Discovering Semantics from Data-Centric XML

Luochen Li[1], Thuy Ngoc Le[1], Huayu Wu[2],
Tok Wang Ling[1], and Stéphane Bressan[1]

[1] School of Computing, National University of Singapore
{luochen,ltngoc,lingtw,step}@comp.nus.edu.sg
[2] Institute for Infocomm Research, Singapore
huwu@i2r.a-star.edu.sg

Abstract. In database applications, the availability of a conceptual schema and semantics constitute invaluable leverage for improving the effectiveness, and sometimes the efficiency, of many tasks including query processing, keyword search and schema/data integration. The Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) model is a conceptual model intended to capture the semantics of object classes, object identifiers, relationship types, etc., underlying XML schemas and data. We refer to the set of these semantic concepts as the ORA-semantics. In this work, we present a novel approach to automatically discover the ORA-semantics from data-centric XML. We also empirically and comparatively evaluate the effectiveness of the approach.

1 Introduction

To improve the conceptual quality, we needs to discover the intended semantics in the logical XML schemas and data. This requires finding such semantic information as object classes, relationship types, object identifiers (OIDs), etc., as present in conceptual models for semi-structured data such as Object-Relationship-Attribute for Semi-Structured data (ORA-SS) [6]. We refer to this semantics as the ORA-semantics. Once discovered, the ORA-semantics is useful not only for users to understand the data and schemas but also for improving both the effectiveness and efficiency of processing. Let us use the XML document in Fig. 1 to illustrate how the availability of such semantics help applications.

XML Query Processing
To process an XPath query, e.g. *//Student[Matric# ='HT001']/Name*, most approaches match the query pattern to the data to find all occurrences. However, if we have the semantics that *Matric#* is the OID of student, after getting an answer, we can stop searching the rest of data.

XML Keyword Search
The use of semantics in current keyword search approaches [7] is still on object level. For a query {CS5201, CS5208} to find common information of two courses, only by knowing there is a relationship type between object classes *Student* and *Course*, one can infer the meaningful answer should be all students taking these two courses. Otherwise, the root node will be returned by most LCA-based XML keyword search approaches [12].

H. Decker et al. (Eds.): DEXA 2013, Part I, LNCS 8055, pp. 88–102, 2013.

Schema/Data Integration

Most existing approaches [1] integrate elements based on their structural and linguistic similarities. *Grade* is an attribute of the relationship type between *Course* and *Student*. Without this semantics, when we integrate this schema with another, in which *Student* has an object attribute *Grade* which means the year of his study in school, we may wrongly integrate these two different attributes with the same attribute name *Grade* and the same parent node *Student*, because of their high structural and linguistic similarities.

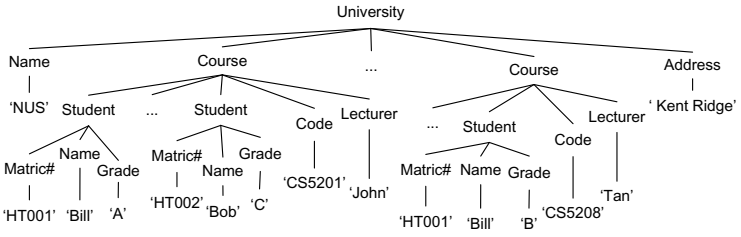


Fig. 1. An XML data tree

However, most practical applications are semantics-less, as most existing XML schema languages, e.g., DTD and XSD, cannot fully represent the semantics such as object class, relationship type, OID, etc. Despite the existence of semantically rich XML models, e.g., ORA-SS, they still requires manual provision of semantics from the initial design or model transformation. We believe only if the automatic semantics discovery technique is developed to a satisfactory level, the achievements in semantics-based query optimization, keyword search, schema/data integration, etc., will be widely adopted by different applications.

In this paper we present a novel approach to automatically discover the ORA-semantics from data-centric XML schemas and data. Different from the existing approaches that only focus on object identification, we consider a comprehensive set of ORA-semantics, including OID, relationship type as well as the distinction between object attribute and relationship attribute.

2 Preliminary

We refer the tree structure derived from XML schemas as *XML schema trees*. For ease of description, all following concepts are defined on XML schema trees.

In XML schema tree, ***object class*** is an internal node representing a real world entity or concept. An object class has a set of ***object attributes*** to describe its properties. Each object class has an ***object identifier (OID)*** to uniquely identify its instance. Several object classes may be connected through a ***relationship type*** which may or may not explicitly appear in the XML schema tree. We call them ***explicit relationship type*** and ***implicit relationship***

type. A relationship type may have a set of ***relationship attributes***. ***Aggregational node*** aggregates its child nodes with identical/similar meaning. ***Composite attribute*** is an object/relationship attribute containing multiple components, each of which can be a single attribute or a composite attribute.

Based on the semantic concepts mentioned above, we define the *ORA-Semantics*, which is the scope of the semantic concepts we consider in this paper.

