Discovering and Maintaining Semantic Mappings between XML Schemas and Ontologies

Yuan An
Drexel University, USA
yuan.an@ischool.drexel.edu

Alex Borgida
Rutgers University, USA
borgida@cs.rutgers.edu

John Mylopoulos
University of Toronto, Canada
jm@cs.toronto.edu

There is general agreement that the problem of data semantics has to be addressed for XML data to become machine-processable. This problem can be tackled by defining a *semantic mapping* between an XML schema and an ontology. Unfortunately, creating such mappings is a tedious, time-consuming, and error-prone task. To alleviate this problem, we present a solution that heuristically discovers semantic mappings between XML schemas and ontologies. The solution takes as input an initial set of simple correspondences between element attributes in an XML schema and class attributes in an ontology, and then generates a set of mapping formulas. Once such a mapping is created, it is important and necessary to maintain the consistency of the mapping when the associated XML schema and ontology evolve. In this paper, we first offer a mapping formalism to represent semantic mappings. Second, we present our heuristic mapping discovery algorithm. Third, we show through an empirical study that considerable effort can be saved when discovering complex mappings by using our prototype tool. Finally, we propose a mapping maintenance plan dealing with schema evolution. Our study provides a set of effective solutions for building sustainable semantic integration systems for XML data.

Categories and Subject Descriptors: Database Management [**Heterogeneous Databases**]:

General Terms: Algorithm and Experiment
Additional Key Words and Phrases: Schema Mapping, XML Data, Ontology, Data Integration, Semantic Web

---

1. INTRODUCTION

There is explosive growth in the amount of XML data published on the Web since XML is becoming a standard format for information exchange on the Web. However, because of the heterogeneity in structures and vocabularies, XML does not support well data integration. To resolve the heterogeneity problem, we need to understand the semantics of XML data. Formal ontologies, as shared conceptualizations of specific domains, carry precise semantics. Capturing the semantics of XML data in terms of ontologies therefore provides a means for integrating heterogeneous XML data sources. For many XML documents satisfying patterns expressed in DTD or XML schema, the semantics can be captured in a formal way, through a semantic mapping relating parts of the schema with logical formulas over predicates introduced by an ontology. In this paper, we study *a heuristic solution to defining and maintaining complex semantic mappings from XML schemas to ontologies*.

Although mappings from XML schemas to ontologies could be as simple as one-to-one correspondences between their constituent parts, in most applications, complex expressions are needed to relate non-trivial structures within corresponding XML schema and ontology. For example, mappings in [Amann et al. 2002; Lakshmanan and Sadri 2003] essentially connect paths in XML to chains of properties in an ontology. Moreinteresting applications of complex mappings can be found in areas such as data integration, as well as peer-to-peer data management systems [Halevy et al. 2003]. Until now, it has been assumed that *humans* specify these complex mapping formulas - a highly non-trivial process, especially since the specifier must be familiar with both the XML schema and the ontology. Since many XML schemas and ontologies are very complicated artifacts, often containing thousands of terms, the entire process is time-consuming and error-prone, and hence calls for support in the form of automated tools.

In this paper, we describe in detail a tool [An et al. 2005a] that assists users in the construction of complex mapping formulas between XML schemas and ontologies, expressed in a subset of First Order Logic. We are motivated by a number of important problems on XML data management including annotating XML data in terms of ontologies, translating XML data into ontologies, and integrating heterogeneous XML data on the Semantic Web. A typical scenario is that the owner of a website (e.g., an online bookstore) wants to make the XML data (e.g., book catalogs) published in the website to be understandable by a software agent (e.g., a book search engine) committed to an existing ontology. A solution is to map the XML data to the ontology used by the software agent.

In an open, dynamic, and distributed information environment such as the Web, information sources are constantly evolving. After semantic mappings between XML schemas and ontologies have been created, it is important and necessary to maintain the consistency of the semantic mappings when schemas or ontologies evolve. In this paper, we extend our ealier work [An et al. 2005a] by proposing strategies for maintaining the mappings under the evolution of XML schema.

```
<docs>
  <article title="The Semantic Web">
    <author id="001">
      <name fn="Tim" ln="Berners-Lee"/>
    </author>
    <author id="002">
      <name fn="James" ln="Hendler"/>
    </author>
    <author id="003">
      <name fn="Ora" ln="Lassila"/>
    </author>
    <contactauthor authorid="001"/>
  </aritle>
  <article title="Toward Principles for Design of Onto. used for
                       Knowl. Sharing">
    <author id="004">
      <name fn="Thomas" ln="Gruber"/>
    </author>
    <contactauthor authorid="004"/>
  </article>
</docs>
```

Fig. 1. An XML document.

1.1 Illustrative Examples

Inspired by the success of the *Clio* tool [Miller et al. 2000; Popa et al. 2002], our tool takes three inputs: an ontology, an XML schema (actually, its unfolding into tree structures that we will call *element trees*), and simple correspondences between XML attributes/"leafs" and ontology datatype properties, of the kind possibly generated by already existing tools (e.g., [Dhamankar et al. 2004; Madhavan et al. 2001; Melnik et al. 2002]). The output is a list of complex formulas representing semantic mappings. The following example illustrates the input/output of the proposed tool.

Example 1.1. *Consider an XML document containing information about articles, authors, and contact authors of articles as shown in Figure 1. The document satisfies the XML Schema specification given in Figure 2 (omitting the definition for the docs element). We want to discover and express the semantic of the XML schema in terms of the ontology in Figure 3. The ontology is described as a UML class diagram.*

Our goal is to produce a logic formula that defines the semantics encoded in the structure of the XML schema. This semantics is expressed by the concepts, attributes, and associations in the ontology. To increase the likelihood of discovering the correct semantics, we assume that some extra information is available as part of the input. This information is specified as a set of simple correspondences between attributes[1] *in the XML schema and datatype properties/attributes in the ontology. The correspondences can be specified by a user or generated by schema matching tools.*

Specifically, we use the following notation to represent the simple correspondences between attributes of the XML schema and datatype properties of the ontology, where the prefixes $\mathcal{X}$ and $\mathcal{O}$ distinguish terms in the XML schema and the ontology.

[1]Include simple type elements, see Section 3.1.

$\mathcal{X}$:*article.@title*$\leftrightsquigarrow\mathcal{O}$:Article.hasTitle
$\mathcal{X}$:*article.author.@id*$\leftrightsquigarrow\mathcal{O}$:Author.hasID
$\mathcal{X}$:*article.author.name.@fn*$\leftrightsquigarrow\mathcal{O}$:Author.hasFirstname
$\mathcal{X}$:*article.author.name.@ln*$\leftrightsquigarrow\mathcal{O}$:Author.hasLastname
$\mathcal{X}$:*article.contactauthor.@authorid*$\leftrightsquigarrow\mathcal{O}$:Author.hasID

The expression on the left-hand side of a correspondence is a path from the root to an attribute in the XML schema, while the right-hand side indicates a datatype property of a concept in the ontology. We will give a formal definition of a correspondence in Section 3.

Having as input the XML schema, the ontology, and the set of such simple correspondences, we expect our tool to generate a set of logical formulas which includes the following one, expressing a possible semantics of the XML schema in terms of the ontology:

$$\begin{array}{ll}
\textit{article}(@\textit{title} = x_1)[& \\
\quad \textit{author}\ (@\textit{id} = x_2)[& \\
\quad\quad \textit{name}\ (@\textit{fn} = x_3,\ @\textit{ln} = x_4)], & \\
\quad \textit{contactauthor}(@\textit{authorid} = x_5)]] \quad \rightarrow & \mathsf{Article}(Y_1), \mathsf{hasTitle}(Y_1, x_1), \\
 & \mathsf{Author}(Y_2), \mathsf{hasID}(Y_2, x_2), \\
 & \mathsf{hasAuthor}(Y_1, Y_2), \\
 & \mathsf{hasFirstname}(Y_2, x_3), \\
 & \mathsf{hasLastname}(Y_2, x_4), \\
 & \mathsf{Author}(Y_3), \mathsf{hasID}(Y_3, x_5), \\
 & \mathsf{Contactauthor}(Y_1, Y_3).
\end{array}$$

In the above mapping formula, the left-hand side expression is a tree-pattern formula which is defined in Section 3.3.

The following example illustrates the problem of maintaining a semantic mapping when the associated XML schema evolves and provides a glimpse of our maintenance plan.

Example 1.2. *A semantic mapping such as the one created in the previous example relates a subgraph of an XML schema with a subgraph of an ontology graph. These two subgraphs should be semantically consistent, i.e., the constraints specified among a set of elements in the XML schema are consistent with the constraints specified among the corresponding set of elements in the ontology.*

Once such a mapping is created, it is important to maintain the consistency when the related schema or ontology evolves. Changes to schemas and ontologies include deletion and addition of elements, restructuring, and constraint update. We aim at incrementally maintaining the consistency of a semantic mapping instead of rediscovering a new semantic mapping when changes happen. For example, for the schema in Figure 2, if a new element @title *is added under the* article *element and the* @title *attribute becomes an attribute of the title element, we only need to update the left-hand side of the existing semantic mapping formula and keep the right-hand side intact. There is no need to execute the mapping discovery algorithm to generate a new mapping formula. A systematic mapping maintenance strategy is presented later.*

```
<xsd:element name="article" type="articleType"/>
  <xsd:complexType name="articleType">
    <xsd:sequence>
      <xsd:element name="author">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" minOccurs="1"
                maxOccurs="1">
              <xsd:complexType>
                <xsd:attribute name="fn"
                 type="xsd:string" use="required"/>
                <xsd:attribute name="ln"
                 type="xsd:string" use="optional"/>
              </xsd:complexType>
            </xsd:element>
          </sequence>
          <xsd:attribute name="id" type="xsd:integer"
           use="required" />
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="contactauthor" minOccurs="1"
          maxOccurs="1">
        <xsd:complexType>
          <xsd:attribute name="authorid"
           type="xsd:integer" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="title" type="xsd:string"
     use="required" />
  </xsd:complexType>
</xsd:element>
```

Fig. 2. An XML schema definition.

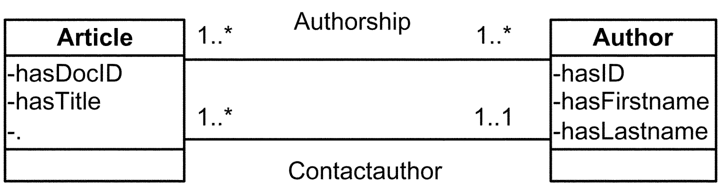

Fig. 3. An ontology.

