PORSCHE: Performance ORiented SCHEma Mediation

Khalid Saleem, Zohra Bellahsene, Ela Hunt

PORSCHE: Performance ORiented SCHEma Matching

Khalid Saleem
LIRMM – UMR 5506 CNRS
University Montpellier 2
161 Rue Ada, F-34392 Montpellier
+33 467 41 85 85
saleem@lirmm.fr

Zohra Bellahsene
LIRMM – UMR 5506 CNRS
University Montpellier 2
161 Rue Ada, F-34392 Montpellier
+33 467 41 85 85
bella@lirmm.fr

Ela Hunt
GlobIS, ETH-Zentrum
CH-8092 Zurich
+41 44 632 74 21
hunt@inf.ethz.ch

ABSTRACT
Semantic matching of schemas in heterogeneous data sharing systems is time consuming and error prone. Existing mapping tools employ semi-automatic techniques for mapping two schemas at a time. In a large-scale scenario, where data sharing involves a large number of data sources, such techniques are not suitable. In this paper we present a method, which creates a mediated schema tree from a large set of input schema trees and defines mappings from the contributing schemas to the mediated schema. It is a two-phase approach. First, we use a set of linguistic matchers, which extract the semantics of all distinct node labels, present in input schemas, and form clusters of semantically similar labels. Second, we use a tree-mining data structure, combined with the similar label clusters, to calculate the context of each node, which is used in mapping. Since the input schemas are trees, our tree mining algorithm uses node ranks calculated by pre-order traversal. Tree mining combined with semantic label clustering minimizes the target search space and improves performance, thus making it suitable for large scale data sharing. We report on experiments with up to 80 schemas containing 83,770 nodes. PORSCHE took 587 seconds to match and merge them to create a mediated schema and to return mappings from input schemas to the mediated schema. We compare the quality of matching of PORSCHE with COMA++ on standard XML schemas, and find them to be very similar to the mappings produced by COMA++.

Categories and Subject Descriptors
I.7.2 [**XML**]: XML and Web Data – *XML Data and schema integration.*

General Terms
Algorithms, Performance, Design, Experimentation.

Keywords
XML schema tree, schema matching, schema mapping, schema mediation, tree mining, large scale.

1. INTRODUCTION
Schema matching relies on discovering correspondences between similar elements of two schemas. Several different types of schema matching [2,4,7,8,9,13,14,15,16,17] have been studied, demonstrating their benefit in different scenarios. In data integration schema matching is of central importance. The need for information integration arises in data warehousing, OLAP, data mashups [11], and workflows. Omnipresence of XML as a data exchange format on the web and the presence of metadata available in that format force us to focus on schema matching, and on matching for XML schemas in particular.

Previous work on schema matching was developed in the context of schema translation and integration [2,4,9], knowledge representation [8,17], machine learning, and information retrieval [7,12]. Most mapping tools map two schemas with human intervention [2,4,7,8,9,13,14,15]. The motivation behind our work is to explore the matching and integration of a large set of schema trees, using scalable syntactic and semantic matching and integration techniques.

We consider schemas as rooted, labelled trees. This supports the computation of contextual semantics in the tree hierarchy. The contextual aspect is exploited by tree-mining techniques, making it feasible to use automated approximate schema matching [7] and integration in a large-scale scenario. The individual semantics of node labels have their own importance. We utilize linguistic matchers to extract the concepts hidden within them. Tree mining techniques by definition extract similar sub tree patterns from a large set of trees and predict possible extensions of these patterns. The pattern size starts from 1 and is incrementally augmented. There are different techniques [1,20] which mine rooted, labelled, embedded or induced, ordered or unordered sub-trees. The first basic function of tree mining is to find sub-tree patterns that are frequent in the given set of trees, which is similar to schema matching activity that tries to find similar concepts among a set of schema trees.

Our Contributions

a) Matching, merging and the creation of a mediated schema with semantically approximate mappings, in one algorithm which has good performance.
b) Use of tokenisation, abbreviation and synonym matching of label tokens, supporting node clustering.
c) Extension of a tree mining data structure [20] to schema matching, using b) to minimize the search space.
d) Ability to produce simple 1:1 mappings, as well as complex mappings, including 1:n (leaf mapped to non-leaf) and n:1 (non-leaf mapped to leaf).
e) Experiments with real XML schema instances (OAGIS, XCBL[1]) showing performance appropriate for a large scale scenario.
f) Comparison with COMA++, showing that PORSCHE is qualitatively similar.

WWW 2007, May 8--12, 2007, Banff, Canada.

[1] OAGIS : http://www.openapplications.org/

XCBL : http://www.xcbl.org/

The reminder of the paper is organized as follows. Section 2 presents the motivations and issues encountered in large-scale schema integration. Section 3 defines the concepts used in the paper. In Section 4 we formalize the semantic integration and mediation problem. Our approach using **P**erformance **OR**iented **SCHE**ma matching (PORSCHE) is detailed in Section 5. Section 6 presents the experimental results comparing our approach with previous work. While Section 7 reviews related work and compares it to ours. Section 8 gives a discussion for future work and Section 9 concludes the paper.

2. MOTIVATIONS IN SCHEMA MEDIATION

Schema Mediation can be defined as integration of a set of input schemas into a single mediated schema, with concepts mappings from the input schemas to the mediated schema.

Heterogeneous data sharing environments require semantic mediation to support query formulation and execution. Such an environment can be dynamic or static. In the late 90s schema integration applications emphasised a static approach, used, for example, in data warehousing. In a data warehouse, schema design (using a mediated schema) is a one-off process. The experts analyse the source schemas and design a centralised data warehouse schema, and the mappings from source schemas mapped to the warehouse schema. The mappings in such applications can be termed ***exact mappings***.