Concept 1. ***ORA-semantics (Object-Relationship-Attribute-semantics)*** *In an XML schema tree, the ORA-semantics is the identification of object class, OID, object attribute, aggregational node, composite attribute and explicit/implicit relationship type with relationship attributes. Each particular semantic concept in ORA-semantics is called an ORA-semantic concept.*

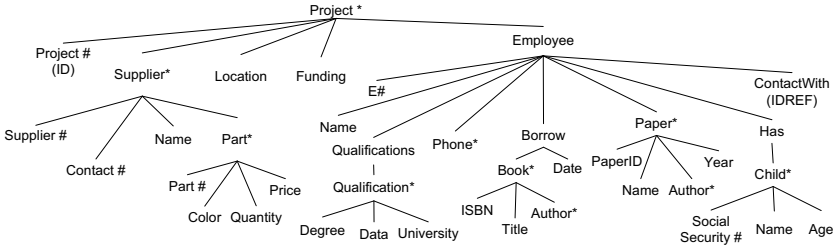


Fig. 2. An XML schema Tree

Example 1. In Fig. 2, we can infer the internal nodes *Project, Supplier, Part, Employee, Book, Paper* and *Child* are object classes, with their OIDs *Project#, Supplier#, Part#, E#, ISBN, PaperID* and *SocialSecurity#*. The internal node *Borrow* is an explicit relationship type between *Employee* and *Book* with a relationship attribute *Date*; the internal node *Has* is an explicit relationship type between *Employee* and *Child* without any relationship attribute. The leaf node *Price* is a relationship attribute of the binary relationship type between *Supplier* and *Part*, and the leaf node *Quantity* is a relationship attribute of the ternary relationship type among *Project, Supplier* and *Part*. The internal node *Qualifications* is an aggregational node, aggregating its child node *Qualification*, which is a composite attribute. All other leaf nodes are object attributes.

3 ORA-Semantics Discovery

We use properties of ORA-semantics, heuristics and data mining techniques to discover the ORA-semantics in data-centric XML schema/XML data. The properties used in our approach conform to the design of the corresponding ORA-SS model or ER model, and the heuristics are summarized based on the characteristics and our observations of different ORA-semantics concepts. In case an XML schema is not available with XML data, XML schema summarization/extraction has been studied in [3]. Fig. 3 shows the road map of our approach.

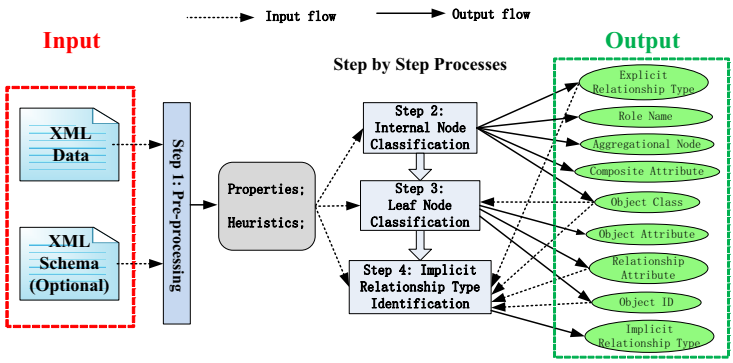


Fig. 3. General process of our automatic semantics discovery approach

3.1 Step 1: Pre-processing

We summarize properties of ORA-semantics concepts. Properties of an ORA-semantics concept are its necessary conditions, which means given an ORA-semantics concept, it must satisfy its properties. E.g, object class has property *'Having more than one child node in its XML schema tree'*, which also conforms to its design in ORA-SS model. We also identify sufficient conditions, by which we can identify a particular ORA-semantics concept, e.g. *'Having an ID attribute in its XML schema as its child node.'* is a sufficient condition for object class. We also proposed heuristics related to ORA-semantics concepts. Some are abstracted from XML schema based on the common way of schema design, and some are discovered from XML data using data mining techniques. We list the properties, sufficient conditions and heuristics for each ORA-semantics concept in Table 1.

In an XML schema tree, a node must be either an internal node or a leaf node. Based on the properties of each ORA-semantics concept in Table 1, internal nodes can be object class, role name, composite attribute, aggregational node and explicit relationship type; while leaf nodes can be OID, object attribute and relationship attribute. We will identify them in 3.2 and 3.3 respectively. There is another ORA-semantics concept, implicit relationship type, which is not explicit shown in the XML schema or XML schema tree. We will identify it in 3.4.

3.2 Step 2: Internal Node Classification

To classify internal nodes, we build a decision tree, Fig. 4, with the properties, sufficient conditions and heuristics in Table 1. We use bottom-up approach so that the category of an internal node can help to identify the category of its parent node. We will explain the decision tree using following rules:

Table 1. Properties, sufficient conditions and heuristics of ORA-semantics concepts