1.2 Overview

The main contributions of this work are as follows: (i) we propose a mapping formalism to capture the semantics of XML schemas based on tree-pattern formulas [Arenas and Libkin 2005a]; (ii) we propose a heuristic algorithm for finding semantic mappings, which are akin to a tree connection embedded in the ontology; (iii) we enhance the algorithm by taking into account information about (a) XML Schema features such as occurrence constraints, `key` and `keyref` definitions, (b) cardinality constraints in the ontology, and (c) XML document design guidelines under the hypothesis that an explicit or implicit conceptual model existed during the process of XML document design; (iv) we adopt the accuracy metric of schema matching [Melnik et al. 2002] and evaluate the tool with a number of experiments; (v) we develop strategies for maintaining the consistency of a semantic mapping when associated XML schema evolves.

The rest of the paper is organized as follows. Section 2 discusses related work, while Section 3 presents formal notations used later on. Section 4 describes some principles, as well as the mapping discovery algorithm. Section 5 reports on empirical studies. Section 6 studies how to maintain the consistency of a semantic mapping under schema evolution. Finally, Section 7 summarizes the results of this work and suggests future directions.

2. RELATED WORK

Much research has focused on converting and storing XML data into relational databases [Shanmugasundaram et al. 1999]. It is natural to ask whether we could utilize the mapping algorithm we have developed in [An et al. 2005b] – for discovering mappings from relational schemas to ontologies – by first converting XML DTDs/schemas into relational tables. Unfortunately, this approach does not work. Among others, the algorithms that generate a relational schema from an XML DTD use backlinks and system generated *id*s in order to record the nested structure, and these confuse the algorithms in [An et al. 2005b], which rely heavily on key and foreign key information.

The *schema mapping* tool *Clio* [Miller et al. 2000; Popa et al. 2002] discovers formal queries describing how target schemas can be populated with data from source schemas, given sets of simple value correspondences. The present work can be viewed as extending *Clio* to the case when the target schema is an ontology treated as a relational schema consisting of unary and binary tables. However, as argued in [An et al. 2005b], the chase algorithm of *Clio* would not produce the desired mappings due to several reasons: (i) the chase only follows nested referential constraints along one direction, while the intended meaning of an XML element tree may follow a binary relationship along either direction (see also Section 4.1); (ii) *Clio* does not exploit occurrence constraints in the XML schema. These constraints carry important semantic information in searching for “reasonable” connections in the ontology.

The Xyleme [Delobel et al. 2003] project is a comprehensive XML data integration system which includes an automatic mapping generation component. A mapping rule in terms of a pair of paths in two XML data sources is generated based on term matching and structural, context-based constraints. Specifically, terms of paths are first matched syntactically and semantically. Then the structural information is exploited. Our work difiers from it significantly in that we propose to discover the mappings between tree structures in XML data and that in ontologies. The discovery is guided by a forward engineering process.

The problem of *reverse engineering* is to extract a conceptual schema (UML diagram, for example) from an XML DTD/schema [Jensen et al. 2003]. The major difierence between *reverse engineering* and our work is that we are given an existing ontology, and want to interpret the XML data in terms of it, whereas reverse engineering aims to construct a new one.

Schema Matching [Dhamankar et al. 2004; Madhavan et al. 2001; Melnik et al. 2002] identifies semantic relations between schema elements based on their names, data types, constraints, and structures. The primary goal is to find the one-one simple correspondences which are part of the input for our algorithm.

Finally, our earlier work [An et al. 2005a] develops and presents the main ideas of the mapping discovery algorithm. However, it does not consider the mapping maintenance problem.

3. FORMAL PRELIMINARIES

In this section, we define some formal notations used in later sections.

3.1 XML Data Model and XML Schema

An XML document is typically modeled as a node-labeled graph. For our purpose, we assume that each XML document is described by an XML schema consisting of a set of element and attribute type definitions. Specifically, we assume the following countably infinite disjoint sets: **Ele** of element names, **Att** of attribute names, and **Dom** of simple type names including the built-in XML Schema datatypes. Attribute names are preceded by a "@" to distinguish them from element names. Given finite sets $E \subset \mathbf{Ele}$ and $A \subset \mathbf{Att}$, an XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$ specifies the type of each element ℓ in E, the attributes that ℓ has, and the datatype of each attribute in A. Specifically, we use the following abstract syntax to define an XML schema. An element type τ is defined by the grammar $\tau ::= \varepsilon \mid \mathsf{Sequence}[\ell_1 : \tau_1, ... \ell_n : \tau_n] \mid \mathsf{Choice}$ $[\ell_1 : \tau_1, ..., \ell_n : \tau_n]$ for $\ell_1, ..., \ell_n \in E$, where ε stands for the empty type, and Sequence and Choice are complex types. Each element has associated two occurrence constraints: *minOccurs*, indicating the minimum number of occurrence, and *maxOccurs*, indicating the maximum number. (We mark with * multiply occurring elements.) The set of attributes of an element $\ell \in E$ is defined by the function $\rho : E \rightarrow 2^A$; and the function $\kappa : A \rightarrow \mathbf{Dom}$ specifies the datatypes of attributes in A. Each datatype name associates with a set of values in a domain **Dom**. In this paper, we do not consider the *simple type elements* (corresponding to DTD's **PCDATA**), assuming instead that they have been represented using attributes[2]. As usual in XML, attributes are single-valued. Furthermore, a special element $\underline{r} \in E$ is the root of each XML schema, and we assume that for any two element $\ell_i, \ell_j \in E$, $\rho(\ell_i) \cap \rho(\ell_j) = \emptyset$.

For example, the XML schema describing articles and authors in Figure 2 has the following specification, where $(\ell)^\tau$ represents the type of an element ℓ:
$E = \{article,\ author,\ contactauthor,\ name\}$,
$A = \{@title,\ @id,\ @authorid,\ @fn,\ @ln\}$,
$(article)^\tau = \mathsf{Sequence}[(author)* :(author)^\tau,\ contactauthor{:}\varepsilon]$,
$(author)^\tau = \mathsf{Sequence}[name{:}\varepsilon]$,
$\rho(article) = (@title)$, $\rho(author) = (@id)$, $\rho(contactauthor) = (@authorid)$,

[2]Multivalued PCDATA elements are encoded by adding an additional element with one attribute

$\rho(name)$ = (@*fn*,@*ln*), κ(@*title*) = String, κ(@*authorid*) = Integer, κ(@*id*)= Integer, κ(@*fn*)= String, κ(@*ln*)= String, and the element *article* is the root. Note that for the *article* element, *contactauthor* only occurs once, while *author* may occur many times. For the *author* element, *name* occurs once. The XML Schema Language [Fallside and Walmsley 2004] is an expressive language that can also express **key** and **keyref** constraints.

Unlike relational databases where data are stored in relations comprising tuples of values, data in XML documents are organized in graph (tree) structures. An XML document $\mathcal{X} = (N, <, \underline{r}, \lambda, \eta)$ over (E, A) consists of a set of nodes N, a child relation < between nodes, a root node $\underline{r}$, and two functions:

—a labeling function $\lambda: N \rightarrow E \cup A$ such that if $\lambda(v) = \ell \in E$, we say that v is in the element type ℓ; if $\lambda(v) = @a \in A$, we say that v is an attribute $@a$;
—a partial function $\eta: N \rightarrow Dom$ for every node v with $\lambda(v) = @a \in A$, assigning values in domain Dom that supplies values to simple type names in **Dom**.

An XML document $\mathcal{X} = (N, <, \underline{r}, \lambda, \eta)$ conforms to a schema $\mathcal{S} = (E, A, \tau, \rho, k)$, denoted by $\mathcal{X} \models \mathcal{S}$, if:

(1) for every node v in $\mathcal{X}$ with children $v_1, ..., v_m$ such that $\lambda(v_i) \in E$ for $i = 1, ..., m$, if $\lambda(v) = \ell$, then $\lambda(v_1),..., \lambda(v_m)$ satis˙es $\tau(\ell)$ and the occurrence constraints.
(2) for ever node v in $\mathcal{X}$ with children $u_1, ..., u_n$ such that $\lambda(u_i) = @a_i \in A$ for $i = 1, ..., n$, if $\lambda(v) = \ell$, then $\lambda(u_i) = @a_i \in \rho(\ell)$, and $\eta(u_i)$ is a value having datatype $k(@a_i)$.

An XML schema can be viewed as a directed node-labeled graph called *schema graph* consisting of the following edges: parent-child edges $e = \ell \rightarrow \ell_i$ for elements ℓ, $\ell_i \in E$ such that if $\tau(\ell)$= Sequence[...$\ell_i : \tau_i$...] or Choice[...$\ell_i : \tau_i$...]; and attribute edges $e = \ell \Rightarrow \alpha$ for element $\ell \in E$ and attribute $\alpha \in A$ such that $\alpha \in \rho(\ell)$. For a parent-child edge $e = \ell \rightarrow \ell_i$, if the *maxOccurs* constraint of ℓ_i is 1, we show the edge to be functional, drawn as $\ell \Rightarrow \ell_i$. Since attributes are single-valued, we always draw an attribute edge as $\ell \Rightarrow \alpha$. The schema graph corresponding to the XML schema in Figure 2 is shown in Figure 4. In the figure, a single-line arrow pointing to a child element from a parent element represents a parent-child relationship where the *maxOccurs* constraint of the child element is multiple, while a double-line arrow

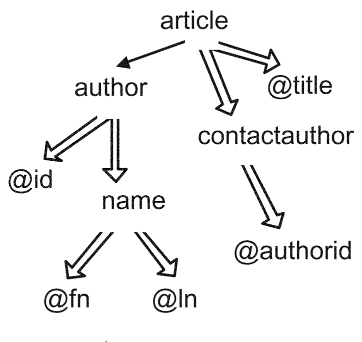

Fig. 4. An XML schema graph.

indicates that the *maxOccurs* constraint of the child element is 1.

Elements and attributes as nodes in a schema graph are located by path expressions. For our purposes, we use a simple path expression $Q = \varepsilon | \ell . Q$ and introduce the notion of *element tree*.

