XML Grammar Similarity: Breakthroughs and Limitations

Joe TEKLI, Richard CHBEIR* and Kokou YETONGNON

LE2I Laboratory – University of Bourgogne

Engineer's Wing

BP 47870 21078 Dijon CEDEX FRANCE

Phone: (+33) 380393655 – Fax: +33380396869

Email: {joe.tekli, richard.chbeir, kokou.yetongnon}@u-bourgogne.fr

Keywords: Semi-structured XML data, XML Grammar, DTD, Structural similarity, Schema Matching.

XML Grammar Similarity: Breakthroughs and Limitations

ABSTRACT

In recent years, XML has been established as a major means for information management, and has been broadly utilized for complex data representation particularly web and multimedia data. Owing to an unparalleled increasing use of the XML standard, developing efficient techniques for comparing XML grammars/schemas becomes essential in various applications domains, such as data integration and data warehousing. In this paper, we provide an overview of XML grammar similarity. We present the state of art related to XML grammar structural comparison. We discuss the features and functionalities of existing approaches, identifying theirs strengths and weaknesses. We consequently try to specify the characteristics of an improved grammar comparison method. Some emergent future research directions are also mentioned.

INTRODUCTION

W3C's **XML** (eXtensible Mark-up Language) has recently gained unparalleled importance as a fundamental standard for efficient data management and exchange. The use of XML covers data representation and storage, database information interchange, data filtering, as well as web applications interaction and interoperability. XML has been intensively exploited in the multimedia field as an effective and standard means for indexing, storing, and retrieving complex multimedia objects. SVG[1], SMIL[2], X3D[3] and MPEG-7[4] are only some examples of

[1] WWW Consortium, SVG, http://www.w3.org/Graphics/SVG/.
[2] WWW Consortium, SMIL, http://www.w3.org/TR/REC-smil/.
[3] Web 3D, X3D, http://www.web3d.org/x3d/.
[4] Moving Pictures Experts Group, MPEG-7 http://www.chiariglione.org/mpeg/standards/mpeg-7/.

XML-based **multimedia data representations**. With the ever-increasing web exploitation of XML, there is an emergent need to automatically process XML documents and grammars for similarity classification and clustering, information extraction and search functions. All these applications require some notion of **structural similarity**, **XML** representing semi-structured data. In this area, most work has focused on estimating similarity between XML documents (i.e., data layer). Nonetheless, few efforts have been dedicated to comparing XML grammars (i.e., type layer).

Computing the structural similarity between **XML documents** is relevant in several scenarios such as change management [(Chawathe S., Rajaraman A., Garcia-Molina H., and Widom J., 1996), (Cobéna G., Abiteboul S. and Marian A., 2002)], XML structural query systems (finding and ranking results according to their similarity) [(Schlieder T., 2001), (Zhang Z., Li R., Cao S. and Zhu Y., 2003] as well as the structural clustering of XML documents gathered from the web [(Nierman A. and Jagadish H.V., 2002), (Dalamagas, T., Cheng, T., Winkel, K., and Sellis, T., 2006]. On the other hand, estimating similarity between **XML grammars** is useful for data integration purposes, in particular the integration of DTDs/schemas that contain nearly or exactly the same information but are constructed using different structures [(Doan A., Domingos P. and Halevy A.Y., 2001), (Melnik S., Garcia-Molina H. and Rahm E., 2002)]. It is also exploited in data warehousing (mapping data sources to warehouse schemas) as well as XML data maintenance and schema evolution where we need to detect differences/updates between different versions of a given grammar/schema to consequently revalidate corresponding XML documents (Rahm E. and Bernstein P.A., 2001).

The goal of this study is to briefly review XML grammar **structural similarity** approaches. Here, we provide a unified view of the problem, assessing the different aspects and techniques related to XML grammar comparison. The remainder of this paper is organized as follows. Section 2 presents an overview of XML grammar similarity, otherwise known as **XML schema matching**. Section 3 reviews the state of the art in XML grammar comparison methods. Section 4 discusses the main criterions characterizing the effectiveness of XML grammar similarity approaches. Conclusions and current research directions are covered in Section 5.

