T H E N A T I O N A L U N I V E R S I T Y
o f S I N G A P O R E

S c h o o l o f C o m p u t i n g
Computing 1, 13 Computing Drive, Singapore 117417

TRA6/13

Discovering Semantics from Data-Centric XML

Luochen Li, Thuy Ngoc Le, Huayu Wu, Tok Wang Ling and Stephane Bressan

June 2013

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

Discovering Semantics from Data-Centric XML

Luochen Li [#], Thuy Ngoc Le [#], Huayu Wu [*], Tok Wang Ling [#], and Stephane Bressan [#]

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

Abstract. In database applications in general, and in applications using XML in particular, the availability of a conceptual schema or of elements of semantics constitute invaluable leverage for improving the effectiveness, and sometimes the efficiency, of many tasks including query processing, keyword search and schema and data integration. The Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) model is a conceptual model designed to capture the semantics of the important constructs in a variety of data models in general and in semi-structured data models in particular. It is specifically intended to capture the semantics of object classes, object identifiers, relationship types, object attributes and relationship attributes underlying XML schemas and data. For a given application, we refer to the set of instances of these semantic concepts as the ORA-semantics of the application. While ORA-SS can be used a priori for the design of new applications, we are interested, in this work, in the automatically discovering of the ORA-semantics from existing XML data and XML schemas. In this paper, we present a novel approach to automatically discover the ORA-semantics from XML schemas and XML data. The approach we proposed is based on a set of techniques and heuristics that identify the different semantic concepts. We empirically and comparatively evaluate the effectiveness of the approach. We show by experiments that the semantics discovered by our approach has more than 90% accuracy.

Keywords: XML, Semantics, Data Mining

1 Introduction

Actors in several industry sectors such as the automotive industry and the chemical industry are now commonly using XML to define standards, in the form of XML schemas, for the representation of technical and business data, and to electronically interchange data among business partners. Indeed, XML can effectively replace archaic electronic data interchange formats, which were confusing physical, logical and conceptual levels in the name of efficiency, and which

were generally hard to understand by users, thus creating inflexible information systems. Orthogonal techniques such as compression can provide the wired efficiency. The conceptual quality is provided by the logical nature of the XML data model and the suite of tools it techniques that accompany it such as keyword search and techniques for schema and data integration.

To improve the conceptual quality, one 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), object attributes and relationship attributes, as present in conceptual models for semi-structured data such Object-Relationship-Attribute for Semi-Structured data (ORA-SS) [9]. 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. We use the XML document in Fig. 1 and the examples to illustrate how the availability of such semantics positively impacts applications.

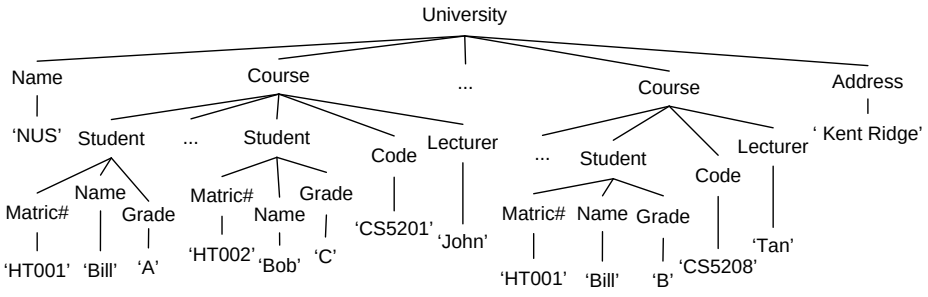


Fig. 1. An XML data tree

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 and *Matric#* functionally determines *Name* as each *Student* instance only has one name, after getting an answer, we can stop searching the rest of the data. Other situations that semantics improves XML query processing are shown in [21].

XML keyword search. Semantics-based XML keyword search approaches have been proposed to improve search efficiency and quality, such as in [20]. However, the use of semantics in current works is still shallow (only on object level, without touching the relationship). For some queries, e.g., {CS5201, CS5208}, only by discovering that there is a relationship type between the object classes *Student* and *Course*, one can infer the meaningful answer of this query should be all the students taking the two courses, e.g., the student with *Matric#* of 'HT001' is one of the answers. Otherwise, the root node

will be returned, as in most LCA-based XML keyword search approaches [23, 16, 22].

Schema/data integration. Most existing approaches (e.g., [1, 8]) integrate elements based on their structural and linguistic similarities. However, this information is still coarse-grained. In the corresponding XML schema of the XML data tree in Fig.1, *Grade* is an attribute of the relationship type between *Course* and *Student*. Without this semantics, when we integrate this schema with another schema, in which *Student* has an object attribute *Grade* which means the year of his/her study in school, we may wrongly integrate these two different attributes which have the same attribute name *Grade* and the same parent object *Student*, because of their high structural and linguistic similarities.

Although semantics-based XML processing is attracting more and more research attention, unfortunately, most practical applications are still semantics-less. The main issue here is the availability of such semantic information. Most existing XML schema languages used by applications, e.g., DTD and XSD, cannot fully represent the semantics such as object class, relationship type, OID, object attribute and relationship attribute. Despite the existence of semantically rich XML models, e.g., ORA-SS [9], such model still requires manual provision of semantics from the initial design or during model transformation. We believe only if the automatic semantics discovery technique is developed to a satisfactory level, the research achievements in semantics-based query optimization, semantics-based keyword search, semantics-based schema/data integration and so on, will be widely adopted by different applications.

Different from the existing semantics inference approaches [3, 10], which only identify objects in XML data or object classes in XML schema, we consider a more comprehensive set of semantic concepts. In this work, we define these semantic concepts as the ORA-semantics (formal definition will be given in Section 2), and propose a novel step-by-step approach to discover these ORA-semantics with the following four steps (including the pre-processing step): (1) We discover the properties of each semantic concepts and propose some related heuristics (if any) based on the characteristics of XML schema and XML data. Most of these properties are captured and represented in the semantic model for XML data, ORA-SS [9]. (2) We use the properties of each semantic concept to distinguish object class from other semantic concepts such as composite attribute, aggregational node and explicit relationship type. For those semantic concepts which cannot be distinguished, we use the related heuristics to help distinguishing them. (3) Together with properties and heuristics discovered in (1), we also utilize the statistic information with data mining techniques to identify OIDs, and then distinguish between object attributes and relationship attributes using the identified OID; (4) We discover the relationship types using the results from the previous steps and functional/multi-valued dependency extracted from the XML data.