ORA-semantics	Properties (Necessary Conditions)	Sufficient Conditions	Heuristics / Observations	Examples
Object Class	**O1)** It is an internal node;	**A)** It has ID attribute in its XML schema;(E.g. *Project*)		*Supplier*
	O2) It has more than one child node;			*Employee*
	O3) It has at least one FD/MVD among its EDLNs;			*Part*
	O4) Not all nodes in the LHS of each of its FDs/MVDs are IDREF attribute;			*etc.*
Explicit Relationship Type	**E1)** It is an internal node;		**H1**) Its tag name can be a verb form.	*Borrow*
	E2) It has at least one object class, IDREF(S) attribute or role name as descendant node.			*Has*
	E3) If it has at least one FDs/MVDs among its EDLN(s), then all nodes in the LHS of each of its FDs/MVDs are IDREF attributes;			*RentBy*
	E4) Its EDLN(s) should be relationship attribute;			*Buy*
Aggregational Node	**A1)** It is an internal node;		**H2)** Its tag name is the plural form of the tag name of its only child node;	*Qualifications*
	A2) It has only one child node;			
	A3) Its child node is a repeatable node;			
Composite Attribute	**C1)** It is an internal node;			*Qualification*
	C2) It has more than one child node;			
	C3) It does not have FD/MVD among its EDLNs;			
	C4) It hasn't any object class, IDREF(s) attribute or role name as its descendant node;			
OID of object class	**OID1)** It is a leaf node;	**B)** It is specified as ID attribute in XML schema; (E.g. *Project #*)		*Project#*
	OID2) Together with OID(s) of some(zero or more) of its ancestor object class(es), they can functionally multi-valued determine all EDLN(s) of the object class;			*ISBN*
				etc.
Object Attribute	**OA1)** It is a leaf node;			*Location*
	OA2) It can be functionally/multi-valued determined by the OID of its lowest ancestor object class;			*Address*
	OA3) Its lowest ancestor object class is the object class it belongs to;			*Author*
				etc.
Relationship Attribute	**RA1)** It is a leaf node;			*Quantity*
	RA2) It cannot be functionally/multi-valued determined by the OID of its lowest ancestor object class;			*Price*
	RA3) It can be functionally/multi-valued determined by OIDs of all object classes involved in the relationship type to which the relationship attribute belongs;			
	RA4) It is an EDLN of an explicit relationship type or EDLN of the lowest object class that involves in an implicit relationhsip type to which the relationship attribute belongs;			
Role Name	**R1)** It is an internal node;		**H3)** Its tag name shares high linguistic similarity with or being a specialization of the tag name of the object class which the IDREF(S) attribute references;	*Landlord*
	R2) It has only one child node;			*Tenant*
	R3) Its child node is not a repeatable node;			
	R4) Its child node is an IDREF(S) attribute;			

Rule 1. *[**Object Class** vs. **OID**] Given an XML schema tree, if an internal node has an ID attribute*[1] *specified in its XML schema as its child, then this internal node is an object class, and the ID attribute is the OID of the object class.*

Rule 1 is obvious. However, some OIDs may not or cannot be specified as ID attribute in the corresponding XML schema because of the limitation of XML schema language. In XML data, the value of an ID attribute is required to be unique for the corresponding object in the whole document, which makes it impossible for some object classes to have ID attribute being specified in their XML schemas. E.g, in Fig. 2, *Project#* is specified as OID for object class *Project* by ID attribute, but *Supplier#* and *Part#* cannot. Otherwise, a supplier can only supply one project and a part can only be supplied by one supplier. Because of this, we use following rules to classify the rest of the internal nodes.

Concept 2. ***Exclusive Descendant Leaf Node (EDLN)*** *In XML schema tree, an exclusive descendant leaf node of an internal node* i *is a leaf node, which is also a descendant node of* i*, but not a descendant node of any other object class which is also a descendant node of* i.

[1] ID attribute is specified in DTD. In XSD there is a similar concept, key element, which can also be used to identify object class and its OID. For simpleness, Rule 1 is illustrated using ID attribute, but key element also applies. Detail of key element in XSD is given in our technical report [5].

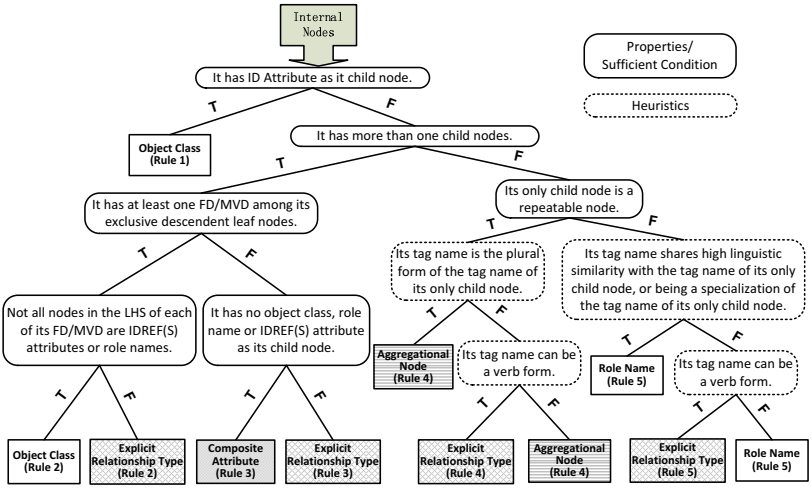


Fig. 4. Decision Tree for Internal Node Classification

The intuitive meaning of EDLN is: given an internal node i, each EDLN of i is a leaf node under i, but there is no other object class between the EDLN and i. E.g, in Fig. 2, the EDLNs of object class *Project* are: *Location* and *Funding*.