A semantic mapping from an XML schema to an ontology consists of a set of mapping formulas each of which is from an element tree (not a graph!) to a conjunctive formulas in the ontology. An *element tree* can be constructed through a depth first search (DFS), for every node in the element graph. The DFS process first creates an empty element graph, and creates a new node for each unmarked node during the traversal of the original schema graph. Mark each node in the schema graph as "visited" when it is reached the first time and unmarked when all of its descendent's have been traversed. (This has the eflect of duplicating subgraphs pointed at from multiple nodes.) Regular edges are created in the element graph whenever there is a traversal from a DFS parent node to its unmarked children in the original schema graph. If an already marked node is reached, then a "back" edge (using dashed line) is added in the element graph from the DFS parent to this marked child, but the DFS does not follow this edge. For example, Figure 5 (a) shows a schema graph with a cycle and a node with multiple parents. Figure 5 (b), (c), and (d) are the element graphs created by the DFS process starting at the elements controls, employee, and manager, respectively. In the figure, a dashed-line arrow represents a back edge created in the DFS process.

Next, we convert the element graphs into element trees by ignoring or unfolding the back edges, depending on our needs. To unfold a back edge from a node ℓ_i to a node ℓ_j, we connect ℓ_j and all the contents descending ℓ_j until ℓ_i to ℓ_i, and then remove the back edge. The occurrence constraint of the newly created edge from ℓ_i to ℓ_j is the same as that of the back edge. Figure 6 (c) and (d) are the element trees obtained from the element graphs in Figure 5 (c) and (d), respectively, by unfolding the back edges, while Figure 6 (b) is the element tree obtained from the element graph in Figure 5 (b) by ignoring the back edge. For the sake of simplicity, we specify each element tree as rooted at the element from which the tree is constructed, ignoring the path from the root to the element in the original schema graph.

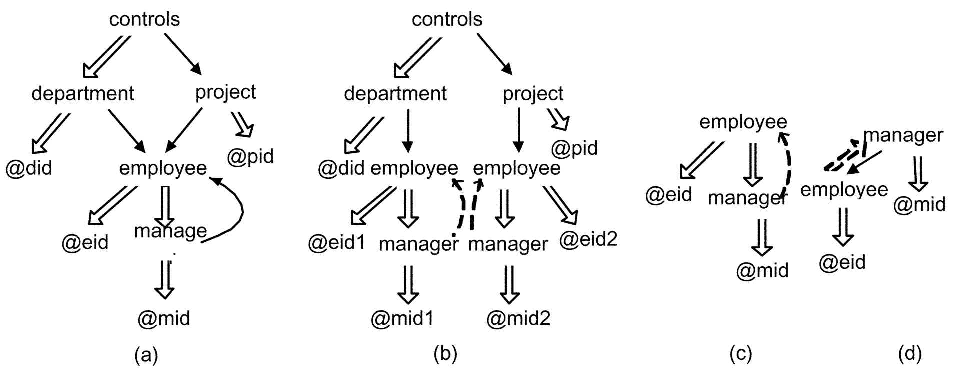

Fig. 5. Schema graph and element graphs.

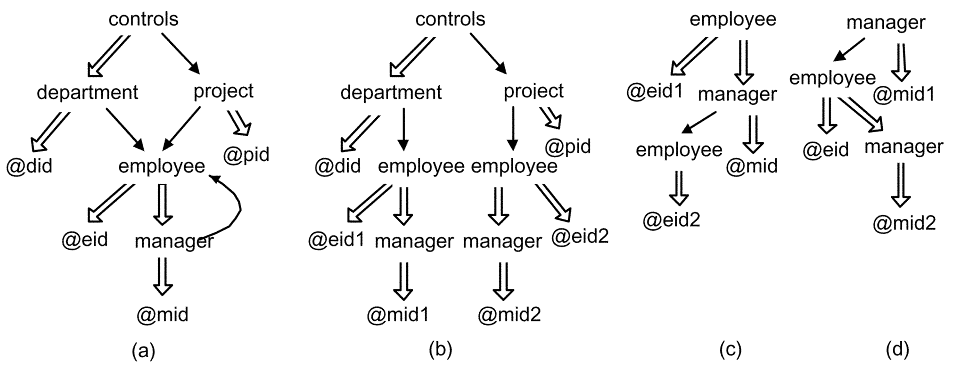

Fig. 6. Schema graph and element trees.

3.2 Ontologies and Ontology Graphs

In this paper, we do not restrict ourselves to any particular language for describing ontologies. Instead, we use a generic conceptual modeling language (CML), which contains *common* aspects of most semantic data models, UML, ontology languages such as OWL, and description logics. Specifically, the language allows the representation of *classes/concepts* (unary predicates over individuals), *object properties/ relationships* (binary predicates relating individuals), and *datatype properties/attributes* (binary predicates relating individuals with values such as integers and strings); attributes are single valued in this paper. Concepts are organized in the familiar ISA hierarchy, and subclasses of a superclass can be either disjoint or overlapping. Relationships, and their inverses (which are always present), are subject to constraints such as specification of domain and range, plus cardinality constraints, which here allow 1 as lower bounds (called *total* relationships), and 1 as upper bounds (called *functional* relationships).

We shall represent a given ontology using a labeled directed graph, called an *ontology graph*. We construct the ontology graph from an ontology as follows: We create a concept node labeled with C for each concept C, and an edge labeled with p from the concept node C_1 to the concept node C_2 for each object property p with domain C_1 and range C_2; for each such p, there is also an edge in the opposite direction for its inverse, referred to as p^{-}. For each attribute f of concept C, we create a separate attribute node denoted as $N_{f,C}$, whose label is f, and add an edge labeled f from node C to $N_{f,C}$. For each ISA edge from a subconcept C_1 to a superconcept C_2, we create an edge labeled with ISA from concept node C_1 to concept node C_2 with cardinality 1..1 on the C_2 side (a C_1 must be a C_2), and 0..1 on the C_1 side. For the sake of succinctness, we sometimes use UML notations, as shown in Figure 3, to represent the ontology graph. Note that in such a diagram, instead of drawing separate attribute nodes, we place the attributes inside the rectangle concept nodes; and relationships and their inverses are represented by a single undirected edge. The presence of such an undirected edge, labeled p, between concepts C and D will be written in text as $\boxed{C}$ ---p--- $\boxed{D}$. It will be important for our approach to distinguish *functional edges* – ones with upper bound cardinality of 1, and their composition:

functional paths. If the relationship p is functional from C to D, we write $\boxed{\text{C}}$ ---p->-- $\boxed{\text{D}}$. For expressive CMLs such as OWL, we may also connect C to D by p if we find an existential restriction stating that each instance of C is related to *some* instance or *only* instances of D by p.

3.3 The Mapping Formalism

In this paper, we attempt to discover a semantic mapping from an XML schema to an ontology, given a set of simple correspondences. A *correspondence* $\mathcal{X}$:P:@$c \leftrightsquigarrow \mathcal{O}$:D.f relates the attribute "@c" of an element ℓ reached by the simple path P in an element tree to the datatype property f of class D in an ontology. A simple path P is always relative to the root of a tree. For example, we can specify the following correspondences for the element tree in Figure 6 (c):

$\mathcal{X}$:*employee*.@*eid*1 $\leftrightsquigarrow$ $\mathcal{O}$:Employee.hasId,
$\mathcal{X}$:*employee.manager*.@*mid* $\leftrightsquigarrow$ $\mathcal{O}$:Employee.hasId.
$\mathcal{X}$:*employee.manager.employee*.@*eid*2 $\leftrightsquigarrow$ $\mathcal{O}$:Employee.hasId

where Employee is a concept in an ontology and hasId is an attribute of the concept Employee. Formally, a correspondence L will be a mathematical relation $L(P, @c, D, f, N_{f,D})$, where the first two arguments determine unique values for the last three.

We now turn to the mapping language relating a formula representing an element tree with a conjunctive formula in an ontology. On the XML side, the basic components are *attribute formulas* [Arenas and Libkin 2005b], which are specified by the syntax $\alpha ::= \ell | \ell(@a_1 = x_1, .., @a_n = x_n)$, where $\ell \in E$, $@a_1, .., @a_n \in A$; E and A are element names and attribute names, respectively, while variables $x_1, .., x_n$ are the free variables of α. Tree-pattern formulas over an XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$ are defined by $\psi ::= \alpha | \alpha[\varphi_1, .., \varphi_n]$, where α ranges over attribute formulas over (E,A). The free variables of a tree formula ψ are the free variables in all the attribute formulas that occur in it. For example,

$employee(@eid1 = x_1)[manager(@mid = x_2)[employee(@eid2 = x_3)]]$

is the tree formula representing the element tree in Figure 6 (c).

A *mapping formula* between an element tree and an ontology then has the form $\Psi(X) \rightarrow \Phi(X, Y)$, where $\Psi(X)$ is a tree formula in the XML schema and $\Phi(X, Y)$ is a conjunctive formula in the ontology. For example, given an ontology containing a concept Employee, with an attribute hasId, and a functional property hasManager (whose inverse is manages, which is not functional), the following mapping formula ascribes a semantics of the element tree in Figure 6 (c):

$$\begin{array}{ll} employee(@eid1 = x_1)[& \\ \quad manager\ (@mid = x_2)[& \\ \qquad employee\ (@eid2 = x_3)\ ]] & \rightarrow \text{Employee}(Y_1), \text{hasId}(Y_1, x_1), \\ & \text{Employee}(Y_2), \text{hasId}(Y_2, x_2), \\ & \text{hasManager}(Y_1, Y_2), \text{Employee}(Y_3), \\ & \text{hasId}(Y_3, x_3), \text{manages}(Y_2, Y_3). \end{array}$$

Since we maintain the unique name assumption for attributes, we can drop the variable names x_is, and just use attribute names in formulas. The variables Y_js are implicitly existentially quantified and refer to individuals in the ontology.

3.4 The Mapping Discovery Problem

Having the formalism for specifying mappings between XML schemas and ontologies, we now turn to the problem of discovering such mappings.

Semantic Mapping Discovery Problem (X-to-O problem). *Given an XML schema $\mathcal{S} = (E, A, \tau, \rho, \kappa)$, an ontology $\mathcal{O}$, and a set of correspondences L from attributes of elements in $\mathcal{S}$ to attributes of concepts/classes in $\mathcal{O}$. For an element tree T, find an association δ_τ in the ontology $\mathcal{O}$ such that T and δ_τ* are "*semantically similar*" *in terms of modeling a subject matter.*

The input of the X-to-O problem is an XML schema, an ontology, and a set of correspondences from attributes of elements in the schema to attributes/datatype properties of concepts in the ontology. An XML document stores attribute values organized into a graph, while an ontology specifies concepts, attributes of concepts, and relationships between concepts. Our solution for discovering the semantic mapping from an XML schema to an ontology exploits the principles that convert a conceptual model into a "good" XML schema. Focusing on semantics discovery, we assume the input XML schema has been transformed into element tree(s).