OVERVIEW

Identifying the similarities among grammars/schemas[1], otherwise known as **schema matching** (i.e., **XML schema matching** w.r.t. XML grammars), is usually viewed as the task of finding correspondences between elements of two schemas (Do H.H., Melnik S. and Rahm E., 2002). It has been investigated in various fields, mainly in the context of **data integration** [(Rahm E. and Bernstein P.A., 2001), (Do H.H., Melnik S. and Rahm E., 2002)], and recently in the contexts of **schema clustering** (Lee M., Yang L., Hsu W. and Yang X., 2002) and **change detection** (Leonardi E., Hoai T.T., Bhowmick S.S. and Madria S., 2006).

In general, a schema consists of a set of related elements (entities and relationships in the ER model, objects and relationships in the OO model, …). In particular, an XML grammar (DTD or XML Schema) is made of a set of XML elements, sub-elements and attributes, linked together via the containment relation. Thus, the schema matching operator can be defined as a function that takes two schemas, S_1 and S_2, as input and returns a mapping between them as output (Rahm E. and Bernstein P.A., 2001). Note that the mapping between two schemas indicates which elements of schema S_1 are related to elements of S_2 and vice-versa.

[1] In the remainder of this paper, terms *grammar* and *schema* are used interchangeably.

The criteria used to match the elements of two schemas are usually based on heuristics that approximate the user's understanding of a good match. These heuristics normally consider the linguistic similarity between schema element names (e.g. string edit distance, synonyms, hyponyms, …), similarity between element constraints (e.g. '?', '*' and '+' in DTDs[1]), in addition to the similarity between element structures (matching combinations of elements that appear together). Some matching approaches also consider the data content (e.g. element/attribute values) of schema elements (if available) when identifying mappings (Doan A., Domingos P. and Halevy A.Y., 2001). In most approaches, scores (similarity values) in the *[0, 1]* interval are assigned to the identified matches so as to reflect their relevance. These values then can be normalized to produce an overall score underlining the similarity between the two grammars/schemas being matched. Such overall similarity scores are utilized in (Lee M., Yang L., Hsu W. and Yang X., 2002), for instance, to identify clusters of similar DTD grammars prior to conducting the integration task.

STATE OF THE ART

Schema matching is mostly studied in the relational and Entity-Relationship models [(Larson J., Navathe S.B. and Elmasri R., 1989), (Milo T. and Zohar S., 1999), (Castano S., De Antonellis V. and Vimercati S., 2001)]. Nonetheless, research in schema matching for XML data has been gaining increasing importance in the past few years due to the unprecedented abundant use of XML, especially on the web. Different kinds of approaches for comparing and matching XML grammars have been proposed.

[1] These are operators utilized in DTDs to specify constraints on the existence, repeatability and alternativeness of elements/attributes. With constraint operators, it is possible to specify whether an element is optional ('?'), may occur several times ('*' for 0 or more times and '+' for 1 or more times), some sub-elements are alternative w.r.t. each other ('|' representing the *Or* operator), or are grouped in a sequence (',' representing the *And* operator).