Rule 2. *[**Object Class** vs. **Explicit Relationship Type**] Given an XML schema tree, let* i *be an internal node with more than one child nodes and there is at least one Functional/Multi-valued Dependency(FD/MVD) among its EDLNs. If for each FD/MVD, there is a left hand side (LHS) node which is not an IDREF attribute or role name then* i *is an object class, else* i *is an explicit relationship type.*

An object class must have more than one child node (`O2` in Table 1), which conflicts with the properties of aggregation node (`A2`) and role name (`R2`), and at least one FD/MVD among its EDLNs (`O3`) that conflict with composite attribute (`C3`). However, explicit relationship type also has these characteristics. To distinguish object class from explicit relationship type, we check whether there is a LHS node in FD/MVD which is not an IDREF attribute or role name (`O4`, `E3`). This is because the FD/MVD among the EDLNs of an explicit relationship type must involve a relationship attribute, which is functionally/multi-valued determined by the OIDs of all involved object classes, and these OIDs can only be represented as IDREF attributes or role names if it is an EDLN of the explicit relationship type. On the other hand, for FD/MVD among the EDLNs of an object class, its LHS should contain the OID of this object class, which is not an IDREF attribute or role name. FDs/MVDs in XML can be identified in [13].

Rule 3. *[**Composite Attribute** vs. **Explicit Relationship Type**] Given an XML schema tree, let* i *be an internal node with more than one child node and there is no FD/MVD among its exclusive descendant leaf nodes. If* i *does not*

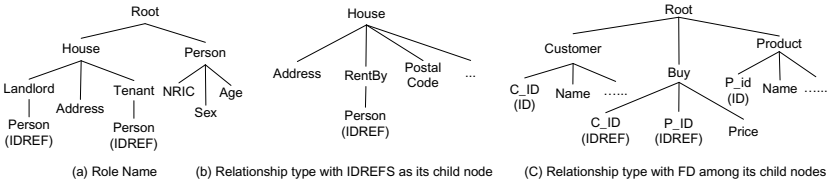


Fig. 5. Internal nodes with IDREF(S) in XML schema tree

have object class, role name or IDREF(S) attribute as its child node, then i *is a composite attribute, else* i *is an explicit relationship type.*

Composite attributes have more than one child node (`C2`), which distinguishes it from aggregation nodes (`A2`) and role names (`R2`). As discussed before, composite attribute has been distinguished from object class. To distinguish composite attribute from explicit relationship type (e.g., composite attribute *Qualification* and explicit relationship type *Borrow* in Fig.2), we check whether it has object class, role name or IDREF(S) attribute as its child node (`C4`, `E2`). This is because explicit relationship types should have at least one object class, IDREF(S) attribute or role name as its descendant nodes to represent the involved object class, while composite attributes should not.

Because of the space limit, we leave the rules to identify aggregational node and role name in our technical report [5]. In the following section, we aim to classify them into OID, object attribute and relationship attribute.

3.3 Leaf Node Classification

OID Discovery. As stated in Rule 1, OID can be explicitly specified in the XML schema with ID attribute, which is a sufficient condition to identify OID. Here we only consider the case that the single-attributed OID is not specified in the XML schema (e.g., *ISBN* in Fig. 2), or the OID contains multiple attributes. Before we explain our approach, we first introduce a concept named *Super OID*.

Concept 3. ***Super OID*** *The super OID of an object class o is a minimal set of nodes which contains a subset of the exclusive descendant leaf nodes of o and the OIDs of some ancestor object classes of o. Super OID of o can functionally/multi-valued determine all exclusive descendant leaf nodes of o.*

In an XML schema tree, given an object class o, its EDLNs may be object attributes of o, or attributes of a relationship type in which o participates. Based on the definition of super OID, the properties of OID and object/relationship attribute (i.e. `OID2`, `OA2`, `RA2` and `RA3` in Table.1), the super OID of o can functionally/multi-valued determine both object attributes and relationship attribute of o, while the OID of o can only functionally/multi-valued determine the object attributes of o. The rationale of our approach to identify OIDs is that given an object class o, there is a minimal attribute set S formed by the OID of

o and the OIDs of some ancestor object classes of o, which functionally/multi-valued determine all EDLNs of o. If there is no relationship attribute being EDLN of o, no OID of ancestor object class of o will be included in S. Thus, from the super OID of object class o, its corresponding OID can be derived by excluding all OID(s) of the ancestors object class(es) of o.

We proposed a top-down approach (the OID of an ancestor object class may be needed to identify the OIDs of its descendant object classes) shown in Algorithm 1 to identify the OID of each identified object class without ID attribute being specified in its XML schema. Given an object class o, we create a set $SupEDLN_o$, which is a superset of its exclusive descendant leaf nodes, denoted as $EDLN(o)$, and include OIDs of all its ancestor object classes. In $SupEDLN_o$, we identify the super OID of o. There may be more than one super OID for o. For each super OID, we can get an OID candidate for o by excluding all OID(s) of ancestor object class(es) of o. (Details about choosing which OID candidate as the OID will be discussed later.) For the object class without ancestor object class, its only super OID will be the same as its OID.