4. MAPPING DISCOVERY ALGORITHM

Now we turn to the algorithm for discovering semantic mapping from an element tree to an ontology. The algorithm assumes a set of correspondences have been given. First, we analyze the structure of an XML element tree to lay out several principles for the algorithm.

4.1 Principles

We start from a methodology presented in the literature [Embley and Mok 2001; Kleiner and Lipeck 2001] for designing XML DTDs/schemas from a conceptual model (CM). We begin with the basic modeling constructs for concepts, attributes, and binary relationships.

4.1.1 *Basic Conceptual Models.* As with relational schemas, there is a notion of XML normal form (XNF) for evaluating the absence of redundancies and update anomalies in XML schemas [Embley and Mok 2001]. The methodology in [Embley and Mok 2001] claims to develop XNF-compliant XML schemas from conceptual models (CMs). It turns out that these "good" XML schemas are trees embedded in the graph representations of the CMs. Using the term "*element tree*" instead of "*schema tree*" in [Embley and Mok 2001], we briey describe the algorithm of [Embley and Mok 2001] (called *EM-algorithm*).

Example 4.1. *A CM containing only binary relationships between concepts is referred to as a* "binary and canonical hypergraph" *in [Embley and Mok 2001]. For*

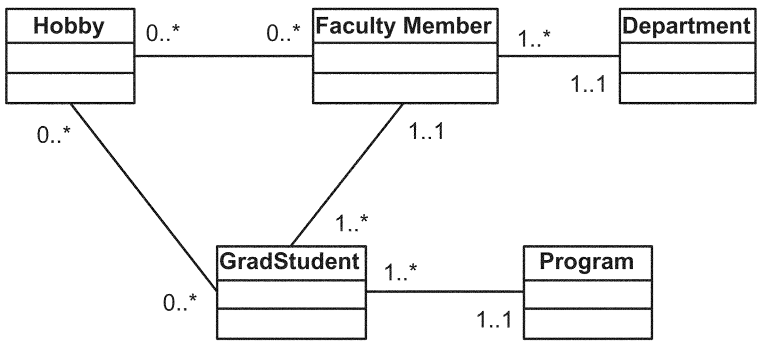

Fig. 7. A Sample CM graph.

such a CM H, the EM-algorithm *derives an element tree T such that T is in XNF and every path of T reflects a sequence of some connected edges in H. For example, starting from the* Department *node of the CM in Figure 7 the following element tree (omitting attributes) T is obtained by the EM-algorithm*:

$$\begin{array}{l} Department[\\ \quad (FacultyMember[\\ \qquad (Hobby)*, (GradStudent[\\ \qquad\qquad Program, (Hobby)*])*])*], \end{array}$$

where we use [ ] to indicate hierarchy and ( ) to indicate the multiple occurrences of a child element (or non-functional edges) in element trees.*

In essence, EM-algorithm *recursively constructs the element tree T as follows: it starts from a concept node N in CM, creates tree T rooted at a node R corresponding to N, and constructs the direct subtrees below R by following nodes and edges connected to N in CM. Finally, a largest hierarchical structure embedded within CM is identified and an edge of T reflects a semantic connection in the CM.*

Given an XNF-compliant element tree T and the CM from which T was derived, we may assume that there is a *semantic tree* S embedded in the CM graph such that S is isomorphic to T. If the correspondences between elements in T and concepts in the CM were given, we should be able to identify S.

Example 4.2. *Suppose elements in the element tree T of Example 4.1 correspond to the concepts (nodes) in Figure 7 by their names. Then we can recover the semantics of T recursively starting from the bottom. For the subtree T'*

$$\begin{array}{l} GradStudent[\\ \qquad Program, (Hobby)*], \end{array}$$

the edge $\mathcal{X}$:GradStudent[3] $\Rightarrow$ *$\mathcal{X}$:Program in T' is functional and $\mathcal{X}$:GradStudent* $\rightarrow$ *$\mathcal{X}$:Hobby is non-functional. In the CM graph, we can take the concept* GradStudent *as the root. Then we seek for a functional edge from the concept* GradStudent *to the concept* Program *and a* $1:N$ *or* $M:N$ *edge from* GradStudent *to the concept* Hobby.

[3]We use the notation *$\mathcal{X}$:Element* to distinguish an element in an XML schema.

The result is the semantic tree S' consisting of two edges: $\boxed{\text{GradStudent}}$ --->-- $\boxed{\text{Program}}$ *and* $\boxed{\text{GradStudent}}$ ----- $\boxed{\text{Hobby}}$.

Having identified S', we now move one layer up to search for a semantic tree S'' corresponding to the following subtree T''

$$\begin{array}{l} \mathit{FacultyMember}[\\ \quad (\mathit{Hobby})*, (\mathit{GradStudent}[\\ \qquad \mathit{Program}, (\mathit{Hobby})*])*]. \end{array}$$

The edge $\mathcal{X}$:FacultyMember $\rightarrow$ $\mathcal{X}$:Hobby in T'' is non-functional, and the edge from $\mathcal{X}$:FacultyMember to $\mathcal{X}$:GradStudent, the root of tree S', is non-functional as well. Hence, in the CM, we build the tree S'' using the $M : N$ edge from the concept FacultyMember *to the concept* Hobby *and the $1 : N$ edge from* FacultyMember *to the concept* GradStudent.

Finally, we are ready to build a semantic tree S corresponding to the entire tree T

$$\begin{array}{l} \mathit{Department}[\\ \quad (\mathit{FacultyMember}[\\ \qquad (\mathit{Hobby})*; (\mathit{GradStudent}[\\ \qquad\quad \mathit{Program}, (\mathit{Hobby})*])*])*]. \end{array}$$

Since we have identified a semantic tree S'' corresponding to T'', what we have to do now is to connect the concept Department *to the root of S'', which is the concept* FacultyMember. *The connection should be a $1 : N$ or $N : M$ edge according to the occurrence constraint of the FacultyMember element.*

Figure 8 shows the final semantic tree S identified from the CM in Figure 7, where we use a line with arrow to indicate a functional edge. Notice that the shared concept Hobby *gets duplicated in the CM graph.*

In an element tree T, attributes are the leaves of T and often correspond to the datatype properties of concepts in a CM. Our algorithm assumes that the user specifies the correspondences from XML attributes to datatype properties in an ontology, i.e., a CM, manually or using some existing schema matching tools. Given an element

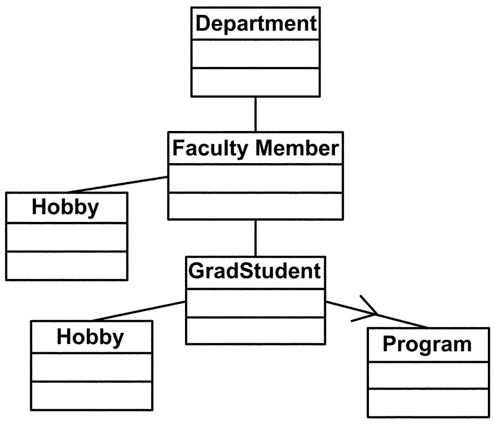

Fig. 8. The identified semantic tree.

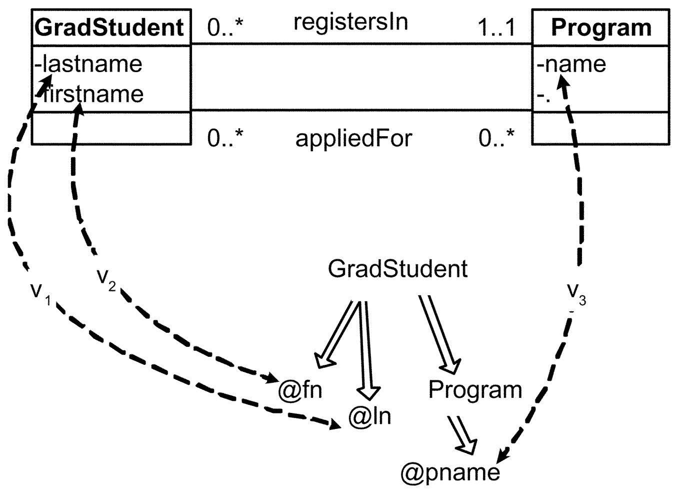

Fig. 9. A small ontology and an element tree.

tree, an ontology, and a set of correspondences, the algorithm attempts to identify the root of a semantic tree corresponding to the element tree and use "semantically matched" edges to connect the root to remaining nodes. This process is recursive and in a bottom-up fashion.

Example 4.3. *Given an element tree T*

GradStudent(@*ln*,@*fn*)[
Program(@*pname*)],

and an ontology/CM shown in Figure 9. Suppose the user specifies the following correspondences from attributes of elements in T to datatype properties of concepts in the ontology

v_1: $\mathcal{X}$:*GradStudent.@ln*$\leftrightsquigarrow\mathcal{O}$:GradStudent.lastname,
v_2: $\mathcal{X}$:*GradStudent.@fn*$\leftrightsquigarrow\mathcal{O}$:GradStudent.firstname,
v_3: $\mathcal{X}$:*GradStudent.Program.@pname*$\leftrightsquigarrow\mathcal{O}$:Program.name.

In a recursive and bottom-up fashion, we build a semantic tree S corresponding to T starting from the leaf @pname. The correspondence v_3 gives rise to the semantic tree S' for the leaf @pname, where S' is the concept Program. *For the subtree Program(@pname), the semantic tree is S' as well because there are no other correspondences involving the element $\mathcal{X}$:Program. At this level, there are two other subtrees: @fn and @ln. The semantic tree for both @fn and @ln is the concept* GradStudent *according to the correspondences v_1 and v_2. Let us refer to this semantic tree as S''. In connecting S''* to *S', a possible solution is to assume that the root of S'' corresponds to the element tree root $\mathcal{X}$:GradStudent. Therefore the connection is a functional edge from the root of S'',* GradStudent, *to the root of S',* Program, *because the connection from the element $\mathcal{X}$:GradStudent to the element $\mathcal{X}$:Program is functional (the occurrence constraint on $\mathcal{X}$:Program is 1). Consequently, we identify the semantic tree S as the connection* $\boxed{\text{GradStudent}}$ --registersIn->- $\boxed{\text{Program}}$ *in the ontology.*

The *first principle* of our mapping discovery algorithm is to identify the root of a semantic tree and to construct the tree recursively by connecting the root to its direct subtrees using edges in the ontology graph. More precisely, for the node v_1 and its child v_2 in an element tree, if a node N_1 in an ontology is identified for the root of a semantic tree for interpreting the tree at v_1 and a node N_2 is the root of a semantic tree for the subtree at v_2, then we connect N_1 to N_2 using an edge having cardinality constraints compatible with the occurrence constraints of the edge from v_1 to v_2 in the element tree.