LSD: Among the early schema matching approaches to treat XML grammars is *LSD* (Learning Source Description) (Doan A., Domingos P. and Halevy A.Y., 2001). It employs machine learning techniques to semi-automatically find mappings between two schemas. A simplified representation of the system's architecture is provided in Figure 4. *LSD* works in two phases: a *training phase* and a *mapping phase*. For the training phase, the system asks the user to provide the mappings for a small set of data sources, and then uses these mappings to train its set of learners. Different types of learners can be integrated in the system to detect different types of similarities (e.g. *name matcher* which identifies mappings between XML elements/attributes w.r.t. their name similarities: semantic similarities – synonyms, hyponyms, … – string edit distance, …). The scores produced by the individual learners are combined via a meta-learner, to obtain a single matching score for each pair of match candidates. Once the learners and the meta-learner have been trained (i.e. the training phase), new schemas can be applied to the system to produce mappings (i.e. the matching phase). A special learner is introduced in LSD to take into account the structure of XML data: the *XML learner*. In addition, LSD incorporates domain constraints as an additional source of knowledge to further improve matching results. LSD's main advantage is its extensibility to additional learners that can detect new kinds of similarities (e.g. similarities between data instances corresponding to the compared schema elements, learners that consider thesauri information …) (Doan A., Domingos P. and Halevy A.Y., 2001). However, its main drawback remains in its *training phase* which could require substantial manual effort prior to launching the matching process.

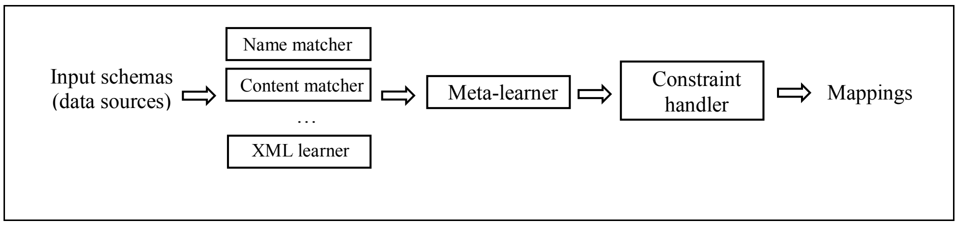

Fig. 1. Simplified representation of *LSD*'s architecture (Doan A., Domingos P. and Halevy A.Y., 2001).

In contrast with the machine learning-based method in (Doan A., Domingos P. and Halevy A.Y., 2001), most XML schema matching approaches employ graph-based schema matching techniques, thus overcoming the expensive pre-match learning effort.

DTD Syntactic Similarity: In (Su H., Padmanabhan S. and Lo M.L., 2001), the authors propose a method to identify syntactically similar DTDs. DTDs are simplified by eliminating null element constraints ('?' is disregarded while '*' is replaced by '+') and flattening complex elements (e.g. '((a, b+) | c)' is replaced by 'a, b+, c', the *Or* operator '|' being disregarded). In addition, sub-elements with the same name are merged to further simplify the corresponding DTDs. A DTD is represented as a rooted directed acyclic graph *G* where each element *e* of the DTD underlines a sub-graph *G(e)* of *G*. A bottom-up procedure starts by matching leaf nodes, based on their names, and subsequently identifies inner-node matches. While leaf node matching is based on name equality, matching inner-nodes comes down to computing a graph distance between the graph structures corresponding to the elements at hand. Intuitively, the graph distance, following (Su H., Padmanabhan S. and Lo M.L., 2001), represents the number of overlapping vertices in the two graphs being compared. Thus, in short, matching two DTDs D_1 and D_2 in (Su H., Padmanabhan S. and Lo M.L., 2001) comes down to matching two elements graphs $G(r_1)$ and $G(r_2)$, where r_1 and r_2 are the root elements of D_1 and D_2 respectively. Note that (Su H., Padmanabhan S. and Lo M.L., 2001)'s approach treats recursive element declarations (e.g. <!ELEMENT a (a)>).

Cupid: In (Madhavan J., Bernstein P. and Rahm E., 2001), the authors propose *Cupid*, a generic schema matching algorithm that discovers mappings between schema elements based on linguistic and structural features. In the same spirit as (Su H., Padmanabhan S. and Lo M.L., 2001), schemas are represented as graphs. A bottom-up approach starts by computing leaf node similarities based on the linguistic similarities between their names (making use of an external thesaurus to acquire semantic similarity scores) as well as their data-type compatibility (making use of an auxiliary data-type compatibility table assigning scores to data-type correspondences, e.g. *Integer*/*Decimal* are assigned a higher score than *Integer*/*String*). Consequently, the algorithm utilizes the leaf node similarity scores to compute the structural similarity of their parents in the schema hierarchy. A special feature of (Madhavan J., Bernstein P. and Rahm E., 2001)'s approach is its recursive nature. While two elements are similar if their leaf sets are similar, the similarity of the leaves is also increased if their ancestors are similar.

