalence of structures cardinality. The value $\varepsilon_{str1\rightarrow str2}$ determines the equivalence between structures cardinality. A structure, for example $str1$, can be represented by a single node. In this case, $SimStr$ evaluates the similarity of this node with a node from $str2$. The *required* operator is implied whenever there is no other operator for a node or a structure. If both structures $str1$ and $str2$ are simple nodes the similarity value for them is depicted in Table 1. If one node is a *choice* structure the formula of $SimChoice$ is used: $SimStr_{str1\rightarrow str2} = Sim_{str1\rightarrow str2} = SimChoice_{str1\rightarrow str2} * \varepsilon_{str1\rightarrow str2}$. Note that in this case $\varepsilon_{str1\rightarrow str2}$ can be different than 1 if different operators are associated with the *choice* structures.

$SimStr$ values are interpreted as follows. If $SimStr_{str1\rightarrow str2} = 100\%$ and $SimStr_{str2\rightarrow str1} = 80\%$, then it means that structure $str1$ is included in $str2$. $Str2$ has either (1) additional nodes, sequences, or *choices*; (2) additional alternatives in its *choice* nodes; or (3) more general operators for nodes[3].

For example, consider the structures defined in Figure 6. In example (a) structures $str1$ and $str2$ are sequences, with $str1$ containing two nodes: $a+$ and b, and $str2$ having three nodes a, b, and c. The similarity value for them is calculated as follows:

$$SimStr_{(a)str1\rightarrow str2} = \frac{SimStr_{(a)str11\rightarrow str21} + SimStr_{(a)str12\rightarrow str22}}{2} * \varepsilon_{+\rightarrow +} * 100\%$$

[3]The most general operator is *; the operator + is more general than R but not than ? and *; the optional operator ? is more general than R (see Figure **??**)

$$SimStr_{(a)str1 \rightarrow str2} = \frac{\varepsilon_{a+ \rightarrow a} + \varepsilon_{b \rightarrow b}}{2} * \varepsilon_{+ \rightarrow +} * 100\%$$

Using the example values from Table 2 for nodes inclusion, node $a+$ from $str1$ is 50% equivalent with node a from $str2$, and nodes b are 100% equivalent. Thus, $SimStr_{(a)str1 \rightarrow str2} = 75\%$. This means that 75% of the structure $str1$ is included in the structure $str2$. To determine the inclusion of structure $str2$ in $str1$ we calculate $SimStr_{(a)str2 \rightarrow str1}$.

$$SimStr_{(a)str2 \rightarrow str1} = \frac{SimStr_{(a)str21 \rightarrow str11} + SimStr_{(a)str22 \rightarrow str12} + SimStr_{(a)str23 \rightarrow str13}}{3} * \varepsilon_{+ \rightarrow +}$$

Structure $str13$ does not exists, so $SimStr_{(a)str23 \rightarrow str13} = 0$. Thus, $SimStr_{(a)str2 \rightarrow str1} = \frac{1+1+0}{3} * 1 * 100\% = 66.67\%$. This means that 66.67% of structure $str2$ is found in structure $str1$.

$\varepsilon_{x \rightarrow y}$	R	?	+	*	ϕ
R	1	1	1	1	0
?	0.5	1	0.4	1	0
+	0.5	0.2	1	1	0
*	0.4	0.5	0.9	1	0

Table 2: Example of nodes equivalence (ϕ = non-existent node)

In example (b) from Figure 6, the structure $str1$ contains two substructures: a sequence $str11$ made of two nodes and $str12$ made of one node. Similarly, the structure $str2$ has a sequence and a node. The similarity value is calculated for one sequence at a time.

$$SimStr_{(b)str1 \rightarrow str2} = \frac{SimStr_{(b)str11 \rightarrow str21} + SimStr_{(b)str12 \rightarrow str22}}{2} * \varepsilon_{+ \rightarrow +} * 100\%$$

Thus, $SimStr_{(b)str1 \rightarrow str2} = \frac{(\varepsilon_{a+ \rightarrow a} + \varepsilon_{b \rightarrow b})/2 * \varepsilon_{+ \rightarrow +} + \varepsilon_{c \rightarrow c}}{2} * \varepsilon_{+ \rightarrow +} * 100\% = 62.50\%$, states that 62.50% of $str1$ is included in $str2$. $SimStr_{(b)str2 \rightarrow str1}$ is computed similarly and is equal to 100%. Both structures $str1$ and $str2$ have the same number of nodes and for each node in one structure there is an equivalent node in the other. The difference in the similarity values is given by the expression operators making $str1$ a more general structure than $str2$.