Evidently, identifying the root of a semantic tree is the major obstacle. The following example illustrates the problem for an XML schema which is not XNF compliant. Such a schema can be easily encountered in reality.

Example 4.4. *Given an element tree*

$$\begin{array}{l} GradStudent[\\ \quad Name(@ln, @fn), Program(@pname)]. \end{array}$$

Suppose the user specified the following correspondences

v_1: $\mathcal{X}$*:GradStudent.Name.@ln*$\leftrightsquigarrow\mathcal{O}$:GradStudent.lastname,
v_2: $\mathcal{X}$*:GradStudent.Name.@fn*$\leftrightsquigarrow\mathcal{O}$:GradStudent.firstname,
v_3: $\mathcal{X}$*:GradStudent.Program.@pname*$\leftrightsquigarrow\mathcal{O}$:Program.name,

from the attributes of elements to the datatype properties of concepts in the ontology shown in Figure 10.

For the element $\mathcal{X}$:Name and the element $\mathcal{X}$:Program, we can identify two sub-trees, the concept GradStudent *and the concept* Program *by using the correspondences. For the element $\mathcal{X}$:GradStudent, we have to use the two identified sub-trees to build the final semantic tree. Since both $\mathcal{X}$:Name and $\mathcal{X}$:Program occur once and are at the same level, the question is which concept node is the root of the* final semantic tree? GradStudent *or* Program*? Since the order of nodes on the same level of the element tree does not*

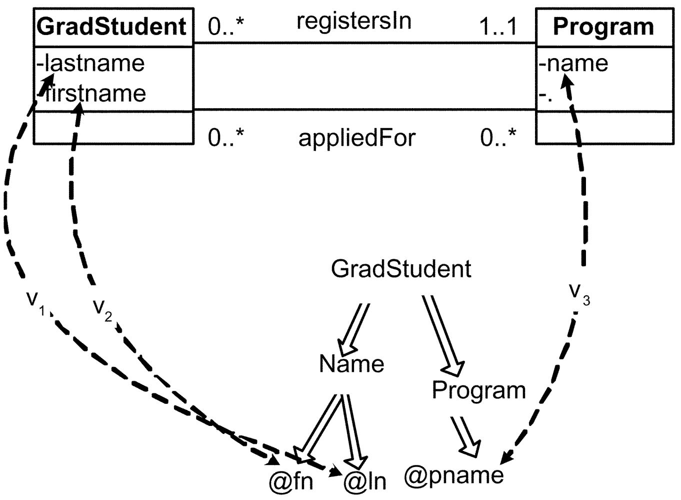

Fig. 10. An element tree and an ontology.

matter, both are potential roots. Therefore, the mapping algorithm should recover functional edges from GradStudent *to* Program *as well as from* Program *to* GradStudent, *if any.*

This leads to the *second principle* of our algorithm. Let v_1 and v_2 be two nodes in an element tree (an element tree has element nodes and attribute nodes). Let v_2 be a child of v_1 and the maximum occurrence constraint for v_2 is 1. For each concept C in an ontology graph such that C has been identified as the root of a semantic tree for the subtree at v_2, C is a potential root for building a semantic tree for the element tree at v_1. If v_1 does not have a child whose maximum occurrence constraint is 1, then we find a concept node as the root of a semantic tree for the element tree at v_1 as follows. The root connects to its children using non-functional paths. The tree consisting the root and its children is the minimum one if there are other trees formed by other roots connecting to the same set of children.

Unfortunately, not every functional edge from a parent node to a child node in an element tree represents a functional relationship. Specifically, some element tags are actually the collection tags. The following example illustrates the meaning of a collection tag.

Example 4.5. *Figure 11 depicts an element tree and the correspondences from the element tree to a CM. The element tree and the correspondences are written in text as follows.*

GradStudent[
    Name(@*ln*, @*fn*), *Hobbies*[
        (*Hobby*(@*title*))*]]

$\mathcal{X}$:*GradStudent.Name.@ln*$\leftrightsquigarrow\mathcal{O}$:GradStudent.lastname,
$\mathcal{X}$:*GradStudent.Name.@fn*$\leftrightsquigarrow\mathcal{O}$:GradStudent.firstname,
$\mathcal{X}$:*GradStudent.Hobbies.Hobby.@title*$\leftrightsquigarrow\mathcal{O}$:Hobby.title.

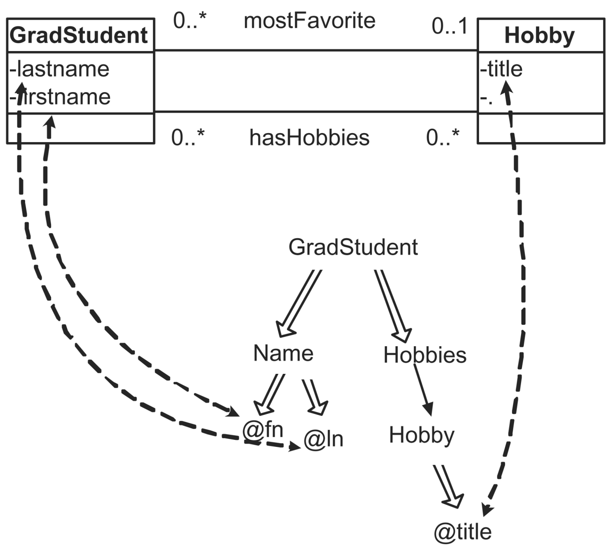

Fig. 11. An element tree with a collection tag.

The element tag $\mathcal{X}$:Hobbies is a collection tag. It represents a collection of hobbies of a graduate student. Although the edge $\mathcal{X}$:GradStudent $\Rightarrow$ $\mathcal{X}$:Hobbies is functional, $\mathcal{X}$:Hobbies $\rightarrow$ $\mathcal{X}$:Hobby is non-functional. When the concept Hobby *is identified as the root of a semantic tree for the subtree*

$$Hobbies[\\ \quad (Hobby(@title))*],$$

Hobby *should not be considered as a potential root of the semantic tree for the entire element tree.*

Eliminating concepts corresponding to collection tags from the set of the potential roots is our *third principle.*

In most cases, we try to discover the semantic mapping between an XML schema and an ontology such that they were developed independently. In such cases, we may not be able to find an isomorphic semantic tree S embedded in the ontology graph. For example, for the element tree

$$City(@cityName)[\\ \quad Country\ (@countryName)],$$

if a CM with a path $\boxed{\text{City}}$ -- locatedIn -- > - $\boxed{\text{State}}$ -- locatedIn -- > -$\boxed{\text{Country}}$ is used for interpreting the element tree, the entire path is a possible answer. The *fourth principle* for discovering mappings is to find shortest paths in an ontology graph instead of restricting to single edges. The composed cardinality constraints of a path should be compatible with the occurrence constraints of the corresponding edge in the element tree.

Even though we could eliminate some collection tags from the set of potential roots to reduce the number of possible semantic trees, there may still be too many possibilities if the ontology graph is large. To further reduce the size of the set of potential roots, we can make use of the key and keyref constructs in an XML schema.

Example 4.6. *Given the element tree*

Article[
Title(@*title*), *Publisher*(@*name*), *ContactAuthor*(@*contact*), (*Author*(@*id*))*].
If the attribute @title is defined as the key for the element $\mathcal{X}$:Article, then we should only choose the concept corresponding to @title as the root of the semantic tree, eliminating the classes corresponding to @name and @contact (chosen by the second principle). Alternatively, if @contact is defined as a keyref referencing some key, we can also eliminate the class corresponding to @contact.

So our *fifth principle* is to use key and keyref definitions to restrict the set of potential roots.

4.1.2 *Reified Relationships.* To represent n-ary relationships in the conceptual modeling language (CML), one needs to use *reified relationship* (*classes*). For example, an ontology may have class Presentation connected with functional *roles* to classes Author, Paper, and Session, indicating participants. It is desirable to recover reified relationships and their role connections from an XML schema. Suppose the element tree.

$$Presentation[\quad Presenter(@author),\ Paper(@title),\ Session(@eventId)];$$

represents the above ternary relationship. Then, in the ontology, the root of the semantic tree is the *reified relationship class* Presentation, rather than any one of the three classes which are role fillers. The *sixth principle* then is to look for *reified relationships* for element trees with only functional edges from a parent to its children that correspond to separate classes[4].

4.1.3 ISA *Relationships.* In [Embley and Mok 2001], ISA relationships are eliminated by collapsing superclasses into their subclasses, or vice versa. If a superclass is collapsed into subclasses, correspondences can be used to distinguish the nodes in the ontology. If subclasses are collapsed into their superclass, then we treat the ISA edges as special functional edges with cardinality constraints $0:1$ and $1:1$. The *last principle* is then to follow ISA edges whenever we need to construct a functional path[5].

4.2 Algorithm

First, to get a better sense of what we are aiming for, we present the encodeTree(S, L) procedure, which translates an ontology subtree $\mathcal{S}$ into a conjunctive formula, taking into account the correspondences L [An et al. 2006].

```
Function encodeTree($S$,$L$)
Input subtree $S$ of ontology graph, correspondences $L$ from attributes of element tree to datatype properties of class nodes in $S$.
Output variable name generated for root of $S$, and conjunctive formula for the tree.
Steps:
(1) Suppose $N$ is the root of $S$, let $\Psi = \{\}$.
(2) If $N$ is an attribute node with label $f$, find $@d$ such that $L(\_, @d, \_, f, N) = true$,
    return ($@d$, $true$).
(3) If $N$ is a class node with label $C$, then introduce new variable $Y$; add conjoint
    $C(Y)$ to $\Psi$; for each edge $p_i$ from $N$ to $N_i$:
    (a) let $S_i$ be the subtree rooted at $N_i$;
    (b) let $(v_i; \phi_i(Z_i))$=encodeTree($S_i$, $L$);
    (c) add conjunct $p_i(Y, v_i) \wedge \phi_i(Z_i)$ to $\Psi$;
(4) return $(Y, \Psi)$.
```

The following procedure constructTree(T, L) generates the subtree of the ontology graph for the element tree after appropriately replicating nodes[6] in the ontology graph.