The main contributions of this paper include:

1. We reveal the importance of semantics in various XML applications including query processing, keyword search and schema/data integration, and show how semantics can be used to help increasing their efficiencies and accuracies. Our automatic XML semantic discovery approach will work as a foundation for all semantics-based XML models and semantics-based XML query processing, keyword search and schema/data integration techniques.
2. We discover and summarize the properties and related heuristics for different semantic concepts of XML database based on their characteristics and the semantic model for XML database (ORA-SS). These properties and heuristics will be used in our automatic semantics discovery approach.
3. We propose a novel approach to automatically discover semantics from XML schema. Different from the existing semantics inference approaches mentioned in [3, 10], which only identify objects in XML data or object classes in XML schema, our approach considers a more comprehensive set of semantic concepts. In particular, we also discover OIDs, role names, aggregational nodes, composite attributes, relationship types and distinguish between object attributes and relationship attributes.
4. To validate our automatic semantic discovery approach, we conduct experiments over 15 real XML schemas/data and 5 XML schemas translated from real relational data sets by PhD students doing research in XML, to show the ORA-semantics discovered by our approach has high accuracy.

The rest of this paper is organized as follows. In Section 2, we clarify the concepts of semantics used in our approach. In Section 3, we present our automatic ORA-semantics discovery approach. In Section 4, we conduct experiments to show the quality of our approach and compare it with pure data mining approach. Related work is reviewed in Section 5. Conclusion and future work are shown in Section 6.

2 Preliminary

In this section, we describe the concepts used in our approach. We refer to the tree structure derived from an XML schema as an *XML schema tree*. An XML schema tree captures the structural information in an XML schema. For ease of description, all the following concepts are defined on an XML schema tree. Some concepts are adopted from ORA-SS model [9].

In the XML schema tree, there are two types of node: *leaf node* and *internal node*. *Object class* is an internal node which represents a real world entity or concept. An object class has a set of *object attributes* as its child nodes or descendant nodes. Each object class should also have an *object identifier (OID)*

in its object attributes to uniquely identify each object 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 those relationship types which explicitly appear in the XML schema tree as *explicit relationship type* and those not appearing as any node in the XML schema tree as *implicit relationship type*. A relationship type may also have a set of *relationship attributes*. Moreover, we also notice the *identifier dependency (IDD) relationship type* between a normal object class and a *weak object class*, which is similar to the *weak entity type* in ER model [2]. If a relationship type explicitly appears as an internal node, its *relationship attributes* will appear as its child nodes or sometimes descendant nodes. Otherwise, its relationship attributes normally appear under the lowest object class that involves in that relationship type. In the XML schema tree, *role name*, *aggregational node* and *composite attribute* are also represented as internal nodes. Role name is the node with an alias of a certain object class, which uses IDREF(S) attribute to reference the original object class which stores full information about that object class. Aggregational node aggregates its child nodes with identical/similar meaning, and serves as a structural node without extra semantics besides the semantics of its child nodes. Composite attribute is a special kind of object/relationship attribute that contains multiple components, each of which can be a single attribute or another composite attribute. *weak object class* with *identifier dependency (IDD)* with another object class, is similar to the *weak entity type* in ER model.

Based on the 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, role name, object identifier (OID), object attribute, aggregational node, composite attribute and explicit/implicit relationship type with relationship attributes.*

All semantic concepts contained in the ORA-semantics can also be found in the ORA-SS model [9] and ER model [2]. For years these semantic concepts have been proved useful for database designers to easily model the complex cases and design reasonable XML and relational databases.

Example 1. Fig.2 shows an XML schema tree. We can infer that the internal nodes *Project, Supplier, Part, Employee, Book, Paper* and *Child* are object classes, with their OIDs *Project# (specified as an ID attribute in its schema), Supplier#, Part#, E#, ISBN, PaperID* and *FirstName* together with *LastName* respectively. The internal node *Borrow* is an explicit relationship type between object classes *Employee* and *Book* with a relationship attribute *Date*, and the internal node *Has* is also an explicit relationship type between object classes *Employee* and *Child* without any relationship attribute. The leaf node *Price* is a relationship attribute of a binary relationship type between object classes *Supplier* and *Part*, and the leaf node *Quantity* is a relationship attribute of a ternary relationship type among object classes *Project, Supplier* and *Part*. There is identifier dependency(IDD) between *Book* and *Chapter*, which makes *Chapter*

a weak object class with its OID {*ISBN, C#*}. Furthermore, the internal node *Qualification* is a composite attribute and the internal node *Qualifications* is an aggregational node. All the other leaf nodes are object attributes of their corresponding object classes.

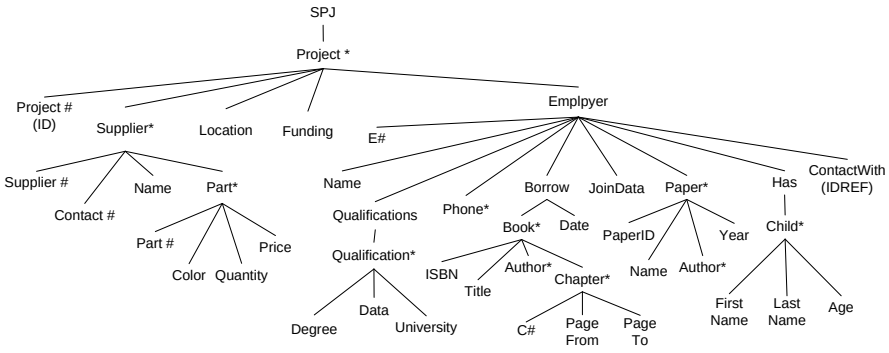


Fig. 2. An XML schema Tree

3 Semantics Discovery

We use properties of ORA-semantics, heuristics and data mining techniques to discover the ORA-semantics with XML schema and XML data as inputs. 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.

XML schema summarization/extraction has been studied in [5], in case an XML schema is not available alongside XML data. The general process of our approach is shown in Fig.3. As mentioned before, our approach includes four steps: (1) Pre-processing. It is a one-time effort and it extracts the properties and heuristics for each ORA-semantics concept. (2) Internal node classification. It uses bottom-up approach to classify all internal nodes in the XML schema tree into one of the following category of ORA-semantics concepts: object class, role name, composite attribute, aggregational node and explicit relationship type; (3) Leaf node classification. It uses a top-down approach to identify OID for each object class, and then distinguishes between object attributes and relationship attributes using the identified OIDs; (4) Implicit relationship type identification. It identifies implicit relationship types, which are not represented in the XML schema.