In a dynamic environment, individual data sources and their schemas are independent and free to change. A pre-designed mediated schema is no answer in a dynamically changing world. The mediated schema has to change, to match the changes in source schemas, by changing the mappings between the source and the mediated schema. An example of such a requirement can be seen in catalogue mappings in web based B2B data exchange. For example, at ebay.com, individual vendors map their catalogue to the ebay catalogue. A query placed on ebay returns results from all the vendors who have mapped their schemas to the ebay catalogue. At any time ebay can enhance its catalogue or any individual vendor can withdraw or change its catalogue structure.

There are numerous issues in the semantic integration of a large number of schemas. Beside mapping quality, performance is also very important. Semantic Web, by definition, offers a large-scale dynamic environment where individual service providers are independent. In such a situation the mappings can never be exact, rather they are approximate [5,8].

In this paper we focus on a large number of schemas, automated matching, and performance. We explore mediated schema generation. For a given batch of schemas and a chosen mediated schema, we efficiently carry out the construction of a large mediated schema, which integrates all given schemas. To enhance the speed and lower the cost of data integration, we remove the need for human intervention. Previous work on matching two large schemas has been presented in [15] using COMA++[2] tool. In this work [15], first, user divide the schema into fragments and then each fragment from source schema is mapped to target schema fragments to find inter-fragment matchings. Next, these fragment mappings are merged to compute the schema level mappings.

Schema size, batch size and matching algorithms

Performance is an open issue in schema matching [15,16,17]. The complexity of the schema matching task is typically proportional to the size of participating schemas, and the number of match algorithms employed, i.e. O(NMA), where N and M are node counts in the source and target schema and A is the number of algorithms applied [2,14]. The quality of mappings depends on the type of matching algorithms and the way they are combined, for instance their execution order.

Here we present a new method for schema matching and integration. The method is a hybrid algorithm which matches, maps and integrates schemas. It uses extended tree mining data structures for ***performance oriented approximate schema matching for XML data sets***.

3. PRELIMINARIES

3.1 Match Operator

Schema matching finds similarities between elements of one schema and the elements of another schema. There are three basic match cardinalities at node level as discussed in [16].

i) 1:1 – one node of source schema corresponds to one element in the target schema,

ii) 1:n – one node in the source schema is equivalent to a composition of n nodes in the target schema,

iii) n:1 - n number of nodes in source schema compositely map to one node in target schema.

Since we are matching tree structures, where the leaf nodes access data, we emphasize more on leaf node matching. Our categorization of node match cardinality is driven by its leaf or non-leaf status, as given in Table 1.

Table 1 : Match Categorization

Source Node	Target Node	Match Cardinality
leaf	leaf	1:1
non-leaf	non-leaf	1 :1
leaf	non-leaf	1 :n
lon-leaf	leaf	n :1

Semantically, a match between two nodes can be either an equivalence or a partial equivalence. In a partial match, the similarity is partial, e.g. Name = 'John M. Brown' in source schema is partially matched to LastName='Brown' and FirstName ='John' in the target schema, because Name also contains the MiddleInitial='M' information. If there are several possible matchings of the source element to the mediated schema, best/most correct match can be selected. The choice can depend upon some match quality confidence computed at run time [2,9,10,16].

3.2 Definitions

Semantic matching requires the comparison of concepts which are structured as schema elements. Node labels of schema elements can be considered as concepts and each element's contextual placement in the schema enhances the semantics of the concept. For example in Figure 1, element author/***name*** and publisher/***name*** are similar labels but their contexts are different,

which makes the two elements conceptually disjoint. In an XML tree, the combination of the element label and the structural placement of the element produce the concept.

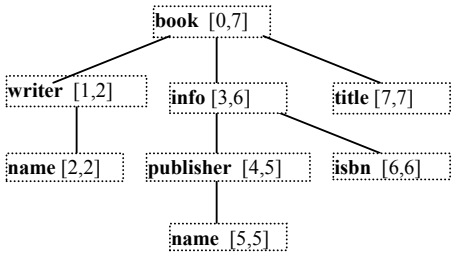

Figure 1. Example of an XML schema tree with abbreviated labels and [number, scope] marked for each node

Definition 1 (*Schema Tree*)
A schema is a rooted, labelled tree [18]. We call it a ***schema tree.*** A schema tree, **S**=(**V**, **E**) is a directed, acyclic, connected graph, with **V** = {0,1,...,n}, a set of nodes, and **E** = {(x,y) | x,y ∈ **V** }, a set of edges. One distinguished node r ∈ **V** is designated the root, and for all x ∈ **V,** there is a unique path from r to x. Further, ℓ: **V** → ***L*** is a labelling function mapping nodes to labels in ***L*** = {ℓ_1,ℓ_2, …}, and VT: ***L*** → $\mathbf{V_i}$ is a function, which returns a set of nodes $\mathbf{V_i} \subseteq \mathbf{V}$ with label $\ell \in \boldsymbol{L}$.

Definition 2 (*Ordered Tree*)
In an ordered tree the children of each node are ordered 1 to k, otherwise, the tree is unordered. We order each schema tree using pre-order traversal (Figure 1, node number).

Definition 3 (*Ancestor-Descendant Relationship*)
If x,y ∈ **V** and there is a path from x to y, then x is called an ancestor of y (and y a descendant of x), denoted $x \leq_p y$, where p ∈ ***N*** is the length of the path from x to y. If $x \leq_1 y$ (x is an immediate ancestor), x is called the parent of y, and y the child of x. If x and y have the same parent, they are called siblings, and if they have a common ancestor, they are called cousins, provided they are at same level in the tree and are not siblings.