Function constructTree(T, L)
Input an element tree T, an ontology graph, and correspondence L from attributes in T to datatype properties of class nodes in the ontology graph.
Output set of (subtree S, root R, *collectionTag*) triples, where *collectionTag* is a

[4]If a parent functionally connects to only two children, then it may represent an M:N binary relationship. So recover it as well.
[5]Thus, ISA is taken care of in the forthcoming algorithm by proper treatment of functional path.
[6]Replications are needed when multiple attributes correspond to the same datatype property. See [An et al. 2005b] for details.

boolean value indicating whether the root corresponds to a collection tag.

Steps:

```
(1) Suppose $N$ is the root of tree $T$.
(2) If $N$ is an attribute, then find $L(\_, N, \_, \_, R) = true$; return $(\{R\}, R, false)$.
    /*the base case for leaves.*/
(3) If $N$ is an element having $n$ edges $\{e_1, ..., e_n\}$ pointing to $n$ nodes $\{N_1, ..., N_n\}$,
    let $T_i$ be the subtree rooted at $N_i$,
    then compute $(S_i, R_i, collectionTagi)$ = constructTree($T_i$, $L$) for $i = 1, ..., n$;
    (a) If $n = 1$ and $e_1$ is non-functional, return $(S_1, R_1, true)$;/*$N$ probably is a collection tag representing a set of instances each of which is an instance of the $N_1$ element.*/
    (b) Else if $n = 1$ and $e_1$ is functional return $(S_1, R_1, collectionTag_1)$.
    (c) Else if $R_1$=$R_2$=...=$R_n$, then return (combine($S_1$, ..., $S_n$), $R_1$, $false$)[7].
    (d) Else let $F$=$\{R_{j1}, ..., R_{jm}|$ s.t. $e_{jk}$ is functional and $collectionTag_{jk} = false$
        for $k = 1, ..., m, j_k \in \{1, ..., n\}\}$ and $NF$=$\{R_{i1}, ..., R_{ih}|$ s.t. $e_{i_k}$ is non-functional,
        or $e_{i_k}$ is functional and $collectionTag_{i_k} = true$ for $k = 1, ..., h, i_k \in \{1, ..., n\}\}$,
        let $ans = \{\}$, /*separate nodes according to their connection types to $N$.*/
        i.   Try to limit the number of nodes in $F$ by considering the following cases:
             1) keep the nodes corresponding to key elements located on the highest
             level; 2) keep those nodes which do not correspond to keyref elements.
        ii.  If $NF = \emptyset$, find a reified relationship concept $R$ with $m$ roles $r_{j1}, ..., r_{jm}$
             pointing to nodes in $F$, let $S$ = combine($\{r_{jk}\}$, $\{S_{jk}\}$) for $k = 1, ..., m$; let
             $ans = ans \cup (S, R, false)$. If $R$ does not exist and $m = 2$, find a non-functional
             shortest path $p$ connecting the two nodes $R_{j1}$, $R_{j2}$ in $F$; let $S$ =
             combine($p$, $S_{j1}$, $S_{j2}$); let $ans = ans \cup (S, R_{j1}, false)$. /*$N$ probably represents
             an n-ary relationship or many-many binary relationship (footnote of the
             sixth principle.)*/
        iii. Else for each $R_{jk} \in F$ $k = 1, ..., m$, find a shortest functional path $p_{jk}$ from
             $R_{jk}$ to each $R_{jt} \in F \backslash R_{jk}$ for $t = 1, ..., k-1, k+1, ..., m$; and find a shortest
             non-functional path $q_{i_r}$ from $R_{jk}$ to each $R_{i_r} \in NF$ for $r = 1, ..., h$; if $p_{jk}$ and
             $q_{i_r}$ exist, let $S$ = combine($\{p_{jk}\}$, $\{q_{i_r}\}$, $\{S_1, ..., S_n\}$); let $ans = ans \cup (S, R_{jk}, false)$.
             /*pick an root and connect it to other nodes according to their
             connection types.*/
        iv.  If $ans \neq \emptyset$, return $ans$; else find a minimum Steiner tree $S$ connecting $R_1$,
             ..., $R_n$, return $(S, R_1, false)$. /*the default action is to find a shortest Steiner
             tree.*/
```

5. EXPERIMENTAL EVALUATION

We have implemented the mapping discovery algorithm and conducted a set of experiments to evaluate its eﬁectiveness and usefulness.

Measures for mapping quality and accuracy. We first attempt to use the notions of *precision* and *recall* for the evaluation. Let R be the number of correct

[7]Function combine merges edges of trees into a larger tree.

mapping formulas of an XML schema, let I be the number of correctly identified mapping formulas by the algorithm, and let P be the total number of mapping formulas returned. The two quantities are computed as: *precision* $= I/P$ and *recall* $= I/R$. Please note that for a single input element tree T, which has a single correct mapping formula, the algorithm either produces the formula or not. So the *recall* for T is either 0 or 1, but the *precision* may vary according to the number of output formulas. For measuring the overall quality of the mapping results, we computed the average precision and recall for all tested element trees of an XML schema.

However, precision and recall alone cannot tell us how useful the algorithm is to users. The purpose of our tool is to *assist* users in the process of constructing complex mappings, so that productivity is enhanced. Consider the case when only one semantic mapping is returned. Even if the tool did not find the exactly right one, it could still be useful if the formula is accurate enough so that some labor is saved. To try to measure this, we adopt the accuracy metric for schema matching [Melnik et al. 2002]. Consider the mapping formula $\Phi(X) \rightarrow \Psi(X, Y)$ with the formula $\Phi(X)$ encoding an element tree. The formula $\Psi(X, Y)$ encodes a semantic tree $S = (V, E)$ by using a set of unary predicates for nodes in V, a set of binary predicates for edges in E, and a set of variables, Y, assigned to each node (there are predicates and variables for datatype properties as well). For a given element tree T, writing the complex mapping formula consists of identifying the semantic tree and encoding it into a conjunctive formula (which could be treated as a set of atomic predicates). Let $\Psi_1 = \{a_1(Z_1), a_2(Z_2), ..., a_m(Z_m)\}$ encode a tree S_1, let $\Psi_2 = \{b_1(Y_1), b_2(Y_2), ..., b_n(Y_n)\}$ encode a tree S_2. Let $D = \Psi_2 \setminus \Psi_1 = \{b_i(Y_i)|$ s.t. for a given partial one-one function $f\colon Y \rightarrow Z$ representing the mapping from nodes of S_2 to nodes of S_1, $b_i(f(Y_i)) \in \Psi_1\}$. One can easily identify the mapping $f\colon Y \rightarrow Z$ by comparing the two trees S_2 and S_1 (recall an ontology graph contains class nodes as well as attribute nodes representing datatype properties) so we consider that it comes for free. Let $c = |D|$. Suppose Ψ_1 to be the correct formula and Ψ_2 to be the formula returned by the tool for an element tree. To reach the correct formula Ψ_1 from the formula Ψ_2, one needs to delete $n - c$ predicates from Ψ_2 and add $m - c$ predicates to Ψ_2. On the other hand, if the user creates the formula from scratch, m additions are needed. Let us assume that additions and deletions need the same amount of effort. However, browsing the ontology for correcting formula Ψ_2 to formula Ψ_1 is different from creating the formula Ψ_1 from scratch. So let α be a cost factor for browsing the ontology for correcting a formula, and let β be a factor for creating a formula. We define the accuracy or labor savings of the tool as *labor savings* $= 1 - \frac{\alpha[(n-c)+(m-c)]}{\beta m}$. Intuitively, $\alpha < \beta$, but for a worst-case bound let us assume $\alpha = \beta$ in this study. Notice that in a perfect situation, $m = n = c$ and *labor savings* $= 1$.

Schemas and ontologies. To evaluate the tool, we collected 9 XML schemas varying in size and nested structure. The 9 schemas come from 4 application domains, and 4 publicly available domain ontologies were obtained from the Web and literature. Table I and II shows the characteristics of the schemas and the ontologies; the column heads are self-explanatory. The *company* schema and ontology are obtained from [Kleiner and Lipeck 2001] in order to test the principles

Table I. Characteristics of test XML schemas.

XML Schema	Max Depth (DFS) in Schema Graph	# Nodes in Schema Graph	# Attributes in Schema Graph
Company	6	30	17
Conference	5	21	12
UT DB	6	40	20
Mondial	6	214	93
DBLP 1	3	132	63
DBLP 2	5	29	11
SigmodRecord	3	16	7
Amalgam 1	3	117	101
Amalgam 2	3	81	53

Table II. Ontology summary.

Ontology	# Nodes	# Links
Company	18	27
KA	105	4396
KA	105	4396
CIA factbook	52	77
Bibliographic	75	749
Bibliographic	75	749
Bibliographic	75	749
Bibliographic	75	749
Bibliographic	75	749

of the mapping construction. The *conference* schema is obtained from [Lee and Chu 2000]. *UT DB* is the schema used for describing the information of the database group in University of Toronto. *SigmodRecord* is the schema for SIGMOD record. The rest of the schemas are obtained from the *Clio* test suite (http://www.cs.toronto.edu/db/Clio). The KA ontology, CIA factbook, and the Bibliographic-Data are all available on the Web.

Experimental results. Our experiments are conducted on a Dell desktop with a 1.8 GHZ Intel Pentium 4 CPU and 1G memory. The first observation is the efficiency. In terms of the execution times, we observed that the algorithm generated results on average in 1.4 seconds which is not significantly large, for our test data.