Similarity Flooding: In (Melnik S., Garcia-Molina H. and Rahm E., 2002), the authors propose to convert XML schemas into *directed labeled graphs* where each entity represents an element or attribute identified by its full path (e.g. /element_1/subelement_{11}/subelement_{111}) and each literal represents the corresponding element/attribute name or a primitive type (e.g. xsd:string). From the graphs of the pair of schemas being compared, a *pair-wise connectivity graph* (PCG) made of node pairs, one for each graph, is constructed. Consequently, an initial similarity score, using classic string matching (i.e. string edit distance) between node labels, is computed for each node pair contained in the PCG. The initial similarities are then refined by propagating the scores to their adjacent nodes. Finally, a filter is applied to produce the best matches between the elements/attributes of the two schemas. In fact, the approach in (Melnik S., Garcia-Molina H. and Rahm E., 2002) is based on the intuition that elements of two schemas are similar if their adjacent nodes are similar.

Coma: A platform to combine multiple matchers in a flexible way, entitled *Coma*, is provided in (Do H.H. and Rahm E., 2002). In schema matching terms, *Coma* is comparable to

LSD (Doan A., Domingos P. and Halevy A.Y., 2001), both approaches being regarded as *combinational*, i.e. they combine different types of matchers. Nonetheless, *Coma* is not based on machine learning in comparison with *LSD*. The authors in (Do H.H. and Rahm E., 2002) provide a large spectrum of matchers, introducing a novel matcher aiming at reusing results from previous match operations. They propose different mathematical methods to combine matching scores (e.g. max, average, weighted average, …). The intuition behind the new reuse matcher is that many schemas to be matched are similar to previously matched schemas. Following (Do H.H. and Rahm E., 2002), the use of dictionaries and thesauri already represents such a reuse-oriented approach utilizing confirmed correspondences between schema element names and data-types. Thus, the authors propose to generalize this idea to entire schemas. The reuse operation can be defined as follows: given two match results, $S_1 \leftrightarrow S_2$, and $S_2 \leftrightarrow S_3$, the reuse operation derives a new match result: $S_1 \leftrightarrow S_3$ based on the previously defined matches (e.g. "CustName" ↔ "Name" and "Name" ↔ "ClientName", imply "CustName" ↔ "ClientName"). *Coma* could be completely automatic or iterative with user feedback. Aggregation functions, similar to those utilized to combine matching scores from individual matchers, are employed to attain an overall similarity score between the two schemas being compared.

XClust: Despite their differences, the approaches above map an XML grammar (i.e. a DTD or XML Schema) into an internal schema representation proper to the system at hand. These internal schemas are more similar to data-guides (Goldman R. and Widom J., 1997) for semi-structured data than to DTDs or XML Schemas. In other words, constraints on the repeatability and alternativeness of elements (underlined by the '?', '*', +' as well as the *And* ',' and *Or* '|' operators in DTDs for instance) are disregarded in performing the structural match. An approach, called *XClust,* taking into account such constraints is proposed in (Lee

M., Yang L., Hsu W. and Yang X., 2002). The authors of *XClust* develop an integration strategy that involves the **clustering of DTDs**. The proposed algorithm is based on the semantic similarities between element names (making use of a thesauri or ontology), corresponding immediate descendents as well as sub-tree leaf nodes (i.e. leaf nodes corresponding to the sub-trees rooted at the elements being compared). It analyses element by element, going through their semantic/descendent/leaf node characteristics, and considers their corresponding cardinalities (i.e. '?', '*' and '+') in computing the similarity scores. While the internal representation of DTDs, with *XClust*, is more sophisticated than the systems presented above, it does not consider DTDs that specify alternative elements (i.e. the *Or* '|' operator is disregarded).