Definition 4 (*Node Scope*)
Since the schema tree is ordered (Def. 2), nodes x ∈ **V** are numbered according to their position in the de pre-order traversal of the tree S (for example, the root is numbered 0). Let T(x) denote the sub-tree rooted at x, and let y be the rightmost leaf (or highest numbered descendant) under x. Then the scope of x is defined as scope(x)=[x,y]. Intuitively, scope(x) is the range of nodes under x, and includes x itself, see Fig. 1. The count of nodes in T(x) is ***y-x+1***.

Definition 5 (*Tokenization*)
A label ℓ is a composition of ***m*** strings, called Tokens. t: **L** → **T**is a tokenization function which maps a label to a set of tokens **T**= t_1,t_2, …}.

Tokenization [8] can help in establishing similarity between two labels. For example label (DateOfBirth) = {date, of, birth} and label (BirthDate) = {birth, date}. Since 'of' is a preposition, it can be discarded. This produces 100% similarity for the two labels, given the identical token sets {date, birth}.

Definition 6 *(Token Semantics*)
Token semantics discovery is the process of lemmatization of each token and finding natural language grammatical meaning of respective lemma (verb, noun, abbreviation etc.). For example, in labels 'IssuedAt' and 'IssuedOn', lemma for the token 'Issued' is 'issue', which is a verb. Using some external natural language oracles, one can infer different semantics for the two labels. For label 'IssuedAt', the reference is towards a place, whereas 'IssuedOn' refers to a date. If the two labels are ' IssueAt' and 'IssueOn', the semantics may be different, as lemma 'issue' is a noun and not a verb. A function C(t) returns concept of token t as the lemma which contains its grammatical interpretation.

C(t):t→ lemma(t)

Definition 7 (*Label Semantics*)
Label semantics corresponds to the conceptual meaning of the label (irrespective of its relation to the node it is related to). It is a composition of concepts attached to the tokens making up the label i.e.,

C_l: l → (C (t_1),C (t_2), ……,C (t_m)).

Definition 8 (*Node Semantics*)
Node semantics C_x combines the semantics of the node label $C(l_x)$ with its contextual placement in the tree TreeContext(x) [8].

C_x: x → ($C(l_x)$, TreeContext(x)).

TreeContext of a node is calculated using the node number and scope. These properties encode structural semantics. This is illustrated in Example 1, Section 3.3.

3.3 Scope Properties

Scope properties give us the contextual placement of a node in the tree and are explained in detail, from the tree mining perspective [20]. The properties are simple integer operations.

Unary Properties, given a node x [X,Y]

Property 1. Leaf Node(x) : X=Y.
Property 2. Non-Leaf Node(x): X<Y.

Binary Properties
Given x [X,Y], xd [Xd,Yd], xa [Xa,Ya], and xr[Xr,Yr].

Property 3. Descendant (x,xd), xd is a descendant of x : Xd>X and Yd<=Y.
Property 4. Descendant Leaf (x,xd) (combination of Property 1 and 3) : Xd>X and Yd ≤Y and Xd=Yd.
Property 5. Ancestor (xa,a) (complement of Property 3) xa is ansector of x : Xa<X and Ya>=Y.
Property 6. Right Hand Side Nodes with Non-Overlapping Scope : xr is Right Hand Side Node of x I: Xc>Y.

Example 1: In Figure 1, Property 1 for node **title**[7,7] defines it as a leaf because the node number equals the number of its rightmost child. Property 2, for **publisher** [4,5] defines it is a non-leaf node, as its number is less than the number of its rightmost child. Properties 1 and 2 detect simple and complex elements in an XML schema.■

Example 2: The task is to find in the tree nodes matching of author/**name**. In Figure 1 there are two nodes called **name**:[2,2] and [5,5]. Given synonymy between words **author** and

writer[1,2], we perform the descendant node check on nodes [2,2] and [5,5] with respect to **writer**[1,2]. Node [2,2] is a descendant of [1,2], using Property 3, and node [5,5] is not a descendant of [1,2]. Similarly, property 5 produces Ancestor([4,5],[5,5]) which holds for **publisher**[4,5] and **name**[5,5] and Ancestor([0,7],[4,5]) holds for **book**[0,7] and **publisher**[4,5]. ■

4. DEFINITION OF SEMANTIC MATCHING

INPUT: A set of schema trees $\mathbf{S}=\{S_1,S_2, \ldots, S_u\}$.

OUTPUTS:

a) A *mediated schema tree* S_m, which is a composition of all distinct concepts in S.
$S_m = \mathbf{P}^{u}_{i=1}$, $S_i = \{C_1 \rho C_2 \rho \ldots . \rho C_n\}$ includes all distinct concepts in S (Def. 8). **P** is the composition function and ρ denotes the composition operator.
b) A *set of mappings* $\mathbf{M} = \{M_1, M_2, \ldots., M_w\}$ from the concepts of input schema trees to the concepts in the mediated schema.

The mediated schema tree S_m is a composition of all nodes representing distinct concepts in the set of schemas. During the integration process if a node is not present in S_m, a new edge e' is created in S_m and a node is added to it.

4.1 Semantic Label Matching

Semantic label matching intuitively minimizes the search space of possible mappable target nodes [8,20]. The derivation of concept similarity in two schemas is initiated by comparing their labels. Similarity between labels is either equivalence or partial equivalence, as shown below:

a) Equivalence: $C(l_x) = C(l_y)$ Similar

b) Partial Equivalence: $C(l_x) \cong C(l_y)$

i. More Specific ¦ Is part of : $C(l_x) \subseteq C(l_y)$

ii. More General ¦ Contains : $C(l_x) \supseteq C(l_y)$

iii. Overlaps: : $C(l_x) \cap C(l_y)$

Example 3: Consider labels AuthorName and WriterName. Since Author and Writer are synonyms and Name is shared, conceptually they are equivalent, and AuthorName = WriterName. Similarly, AuthorLastName $\subseteq$ AuthorName, as LastName is conceptually part of Name. Conversely, AuthorName $\supseteq$ AuthorLastName. MiddleLastName and FirstNameMiddle are overlapping, as they share tokens {Name, Middle}..■

5. PORSCHE: Our Approach

We assume that only schema trees are available as input. **PORSCHE** accepts a set of schema trees and outputs the mediated schema tree and the corresponding mappings. We make the following assumptions valid in domain specific schema integration (extended from [18]).