Algorithm 1. Candidate OID Discovery

Input: Identified object classes $\mathbb{O}$; $EDLN(o)$ for each identified object class $o \in \mathbb{O}$;
Output: Candidate OID id_o for each identified object class $o \in \mathbb{O}$

```
1  foreach identified object class o ∈ 𝕆 do
2      SupEDLN_o=EDLN(o);
3      foreach o_i ∈ 𝕆, which is ancestor object class of o do
4          SupEDLN_o = SupEDLN_o ∪ id_oi;          //id_oi is the OID of object class o_i
5      foreach SID_o ⊂ SupEDLN_o do
6          if ∀e ∈ EDLN(o), such that SID_o → e or SID_o ↠ e then
7              if ∄S ⊂ SID_o, such that ∀e ∈ EDLN(o), such that S → e or S ↠ e then
8                  if ∀o_j ∈ 𝕆 with its OID id_oj ∈ EID_o, such that ∃o_k ∈ 𝕆 with its OID
                   id_ok, AD(o_j, o_k) and AD(o_k, o), then id_ok ∈ SID_o then
9                      foreach e ∈ SID_o do
10                         if e ∈ EDLN(o) then
11                             e ∈ id_o;
12                 return id_o as a candidate OID of o.
```

Example 2. In Fig. 2, considering 3 identified object classes *Project*, *Supplier* and *Part* with their EDLNs, suppose we get the following full FDs from the XML data: {*Project#*}→{*Location,Funding*}, {*Supplier#*}→{*Contact#,Name*}, {*Part#*}→{*Color*}, {*Supplier#,Part#*}→{*Price*}, {*Project#,Supplier#,Part#*} →{*Quantity*}. For object class *Project*, we can identify *Project#* as its OID by Rule 1, as it is an ID attribute. For object class *Supplier*, as the attribute *Supplier#* functionally determines all its EDLNs, we identify *Supplier#* as its OID, the same as its super OID. For object class *Part*, we combine its EDLNs and OIDs of its ancestor object classes *Supplier* and *Project*, and use the above given FDs to discover the minimal subsets that functionally determine all its EDLNs to be its super OID, which are {Project#, Supplier#, Part#}, {Supplier#, Part#, Quantity} and {Part#, Quantity, Price}. Then we get {Part#}, {Part#, Quantity} and {Part#, Quantity, Price} as OID candidates of object class *Part*.

We use the following heuristics summarized from our observations to choose the best OID from all OID candidates returned by Algorithm 1.

Observation 1. *[OID] In XML schema tree, given an object class* o*, its OID* id_o *is likely to be designed with some of the following features: (1)* id_o *is a single attribute of* o*; (2) The first child node of* o *is (part of)* id_o*; (3)* id_o *contains substring 'Identifier', 'Number', 'Key' or their abbreviations in its tag name; (4)* id_o *has numeric as (part of) its value, and the numerical part is in sequence.*

Observation 1 is based on structural/linguistic characteristics of OIDs designed in real world. Besides, we have two more observations: (1) the number of object classes without relationship attribute is more than the number of object classes with relationship attributes; (2) the number of relationship attributes of binary relationship type is more than the number of relationship attributes of ternary relationship type, and so on. Based on these observations, we collect 204 object classes with their OIDs being manually specified in XML schemas and extract the statistics mentioned above. Using such statistics, we train a Bayesian Network to rank all OID candidates, and choose the best one as its OID. In Example 2, for the object class *Part*, among all its OID candidates, {Project#, Supplier#,Part#} get the highest ranking using our Bayesian Network ranking model. Thus, {*Part#*} is identified as the OID of object class *Part*. More details of our Bayesian Network ranking model can be found in our technical report [5].

Object Attribute and Relationship Attribute Discovery. For an explicit relationship type, we identify its EDLNs that are not role names, as its relationship attributes based on its property (i.e. `E4` and `RA4` in Table 1). For implicit relationship type, its relationship attributes should appear as EDLNs of the lowest object class participating in the relationship type (`RA4` in Table 1), together with the object attributes of that object class. Based on these, we propose Rule 4 to distinguish object attributes and relationship attributes among the EDLNs of each identified object class with OID identified. We use the properties that object attribute can be functionally/multi-valued determined by OID of the object class it belongs to, while relationship attribute can not, to differentiate them.

Rule 4. *[**Object Attribute** vs. **Relationship Attribute**] Given an object class* o *and its OID, if an exclusive descendant leaf node* e *of* o *can be functionally/multi-valued determined by the OID of* o*, then* e *is an attribute of* o*, otherwise it is an attribute of an implicit relationship type which* o *involves in.*

Example 3. In Figure 2, given the object class *Part* with its OID *Part#*, its child node *Color* is functionally dependent on its OID, while *Quantity* and *Price* are not. Thus, we identify *Color* as an object attribute of *Part*, while *Quantity*, *Price* as relationship attributes of some relationship types that *Part* involves in. The corresponding relationship types will be discovered in the following Step 4.

3.4 Step 4: Implicit Relationship Type Discovery

Recall that explicit relationship type can be identified by Rule 2, 3 in Step 2 in Section 3.2. However, there are some implicit relationship types which are not

explicitly represented as any node in its XML schema tree. In this section, we classify implicit relationship type into four categories: (1) Implicit relationship type with at least one relationship attribute; (2) Implicit relationship type with IDREF(S) attribute; (3) Implicit relationship type with no relationship attribute and no IDREF(S) attribute, and (4) Identifier Dependency (IDD) Relationship Type [6]. Because of the space limit, we only discuss the first two categories in this paper. The other two categories are discussed in our technical report [5].