DTD-Diff: An original approach, entitled *DTD-Diff*, to identify the changes between two DTDs is proposed in (Leonardi E., Hoai T.T., Bhowmick S.S. and Madria S., 2006). It takes into account DTD features, such as element/attribute constraints (i.e. '?', '+', '*' for elements and "Required", "Implied", "Fixed" for attributes), alternative element declarations (via the *Or* '|' operator) as well as entities. DTDs are presented as sets of element type declarations *El* (e.g. <*!ELEMENT* $ele_1(ele_2, ele_3)$>), attribute declarations *A* (e.g. <*ATTLIST* ele_1 att_1 *CDATA*>) and entity declarations *En* (e.g. <*!ENTITY* ent_1 *"..."*>). Specific types of changes are defined w.r.t. to each type of declaration. For instance, insert/delete element declaration (for the whole declaration), insert/delete leaf nodes in the element declaration (e.g. $Del_{Leaf}(ele_2)$ in <*ELEMENT* $ele_1(ele_2, ele_3)$>), insert/delete a sub-tree in the element declaration (e.g. $Del_{SubTree}(ele_2, ele_3)$ in <*ELEMENT* $ele_1(ele_2, ele_3)$>), … The algorithm takes as input two DTDs $D_1=(El_1, A_1, En_1)$ and $D_2=(El_2, A_2, En_2)$ representing old and new versions of a DTD, and returns a script underlining the differences between D_1 and D_2. The authors in (Leonardi E., Hoai T.T., Bhowmick S.S. and Madria S., 2006) show that their approach yields near optimal results when the percentage of change between the DTDs being treated is higher to 5%. That is because, in some cases, some operations might be confused with others, e.g. the move operation might be detected as a pair of deletion and insertion operation.

A taxonomy covering the various XML grammar comparison approaches is presented in Table 1.

DISCUSSION

It would be interesting to identify which of the proposed matching techniques performs best, i.e. can reduce the manual work required for the match task at hand [Do *et al.* 2002]. Nonetheless, evaluations for **schema matching** methods were usually done using diverse methodologies, metrics, and data making it difficult to assess the effectiveness of each single system (Do H.H., Melnik S. and Rahm E., 2002). In addition, the systems are usually not publicly available to apply them to a common test problem or benchmark in order to obtain a direct quantitative comparison. In [(Rahm E. and Bernstein P.A., 2001), (Do H.H., Melnik S. and Rahm E., 2002)], the authors review existing generic schema matching approaches without comparing them. In (Do H.H., Melnik S. and Rahm E., 2002), the authors attempt to assess the experiments conducted in each study to improve the documentation of future schema matching evaluations, in order to facilitate the comparison between different approaches.

Here, in an attempt to roughly compare different XML-related matching methods, we focus on the initial criteria to evaluate the performance of schema matching systems: the manual work required for the match task. This criterion depends on two main factors: i) the level of simplification in the representation of the schema, and ii) the combination of various match techniques to perform the match task.