5.1 Assumptions

a) Schemas in the same domain contain the same domain concepts, but differ in structure and concept naming.

b) In one schema different labels for the same concept are rarely present.

c) Only one type of matching between two labels is possible. For example, author is a synonym of writer.

d) We select the input schema with the highest number of nodes as the initial mediated schema. Since each node represents a concept, this covers the maximum number of concepts. This minimizes the addition of new concepts (nodes not present in the mediated schema) to the mediated schema and should improve performance.

e) We perform semantic comparisons between the labels of the mediated schema and the labels not present in the mediated schema (based on assumption b). This minimizes the target search space for similar labels.

f) A node from the input schema is only matched to the cluster of similar label nodes present in the mediated schema.

5.2 PORSCHE Architecture

PORSCHE architecture covers the complete semantic integration process involving schema trees in a large-scale scenario. A diagram of the architecture is shown in Figure 2.

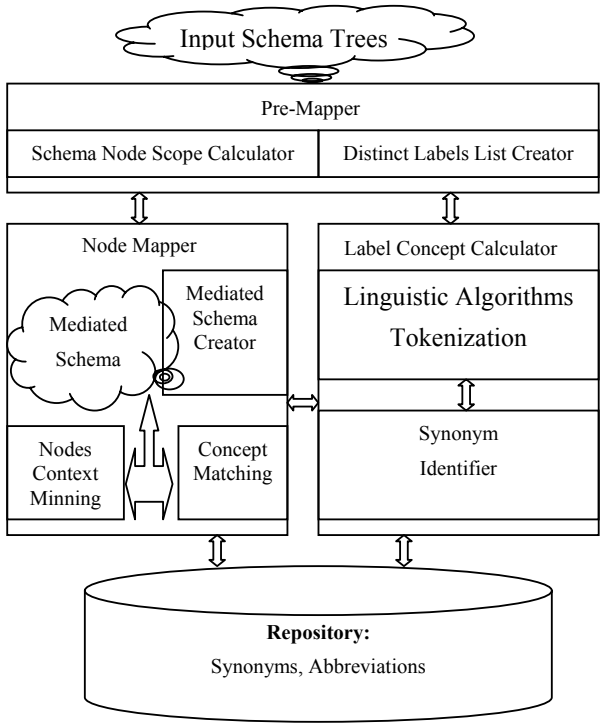

Figure 2. PORSCHE Architecture

The application is divided in three parts: Pre-Mapper, Label Concept Calculator and Node Mapper. Schema trees are input to the system as a stream of XML. Pre-Mapper calculates scope and node number for each of the nodes in the input schema trees. A listing of nodes and a list of distinct labels for each tree is constructed. In a schema tree, if several nodes have the same label, each corresponding node label is considered as distinct, as it represents a contextually different node and has distinct semantics. For example, name in author→name and publisher→name are two distinct node labels.

In the next phase, a linguistic matcher identifies semantically distinct node labels in the input trees. It uses an abbreviation table

and tokenizes the labels. Then it derives the meaning for each individual token and combines these meanings to form a label concept. The comparisons of labels are based on similar token sets or similar synonym token sets.

Example 4 : Consider labels "POShipOn" and "PurchaseOrderDeliverOn". In the abbreviation table PO stands for {purchase, order} and in the synonym table 'deliver'='ship' and 'on'='date'. This implies that the two labels have similar token sets.■

Mediated Schema Creator takes the input schema tree with the highest number of nodes and then takes each schema in turn and merges it with the mediated schema. This requires matching, mapping and merging. Concepts from input schemas are matched to the mediated schema. The algorithm traverses the input schema depth-first, mapping parents before siblings. If a new concept is found, with no match in the mediated schema, a new concept node is created and added to the mediated schema. It is the right most child leaf node added to the node in the mediated schema to which the parent of current node is mapped. This new node is used as the target node in the mapping. The algorithm combines node label similarity and contextual positioning in the schema tree, calculated with the help of properties defined in Section 3.3.

5.3 Algorithms

Pre-Mapper comprises a number of functions: (1) scope calculation, (2) parent node number calculation, (3) initial mediated schema detection and (4) creation of data structures for matching, shown in Tables 1 and 2. Those data structures are updated by **NodeMapper** algorithm. The novelty of our method is that *similar labels are clustered together* to support fast calculation of possible matches. The best match is selected, based upon the contextual placement of nodes. (1), (2), (3) and (4) require one traversal of each tree.

NodeMapper algorithm, Figure 3, covers both schema matching and integration. It handles the semantics of node labels along with the node's tree context, to compute the mapping. The algorithm creates mappings for almost every input node, in particular for the leaves, since the data resides there. The algorithm accepts as input a set of schemas S and selects the initial mediated schema S_m. It outputs the mediated schema S_m' with added nodes (if any) and a set of mappings M, from input schema nodes to the mediated schema.