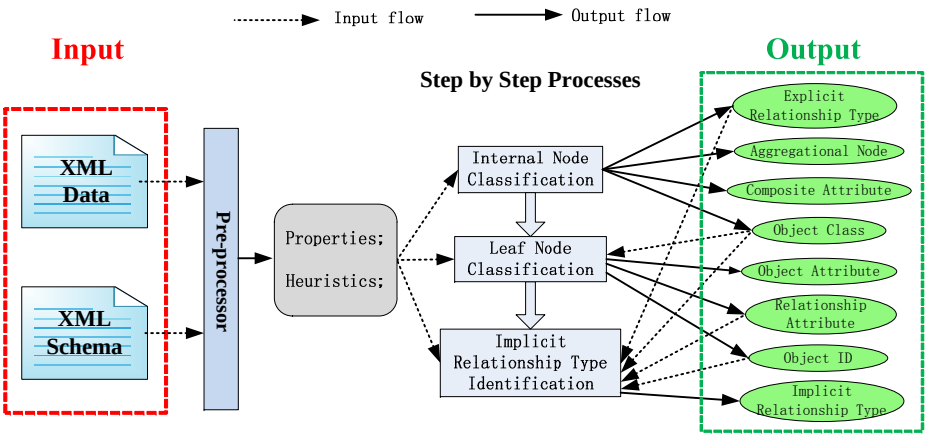


Fig. 3. General process of our automatic semantics discovery approach

3.1 Pre-processing

Recall the semantic model ORA-SS, which is proposed and designed to capture and represent the semantic concepts in XML database. In this semantic model, all semantic concepts are designed with their own properties, especially their hierarchical structures. Thus, in this subsection, we extract and summarize the properties for each ORA-semantics concept from the ORA-SS model. Some properties can also be obtained from the ER model. Although ER model is proposed for relational database, it captures and represents the same semantic concepts, just in a different form.

Properties of an ORA-semantics concept are also necessary conditions of it, which means given an identified ORA-semantics concept, it must satisfy its properties. For example, for object class, it has a property that *"Having more than one child node in its XML schema tree"*, which also conforms to its design in ORA-SS model. Besides the properties (necessary conditions) of each ORA-semantics concept, we can also propose the sufficient condition for some ORA-semantics concepts, which means by satisfying this sufficient condition, we can surely identify an input node as a certain ORA-semantics concept. A sufficient condition for an ORA-semantics concept may or may not be a property of this ORA-semantics concept. For example, for object class, *"Having an ID attribute specified in its XML schema (XML DTD in particular) as its child node in its XML schema tree."* is a sufficient condition to identify an object class.

Besides, we also proposed some heuristics related to different ORA-semantics concepts. All these heuristics are summarized and proposed based on the characteristics and our observations of these ORA-semantics concepts. Some of these characteristics are directly discovered from the XML schema based on the common way of designing the XML schema, and some are discovered from the XML data using data mining approaches and statistic information.

Table 1. Properties(necessary conditions), sufficient conditions and heuristics of different ORA-semantics concepts

ORA-semantics	Properties (Necessary Conditions)	Sufficient Conditions	Heuristics/Observations	Examples
Object Class	**O1)** It is an internal node; **O2)** It has more than one child node; **O3)** It has at least one FD/MVD among its EDLNs; **O4)** Not all nodes in the LHS of each of its FDs/MVDs are IDREF attribute or rule name;	**A)** It has ID attribute in its XML schema; (E.g. *Project*)		*Supplier* *Employee* *Part* *Paper* *etc.*
Weak Object Class	**W1)** It is an internal node; **W2)** It has more than one child node; **W3)** It does not have FD/MVD among its EDLNs; **W4)** It does not has any object class, IDREF(s) attribute or role name as its descendant node; **W5)** It has a child node which cannot functional/ multi-valued determine any other EDLN(s) of the weak object calss, but together with the OID of its lowest ancestor object class, they can functional/multi-valued determine all EDLN(s) of the weak object calss;			*Chapter*
Role Name	**R1)** It is an internal node; **R2)** It has only one child node; **R3)** Its child node is not a repeatable node; **R4)** Its child node is an IDREF(S) attribute;		**H1)** Its tag name shares high linguistic similarity with the tag name of the object class which the IDREF(S) attribute references;	*Landlord* *Tenant*
Explicit Relationship Type	**E1)** It is an internal node; **E2)** It has at least one object class, IDREF(S) attribute or role name as its descendant node. **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 attribute or rule name; **E4)** Its EDLN(s) should be relationship attribute;			*Borrow* *Has* *RentBy* *Buy*
Aggregational Node	**A1)** It is an internal node; **A2)** It has only one child node; **A3)** Its child node is a repeatable node;		**H2)** Its tag name is the plural form of the tag name of its only child node;	*Qualifications*
Composite Attribute	**C1)** It is an internal node; **C2)** It has more than one child node; **C3)** It does not have FD/MVD among its EDLNs; **C4)** It does not has any object class, IDREF(s) attribute or role name as its descendant node;			*Qualification*
OID of object class	**OID1)** It is a leaf node; **OID2)** Together with the 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;	**B)** It is specified as ID attribute in its XML schema; (E.g. *Project #*)		*Project#* *Supplier#* *ISBN* *etc.*
Object Attribute	**OA1)** It is a leaf node; **OA2)** It can be functionally/ multi-valued determined by the OID of its lowest ancestor object class; **OA3)** Its lowest ancestor object class is the object class it belongs to;			*Location* *Address* *Author* *etc.*
Relationship Attribute	**RA1)** It is a leaf node; **RA2)** It cannot be functionally/ multi-valued determined by the OID of its lowest ancestor object class; **RA3)** It can be functionally/ multi-valued determined by OIDs of all object classes involved in the relationship type to which the the relationship attribute belongs; **RA4)** It is an EDLN of an explicit relationship type or an EDLN of the lowest object class which involves in the implicit relationahsip type to which the relationship attribute belongs;			*Quantity* *Price*
IDREF(S) Attribute	**I1)** Its value range is a subset of the value range of the OID of the object class(es) which it references.		**H3)** Its tag name shares high linguistic similarity with the tag name of the object class(es) which it references, or the corresponding OID(s).	*ContactWith*

In the following Table 1, for each ORA-semantics concept considered in our approach, we list its properties (necessary condition), sufficient conditions, related heuristics and example instances in Fig.2. All properties related to the node structure mean the structure in the corresponding XML schema tree, and the examples are from Fig.2 and Fig.5.

Before describing details of our approach, we separate all ORA-semantics concepts considered in this paper into two groups based on their first properties, so that we can introduce our approach easier:

1. It is an internal node in its XML schema tree;

   or

2. It is a leaf node in its XML schema tree;

ORA-semantics are discovered from the XML schema and XML data. In the corresponding XML schema tree, a node must be either an internal node or a

leaf node. Based on the properties of each ORA-semantics concept, all ORA-semantics concept can be grouped as follows: in the group of internal node, they are: object class, role name, composite attribute, aggregational node and explicit relationship type; in the group of leaf node, they are OID, object attribute and relationship attribute. We will discover these two groups of nodes in 3.2 and 3.3 respectively. Furthermore, there is another ORA-semantics concept which is not explicit shown in the XML schema or XML schema tree. It is implicit relationship type, and we will discover it in 3.4.

3.2 Internal Node Classification

The inputs of this step are all internal nodes in the XML schema tree, which can be easily transferred from any XML schema. We classify internal nodes in the XML schema tree into five categories: object class, role name, aggregational node, composite attribute, and explicit relationship type. The goal of this step is classifying all input internal nodes into one of the above five categories.