In example (c) from Figure 6, both structures are formed by three nodes but grouped differently in sequences. In $str2$ the nodes a and b are grouped in a repeatable sequence. In $str1$ the nodes $a+$ and b are not separated by c but it can be considered that there is a *required* sequence

(a)	$str1 : (a+, b)+$	$str11 : a+, str12 : b$
	$str2 : (a, b, c)+$	$str21 : a, str22 : b, str23 : c$
(b)	$str1 : ((a+, b)+, c+)+$	$str11 : (a+, b)+, str12 : c+$
	$str2 : ((a, b)+, c)+$	$str21 : (a, b)+, str22 : c$
(c)	$str1 : (a+, b, c)$	$str11 : (a, b), str12 : c$
	$str2 : ((a, b)+, c)*$	$str21 : (a, b)+, str22 : c$

Figure 6: Determining sequence equivalence for multiple sequences

that groups them in $str1 : ((a+, b), c)$. This gives the advantage of comparing the two sequences containing the nodes a and b and give a better similarity value between $str1$ and $str2$. Conversely, if $str1$ is compared to the structure (a, b, c) the nodes $a+$ and b must not be grouped separately as both structures have only a simple sequence.

$$SimStr_{(c)str1 \to str2} = ((\varepsilon_{a+\to a} + \varepsilon_{b\to b})/2 * \varepsilon_{R\to +} + \varepsilon_{c\to c})/2 * \varepsilon_{R\to *} * 100\% = 83.33\%$$

$$SimStr_{(c)str2 \to str1} = ((\varepsilon_{a\to a+} + \varepsilon_{b\to b})/2 * \varepsilon_{+\to R} + \varepsilon_{c\to c})/2 * \varepsilon_{*\to R} * 100\% = 30\%$$

$Str1$ is a flat structure compared to $str2$ which contains other nested structures. $Str1$ is found in $str2$ in terms of 83.33%, while $str2$ is found in $str1$ in terms of 30%. They both contain the same nodes but are grouped differently. The difference in percentages is generated by (1) the nested sequence $(a, b)+$ compared with $a+, b$, and (2) the * operator.

The order in which nodes appear in a structure is unimportant in determining if it is equivalent to another structure. Thus, *sequence* and *all* are considered to be equivalent and the equivalence measure for *all* structures is computed using the formula defined in Equation 10.

7.2 Xequiv applied to nested and non-nested structures

Consider the examples from Figure 7. The connection between employees who work on projects is preserved either using references (examples (a) and (c)), either through a nested structure Figure 7(d). The example from Figure 7(b) provides some connection between employees and projects but without checking the foreign key integrity. Are they all equivalent? There is no mechanism in the literature to clearly compare them and determine their equivalence. This section is dedicated to solving this problem.

If examples (a) and (d) from Figure 7 are compared, the equivalence algorithms find the node employee.pid is an extra node in example (a), thus reducing the equivalence measure of the

(a)	<!ELEMENT company (employee, project)>
	<!ELEMENT employee (eid, pid, name)>
	<!ELEMENT project (pid, description)>
	eid and project.pid primary keys, employee.pid is keyref to project.pid
(b)	<!ELEMENT company (employee, project)>
	<!ELEMENT employee (eid, pid, name)>
	<!ELEMENT project (pid, description)>
(c)	<!ELEMENT company (employee, project)>
	<!ELEMENT employee (eid, name)>
	<!ELEMENT project (pid, eid, description)>
	employee.eid and pid are primary keys, project.eid is keyref to employee.eid
(d)	<!ELEMENT company (employee)>
	<!ELEMENT employee (eid, project, name)>
	<!ELEMENT project (pid, description)>
	eid and project.pid primary keys

Figure 7: Simple possible equivalent schemas

two structures. Since a corresponding node to employee.pid is not necessary in the former structure to link an employee to a project as this is done by the nested feature we have two options to remedy this drawback. The first option is to eliminate nodes which represent references such as employee.pid. Unfortunately, this wrongly determines structures (a) and (b) that have the same nodes to be 100% equivalent even though (b) is missing an important reference. Another option is to add fake references in nested structures in the preparation part of the RA. For example, in the structure (d) we could add either pid in the employee node or eid in the project node, thereby generating two alternative structures that have different equivalence measures to structure (a). As we do not know if structure (d) is compared to (a) or (c), we must add a reference node that will determine the same similarity value between (a) and (d) as between (c) and (d). Thus, we define an additional reduction rule that takes care of references.