Initially, S_m is adopted as the final mediated schema S_m' (l. 1). Match for every node (l. 3) of each input schema (l. 2) is calculated, mapping it to the mediated schema. For each input node v, a set Vt of possible mappable target nodes in the mediated schema is created, constituting the target search space for v. The criterion for the creation of this set of nodes is node label equivalence or partial equivalence (l. 4,5). Vt can have zero (l. 24), one (l. 8) or more than one (l. 18) nodes. Checks in lines 9, 14 and 16 are used to compare the tree context of nodes v (input node) and vt (possible target node in Vt), to ensure e.g. that a leaf node is mapped to another leaf node.

Second check, ancestorMap (l. 10), ensures that at some point up the ancestor hierarchy of v and vt, there has been a mapping. This is to ensure that subtrees containing v and vt correspond to similar concept hierarchies. This increases the match confidence of nodes v and vt. Similarly if the target search space Vt has more than one node, we check the *descendant* property (line 19), for each node vt in Vt, Descandant(map(parent(v)),vt). This verifies that vt lies in the sub-tree of node to which parent node of v is mapped, that is with in the scope.

Algorithm : NodeMapper

Input: S, Sm // *S : set of Schema Trees*
// *Sm : Mediated Schema Tree*
Output: M, Sm' // *M: set of mappings; initially empty*
// *Sm': Sm appended with new nodes*

```
Begin
1  Sm'←Sm
2  for each Si ∈ S do                          // 1≤ i ≤ u
3     for each v ∈ V(Si) do   // V : nodes of Si
4        Lv ← l(v)                             // Lv : label of node v
5        Lvsl←SimilarLabels(Lv, Sm')
         // Lvsl : set of labels in Sm' similar to Lv
         // Lvsl ⊆ Lm where Lm:  set of node labels in Sm'
6        if (Lvsl ¬ empty) then
7           Vt←VT(Lvsl) //Vt : {vt1, vt2 ….vt|Lvsl|} : Def. 1
            // Vt: set of nodes in Sm' corresponding  to
            // labels in Lvsl, vt: one of the nodes in Vt
            // VT is applied to each label in Lvsl in turn
8           if (|Vt| = 1) then
9              if ((Leaf(v)∧Leaf(vt1)) ∨  // property 1,2
                   ((¬ Leaf(v))∧(¬ Leaf(vt1)))) then
10                if (ancestorMap(v,vt1))
11                   m← map11 (v,vt1)
12                else (vt1, Sm')=addNewNode(Sm',v)
13                   m ← map11 (v, vt1)
14             if (Leaf(v)∧(¬Leaf(vt1))) then
15                m← map1n (v,leafNodes(vt1))
16             if (¬(Leaf(v))∧(Leaf(vt1))) then
17                m← mapN1 (v,vt1)
18          else        // i.e. |Vt| >1
19             for  each vt ∈ Vt do  // Vt:{vt1,vt2,..,vtm}
                  // function true for only one node or none
20                if (descendant(v,vt) = true)      // property 3
21                   m← map11 (v,vt)
22                else (vt, Sm')← addNewNode (Sm',v)
23                   m ← map11 (v,vt)
24       else (vt, Sm')= addNewNode (Sm',v)
25          m ← map11 (v,vt)
26       M=M ⋃ m
27 return M, Sm'
End
```

Figure 3. Schema mediation algorithm

Function addNewNode( ) (l. 12,22,24) adds a new node vt as the rightmost sibling of node to which parent node of v is mapped. Mapping function map11( ), makes a 1:1 map from v to vt, whereas map1n( ) creates a mapping from v (identified as leaf) to a set of leaf nodes of subtree rooted at node vt (identified as non leaf node), which is a 1:n mapping. This mapping is considered to be an approximate mapping but follows the semantics that leaf node should map to leaf nodes. And similarly mapn1( ) is a composite mapping of leaf nodes under v, to the leaf node vt.

NodeMapper algorithm integrates all input nodes into the mediated schema and creates corresponding mappings from input schemas to the mediated schema.

5.4 Example of Schema Integration

Figure 4 shows two trees after executing **Pre-Mapper.** A list of labels created in this traversal is shown in Table 2a. The two nodes with the same label 'name' but different parents are shown to be disjoint. The last entry in the list is a label 'ROOT' for the root node of mediated schema. A matrix of size ***um*** is created, where ***u*** is the number of schemas and ***m*** the number of distinct labels in all schemas (the length of the label list), see Table 2.b. In the matrix each row represents an input schema tree. Each non-null entry contains the node scope, parent node link and the mapping, which is initially null. Matrix columns are ordered according to the order of nodes in the label list.

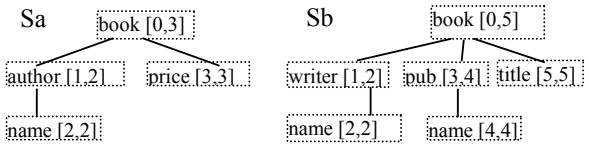

Figure 4. Two input schema trees

The largest schema tree Sb, Figure 4, is selected as the initial mediated schema Sm. A list of size ***m***, Tab. 1.c, is created to hold Sm, assuming the same column order as in Table 1.a and 1.b. In the label list the semantically similar (equivalent or partially equivalent) labels are detected. Author is equivalent to writer.

The node mapping algorithm takes the data structures in Table. 1 as input, and produces mappings shown in Table. 3.b and the integrated schema in Table 3.c. In the process, the input schema Sa is directly mapped to mediated schema Sb. The mapping is taken as the column number (Table 3.b <column number>) of node. Saving mappings as column number gives us the flexibility to add new nodes to mediated schema tree. Scope values of some existing nodes are affected because of addition of new nodes, but column numbers of all previous nodes remain the same. Thus intuitively none of the existing mappings are affected. All mappings in this case are one to one.