In order to classify the internal node, we build a decision tree (shown in Fig.4) using the properties, sufficient conditions and heuristics listed in Table1 as building block.

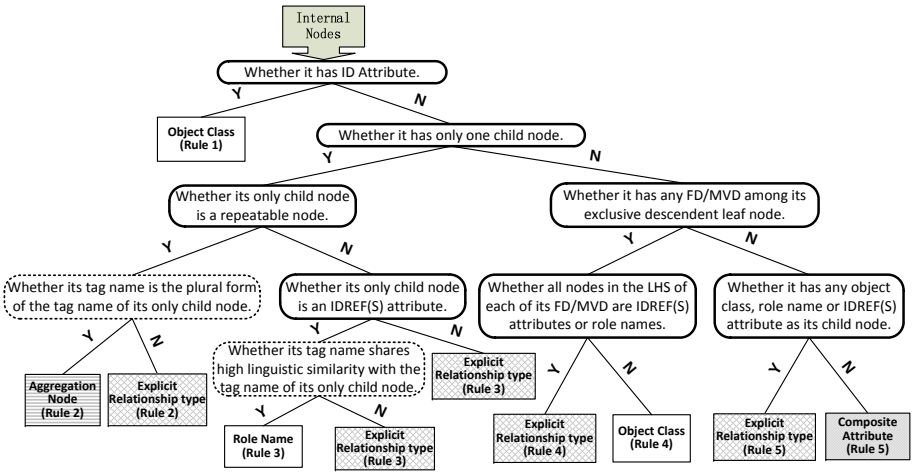


Fig. 4. Decision Tree for Internal Node Classification

In the decision tree, properties and sufficient conditions are represented using bold line rectangles; while heuristics are represented using while dotted line rectangles. Sufficient condition can be directly used as a rule for the corresponding ORA-semantics concept. The differences between properties and heuristics are:

1 Properties are necessary conditions for the corresponding ORA-semantics concept, while heuristics are not, and they are just characteristics of the cor-

responding ORA-semantics concept or some naming convention summarized from observation.

2 Properties can be used as filters to filter out incorrect categories of ORA-semantics concept for each input internal node, while heuristics can only be used to increase the probability of an internal being a certain ORA-semantics concept when the properties are not enough to distinguish them.

3 Properties/sufficient conditions are correct for the corresponding ORA-semantics concept, while heuristics are not guaranteed to be 100% correct.

The pseudocode of our internal node classification is given in Algorithm 1. We use bottom-up approach in our algorithm so that the category of an internal node can help to identify the category of its parent internal node. Furthermore, decision tree shows that our approach for internal node classification is complete, which means all internal nodes in a XML schema tree can be classified into one of the above five categories. For ease of description, we represent the decision tree using the following five rules, and the number in the leaf node of the decision tree is the corresponding rule number that can identify this ORA-semantic concept.

Algorithm 1: Internal Node Classification

Input: Internal nodes $\mathbb{N}$ in XML schema tree, XML data
Output: Identified object classes $\mathbb{O}$, Identified role name $\mathbb{R}$, Identified composite attributes $\mathbb{C}$, Identified aggregational nodes $\mathbb{A}$ and Identified explicit relationship type $\mathbb{ER}$

1 **foreach** *internal node* $n \in \mathbb{N}$ *in bottom-up order* **do**
2 Put n into the decision tree;
3 The decision tree returns the classification of n as object class, role name, composite attribute, aggregational node or explicit relationship type.

Rule 1 *[**Object Class** and **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.*

Rule1 is a sufficient condition for object class, which means every internal node having an ID attribute specified in the XML schema as its child must be an object class. For example, in the XML schema tree in Fig.2, the internal node *Project* with its ID attribute *Project#* should be identified as an object class. Notice that 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.

[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.

For example, in Fig.2, *Project#* can be specified as OID for object class *Project* by ID attribute, but both *Supplier#* and *Part#* cannot. Otherwise, a supplier can only supply one project and a part can only be supplied by one supplier, because of the limitation that the value of an ID attribute cannot appear twice in the same XML data.

Besides XML DTD, XSD (XML Schema) is another kind of XML schema which is also frequently used in XML applications. As mentioned before, in XSD, there is a kind of element node named key element, which is designed for the same purpose as ID attribute in DTD. Key element must contain a selector element and a field element in order. The selector element contains an XPath expression specifying the set of elements across which the values specified by field element must be unique, and correspondingly, each field element contains an XPath expression specifying the values that must be unique for the set of elements specified by the selector element. However, similar to ID attribute in DTD, the value of each field element is required to be unique for the corresponding object among the set of elements specified by the selector element. Although the selector element largely reduces the range in which the field element must be unique, it will still encounter the same problem as mentioned in the previous example under some schemas.

Next, we use the following rules to classify the rest of the internal nodes, including those object classes without ID attribute specified in their XML schemas.

Rule 2 *[**Aggregational Node** and **Explicit Relationship Type**] Given an XML schema tree, let* i *be an internal node, that has only one child node* c *and* c *is a repeatable node*[2]*; if the tag name of* i *is the plural form of the tag name of* c*, then* i *is an aggregational node, else it is an explicit relationship type.*

The first part of Rule2 (i.e., before the semicolon) is constructed from the properties of aggregational node. By comparing them to the properties of other ORA-semantics concepts in Table1, we can easily find out `A2` conflicts with `O2` and `C2`, `A3` conflicts with `R3`, which means we can distinguish aggregational node from object class, role name and composite attribute. However, these properties are not enough to distinguish between aggregational node and explicit relationship type, such as the internal node *Qualification* and *Has* in Fig.2, and we need related heuristics to help.

Recall that aggregational node is a structural node for aggregating its repeatable child node. Thus, based on the characteristic of aggregational node, it is reasonable to have the heuristic `H3` *"Its tag name should be the plural form*[3] *of the tag name of its child node"*, which is the second part of Rule2 (i.e., after the semicolon). For example, in Fig.2, the internal node *Qualifications* should be identified as an aggregational node, which aggregates its child node *Qualification*. On the other hand, an explicit relationship type may also have only one child node and the child node is repeatable, such as node *Has* in Fig.2, but the

[2] Repeatable node is the node which can occur multiple times in the corresponding XML data.

[3] Plural form also includes appending *'s'* to an abbreviation, such as *quals* being the plural form of *qual*, which is the abbreviation of *qualification*.

tag name of an explicit relationship type should not be the plural form of the tag name of its child node.