Rule 8 The connection between a structure S2 with the primary key KEY2 nested inside another structure S1 with the primary key KEY1 is preserved by adding a *choice* reference structure formed by primary keys KEY1||KEY2 inside the nested structure. ■

Rule 8 is contained in the preparation part of the RA and is applied after Rule 3 when sequences are still nested. References are included in the inner structure to borrow its operator, thereby to preserve the cardinality of the nested structure. If the outer structure S1 contains only S2 and no additional elements but is part of a structure S0 with primary key KEY0, then the *choice* structure KEY0||KEY2 is added inside S2.

Referring to example (d) from Figure 7, the reference node is formed by (eidREF||pidREF) as eid and pid are primary keys.

$$company(employee(eid, project(pid, description), name)$$
$$\overset{R8}{\Rightarrow} company(employee(eid, (eidREF||pidREF, pid, description), name)$$

We represent the *choice* structure for references using a double line || as this is evaluated differently from a regular alternative construction. Only one element of the alternative structure for references is going to be found (if any) in the other schema. Thus, contrary to *SimChoice* previously defined, that must determine how many alternative options from one schema are found in the other schema, *SimChoiceRef* must evaluate if there is any corresponding reference (see Equation 11). Thus, only one $\varepsilon_{x_i \to y_j}$ is greater than zero from the components of the maximum function, where x_i and y_j are reference alternatives from XRTS1 and XRTS2, respectively.

$$SimChoiceRef = Max(\varepsilon_{x_i \to y_j}) * \varepsilon_{REF} * 100\% \tag{11}$$

A correct equivalence evaluation must also consider the existence of primary keys. If XRTS3 is defined as Employee(eid), with eid primary key and XRTS4 as Employee(eid), they are not 100% equivalent. Thus, we revise the similarity formula (Equation 10) to multiply the node equivalence to the key equivalence for primary keys. For example if eid is a primary key its equivalence is the product $\varepsilon_{eid} * \varepsilon_{KEY}$, where $\varepsilon_{KEY} = 1$ if both nodes are primary keys, and < 1 if only one of them is a primary key.

The examples from Figure 7, are reduced to the structures detailed in Figure 8 with the KEY suffix for primary keys and REF for references. By defining $\varepsilon_{KEY} = 0.7$ and $\varepsilon_{REF} = 0.6$ and using the operators equivalence defined in Table 2, schema (a) is similar to the the rest of the schemas in the proportions presented in Figure **??**, where x represents a generic schema and ε_{eid} is the short form for $\varepsilon_{(a)eid \to (x)eid}$.

(a)	company(eidKEY, pidREF, name, pidKEY, description)
(b)	company(eid, pid, name, pid, description)
(c)	company(eidKEY, name, pidKEY, eidREF, description)
(d)	company (eidKEY, eidREF||pidREF, pid, description, name)

Figure 8: Reduced schemas

$$SimStr_{(a)\to(x)} = (\varepsilon_{eid} * \varepsilon_{eidKEY} + \varepsilon_{pid} * \varepsilon_{pidREF} + \varepsilon_{name} + \varepsilon_{pid} * \varepsilon_{pidKEY} + \varepsilon_{description})/5 * 100\%$$
$$SimStr_{(a)\to(b)} = (0.7 + 0.6 + 1 + 0.7 + 1)/5 * 100\% = 80\%$$
$$SimStr_{(a)\to(c)} = (1 + 0 + 1 + 1 + 1)/5 * 100\% = 80\%$$
$$SimStr_{(a)\to(d)} = (1 + 1 + 1 + 1 + 1)/5 * 100\% = 100\%$$

Figure 9: Structural similarity of reduced schemas

8 Example

8.1 Schema reduction using the Reduction Algorithm

Consider the two examples depicted in Figures 1 and 2. We want to determine how equivalent they are and how much of schema from Figure 1 (called XSD1) is found in the schema from Figure 2 (called XSD2). We start by applying the Reduction Algorithm to XSD1 and XSD2. We exemplify using XSD1; the reduction of XSD2 is found elsewhere [8]. First, XSD1 is translated using the nested DTD representation.

$$\overset{R2}{\Rightarrow} company1_E(employee_E + (eid_E(eid_T)|sin_E(sin_T), name_E(name_T), address_A, pid_E * (pid_T)|task\_name_E, (task\_name_T)), project_E * (pid_E(pid_T), description_E (description_T), budget_E(budget_T)|manager_E(manager_T)|location_E(location_T)), task_E + (task\_name_E(task\_name_T), date_E(date_T)))$$

Rule 2 is applied to transform the attribute node $address_A$ in an element and text node and next Rule 3 eliminates text nodes. All the nodes are element nodes, and as there is no fear for confusion and we drop the suffix E. We use suffixes KEY and REF for key and reference nodes.

$$\overset{R2,R3}{\Rightarrow} company1(employee+(eid|sin, name, address, (pidREF*|task\_nameREF)+), project* (pidKEY, description, budget|manager|location), task + (task\_nameKEY, date))$$

Part 2 of the Reduction Algorithm ends by applying Rules 6 and 7 from bottom-up to reduce subelement-only elements and generate XRTS1.