Level of Simplification

While most methods consider, in different ways, the linguistic as well as the structural aspects of XML grammars, they generally differ in the internal representations of the grammars. In other words, different simplifications are required by different systems inducing simplified schema representations upon which the matching process is executed. These simplifications usually target constraints on the repeatability and alternativeness of elements (e.g. '+', '*', … in DTDs). Hence, we identify a correspondence between the level of simplification in the grammar representations and the amount of manual work required for the match task: *the more schemas are simplified, the more manual work is required to update the match results by considering the simplified constraints*. For instance, if the *Or* operator was replaced by the *And* operator in the simplified representation of a given schema (e.g. *(a | b)* was replaced by *(a , b)* in a given DTD element declaration), the user has to analyze the results produced by the system, manually re-evaluating and updating the matches corresponding to elements that were actually linked by the *Or* operator (i.e. alternatives) prior to the simplification phase. Following this logic, *XClust* (Lee M., Yang L., Hsu W. and Yang X., 2002) seems more sophisticated than previous matching systems in comparing XML grammars (particularly DTDs) since it induces the least simplifications to the grammars being compared: it only disregards the *Or* operator. On the other hand, *DTD-Diff* (Leonardi E., Hoai T.T., Bhowmick S.S. and Madria S., 2006), which is originally a DTD change detection system, could be adapted/updated to attain an effective schema matching/comparison tool since it considers the various constraints of DTDs, not inducing any simplifications.

Combination of Different Match Criterions

On the other hand, the amount of user effort required to effectively perform the match task can also be alleviated by the combination of several **matchers** (Do H.H. and Rahm E., 2002), i.e. the execution of several matching techniques that capture the correspondences between schema elements from different perspectives. In other words, the schemas are assessed from different angles, via multiple match algorithms, the results being combined to attain the best matches possible. For this purpose, existing methods have allowed a so-called *hybrid* or *composite* combination of matching techniques (Rahm E. and Bernstein P.A., 2001). A hybrid approach is such as various matching criteria are used within a single algorithm. In general, these criteria (e.g., element name, data type, …) are fixed and used in a specific way. In contrast, a composite matching approach combines the results of several independently executed matching algorithms (which can be simple or hybrid).

Hypothetically, hybrid approaches should provide better match candidates and better performance than the separate execution of multiple **matchers** (Rahm E. and Bernstein P.A., 2001). Superior matching quality should be achieved since hybrid approaches are developed in specific contexts and target specific features which could be overlooked by more generic combinational methods. It is important to note that hybrid approaches usually provide better performance by reducing the number of passes over the schemas. Instead of going over the schema elements multiple times to test each matching criterion, such as with combinational approaches, hybrid methods allow multiple criterions to be evaluated simultaneously on each element before continuing with the next one.

On the other hand, combinational approaches provide more flexibility in performing the matching, as is it possible to select, add or remove different algorithms following the matching task at hand. Selecting the matchers to be utilized and their execution order could be done either automatically or manually by a human user. An automatic approach would reduce the number of user interactions, thus improving system performance. However, with a fully automated approach, it would be difficult to achieve a generic solution adaptable to different application domains.

In the context of XML, we know of two approaches that follow the composite matching logic, *LSD* (Doan A., Domingos P. and Halevy A.Y., 2001) and *Coma* (Do H.H. and Rahm E., 2002), whereas remaining matching methods are hybrid (e.g., *Cupid* (Madhavan J., Bernstein P. and Rahm E., 2001), *XClust* (Lee M., Yang L., Hsu W. and Yang X., 2002), …). While *LSD* is limited to matching techniques based on machine learning, *Coma* (Do H.H. and Rahm E., 2002) underlines a more generic framework for schema matching, providing various mathematical formulations and strategies to combine matching results.

Therefore, in the context of XML, effective **grammar matching**, minimizing the amount of manual work required to perform the match task, requires:

- Considering the various characteristics and constraints of the XML grammars being matched, in comparison with existing 'grammar simplifying' approaches.
- Providing a framework for combining different matching criterions, which is both *i)* flexible, in comparison with existing static *hybrid* methods, and *ii)* more suitable and adapted to XML-based data, in comparison with the relatively generic *combinational* approaches.

CONCLUSION

As the web continues to grow and evolve, more and more information is being placed in structurally rich documents, particularly XML. This structural information is an important clue as to the meaning of documents, and can be exploited in various ways to improve data management. In this paper, we focus on XML grammar structural similarity. We gave a brief overview of existing research related to XML structural comparison at the type layer. We tried to compare the various approaches, in an attempt to identify those that are most adapted to the task at hand.