Rule 3 *[**Role Name** and **Explicit Relationship Type**] Given an XML schema tree, let* i *be an internal node, that has only one child node* c *and* c *is not a repeatable node, but an IDREF(S) attribute; if the tag name of* i *shares high linguistic similarity with the tag name of* c*, then* i *is a role name, else* i *is an explicit relationship type.*

The first part of Rule3 (i.e., before the semicolon) is constructed from the properties of role name. By comparing them to the properties of other ORA-semantics concepts in Table1, we can easily find out `R2` conflicts with `O2` and `C2`, `R3` conflicts with `A3`, `R4` conflicts with `C4`, which means we can distinguish role name from object class, aggregational node and composite attribute. However, these properties are not enough to distinguish between role name and explicit relationship type, such as the internal node *Landlord* and *RentBy* in Fig.5, and we need related heuristics to help.

Being the node with an alias of a certain object class, it is designed based on the fact that both the role name and the corresponding object class are representing the same concept. This makes it reasonable for the tag name of the role name and the tag name of the corresponding object class share high linguistic similarity. For example, in Fig.5, given an object class *Person* with its ID attribute, it may be referenced elsewhere by an IDREF attribute under a role name *Landlord*. In order to compare the similarity between two tag names, some research [11, 19, 17] has already been done on comparing linguistic similarity between different words, such as using the information from WordNet[4] for linguistic and semantic comparisons. However, for explicit relationship type, which may also share the properties of role name, it rarely shares high linguistic similarity with its child node, such as *RentBy* in Fig.5.

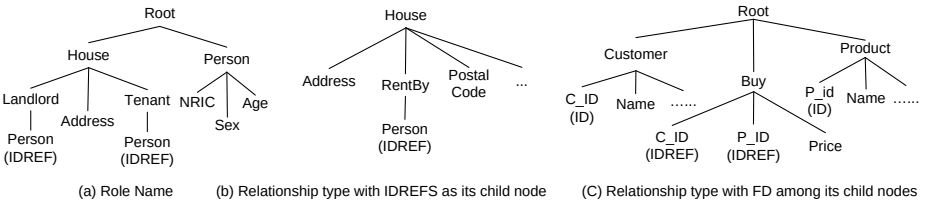


Fig. 5. Internal nodes with IDREF(S) in XML schema tree

Before we introduce the next rule, let us introduce a concept named exclusive descendant leaf node (EDLN):

[4] http://wordnet.princeton.edu/

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.

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. For example, in Fig.2, given an object class *Project*, its EDLNs include: *Location* and *Funding*.

Rule 4 *[**Explicit Relationship Type** and **Object Class**] Given an XML schema tree, let* i *be an internal node with more than one child nodes and there is at least one FD or MVD among its exclusive descendant leaf nodes; if all left hand side (LHS) nodes of those FDs or MVDs are IDREF(S) attributes or role names, then* i *is an explicit relationship type, else* i *is an object class.*

Rule 4 is constructed using properties of object class without any heuristic. By comparing them to the properties of other ORA-semantics concepts in Table1, we can easily find out `O2` conflicts with `A2` and `R2`, `O3` conflicts with `C3`, `O4` conflicts with `E3`, which means we can distinguish object class such as *Supplier*, *Part*, etc. in Fig.2, from role name, aggregational node, composite attribute and explicit relationship type.

Rule 4 utilizes the FDs/MVDs extracted from XML data. FDs/MVDs in XML data, defined in [7], are not exactly the same as FDs/MVDs in relational data. In XML data, FD/MVD is valid only under a header path, which is a path expression starting at the root and ending at a common ancestor node of all nodes involved in the FD/MVD. For example, in Fig. 2, given a header path */SPJ/Project/Supplier*, we have the FD $\{Supplier\#\} \rightarrow \{Name\}$, and the *Name* under *Employee* should not be considered. In our work, we adopt the algorithm in [14, 24, 25] to identify FDs/MVDs from XML data. For simplicity, in the rest of this paper, given a FD/MVD in XML schema tree, we use the path ending at the lowest common ancestor node of all nodes involved in the FD/MVD as its header path, and do not show it explicitly.

Rule 5 *[**Composite Attribute** and **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 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.*

Rule 5 is constructed using properties of composite attribute without any heuristic. By comparing them to the properties of other ORA-semantics concepts in Table1, we can easily find out `C2` conflicts with `A2` and `R2`, `C3` conflicts with `O3`, `C4` conflicts with `E2`, which means we can distinguish composite attribute such as *Qualification* in Fig.2, from object class, role name, aggregational node and explicit relationship type.

3.3 Leaf Node Classification

A. OID Discovery After processing all the internal nodes in the previous step, next step is to identify OID of each identified object class. As stated in Rule 1, OID can be explicitly specified in the XML schema using ID attribute, and this is also a sufficient condition for identifying OID. Thus in this section, we only consider the case that the single-attributed OID is not specified in the XML schema (e.g., *ISBN* of object class *Book* in Fig.2), or the OID contains multiple attributes (e.g., {*FirstName*, *LastName*} of object class *Child* in Fig.2).

In an XML schema tree, the attributes under an object class may be its object attributes or attributes of the relationship type which it participates in. Based on the definition of OID, only its object attributes can be functionally/multi-valued determined by its OID, while relationship attributes cannot. Our approach identify OID for each identified object class based on this heuristic. Before we explain how our approach identifies OIDs, 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 contain a subset of the exclusive descendant leaf nodes of o and the OIDs of some ancestor object classes of o. Super OID of o should functionally/multi-valued determines all exclusive descendant leaf nodes of o.*

In an XML schema tree, given an object class o, its exclusive descendant leaf nodes (EDLNs) may be object attributes of o, or relationship attributes of some relationship type which o participates in. Based on the definition of super OID and the properties of OID, object attribute and relationship attribute (i.e. `OID2`, `OA2`, `RA2` and `RA3` in Table.1), the super OID of o should functionally/multi-valued determine both object attributes and relationship attribute (if any) 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, the attribute set S formed by the OID of o and the OIDs of some of the ancestor object classes of o should functionally/multi-valued determine all EDLNs of o (including both object attributes and relationship attributes). In case of no relationship attribute being EDLN of o, no OID of ancestor object class of o will be included in S. Recall the definition of super OID, the attribute set S is also a super OID of o, which is also a superset of OID of o. Thus, this shows us a way to identify the OID of an object class. By excluding all OID(s) of the ancestor object class(es) of an input object class from its super OID, what left should be its OID.

Based on above analysis, we proposed a top-down approach (as the OID of an ancestor object class may be needed for identifying the OIDs of its descendant object classes) shown in Algorithm 2 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 $EDLN(o)$, also including OIDs of all its ancestor object classes. In $SupEDLN_o$, we identify the super OID of o, which is a minimal subset of $SupEDLN_o$ that functionally/multi-valued determines all

nodes in $EDLN(o)$. Also, we use the heuristic that if the OID of an ancestor object class o_j is in an super OID of o, then the OIDs of all object classes between o_j and o in their XML schema tree should also be included in that super OID. This is because the super OID of o containing the OID of object class o_j means there may be at least one relationship attribute in the exclusive descendant leaf node of o, and there is a corresponding relationship type between o_j and o, and all object classes between them should also participate in that relationship type. As there may be more than one minimal subset of $SupEDLN_o$ that satisfy the above condition, there may be more than one super OID for o. For each super OID, we can get an corresponding OID candidate for o, by excluding all OID(s) of ancestor object class(es) of o from the super OID. For the object class without ancestor object class, because it should not have any relationship attribute in its exclusive descendant leaf nodes, its super OID will be the same as its OID. In Algorithm 2, we use $AD(o, o')$ to represent that o is an ancestor node of o' in their XML schema tree.