NodeMapper for input schema tree Sa (Table 2.b row 1) starts from node Sa[0,3] with label book. The sequence of nodes mapping from the input schema tree follows the depth first traversal. This makes sure that parent nodes are mapped before the siblings. Sa[0,3] is a non-leaf node with only one similar node

Table 2. Before NodeMapper Execution

a. Labels List

0	1	2	3	4	5	6	7	8
author	***book***	***name***	***name***	***price***	***pub***	***title***	***writer***	***ROOT***

b.Input Schema Nodes' Matrix

1,2,0	*0,3,-1*	*2,2,1*		*3,3,0*				
	0,5,-1	***2,2,1***	***4,4,3***		***3,4,0***	***5,5,0***	***1,2,0***	

c. Initial Mediated Schema

	1,6,0	***3,3,2***	***5,5,4***		***4,5,1***	***6,6,1***	***2,3,1***	***0,6,-1***

in the mediated schema tree Sm (Sb in this example) i.e., node 1 at column 1. So label <1> in column 1 for Sa records the mapping Sa[0,3]-> Sb[0,5], see Table 3.b. Information regarding mapping is also saved as '1.0' i.e., label 0 of schema tree 1. Next node to map is node in Sa, author[1,2], similar to writer[1,2] in Sb. Both nodes are internal nodes and the function ancestorMap( ) returns true since parent nodes of both are already mapped. The resulting mapping for label 0 is label <7>. For label 2 'name', there are two possibilities, label 2 (column 2) and label 3 (column 3). Descendant(name,author) is true for node in column 2 and false for 3. Hence <2> is the correct match. The last node in Sa is price[3,3]. There is no node in mediated schema tree with a similar label, so a new node is added to mediated schema, recorded by an entry in the column with label 'price' in the mediated schema list (Table 3.c). This node is created as the right most sibling of node in the mediated tree to which the parent node of current input node is mapped i.e., node with label 'book'. The scope and parent node link is accordingly adjusted for the new node. And a mapping is created from input node to this newly created target node.

Table 3 . After NodeMapper Execution

a. Label List

0	1	2	3	4	5	6	7	8
author	***book***	***name***	***name***	***price***	***pub***	***title***	***writer***	***ROOT***

b. Mapping matrix

1,2,0 <7>	*0,3,-1* <1>	*2,2,1* <2>		*3,3,0* <4>				
	0,5,-1 <1>	*2,2,1* <2>	*4,4,3* <3>		*3,4,0* <5>	*5,5,0* <6>	*1,2,0* <7>	

c. Final Mediated Schema

	1,7,0, ***1.0,*** ***2.0***	***3,3,2*** ***1.2,*** ***2.2***	***5,5,4*** ***2.4***	***7,7,1*** ***1.3***	***4,5,1*** ***2.3***	***6,6,1*** ***2.5***	***2,3,1*** ***1.1,*** ***2.1***	***0,7,-1***

If the new node is added in the middle of the tree then its ancestor's scope is incremented by one and accordingly next node numbers in a pre-order traversal are adjusted in the tree.

The algorithm keeps track of the columns for the next node according to pre-order. Thus the final mediated schema tree can be generated from the final mediated schema row by a traversal starting from the ROOT.

6. EXPERIMENTAL EVALUATION

We examine both the performance and quality of schema matching. Performance is evaluated as the number of schemas or nodes processed versus the time required for matching, merging and mapping. Quality of matches is compared with the match results produced by COMA++ [2].

6.1 Performance Evaluation

The prototype uses Java 5.0. All the schema tree data structures and corresponding mappings data are saved in a database for analysis and statistical evaluation. A PC, Pentium 4-M, 1.80 GHz, 768 MB RAM, running Windows XP was used in this evaluation.

We selected three sets of schema trees from different domains:

1. Domain 1 : BOOKS (Synthetic Schemas)
2. Domain 2: OAGIS (Real Schemas from the Web)
3. Domain3 : XCBL (Real Schemas from the Web)

Table 4. Characterization of schema domains.

	Domain 1 (Real) OAGIS	Domain 2 (Real) XCBL	Domain 3 (Synthetic) Books
Number of Schemas	80	44	176
Avg. nodes per schema	1047	1678	8
Largest/ smallest schema size	3519/ 26	4578/ 4	14/ 5

Experiments were performed with different sizes of sets taken from the core domain sets. Algorithm time execution performance was evaluated under three different label similarity cases.

A) Label String Equivalence

B) Token Set Equivalence

C) Synonym Token Set Equivalence

Our experiments show that the execution time for PORSCHE depends upon the number on the schemas which are being integrated, and appears to be quadratic in the number of nodes compared. Figure 5 shows time in milliseconds for Domains 2 and 3.

Figure 6 shows a comparison of three kinds of matching A, B, and C for sets of 2, 4, 8, 16, 32, 64, 128, 176 from BOOKS. There is no difference in the time performance of various matchers. This is possibly due to the fact that synthetic schemas vary little in their labels.

For Domains 2 and 3, Figures 7 and 8 show the time in (s) against the number of nodes processed, for the three similarity methods. XCBL schemas are slower to match than OAGIS schemas. This is due to the higher average number of nodes in XCLB schemas. It takes less than 550 s to match 80 OAGIS schemas, while 44 XCLB schemas require about 800 s

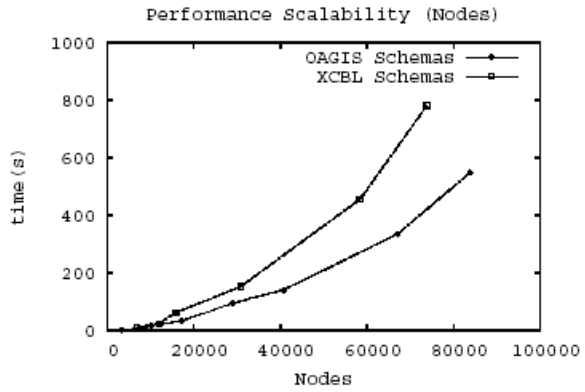

Figure 5. Comparison of schema integration times for real web schemas.