XRTS1: $\overset{R6,R7}{\Rightarrow} company1((eid|sin, name, address, (pidREF * |task\_nameREF)+)+,$

$(pidKEY, description, budget|manager|location)*, (task\_nameKEY, date)+)$

Part 3 continues reduction by transferring the multiplicators + and * inside structures. Thus, XRTN1 is: $company1(eid+|sin+, name+, address+, pidREF*|task\_nameREF+, pidKEY*, description*, budget*|manager*|location*, task\_nameKEY+, date+)$

XSD2 is presented below after Rule 3 was applied (see [8]).

XSD2: $company2(employee + (eidKEY, sin, name, address*, dateOfBirth?, projects?(project + (pidKEY, description?, manager|location, task + (task, date)))))$

Rule 8 is applied to preserve connections between nested structures. There is no primary key for the nested structure *task* so no reference is created.

$\overset{R8}{\Rightarrow} company2(employee + (eidKEY, sin, name, address*, dateOfBirth?, projects?(project + (pidKEY, description?, manager|location, eidREF||pidREF, task + (task, date)))))$

XSD2 becomes XRTS2 at the end of Part 2, and XRTN2 at the end of Part 3 of RA.

XRTS2: $company2(eidKEY, sin, name, address*, dateOfBirth?, (pidKEY, description?, manger|location, eidREF||pidREF, (task, date)+)*)+$

XRTN2: $company2(eidKEY+, sin+, name+, address*, dateOfBirth*, pidKEY*, description*, manager*|location*, eidREF*||pidREF*, task*, date*)$

8.2 Computing equivalence measures

We start by comparing XTRN trees to determine if they are from the same domain. Consider the operators equivalence detailed in Table 2. We determine the node's similarity from the two schemas using Equation 2.

$Sim_{XRTN1 \to XRTN2} = (\frac{Max(\varepsilon_{eid+\to+}, \varepsilon_{sin+\to+})}{2} + \varepsilon_{name+\to+} + \varepsilon_{address+\to*} + (\varepsilon_{pid*\to*} + \varepsilon_{task\_name+\to task*})/2 + \varepsilon_{pid*\to*} + \varepsilon_{description*\to*} + (\varepsilon_{budget*\to\phi} + \varepsilon_{manager*\to*} + \varepsilon_{location*\to*})/3 + \varepsilon_{task\_name+\to task*} + \varepsilon_{date+\to*})/9 * 100\% = 96.20\%$

$Sim_{XRTN2 \to XRTN1} = (Max(\varepsilon_{eid+\to+}, \varepsilon_{sin+\to+}) + \varepsilon_{name+\to+} + \varepsilon_{address*\to+} + \varepsilon_{dateOfBirth*\to\phi} + \varepsilon_{pid*\to*} + \varepsilon_{description*\to*} + (\varepsilon_{manager*\to*} + \varepsilon_{location*\to*})/2 + Max(\varepsilon_{eid*\to\phi}, \varepsilon_{pid*\to*}) + \varepsilon_{task*\to task\_name+} + \varepsilon_{date*\to+}))/12 * 100\% = 68.83\%$

Both values are high enough to suggest that there are common nodes between XRTN1 and

XRTN2. The next step evaluates the similarity between the structures of XRTS1 and XRTS2. To optimize the computation of the structural similarities we use the references determined in Part 3 and include them accordingly into sequences.

$$SimStr_{XRTS1 \to XRTS2} = ((Max(\varepsilon_{eidR \to R}, \varepsilon_{sinR \to R})/2 + \varepsilon_{nameR \to R} + \varepsilon_{addressR \to *} + (\varepsilon_{pid* \to *} * \varepsilon_{REF} + 0)/2 * \varepsilon_{+ \to +})/4 * \varepsilon_{+ \to +} + (\varepsilon_{pidR \to R} * \varepsilon_{KEY} + \varepsilon_{decriptionR \to ?} + (\varepsilon_{budgetR \to \phi} + \varepsilon_{managerR \to R} + \varepsilon_{locationR \to R})/3)/3 * \varepsilon_{* \to *} * \varepsilon_{R \to +} + (\varepsilon_{task\_nameR \to taskR} * \varepsilon_{task\_nameKEY} + \varepsilon_{dateR \to R})/2 * \varepsilon_{+ \to +} * \varepsilon_{R \to *} * \varepsilon_{R \to +})/3 * 100\% = 66\%$$