Figure 12 shows the average precision and recall measures of the 9 mapping pairs. For each pair of schema and ontology, the average precision and recall are computed as follows. For the element trees extracted from the schema graph, a set of correct mapping formulas is manually created. We then apply the algorithm on the element

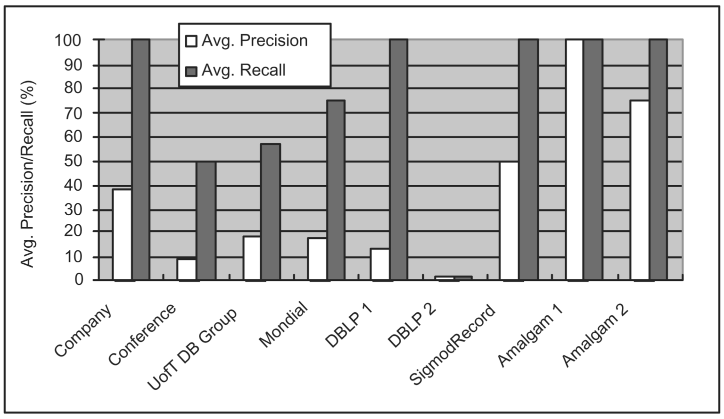

Fig. 12. Average recall and precision for 9 mapping cases.

trees and ontologies to generate a set of formulas. Next we examine each of the generated formulas to count how many are correct and compute the average precision and recall. The overall average precision is 35% and overall average recall is 75%. Notice that we have limited the number of formulas returned by the tool to 10.

Finally, we evaluate the usefulness of the tool. Figure 13 shows the average values of labor savings for the 9 mapping cases. For each mapping case, the average labor savings is computed as follows. Examine each incorrect formula returned by the algorithm and compute its labor saving value relative to the manually created one. Take the average value of the labor savings of all incorrect formulas. Note that even when the correct formula was identified by the algorithm, we still computed the labor savings for all incorrect ones to see how useful the tool is in case only one formula was returned. The overall average labor savings is over 80%, which is quite promising. Especially in view of the pessimistic assumption that $\alpha = \beta$ in the *labor savings* formula, we take this as evidence that the tool can greatly assist users in

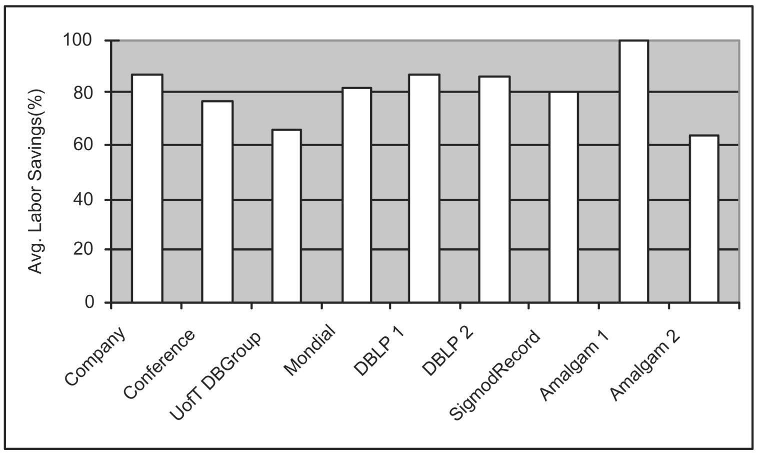

Fig. 13. Average labor savings for 9 mapping cases.

constructing complex mappings between XML schemas and ontologies with a proper schema matching tool as a front-end component.

6. MAINTAINING SEMANTIC MAPPINGS

Creating semantic mappings between XML schemas and ontologies is a complex process. Although we have developed heuristics for assisting people to discover semantic mappings, it still requires human involvement in the process. Once such a semantic mapping has been created, it is important and necessary to help maintain the consistency of the semantic relationship when the schema and ontology evolve. In this section, we study the problem of maintaining a semantic mapping.

The purpose of the maintenance are two-fold: first, to preserve the semantic relationship between the schema and the ontology when the schema or ontology are modified; second, to reuse the existing semantic mapping as much as possible. A similar problem has been studied for adapting schema mappings under schema evolution. Two possible approaches have been proposed in the literature: a schema change approach (SCA) [Velegrakis et al. 2003] and a mapping composition approach (MCA) [Yu and Popa 2005]. Both solutions focus on reusing the semantics encoded in previous mappings for merely adapting the mappings. Schemas are not "synchronized". In our situation, "synchronizing" the ontology and schema associated with a semantic mapping along with adapting the mapping will be essential for achieving desired goals. Consider a very simple case. Suppose the semantics of an XML schema is expressed in terms of an ontology. If the database administrator (DBA) wants to modify an occurrence constraint in the schema by changing it from many to one, it may be desirable to "synchronize" the corresponding cardinality constraint in the ontology accordingly. Although an ontology is commonly considered as a shared conceptualization and tends to be fixed, we assume that we can modify a local copy of the ontology and employ sophisticated version control systems for maintaining difierent versions of ontologies. Version control for ontologies is beyond the scope of this paper. In the sequel, we focus on *mapping maintenance between XML schemas and ontologies when XML schemas evolve*.

Related work includes schema evolution in object-oriented databases (OODB). The problem of schema evolution in OODB is to maintain the consistency of the instances in an OODB when its schema is modified. The challenges are to update the database efficiently and minimize information loss. A variety of solutions, e.g., [Benatallah; Banerjee et al.; Claypool et al.; Ferrandina et al.], have been proposed in the literature. Our problem is different from the schema evolution problem in OODB in that we aim at the semantic consistency between a schema and an ontology. How- ever, we can draw some insights from the extensive study of the schema evolution problem in OODB.

Another mapping maintenance problem, studied in [McCann et al.], mainly focuses on detecting inconsistency of simple correspondences between schema elements when schemas evolve. This problem is complementary to the problem we consider here.

We organize this section as follows. First, we propose the goals of mapping maintenance: We need to understand what is a consistent semantic relationship. Second, we analyze and categorize possible changes to schemas and ontologies. Finally, we present a maintenance plan.

6.1 Goals of Maintaining Semantic Mappings

A semantic mapping formula $\Psi(X) \rightarrow \Phi(X, Y)$ relates a formula $\Psi(X)$ encoding an element tree T with a conjunctive formula $\Phi(X, Y)$ encoding a semantic tree (or *s-tree*) S in an ontology. A consistent mapping means that the element tree and the s-tree S should be "semantically compatible". In other word, the instances of the element tree should be compatible with the instances of the s-tree. To check this condition, we consider the process of translating instances under one structure to instances under another structure.

Let Σ_T be a set of occurrence constraints (see Section 3.1) imposed to the elements in the element Tree T. Let Σ_S be a set of cardinality constraints (see Section 3.2) imposed on the relationships in the semantic tree S. Consider the mapping formula $\Psi(X) \rightarrow \Phi(X, Y)$. For a legal instance of T satisfying Σ_T, we can create an instance of S by instantiating Y using labeled null variables in $\Phi(X, Y)$. We then check whether the new instance of S satisfies the constraints Σ_S. Conversely, for a legal instance of S satisfying Σ_S, we can create an instance of T and check whether the instance satisfies the constraints Σ_T . If for each legal instance of T we can create a legal instance of S, and for each legal instance of S we can create a legal instance of T, then we say that the mapping formula $\Psi(X) \rightarrow \Phi(X, Y)$ is consistent. Our goals for maintaining semantic mappings are as follows.

Goal 1 *For a consistent semantic mapping $\mathcal{M}$ between an XML schema $\mathcal{X}$ and an ontology $\mathcal{O}$, maintain the consistency of $\mathcal{M}$ when $\mathcal{X}$ evolves.*

Goal 2 *For a consistent semantic mapping $\mathcal{M}$ between an XML schema $\mathcal{X}$ and an ontology $\mathcal{O}$, maximize the usage of the existing semantics of $\mathcal{M}$ during the maintenance.*

6.2 Schema Evolution

Changes to XML schemas can be classified along two orthogonal axes. First, on the *action* axis, changes can be classified into (1) changes for adding/deleting elements; (2) changes for merging/splitting elements; (3) changes for moving/copying elements; (4) changes for renaming elements; and (5) changes for modifying constraints. Second, on the *effect* axis, changes can be classified into (i) changes that cause mapping modification; (ii) changes that cause the related schema (or ontology) modification; and (iii) changes that cause both mapping and the related schema (or ontology) modification. These changes can be characterized by mappings [Yu and Popa 2005] or by sequences of evolution primitives [Velegrakis et al. 2003; Banerjee et al.].

In our study, we use a set of simple correspondences to link elements of the previous schema to elements of the new schema after a schema changed. We then analyze the existing semantic mapping and the semantics in the new schema. Through the set of correspondences, we will then (semi-)automatically adapt both the semantic mapping and the schema/ontology to maintain the consistency of the semantic mapping.

Example 6.1. *Figure 14 shows on the left hand side an old XML schema $\mathcal{X}$. On the*

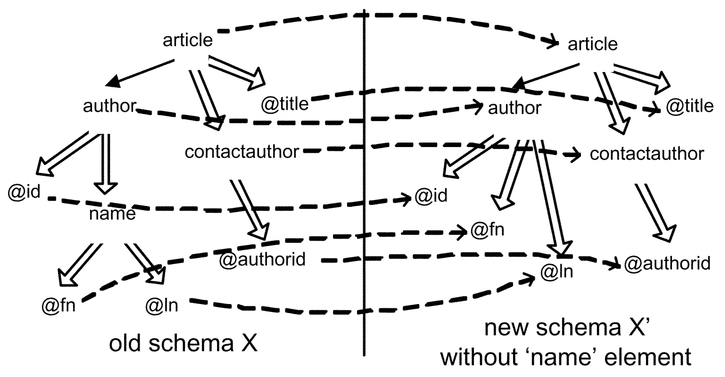

Fig. 14. Capturing the evolution of an XML schema.

right hand side is the new XML $\mathcal{X}'$ which was evolved from $\mathcal{X}$ by removing the Name *element. The dashed arrows from attributes and elements in $\mathcal{X}$ to attributes and elements in $\mathcal{X}'$ capture the commonality/differences between the old XML schema and the new XML schema.*

6.3 Maintenance Plan

We present a plan for maintaining semantic mappings between XML schemas and ontologies. Figure 15 graphically describes the semantic mapping maintenance setting, where the schema $\mathcal{X}$ evolved to a new schema $\mathcal{X}'$. M is the existing semantic mapping; M' is the set of correspondences from elements of $\mathcal{X}'$ to elements of $\mathcal{X}$. The result of the mapping maintenance is to adapt M to a new semantic mapping M'' between $\mathcal{X}'$ and $\mathcal{O}$ (or $\mathcal{O}''$ if the (copy of the) original ontology needs to be modified.)

For a semantic mapping formula $\Psi(X) \rightarrow \Phi(X, Y)$ which relates a formula $\Psi(X)$ encoding an element tree T with a conjunctive formula $\Phi(X, Y)$ encoding an s-tree S in an ontology, the plan consists of several strategies presented as the following.