Algorithm 2: Candidate OID Discovery

Input: Identified object classes $\mathbb{O}$, exclusive descendant leaf nodes for each identified object class
Output: 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); //SupEDLN_o is a super set of exclusive descendant nodes of 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 EID_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 ∄o_p ∈ 𝕆 such that e is the OID of o_p then
11                             e ∈ id_o;
12             return id_o as an OID candidate of o.
```

Example 2. In Fig.2, considering three identified object classes *Project*, *Supplier* and *Part* with their EDLNs, suppose we only get the following full FDs from the XML data: {*Project#*} → {*Location, Funding*}, {*Supplier#*} → {*Contact#, Name*}, {*Part#*} → {*Color*}, {*Supplier#, Part#*} → {*Price*} and {*Project#, Supplier#, Part#*} → {*Quantity*}. For object class *Project*, we can identify *Project#* as its OID by Rule 1, because it is specified as an ID attribute. For object class *Supplier*, as the single 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 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*.

As is shown in Example 2, the super OID may not be unique for some object classes, and our Algorithm 2 may return more than one OID candidate. Different OIDs show us different ORA-semantics. For example, for the object class *Part*, choosing {*Part#, Quantity, Price*} as its OID means it does not have any relationship attribute, while choosing {Part#, Quantity} means it has one relationship attribute *Price*, and choosing *Part#* means it has two relationship attributes *Quantity* and *Price*. Thus, in the following, we use some related heuristics summarized from our observation to help choosing the best OID from all OID candidates returned by Algorithm 2.

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 proposed based on our observation of structural and linguistic characteristics of OIDs designed in real world. Furthermore, we have following two observations: (1) the number of the object classes without relationship attribute is more than the number of object classes with relationship attributes; (2) for relationship attribute, the number of relationship attributes of binary relationship type is more than the number of relationship attributes of ternary relationship type, and so on. Thus, given an object class, the chance of having its super OID, being the same as its OID is higher than containing the OID of only one object class, and the case of containing OID of only one object class is higher than the case of containing OIDs of two or more object classes. Based on these and the heuristics mentioned in Observation 1, we collect 204 object classes with their OIDs being manually specified in their XML schemas and extract the statistic information mentioned above. Using such statistic information, we train a Bayesian Network to rank all OID candidates based on the heuristics for each object class, and choose the best one as its OID. More detail of the Bayesian Network is given in [4]. In Example 2, for the object class *Part*, although both {Project#, Supplier#, Part#}, {Supplier#, Part#, Quantity} and {Part#, Quantity, Price} can functionally determine all its EDLNs, {Project#, Supplier#, Part#} get the highest ranking using our Bayesian Network ranking model. Thus, {*Part#*} is identified as the OID of object class *Part* by our approach.

B. Object/Relationship Attribute Discovery For an explicit relationship type (discovered in Section 3.2), we identify all its EDLNs as its relationship attributes base on its property (i.e. `E4` and `RA4` in Table 1). For implicit relationship type, its relationship attributes should appear as an EDLN of the lowest object class which involves in the relationship type (`RA4` in Table 1), together

with the object attributes of that object class. Thus, based on these properties, we propose Rule 6 to distinguish object attributes and relationship attributes among the EDLNs of each identified object class. We use the properties that object attribute should be functionally/multi-valued determined by OID of the object class it belongs to, while relationship attribute should not, to differentiate them.

Rule 6 *[**Object Attribute** and **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 object 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.

3.4 Implicit Relationship Type Discovery

Recall that explicit relationship type can be identified by Rule 2, 3, 4, 5 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 [9]. In the following, we will introduce all these four categories one by one.

A. 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 3, to identify the implicit relationship type with its degree, and all involved object classes.

Algorithm 3 uses a bottom-up approach to identify the implicit relationship type with its degree, and all the involved object classes. We use $r(\mathbb{C})$ to represent the implicit relationship type among object classes in $\mathbb{C}$, and degree of the implicit relationship type is $|\mathbb{C}|$, which is the number of object classes in $\mathbb{C}$. For each relationship attribute ra, Algorithm 3 creates a set $SemID_{ra}$ containing the OID of its lowest ancestor object class id_{oi}. We use a bottom-up approach so that in each iteration we add the OID of the lowest ancestor object class, which has not been considered along the path from ra to the root, into $SemID_{ra}$. Whenever $SemID_{ra}$ functionally/multi-valued determines ra, we identify the

implicit relationship type $r(\mathbb{C})$, to which ra belongs, and return the involved object classes as those object classes whose OIDs are in $SemID_{ra}$ and the degree of the implicit relationship type as the number of involved object classes.

Algorithm 3:
Implicit Relationship Type with Relationship Attribute

Input: Identified relationship attribute $\mathbb{A}$; Identified object classes $\mathbb{O}$, with their 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 $\mathbb{A}$

foreach *identified relationship attribute* $ra \in \mathbb{A}$ **do**
  o_i = the lowest ancestor object class of ra.
  $\mathbb{C} = \{o_i\}$;
  $SemID_{ra} = id_{oi}$; //id_{oi} `is the OID of object class` o_i;
  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**
    $SemID_{ra} = SemID_{ra} \cup id_{oj}$; //$id_{oj}$ `is the OID of object class` o_j;
    $\mathbb{C} = \mathbb{C} \cup \{o_j\}$;
    if $SemID_{ra} \rightarrow ra$ or $SemID_{ra} \twoheadrightarrow ra$; **then break**;
  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;

Example 4. In Fig.2, given a relationship attribute *Price*, object classes *Project*, *Supplier* and *Part*, as well as their OIDs. By Algorithm 3, we find out that {*Supplier#, Part#*} functionally determines *Price*. Then we know there is an implicit binary relationship type between object classes *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 find out {*Project#, Supplier#, Part#*} functionally determines *Quantity*. Then we know there is an implicit ternary relationship type among object classes *Project*, *Supplier* and *Part*, with relationship attribute *Quantity*.

B. 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). Based on this hint, if an object class has a child node specified as an IDREF(S) attribute, we identify that there is an implicit relationship type between the object class and the object class(es) the IDREF(S) attribute referring to. For some XML schema language (e.g., DTD), we do not know which object class(es) the IDREF(S) attribute references. 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 this kind of implicit relationship type: (1) [H3] The tag name of the IDREF(S) attribute may share high linguistic similarity with the tag name of the object class(es) which it references, or the corresponding OID(s). We can

identify them by research work [11] 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, then 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.

C. Implicit relationship type with no relationship attribute or IDREF(S) The implicit relationship type with no relationship attribute or IDREF(S) will be discovered by Algorithm 4. We start from the object class o located at the lowest level in its XML schema tree, and discover the implicit relationship type involving o and all other involved object classes, which are ancestors of o.

Algorithm 4: Implicit Relationship Type without Relationship Attribute

Input: Identified object classes $\mathbb{O}$, with their OIDs; XML schema tree; XML data
Output: Relationship type $r(\mathbb{C})$ with its involved object classes $\mathbb{C}$ and degree $|\mathbb{C}|$