Implicit Relationship Type with at Least One Relationship Attribute. For each relationship attribute discovered in Section 3.3 (except those being EDLNs of explicit relationship type), there must be an implicit relationship type it belongs to. Based on the property that relationship attribute should be functionally/multi-valued determined by the OIDs of all object classes involved in the implicit relationship type, to which the relationship attribute belongs (i.e. `RA4` in Table 1), we proposed a bottom-up approach, Algorithm 2, to identify the implicit relationship type with its degree, and all involved object classes.

Example 4. In Fig. 2, given a relationship attribute *Price*, object classes *Project*, *Supplier*, *Part*, and their OIDs. By Algorithm 2, we find out {*Supplier#,Part#*} → {*Price*}. Then, there is an implicit binary relationship type between *Supplier* and *Part*, with relationship attribute *Price*. For another relationship attribute *Quantity*, {*Supplier#,Part#*} cannot functionally/multi-valued determine it. Then we add in the OID of object class *Project*, and get {*Project#, Supplier#,Part#*}→ {*Quantity*}. Then there is an implicit ternary relationship type among *Project*, *Supplier* and *Part*, with relationship attribute *Quantity*.

Algorithm 2. Implicit Relationship Type with Relationship Attribute

Input: Relationship attribute $\mathbb{A}$; Object classes $\mathbb{O}$, with OIDs; XML schema tree; XML data
Output: Relationship type $r(\mathbb{C})$ with its involved object classes $\mathbb{C}$ and degree $|\mathbb{C}|$, for each identified relationship attribute in $\mathbb{A}$

1 **foreach** *identified relationship attribute* $ra \in \mathbb{A}$ **do**
2 o_i = the lowest ancestor object class of ra.
3 $\mathbb{C} = \{o_i\}$;
4 $SemID_{ra} = id_{oi}$; //id_{oi} `is the OID of object class` o_i;
5 **foreach** *identified object class* $o_j \in \mathbb{O}$, *along the path from* o_i *to the root in its XML schema tree in bottom-up order* **do**
6 $SemID_{ra} = SemID_{ra} \cup id_{oj}$; //$id_{oj}$ `is the OID of object class` o_j;
7 $\mathbb{C} = \mathbb{C} \cup \{o_j\}$;
8 **if** $SemID_{ra} \rightarrow ra$ or $SemID_{ra} \twoheadrightarrow ra$; **then break**;
9 return implicit relationship type $r(\mathbb{C})$ to which ra belongs, object classes in $\mathbb{C}$ as its involved object classes and $|\mathbb{C}|$ as its degree;

Implicit Relationship Type with IDREF(S) Attribute. In XML schema, some designers may design an implicit relationship type by specifying an IDREF(S) attribute under an object class, which references other object class(es). Thus, if an object class has a child node specified as an IDREF(S) attribute, we identify an implicit relationship type between the object class and the object class(es) the IDREF(S) attribute refers to. For some XML schema language (e.g.,DTD), we do

not know to which object class(es) the IDREF(S) attribute refer. Based on the property and the heuristic of IDREF(S) attribute listed in Table 1, there are two ways to identify the object classes involved in implicit relationship type: (1) [H3] Tag name of the IDREF(S) attribute may share high linguistic similarity with the tag name of the object class(es) to which it refers, or the corresponding OID(s). We can identify them by research work [9] comparing linguistic similarity. E.g., given two object classes *Department* and *Staff* with their OIDs *Dept#* and *Staff#* respectively, if there is an IDREF(S) attribute under *Staff* with its tag name as *Dept#*, we identify an implicit relationship type between *Department* and *Staff*; (2) If we cannot find high linguistic similarity between the IDREF(S) attribute and any object class or OID, we can use the XML data to identify which object class(es) the IDREF(S) attribute references. A property of IDREF(S) attribute is that [I1] the value range of the IDREF(S) attribute in its XML data must be a subset of the value range of the OID of the object class(es) which it references. E.g., in Fig. 2, if we know that every value of the IDREFS attribute *ContactWith* is also found as a value of OID of object class *Supplier*, there is a high possibility that there is an implicit relationship type between *Employee* and *Supplier*. Furthermore, as the property of IDREF(S) attribute, I1 is also used to verify H3. Although neither of the two ways can 100% guarantee that the object class we discover is the corresponding object class which the IDREF(S) references, we will show the accuracy in our experiments.

4 Experiment

We evaluate the proposed approach for discovering the ORA-semantics in the given XML schemas. The experimental data includes 15 real world data-centric XML schemas, e.g., mondial[2] and XMark[3], etc. For all XML schemas used in our experiments, the average number of internal node is 11 and the average maximal depth is 5. To evaluate the accuracy of our approach, we measure precision, recall and F-measure[4] against a gold standard provided by 8 evaluators. Divergence in their opinions is accounted for by means of an uncertainty factor weighting the results. Further details are given in [5].