Despite the considerable work that has been conducted around the XML grammar structural similarity problem, various issues are yet to be addressed.

In general, most XML-related schema matching approaches in the literature are developed for generic schemas and are consequently adapted to XML grammars. As a result, they usually induce certain simplifications to XML grammars in order to perform the matching task. In particular, constraints on the repeatability and alternativeness of XML elements are usually disregarded. On the other hand, existing methods usually exploit individual matching criterions to identify similarities, and thus do not capture all element resemblances. Those methods that do consider several matching criteria utilize machine learning techniques or basic mathematical formulations (e.g., *max*, *average*, …) which are usually not adapted to XML-based data in combining the results of the different matchers. Hence, the effective combination of various matching criterions in performing the match task, while considering the specific aspects and characteristics of XML grammars, remains one of the major challenges in building an efficient XML grammar matching method.

Moreover, providing a unified method to model grammas (i.e., DTDs and/or XML schemas) would help in reducing the gaps between related approaches and in developing XML grammar similarity methods that are more easily comparable.

On the other hand, since XML grammars represent structured information, it would be interesting to exploit the well known tree edit distance technique, thoroughly investigated in XML document comparison, in comparing XML schemas.

Finally, we hope that the unified presentation of XML grammar similarity in this paper would facilitate further research on the subject.

Tab. 1. XML grammar structural similarity approaches.

Approach	Features	Applications
Doan *et al.*, 2001 - *LSD*	– Based on machine learning techniques – Combines several types of learners – Requires a training phase	Schema matching (data integration)
Su *et al.* 2001 - *DTD Syntactic Similarity*	– DTDs are represented as rooted directed acyclic graphs – Null element constrains and alternative elements are disregarded. – Button-up matching process, starting at leaf nodes and propagating to inner ones.	Schema matching (data integration)
Madhavan *et al.* 2001 - *Cupid*	– Schemas are represented as graphs – Bottom-up matching procedure – Matching based on linguistic similarity and data type compatibility	Generic schema matching
Melnik *et al.* 2002 - *Similarity Flooding*	– Schemas are represented as graphs – Construction of a pair-wise connectivity graph – Similarity scores are computed between pairs of labels and propagated to neighbors in the graph	Schema matching (data integration)
Do and Rahm 2002 - *Coma*	– Platform to combine various matchers – Use of mathematical formulations in comparison with LSD based on *ML* – Introduction of the reuse matcher	Schema matching (data integration)
Lee *et al.* 2002 - *XClust*	– Clustering of DTDs – Considers the various DTD constraints (‘?’, ‘*’, ‘+’) to the exception of the Or operator – Based on the semantic similarity between element names, immediate descendents and sub-tree leaf nodes	DTD Clustering
Leonardi *et al.* 2006 – *DTD-Diff*	– Detects the changes between two DTDs – DTDs represented as sets of element/attribute/entity declarations – Specific types of changes are defined w.r.t. each type of declaration – Considers the various DTD constraints on the repeatability and alternativeness of elements	Maintenance of XML documents and DTD evolution

REFERENCES

Castano S., De Antonellis V. and Vimercati S. (2001). Global Viewing of Heterogeneous Data Sources. *IEEE TKDE* 13(2), pp. 277–297.

Chawathe S., Rajaraman A., Garcia-Molina H., and Widom J. (1996). Change Detection in Hierarchically Structured Information. In *Proceedings of ACM SIGMOD,* pp. 493-504.

Cobéna G., Abiteboul S. and Marian A. (2002). Detecting Changes in XML Documents. In *Proceedings of the IEEE Int. Conf. on Data Engineering*, pp. 41-52.

Dalamagas, T., Cheng, T., Winkel, K., and Sellis, T. (2006). A methodology for clustering XML documents by structure. *Information Systems*, 31(3), pp. 187-228.