```
1  foreach identified object class o ∈ 𝕆 do
2      if ∃r which is an implicit relationship type involving o then
3          continue;
4      else
5          ℂ = {o};
6          S = Null;
7          foreach identified object class o_j ∈ 𝕆, along the path from o to the root node in
           its XML schema tree in bottom-up order do
8              S = S ∪ id_oj;                    //id_oj is the OID of object class o_j;
9              ℂ = ℂ ∪ {o_j};
10             if S ↠ id_o; then break;          //id_o is the OID of object class o;
11         return implicit relationship type r(ℂ) with object classes in ℂ as its involved object
           classes and |ℂ| as its degree;
```

Given an object class o with its OID id_o, without any implicit relationship type being identified, similar to Algorithm 3, we recursively add the OID of the lowest ancestor object class of o that has not been considered into a set S and check whether S multi-valued determines id_o. Whenever S multi-valued determines id_o, we identify an implicit relationship type $r(\mathbb{C})$, and return its involved object classes and degree. Object classes in $\mathbb{C}$ are those object classes whose OIDs are in S. The rationale of Algorithm 4 is that without relationship

attribute, given an object class o', if each of its object instance always corresponds to the same set of object instances of its child object class o, there is an implicit binary relationship type between o' and o. Otherwise, this implicit relationship type must also involve some other ancestor object classes of o. And this corresponding relationship can be captured using multi-valued dependency among their OIDs.

Example 5. In Fig. 2, supposing *Project*, *Employee* and *Paper* are identified as object classes, if the same object instance of *Employee* under different *Project* instance always has the same set of *Paper* instances under it in its XML data, then we know there is an implicit binary relationship between object classes *Employee* and *Paper*, regardless of its parent object class *Project*. If under different *Project* instances, the same *Employee* instance has different set of *Paper* instances, as there is no more ancestor object class of *Project*, then we know there is an implicit ternary relationship type among object classes *Project*, *Employee* and *Paper*.

D. Identifier Dependency (IDD) Relationship Type There is a special kind of relationship type called identifier dependency (IDD) relationship type, which means there is a weak object class that can only be identified together with its lowest ancestor object class. For example, in Fig. 2, the relationship type between object classes *Book* and *Chapter* should be an IDD relationship type, which makes *Chapter* a weak object class. Given an object instance of object class *Chapter*, there is no way we can uniquely identify it without knowing to which book it belongs. The properties of weak object class are given in Table 1. For a weak object class, our rules in Section 3.2 may classify it as composite attribute, because weak object class shares all four properties of composite attribute. In order to discovery weak object class together with IDD relationship type, we propose the following rule based on a property of weak object class, which is not shared by other ORA-semantics concepts:

Rule 7 ***[IDD Relationship Type]*** *In an XML schema tree, given an internal node* i *and OID* id_o *of its lowest ancestor object class* o*, if* $\exists$ e*,* e *is an exclusive descendant leaf node of* i *such that (1)* e *cannot functionally/multi-valued determine any other exclusive descendant leaf node of* i*; (2)* e *together with* id_o *can functionally/ multi-valued determine all exclusive descendant leaf nodes of* i*, then* i *is a weak object class and there is an IDD relationship type between object class* o *and weak object class* i.

The intuitive meaning of Rule 7 is the same as the `W5` in Table 1. For weak object class, there must be an exclusive descendant node of it which cannot functionally/multi-valued determine any other exclusive descendant leaf nodes of the weak object class, but can functionally/multi-valued determine them together with the OID of its lowest ancestor object class, which is also the object class having IDD relationship with it. For example, in Fig.2, *C#* of *Chapter* does not functionally/multi-valued determine any other exclusive descendant

leaf nodes of *Chapter*, but *C#* together with OID of its parent object class, {*C#, ISBN*} can functionally determine all of them. Thus *Chapter* is a weak object class, having an IDD relationship with its parent object class *Book*.

4 Experiment

We experimentally 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., the auction data, the university courses data, the mondial data[5] and the XMark data[6], etc. Considering that most practical databases are still in relational model and much XML data are actually translated from relational data and published/exchanged in XML format, we also collected 5 relational data sets such as IMDB[7] and TPC-H[8], etc. We asked 8 PhD students doing research in XML to reasonably design XML schemas based on those 5 relational data sets. We apply our discovery algorithms to each of the above schemas.

