Breakthroughs and Limitations of XML Grammar Similarity

Joe Tekli
University of Bourgogne, France

Richard Chbeir
University of Bourgogne, France

Kokou Yetongnon
University of Bourgogne, France

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 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, Rajaraman, Garcia-Molina, & Widom, 1996; Cobéna, Abiteboul, & Marian, 2002), XML structural query systems (finding and ranking results according to their similarity) (Schlieder, 2001; Zhang, Li, Cao, & Zhu, 2003) as well as the structural clustering of XML documents gathered from the Web (Dalamagas, Cheng, Winkel, & Sellis, 2006; Nierman & Jagadish, 2002). 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, Domingos, & Halevy, 2001; Melnik, Garcia-Molina, & Rahm, 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 & Bernstein, 2001).

The goal of this article 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 article is organized as follows. The second section presents an overview of XML grammar similarity, otherwise known as **XML schema matching**. The third section reviews the state of the art in XML grammar comparison methods. The fourth section discusses the main criterions characterizing the effectiveness of XML grammar similarity approaches. Conclusions and current research directions are covered in the last section.

OVERVIEW

Identifying the similarities among grammars/schemas[5], otherwise known as **schema matching** (i.e., **XML schema matching** with respect to XML grammars), is usually viewed as the task of finding correspondences between elements of two schemas (Do, Melnik, & Rahm, 2002). It has been investigated in various fields, mainly in the context of **data integration** (Do et al., 2002; Rahm & Bernstein, 2001), and recently in the contexts of **schema clustering** (Lee, Yang, Hsu, &

Yang, 2002) and **change detection** (Leonardi, Hoai, Bhowmick, & Madria, 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, etc.). 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 & Bernstein, 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.

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, etc.), similarity between element constraints (e.g., '?', '*' and '+' in DTDs[6]), 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 et al., 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 et al. (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 (Castano, De Antonellis, & Vimercati, 2001; Larson, Navathe, & Elmasri, 1989; Milo & Zohar, 1999). 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.

LSD: Among the early schema matching approaches to treat XML grammars is *LSD* (Learning Source Description) (Doan et al., 2001). It employs machine learning techniques to semiautomatically 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 with respect to their name similarities: semantic similarities – synonyms, hyponyms, etc., string edit distance, etc.). 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, etc.) (Doan et al., 2001). However, its main drawback remains in its *training phase* which could require substantial manual effort prior to launching the matching process.

In contrast with the machine learning-based method in Doan et al. (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, Padmanabhan, and Lo (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