$$SimStr_{XRTS2 \to XRTS1} = ((Max(\varepsilon_{eidR \to R} * \varepsilon_{KEY}, \varepsilon_{sinR \to R}) + \varepsilon_{nameR \to R} + \varepsilon_{address* \to R} + \varepsilon_{dateOfBirth? \to \phi}) * \varepsilon_{+ \to +} + ((\varepsilon_{pidR \to R} * \varepsilon_{KEY} + \varepsilon_{description? \to R} + (\varepsilon_{managerR \to R} + \varepsilon_{locationR \to R})/2 + \varepsilon_{pid* \to *} * \varepsilon_{pidREF}) * \varepsilon_{* \to *} * \varepsilon_{+ \to R} + (\varepsilon_{taskR \to R} + \varepsilon_{dateR \to R})/2 * \varepsilon_{+ \to +} * \varepsilon_{* \to R} * \varepsilon_{+ \to R})/5)/6 * 100\% = 46.50\%$$

The nodes similarity values Sim show that both schemas are from the same domain and refer to the same set of entities (employee, projects, and task) as they have many correspondent nodes. However, the structural similarity values show that they are organized significantly different. XSD1 is less general than XSD2 as more of its structure is included in XSD2 ($SimStr_{XRTS1 \to XRTS2} > SimStr_{XRTS2 \to XRTS1}$).

9 Conclusion and Future Work

Our approach finds equivalent XML schema structures by determining if their XML trees are equivalent. Xequiv first determins if schemas are from the same domain and if there is any similarity between their nodes regarding labels, data types and operators. Secondly, our algorithm focuses on structural organization and considers the number of nodes in structures, operators applied to sequences, nested or linked structures. The elimination of the non-leaf nodes using the Reduction Algorithm [8] makes the nodes path unimportant. This has the advantage of allowing schemas to be equivalent because they refer to the same entity attributes but not necessarily because they share a part of the XML tree. Further research needs to be conducted to asses the efficiency of Xequiv compared to other existent algorithms in the area.

References

[1] Al-Halimi, R. *et al. WordNet. An electronic lexical database* Reading, MIT Press, 1998.

[2] Arenas, M. and Libkin, L. *A normal form for XML documents* in Proceedings of PODS 2002, pages 85-96, 2002.

[3] Bertino, E., Guerrini, G., Mesiti, M. *A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications*, in Information Systems, Volume 29, Issue 1, pages 23-46, 2004.

[4] Boyer, J. *Canonical XML Version 1.0, W3C Recommendation*, White Paper, 15 March 2001. Available at http://www.w3.org/TR/xml-c14n.

[5] Cognitive Science Laboratory, *WordNet*. Available at http://www.cogsci.princeton.edu. (Last searched on May 20, 2004).

[6] Do H.-H., Melnik, S., and Rahm, E. *Comparison of schema matching evaluations* In Web, Web-Services and Database Systems, Lecture Notes in Computer Science, Vol. 2593, pages 221-237, 2003.

[7] Do H.-H. and Rahm, E. *COMA - A system for flexible combination of schema matching approaches*, in Proceedings of the 28th VLDB Conference, China, pages 610-621, 2002.

[8] Duta, A.C., Barker, K., and Alhajj, R. *XML schema reduction algorithm* Technical report June 2004.

[9] Lee, M.L., Yang, L.H., Hsu, W., Yang, X. *XClust: Clustering XML Schemas for Effective Integration* in Proceedings of the eleventh international conference on Information and knowledge management. Virginia USA, pages 292-299, 2002.

[10] Madhavan, J., Bernstein, P.A., Rahm, E. *Generic Schema Matching with Cupid* in Proceedings of the 27th VLDB Conference, pages 49-58, 2001.

[11] Nierman A. and Jagadish, H.V. *Evaluting structural similarity in XML documents*, in the International Workshop on the Web and Databases (WebDB), Madison, WI, pages, June 2002.

[12] Salminen, A. and Tompa, F.Wm. *Requirement for XML Document Database Systems* in Proceedings of the 2001 ACM Symposium on Document Engineering, Atlanta, Georgia, USA, pages 85-94, 2001.

[13] World Wide Web Consortium, *XML Schema Part 0, 1, and 2*, May 2001.