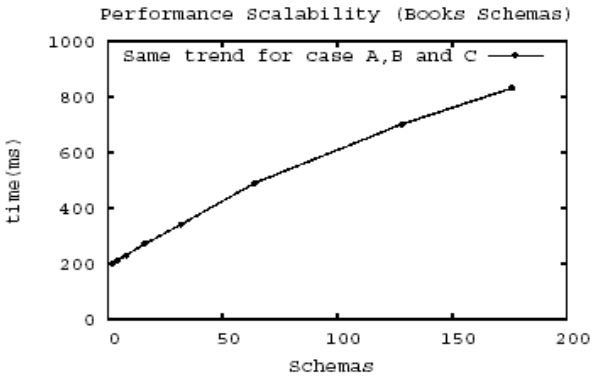

Figure 6. Integration time with reference to the number of schemas in BOOKS

In both cases there is a slight increase in matching times for categories A, B and C, because of different label matching strategies. A is the fastest, as it works only on the labels themselves. B is slightly slower, as labels have to be tokenized and the number of tokens is larger than the number of labels, and C is the slowest, as one need to match synonyms as well. These evaluation cases show that PORSCHE has acceptable performance on an office PC.

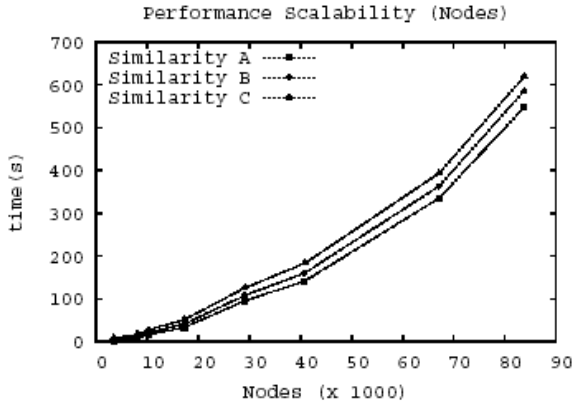

Figure 7. Integration of OAGIS schemas

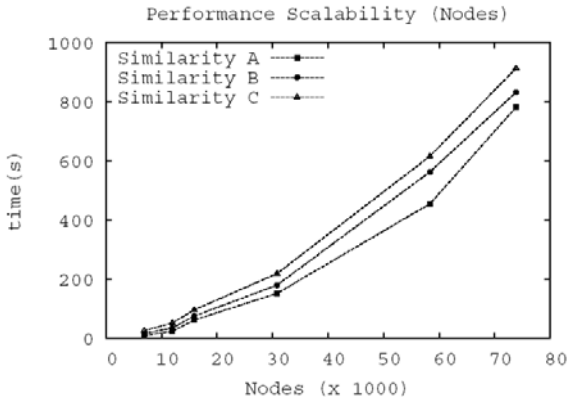

Figure 8. Integration of XCBL schemas

6.2 Quality Comparison

The quality of mappings is compared for the three scenarios A, B and C. The quality of mappings is the highest for case C, synonym token sets equivalence. For quality evaluation we used three sets of two schemas. Those were: (a) Purchase Order schemas from a web site; (b) BOOKS; and (c) OAGIS. The results were compared to COMA++ under similar conditions. Table 5 shows the quality and performance comparison between PORSCHE and COMA++. The abbreviation and the synonym tables were provided by COMA++ demo version.

Table 5. PORSCHE – COMA++ performance comparison

Domain / Schema / Size / Match Tool	Purchase Order		Books		OAGIS	
	S1	S2	S1	S2	S1	S2
	18	14	15	12	2931	475
	T	Q	T	Q	T	Q
PORSCHE	0.2	=	0.2	=	2.5	---
COMA++	5	=	3	=	370	---

*T : Time (s) , Q : Quality comparison to COMA++

We could not carry out a quality comparison for Set C because of the size of schemas and the fact that the abbreviation and synonym tables existed only for Purchase Orders and Books.

Our experiments show that PORSCHE improves on COMA++, as it produces similar integrated schemas **automatically**. COMA++ requires human intervention to select the best match and create mappings after merging.

6.3 Complexity

Given a set of input schemas $\mathbf{S} = \{S_1, S_2, \ldots, S_u\}$, we select as the mediated schema the schema tree with highest number of nodes, max ($N(S_i)$) where $N(S_i)$returns the number of nodes in a schema. We match each node of each input schema with the mediated schema. The number of input schema nodes N_t is given by $N_t = \Sigma^u_{i=1} N(S_i)$. Therefore the complexity of Node Mapper algorithm is $O(N_t \times N(S_m))$. This is quadratic in the size of schema set that is to be integrated. Our experiments confirm this complexity.

7. Related Work

Nearly all schema-matching systems [2,6,7,8,9,13,14,15] compare two schemas at a time and aim for quality matching but *require significant human intervention*. For a limited number of schemas (less than 10) of small size (less than 100 nodes), these matching and integration processes give acceptable performance. Surveys of schema-matching tools [16,17] present the results and their comparison.

The most recent matching tools are S-Match and Coma++. S-Match is a hybrid matcher. It carries out semantic matching by using the Wordnet dictionary. [8] demonstrates better mappings than Coma and Cupid but has worse performance. Coma is a composite matcher, which can reuse previous mappings. It uses user defined synonym and abbreviation tables, along with some basic name matchers. Coma has the advantage of mapping large schemas with the help of user input. The user can identify fragments of the schema to be mapped. This option is intended to manage the namespace/ include characteristics of XML schemas [2,15]. However, human intervention in the schema mapping process is needed. Moreover, the final requirement is an integrated schema. Systems, which just produce mappings and no integrated schema do not support automated data integration suitable for application environments with hundreds of schemas.