4.1 Accuracy of Internal Node Classification

There are totally 512 internal nodes in our input XML schema trees, with their ORA-semantics being labelled (i.e., object class, role name, explicit relationship type, aggregational node or composite attribute). Table 2 shows that the overall accuracy of our rules achieves almost 95% of precision, recall and F-measure. The low precision and recall for explicit relationship type as well as low precision for role name and aggregational node are because the related heuristics used are not as accurate as the properties used in our rules. In Table 3, we show the

[2] http://www.cs.washington.edu/research/xmldatasets/www/repository.html
[3] http://www.xml-benchmark.org/
[4] F-measure = 2 * precision * recall/(precision + recall)

Table 2. Precision, recall and F-measure of internal node classification

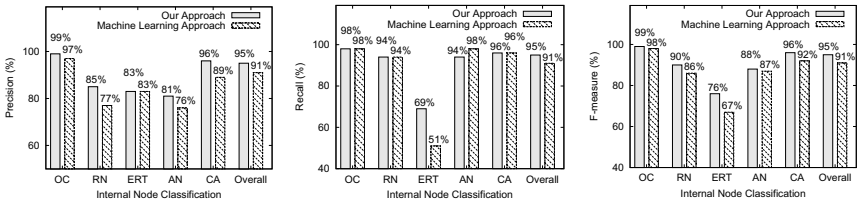
	Object Class	Role Name	Explicit Relationship Type	Aggregational Node	Composite Attribute	Overall
Precision	99.4%	85.0%	82.9%	81.0%	96.3%	94.7%
Recall	98.4%	94.4%	69.4%	94.4%	96.3%	94.7%
F-measure	98.9%	89.7%	76.2%	87.7%	96.3%	94.7%

Table 3. Statistic information of the internal node in experiment data

	Object Class	Role Name	Explicit Relationship Type	Aggregational Node	Composite Attribute	Total
# of nodes	311	18	49	54	80	512
Percentage	60.7%	3.5%	9.6%	10.5%	15.6%	100%

number and percentage of each ORA-semantics concept in all our collected data sets. Object class is one of the most important ORA-semantics concepts, and its identification helps many XML applications to increase their efficiency and effectiveness as introduced in Section 1. There are 311 object classes among all 512 internal nodes, which take up around 60% of all internal nodes. Other ORA-semantics concepts only take up a small percentage of the internal nodes, especially for role name, which takes up less than 5% of the internal nodes.

We also used a machine learning approach to classify the internal nodes for comparison purpose. We use the properties listed in Table 1 with all our experimental data to train a classification model to classify the internal nodes. In order to avoid bias because the selection of training data, we use 3 folds cross-verification with all the input internal nodes. In 3 folds cross-validation, the original input data is randomly partitioned into 3 portions. Of the 3 portions, a single portion is retained as the validation data for testing the trained classification model, and the remaining 2 portions are used as training data. The cross-validation process is then repeated 3 times, with each of the 3 portions used exactly once as the validation data. The 3 results of precision, recall and F-measure then can be averaged to get the overall accuracy of the trained classification model. We compare the accuracy of our rules with the trained classification model in Fig. 6. We choose the frequently used decision tree algorithm C4.5 [10] to build the classification models. The results show our rules work better than the trained classification models, especially for discovering the explicit relationship types. This is because the explicit relationship type can be designed with different structures in XML schema tree; the classification of a descendent node also cannot help classifying ancestor node as in our approach using rules. Furthermore, the decision trees trained from different training data sets are quite different from each other and most of their branches are not as meaningful as our rules, which shows that they are heavily dependent on the training data. More details about the internal node classification test can be found in [5].



(a) Precision (b) Recall (c) F-measure

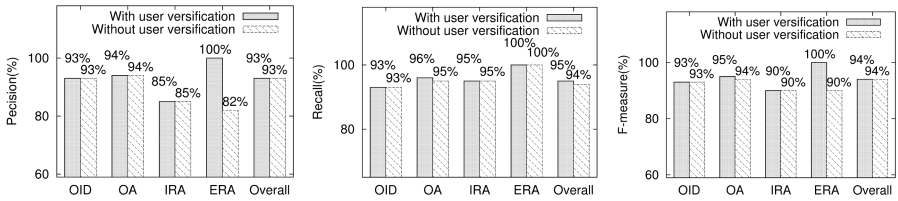
Fig. 6. Comparison of internal node identification (OC: Object Class; RN: Role Name; ERT: Explicit Relationship Type; AN: Aggregational node; CA: Composite Attribute)

4.2 Accuracy of Leaf Node Classification

Recall that our approach may return more than one OID candidates for each object class, thus we build a Bayesian Network to rank all its OID candidates, and choose the highest ranked candidate as its OID. There are 311 object classes with their OIDs in our experimental data. We randomly choose 2/3 of them for training and the rest for testing. From the training data, we collect the statistics of the features mentioned in Observation 1, and build a Bayesian Network, which returns us the probability of an OID candidate being the correct OID based on the statistics. More details of building the Bayesian Network are discussed in [5].

In our step-by-step approach, outputs of the previous step will work as the inputs for a latter step. The accuracy of the latter step is affected by the accuracy of its previous steps. To show the accuracy of each step, we conduct two groups of experiments to evaluate the precision, recall and F-measure of our approach for leaf node classification, one with user verification, which means all object classes have been correctly labelled in XML schema trees, and the other one based on the results of our internal node classification without user verification.