Do H.H., Melnik S. and Rahm E. (2002). Comparison of Schema Matching Evaluations, In *Proceedings of the GI-Workshop on the "Web and Databases"*, pp. 221-237.

Do H.H. and Rahm E (2002). COMA: A System for Flexible Combination of Schema Matching Approaches. In *Proceedings of the 28th VLDB Conference*, pp. 610-621.

Doan A., Domingos P. and Halevy A.Y. (2001). Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. In *Proc. of ACM SIGMOD,* pp. 509-520.

Goldman R. and Widom J. (1997). Dataguides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the 23rd VLDB Conference*, Athens, pp. 436-445.

Larson J., Navathe S.B. and Elmasri R. (1989). Theory of Attribute Equivalence and its Applications to Schema Integration. *IEEE Transactions on Software Engineering*, 15(4), pp. 449-463.

Lee M., Yang L., Hsu W. and Yang X. (2002). XClust: Clustering XML Schemas for Effective Integration. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM)*, pp. 292-299.

Leonardi E., Hoai T.T., Bhowmick S.S. and Madria S. (2006). DTD-Diff: A Change Detection Algorithm for DTDs. *Data Knowledge Engineering, 61(2)*, pp. 384-402.

Madhavan J., Bernstein P. and Rahm E. (2001). Generic Schema Matching With Cupid. In *Proceedings of the 27th VLDB Conference*, pp. 49-58.

Melnik S., Garcia-Molina H. and Rahm E. (2002). Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *Proceedings of the 18th ICDE Conference*, pp. 117-128.

Milo T. and Zohar S. (1999). Using Schema Matching to simplify Heterogeneous Data Translation. In *Proc. of the 25th VLDB Conference*, pp. 122-133.

Nierman A. and Jagadish H. V. (2002). Evaluating structural similarity in XML documents. In *Proceedings of the 5th SIGMOD WebDB Workshop,* pp. 61-66.

Rahm E. and Bernstein P.A. (2001). A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, (10), pp. 334-350.

Schlieder T. (2001). Similarity Search in XML Data Using Cost-based Query Transformations. *In Proceedings of 4th SIGMOD WebDB Workshop*, pp. 19-24.

Su H., Padmanabhan S. and Lo M.L. (2001). Identification of Syntactically Similar DTD Elements for Schema Matching. In *Proceedings of the Second International Conference on Advances in Web-Age Information Management*, pp. 145-159.

Zhang Z., Li R., Cao S. and Zhu Y. (2003). Similarity Metric in XML Documents. *Knowledge Management and Experience Management Workshop*.

TERMS AND DEFINITIONS

XML: eXtensible Markup Language. Developed by WWW consortium in 1998, it has been established as a standard for the representation and management of data published on the web. Its application domains range over database information interchange, web services interaction and multimedia data storage and retrieval.

XML tree: XML documents represent hierarchically structured information and can be modeled as trees, i.e., Ordered Labeled Trees. Nodes, in an XML tree, represent XML elements/attributes and are labeled with corresponding element tag names.

XML Grammar: It is a structure specifying the elements and attributes of an XML document, as well as how these elements/attributes interact together in the document (e.g., DTD – Document Type Definition or XML Schema).

Schema Matching: It is generally viewed as the process of finding correspondences between elements of two schemas/grammars. The schema matching operator can be defined as a function that takes two schemas, S_1 and S_2, as input and returns a mapping between those schemas as output.

Simple Matcher: It is a process that identifies mappings between schema/grammar elements based on a single specific criterion, e.g., element name syntactic similarity, element repeatability constraints…

Hybrid Matcher: It is a matcher that combines multiple criterions within a single algorithm. In general, these criterions (e.g., element name, data type, repeatability constraints…) are fixed and used in a specific way.

Composite Matcher: It combines the results of several independently executed matching algorithms (which can be simple or hybrid). It generally provides more flexibility in performing the matching, as is it possible to select, add or remove different matching algorithms following the matching task at hand.