Another interesting schema matching domain under active research is matching elements across query interfaces of structured Web databases. The elements layout forms a certain hierarchy backed by a database schema. For certain Web domains such as travel, these interfaces can go into thousands. [10, 18] handle holistically the integration of these structured layouts as a mining problem. [18] observes that Web database query interfaces in the same domain are usually semantically similar, as a label is often unambiguous in a domain but it can have several meanings in a dictionary, and synonym labels are rarely co-present in the same schema. However, grouping of elements such as . LastName and FirstName, is common in the schema at the same level with same parent element, to form a larger concept.

8. DISCUSSION FOR FUTURE WORK

PORSCHE is similar to other tools using linguistic techniques like tokenization, use of abbreviations and synonym oracles. At present it uses very limited linguistic methods but the match quality is equivalent to other current tools. The architecture of PORSCHE is flexible, and can accommodate new syntactic and linguistic similarity algorithms.

As future work we plan to investigate the application of this technique for nodes matching for an Information system based on P2P architecture. Secondly, we want to enhance the linguistic matching part of the system. Our study of the tree mining technique reveals that it can be utilized for identifying co-relationships between the co-existence of elements with in schema tree and a forest of schema trees. Thus help in identifying subsumptions and overlap relationships for n:m complex mappings.

9. CONCLUSIONS

We presented a novel schema integration method, PORSCHE, which has shown very promising results for large scale schema integration. It uses a tree based pre-order traversal algorithm for matching, merging and mapping a set of schema trees. To improve performance we adapted a technique from tree mining including the clustering of similar node labels. This minimizes the

target search space for a node match and gives better performance. Selection of the largest schema tree as the mediated schema further enhances the matching process. PORSCHE uses the optimistic top down pre-order match traversal (parents are mapped before children, and left sub-tree is traversed before right sub-tree), since our assumption is that we utilize it in a domain specific environment. This helps in using the structural semantics of nodes for better quality matching.

The novelty of our method is fourfold. First, we support fully automated schema matching. Second, we not only generate matches, but also build an integrated schema at the same time. Third, our approach scales to hundreds of schemas. Fourth, the use of tree mining techniques for schema matching is also new in this field.

10. REFERENCES

[1] Asai, T., Arimura, H., and Nakano, S. Discovering Frequent Substructures in Large Unordered Trees. In Proceedings of ICDS, 2003.

[2] Aumuller, D., Do, H.-H., Massmann, S., and Rahm, E. Schema and ontology matching with COMA++. In Proceedings of SIGMOD, 2005.

[3] Avigdor, G., Ateret, A-T., Alberto, T., and Danilo M. A framework for modeling and evaluating automateic semantic reconciliation. VLDB Journal, Vol. 14, Issue 1, March 2005.

[4] Bernstein, P.A., Melnik, S., Petropoulos, M., and Quix, C. Industrial-Strength Schema Mapping. SIGMOD Record, Vol. 33, No. 4, December 2004.

[5] Chang, K.C., He, B., and Zhang, Z. Mining Semantics for Large Scale Integration on the Web: Evidences, Insights, and Challenges. SIGKDD, Vol. 6, Issue 2, December 2004.

[6] Do, H.-H., Melnik, S., and Rahm, E. Comparison of schema matching evaluations. In Proceedings of the workshop on Web and Databases, 2002.

[7] Doan, A., Madhavan, J., Domingos, P., and Halevy, A. Learning to map ontologies on the semantic web. In Proceedings of the Int'l World Wide Web Conference (WWW), 2003.

[8] Giunchiglia, F., Shvaiko, P., and Yatskevich, M. S-match: an algorithm and an implementation of semantic matching. In Proceedings of ESWS, 2004.

[9] Halvey, A., Ives, Z., Suciu, D., and Tatarinov, I. Schema Mediation in Peer Data Management Systems. In Proceedings of ICDE, 2003.

[10] He, B., and Chang, K.C.-C. A Holistic Paradigm for Large Scale Schema Matching. SIGMOD Record, Vol. 33, No. 4, 2004.

[11] Jhingran, A. Enterprise Information Mashups: Integrating Information, Simply. Keynote Address, VLDB 2006.

[12] Kurgan, L. et al. Semantic Mapping of XML Tags Using Inductive Machine Learning. In Proceedings of ICMLA, 2002.

[13] Lu, J., Wang, S., and Wang, J. An Experiment on the matching and Reuse of XML Schemas. In Proceedings of ICWE 2005.

[14] Madhavan, J., Bernstein, P.A., and Rahm, E. Generic schema matching with Cupid. In Proceedings of the VLDB, 2001.

[15] Rahm, E., Do, H-H., and Masmann, S. Matching Large XML Schemas. ACM SIGMOD Record 33(4):26-31, 2004.

[16] Rahm, E., and Bernstein, P. A. A Survey of Approaches to Automatic Schema Matching. VLDB Journal, 2001.

[17] Shvaiko, P. A classification of schema-based matching approaches. In Proceedings of the workshop at the ISWC, 2004.

[18] Su, W., Wang, J., and Lochovsky, F. Holistic Query Interface Matching using Parallel Schema Matching. In Proceedings of ICDE, 2006.

[19] Tranier, J., Baraer, R., Bellahsene, Z., and Teisseire, M. Where's Charlie: Family Based Heuristics for Peer-to-Peer Schema Integration. In Proceedings of IDEAS, 2004.

[20] Zaki, M.J. Efficiently Mining Frequent Embedded Unordered Trees. *Fundamenta Informaticea*, 2005.