Fig. 7 shows our approach for leaf node classification get above 90% of precision, recall and F-measure. Even without user verification, the precision/recall only drop slightly, as our approach for discovering object class also gets high precision and recall. The low precision of implicit relationship attribute is because



(a) Precision (b) Recall (c) F-measure

Fig. 7. Precision, Recall and F-measure of Leaf Node Classification (OA: Object Attribute, ERA: Explicit Relationship Attribute, IRA: Implicit Relationship Attribute)

its identification is heavily depended on FDs/MVDs among the corresponding XML data, which may not be large enough to return all the correct FDs/MVDs.

4.3 Accuracy of Implicit Relationship Type Discovery

We also conduct experiments on our approach for implicit relationship type discovery. Similar to the leaf node classification, we conduct two groups of experiments to evaluate its accuracy, one with user verification, and the other one without user verification. Fig. 8 shows that our approach to discover implicit relationship types has high precision, recall and F-measure. Because of the space limit, more detailed breakdown is given in [5].

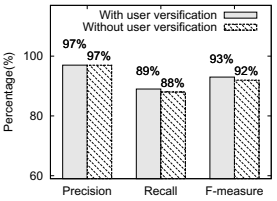


Fig. 8. Precision, Recall, F-measure of Implicit Relationship Type Discovery

5 Related Work

To the best of our knowledge, only a few research works have frontally addressed the problem of automatically discovering the implicit semantics embedded in XML schema and XML data. Most existing works in semantics discovery in XML data only focus on objects. In [2], in the context of view design, all internal nodes in an XML schema tree are considered as object classes. In the context of keyword search, XSeek [7] also infers semantics from XML schemas to identify return nodes. This work infers semantics of objects by using the repeatable node. In [11], the authors build a data graph from an XML document. However, they just focus on objects and properties, still missing lots of meaningful semantics. Compared to our work, the existing works have two major drawbacks. First, they only consider the semantics of object, ignoring many other important ORA-semantics concepts which may play an important role in XML applications, as illustrated. Second, even for object, the inference accuracy of the existing works is quite low. For example, most of them will treat relationship attribute as object attribute when the relationship is implicit. In contrast, our work focuses on a comprehensive set of ORA-semantics concepts, and has high inference accuracy.

Semantics are also captured in other domains. In [8] authors proposed a form-driven approach, which firstly transforms the relational database to a set of form model schemas, each of which is essential a view on the underlying database, and then extracts the corresponding ER schema from them; [4] resolves the reference ambiguation problem, which means an attribute is actually referencing another

attribute but cannot be detected due to the inconsistent name issue, by also considering their neighbor attributes. However, as the underlying data is flat in relational database, and these approaches only try to identify relationships between relations through key-foreign key constraints, they still cannot identify ternary or n-nary relationships as well as the relationship attributes.

6 Conclusion and Future Work

ORA-semantics is important for many XML applications. Existing works in semantics discovery only focus on object, ignoring many other important concepts such as relationships. In this paper we present a novel approach to identify a comprehensive set of ORA-semantics, including object, object ID, explicit/implicit relationship, relationship attribute, etc. We analyze the properties of each semantics concepts, and propose rules and apply data mining techniques to discover them in XML schema and data. We conduct experiments to demonstrate our approach can achieve almost 95% overall precision, recall and F-measure.

We are now investigating those cases that still defeat our proposed approach and consider its combination with additional domain knowledge and ontologies.

References

1. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and ontology matching with COMA++. In: SIGMOD Conference, pp. 906–908 (2005)
2. Chen, Y.B., Ling, T.W., Lee, M.L.: Designing valid XML views. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 463–477. Springer, Heidelberg (2002)
3. Hegewald, J., Naumann, F., Weis, M.: Xstruct: Efficient schema extraction from multiple and large XML documents. In: ICDE Workshops, p. 81 (2006)
4. Kalashnikov, D.V., Mehrotra, S.: Domain-independent data cleaning via analysis of entity-relationship graph. ACM Trans. Database Syst. 31(2), 716–767 (2006)
5. Li, L., Le, T.N., Wu, H., Ling, T.W., Bressan, S.: Discovering semantics from data-centric XML. Technical Report TRA6/13, National University of Singapore
6. Ling, T.W., Lee, M.L., Dobbie, G.: Semistructured database design (2005)
7. Liu, Z., Chen, Y.: Identifying meaningful return information for XML keyword search. In: SIGMOD Conference, pp. 329–340 (2007)
8. Mfourga, N.: Extracting entity-relationship schemas from relational databases: A form-driven approach. In: WCRE, pp. 184–193 (1997)
9. Mizuta, S., Hanya, K.: Specifications of word set in linguistic approach for similarity estimation. In: BICoB, pp. 25–29 (2010)
10. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann (1993)
11. Y.S.: A personal perspective on keyword search over data graphs. In: ICDT (2013)
12. Xu, Y., Papakonstantinou, Y.: Efficient lca based keyword search in XML data. In: EDBT, pp. 535–546 (2008)
13. Yu, C., Jagadish, H.V.: XML schema refinement through redundancy detection and normalization. VLDB J. 17(2), 203–223 (2008)