Strategy 1: Align the element tree T and the s-tree S so that each attribute, element, and edge in T corresponds to a construct or a set of constructs in S. This alignment may indicate that a path in T corresponds to an edge in S or vice versa. Consequently, for each individual attribute, element, or edge in T, it either corresponds to an individual construct in S or it is a part of a compound structure such as a path which corresponds to an individual construct in S.

Strategy 2: If an attribute is added to the XML schema, then locate the construct

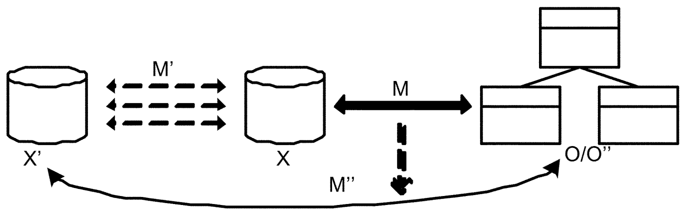

Fig. 15. Maintenance of semantic mapping.

in S corresponding to the element where the attribute is added. Add a new attribute to the construct. If an element added to the schema, add a new concept to the ontology. The new concept is connected to the construct corresponding to the parent element of the new element. Update the mapping formula accordingly.
Strategy 3: If a construct is deleted from either the schema such that some leaves of the trees are removed, then update the mapping formula by removing the deleted construct. For example, if a leaf of T is removed due to the deletion of an attribute in the schema, the mapping formula can be updated accordingly by removing the referred attribute.
Strategy 4: If the element tree T evolves to a new tree T' through restructuring, then use the set of simple correspondences between T and T' to generate an evolution mapping between these two trees. Composing [Yu and Popa 2005] the evolution mapping and the existing mapping formula to generate a new mapping formula relating T' and S.
Strategy 5: If a constraint in the schema is changed, locate the constructs in the ontology which correspond to the structure in the XML schema where the changed constraint was imposed. Update the corresponding constraint in the ontology according to the change to the constraint in the schema. For example, if the occurrence constraint on a parent-child edge is changed from many to single, then the cardinality constraint on the corresponding relationship in the ontology should be updated from many to functional.

With the above strategies, we are able to *incrementally* maintain the consistency of a semantic mapping between an XML schema and an ontology when the schema evolves.

7. CONCLUSIONS

In this paper, we have motivated and defined the problem of discovering complex semantic mappings from XML schemas to ontologies, given a set of simple correspondences from XML attributes to ontology datatype properties. The problem is motivated by the needs to annotate XML data in terms of ontologies, to translate XML data into ontologies [An and Mylopoulos], and to integrate heterogeneous XML data on the semantic web. We presented in detail a novel tool [An et al. 2005a] for semi-automatically constructing complex mappings for users. The experimental results suggest that quite significant savings in human work could be achieved by the use of our tool.

Semantic mappings between XML schemas and ontologies are valuable assets once they are created. However, schemas and ontologies change constantly in open, dynamic, and distributed environments such as the Web. To address this problem, we extend our previous work [An et al. 2005a] by proposing strategies for maintaining a semantic mapping between an XML schema and an ontology under schema evolution. Overall, our work addresses an important problem in managing heterogeneous XML data sets. The results of the entire study provide a set of solutions for building sustainable XML data integration systems using ontologies.

Future work includes developing effective algorithms for ranking the mapping candidates and studying the maintenance problem under ontology evolution. We also plan to use semantic mappings between XML schemas and ontologies to generate direct mappings between XML schemas.

REFERENCES

AMANN, B., BEERI, C., FUNDULAKI, I., AND SCHOLL, M. 2002. Ontology-based integration of xml web resources. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web.* Springer-Verlag, London, UK, 117–131.

AN, Y., BORGIDA, A., AND MYLOPOULOS, J. 2005a. Constructing Complex Semantic Mappings between XML Data and Ontologies. In *Proceedings of the International Conference on Semantic Web (ISWC).* 6–20.

AN, Y., BORGIDA, A., AND MYLOPOULOS, J. 2005b. Inferring complex semantic mappings between relational tables and ontologies from simple correspondences. In *OTM Conferences (2).* 1152–1169.

AN, Y., BORGIDA, A., AND MYLOPOULOS, J. 2006. Discovering the Semantics of Relational Tables through Mappings. *Journal on Data Semantics VII*, 1–32.

AN, Y. AND MYLOPOULOS, J. Translating xml web data into ontologies. In *In the Proceedings of International Workshop on Web Semantics (SWWS'05), Agia Napa, Cyprus. 2005.*

ARENAS, M. AND LIBKIN, L. 2005a. Xml data exchange: consistency and query answering. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems.* ACM Press, New York, NY, USA, 13–24.

ARENAS, M. AND LIBKIN, L. 2005b. XML Data Exchange: Consistency and Query Answering. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS).* 13–24.

BANERJEE, J. ET AL. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *SIGMOD'87.*

BENATALLAH, B. A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases. In *ER'99.*

CLAYPOOL, K. T., JIN, J., AND RUNDENSTEINER, E. SERF: Schema Evolution through an Extensible, Re-usable, and Flexible Framework. In *CIKM'98.*

DELOBEL, C., REYNAUD, C., ROUSSET, M.-C., SIROT, J.-P., AND VODISLAV, D. 2003. Semantic integration in xyleme: a uniform tree-based approach. *Data Knowl. Eng. 44*, 3, 267–298.

DHAMANKAR, R., LEE, Y., DOAN, A., HALEVY, A., AND DOMINGOS, P. 2004. imap: discovering complex semantic matches between database schemas. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data.* ACM Press, New York, NY, USA, 383–394.

EMBLEY, D. W. AND MOK, W. Y. 2001. Developing xml documents with guaranteed "good" properties. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling.* Springer-Verlag, London, UK, 426–441.

FALLSIDE, D. C. AND WALMSLEY, P. October 2004. XML Schema Part 0: Primer Second Edition. In *W3C Recommendation.* http://www.w3.org/TR/xmlschema-0/.

FERRANDINA, F., FERRAN, G., MEYER, T., MADEC, J., AND ZICARI, R. Schema and Database Evolution in the O2 Object Database System. In *VLDB'95.*

HALEVY, A. Y., IVES, Z. G., MORK, P., AND TATARINOV, I. 2003. Piazza: data management infrastructure for semantic web applications. In *WWW '03: Proceedings of the 12th international conference on World Wide Web.* ACM Press, New York, NY, USA, 556–567.

JENSEN, M. R., MOLLER, T. H., AND PEDERSEN, T. B. 2003. Converting xml dtds to uml diagrams for conceptual data integration. *Data Knowl. Eng. 44*, 3, 323–346.

KLEINER, C. AND LIPECK, U. W. 2001. Automatic generation of xml dtds from conceptual database schemas. In *GI Jahrestagung (1).* 396–405.

LAKSHMANAN, L. V. S. AND SADRI, F. 2003. Interoperability on xml data. In *International Semantic Web Conference.* 146–163.

LEE, D. AND CHU, W. W. 2000. Constraints-preserving transformation from xml document type definition to relational schema. In *ER*. 323–338.

MADHAVAN, J., BERNSTEIN, P. A., AND RAHM, E. 2001. Generic schema matching with cupid. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 49–58.

MCCANN, R. ET AL. Maveric: Mapping Maintenance for Data Integration Systems. In *VLDB'05.*

MELNIK, S., GARCIA-MOLINA, H., AND RAHM, E. 2002. Similarity ooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*. 117–128.

MILLER, R. J., HAAS, L. M., AND HERNFIANDEZ, M. A. 2000. Schema mapping as query discovery. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 77–88.

POPA, L., VELEGRAKIS, Y., MILLER, R. J., HERNFIANDEZ, M. A., AND FAGIN, R. 2002. Translating web data. In *VLDB*. 598–609.

SHANMUGASUNDARAM, J., TUFTE, K., ZHANG, C., HE, G., DEWITT, D. J., AND NAUGHTON, J. F. 1999. Relational databases for querying xml documents: Limitations and opportunities. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, Eds. Morgan Kaufmann, 302–314.

VELEGRAKIS, Y., MILLER, R. J., AND POPA, L. 2003. Mapping Adaptation under Evolving Schemas. In *Proceedings of the International Conference on Very Large Data bases (VLDB)*. 584–595.

YU, C. AND POPA, L. 2005. Semantic Adaptation of Schema Mappings when Schema Evolve. In *Proceedings of the International Conference on Very Large Data bases (VLDB).*

Yuan An received a PhD degree from the University of Toronto in 2007 in Computer Science. He has been Assistant Professor in the College of Information Science and Technology at Drexel University since that year. He has research interests in semantic technologies for information integration, information modeling including ontology design, schema mapping, and the semantic web. Dr. An designed and developed the MAPONTO tool for creating semantic mappings between heterogeneous data representation. Dr. An also has more than 10 years working experience in the Information Technology industry. As a principal developer and team leader, he designed and led the development of the management information systems for manufacturers of electronic products, real estate companies, financial and educational institutes in various sizes. Dr. An has a Master's degree in Computer Science from Dalhousie University, Canada. He also earned a Master's degree in Electrical Engineering from Tsinghua University, China, in 1989 and a Bachelor's degree in Electrical Engineering from the same Chinese university in 1987.

Alex Borgida Alex Borgida has been on the faculty of the Department of Computer Science at Rutgers University, after receiving a PhD degree from the University of Toronto. His interests lie at the intersection of Knowledge Representation and Reasoning (esp. description logics), Databases (exceptions, conceptual modeling, schema design and integration), and Software Engineering (formal specification of software, and requirements analysis.)

John Mylopoulos John Mylopoulos earned a PhD degree from Princeton University in 1970 in Electrical Engineering. He has been professor of Computer Science at the University of Toronto (Canada) since that year. His research interests include conceptual modelling, requirements engineering, data semantics and knowledge management. Mylopoulos is a fellow of the American Association for Artificial Intelligence (AAAI) and the Royal Society of Canada (Academy of Sciences). He has served as programme/general chair of international conferences in Artificial Intelligence, Databases and Software Engineering, including IJCAI (1991), Requirements Engineering (1997), and VLDB (2004). He was co-editor-in-chief of the Requirements Engineering Journal, published by Springer-Verlag (2000-07). Since September 2005 Mylopoulos is distinguished professor (chiara fama) of Science at the University of Trento (Italy).