Among the XML schemas we collected for experiments, some of them come with complex structure such as XMark, which has 48 internal nodes and a maximal depth of 8 in its XML schema tree. To evaluate the accuracy of our approach, we measure precision, recall and F-measure[9] against a gold standard provided by 8 evaluators. The evaluators are all PhD students doing research in XML. In order to handle the case that different students may have different understandings of what a node in XML schema should be identified as, we adopt the probability theory and use the probability of a node being identified as each kind of ORA-semantic as its ground truth rather than using only one ground truth for each node being identified. For example, the ground truth of an internal may be it has 75% to be an object class and 25% to be a composite attribute because among 8 PhD students, 6 of them identify it as an object class and 2 of them identify it as a composite attribute. Thus, both object class and composite attribute will be consider as the ground truth with different probability by considering their expected values in our calculation of precision and recall.

4.1 Accuracy of Internal Node Classification

There are totally 505 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 for explicit relationship type and low recall for aggregational

[5] http://www.cs.washington.edu/research/xmldatasets/www/repository.html
[6] http://www.xml-benchmark.org/
[7] http://www.imdb.com/interfaces
[8] http://www.tpc.org/tpch/
[9] F-measure = 2 * precision * recall/(precision + recall)

node are because the related heuristics used are not as accurate as the properties used in our rules. In Table 3, we show the 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 306 object classes among all 505 internal nodes, which take up around 60% of the internal nodes. Other ORA-semantics concepts only take up a small percentage of the internal nodes, especially for role name, which only takes up less than 2% of the internal nodes.

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

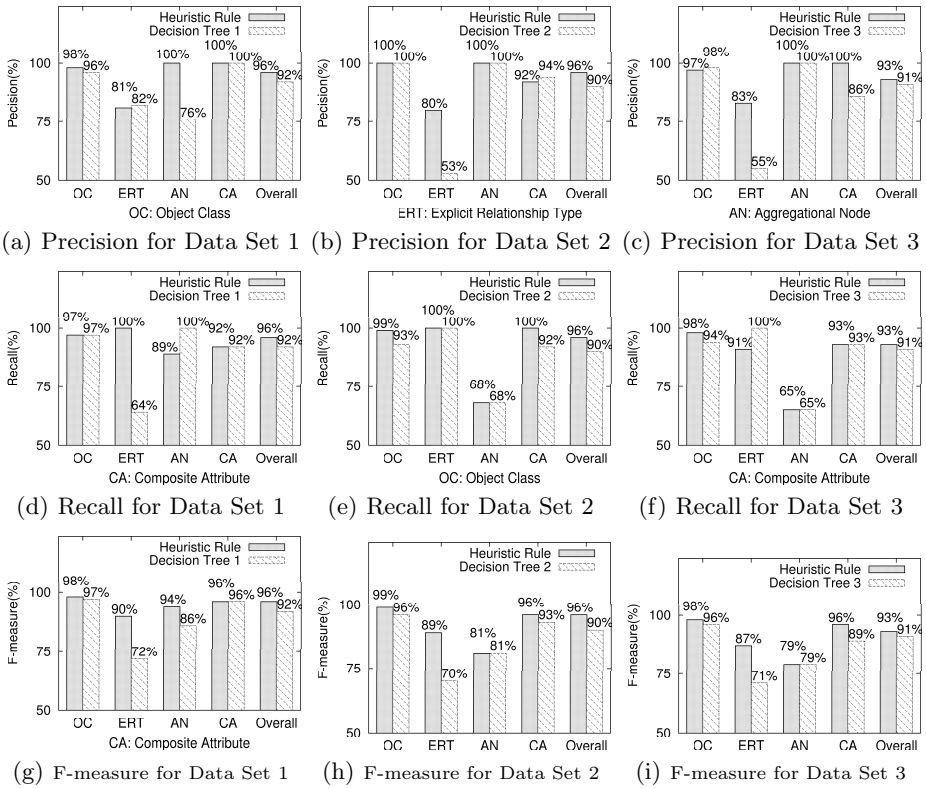
	Object Class	Role Name	Explicit Relationship Type	Aggregational Node	Composite Attribute	Overall
Precision	98.1%	87.5%	73.8%	100.0%	96.6%	94.9%
Recall	97.7%	100%	98.0%	74.6%	95.5 %	94.9%
F-measure	97.9%	93.8%	85.9%	87.3%	96.0%	94.9%

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

	Object Class	Role Name	Explicit Relationship Type	Aggregational Node	Composite Attribute	Total
Num. of nodes	306	7	49	55	88	505
Percentage	60.6%	1.4%	9.7%	10.9%	17.4%	100%

We also used a pure data mining approach to classify the internal nodes for comparison purpose. We use the properties and heuristics used in our rules as features to train a classification model to classify the internal nodes into object class, explicit relationship type, aggregational node or composite attribute (role name is omitted because of the small number of instance of it, which makes the generated decision tree work extremely bad.). In order to avoid bias, we equally divided all internal nodes into 3 groups, and use 2 of them for training and the rest internal nodes for testing. This will return us 3 classification models (may or may not be the same). 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 [12] 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. 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. The data mining approach may be able to work better given a much larger training data set. However, the input data we need is the XML schema rather than XML data, and it cannot be as large as the XML data.



(a) Precision for Data Set 1 (b) Precision for Data Set 2 (c) Precision for Data Set 3

(d) Recall for Data Set 1 (e) Recall for Data Set 2 (f) Recall for Data Set 3

(g) F-measure for Data Set 1 (h) F-measure for Data Set 2 (i) F-measure for Data Set 3

Fig. 6. Comparison of two approaches for internal node identification (OC: Object Class, ERT: Explicit Relationship Type; AN: Aggregational node; CA: Composite Attribute)

4.2 Accuracy of Leaf Node Classification

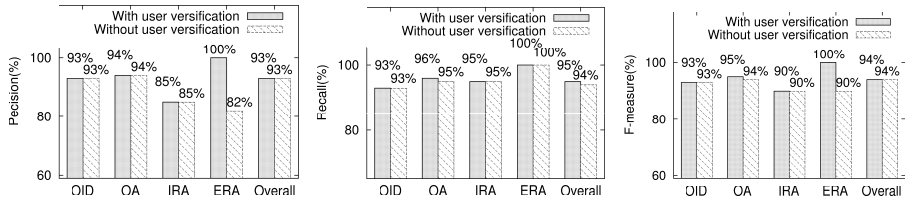
As mentioned in Section 3.3, our approach may return more than one OID candidates for each identified object class, thus we build a Bayesian Network to rank all its OID candidates, and choose the highest ranked candidate as its OID. There are 306 object classes with their OIDs in our experimental data. 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. In Table 4, based on the features mentioned in Observation 1 in Section 3.3, we show the top 10 probabilities of being OID with different features.

Table 4. Features of top 10 ranking of being OID

Rank	Number of Object Attribute	Number of Involved Level	Contains First Child Node or Not	Contains Keyword Substring or Not	Numerical Value with Pattern	Probability of Being OID
1	1	1	T	T	T	0.977
2	1	1	T	T	F	0.977
3	1	1	T	F	T	0.932
4	1	1	T	F	F	0.931
5	1	2	T	T	T	0.876
6	1	2	T	T	F	0.876
7	1	1	F	T	T	0.758
8	1	1	F	T	F	0.758
9	2	1	T	T	T	0.692
10	1	2	T	F	T	0.689

In our step-by-step approach, some 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. In order to show the accuracy of each step separately, 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. Furthermore, recall that we discover explicit relationship type in the step of internal node classification, and all exclusive descendant nodes of this explicit relationship type will be identified as its relationship attributes.



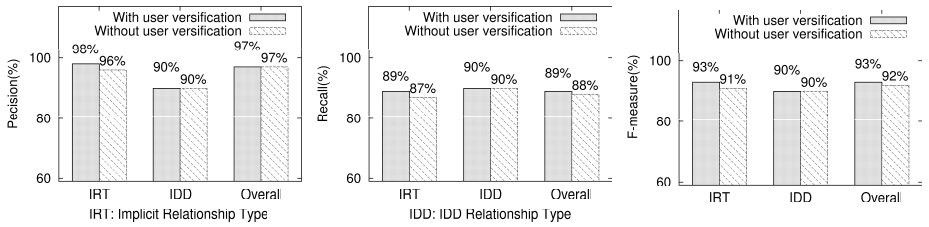
(a) Precision (b) Recall (c) F-measure

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

Fig.7 shows that our approach for leaf node classification can get above 90% of precision, recall and F-measure. Even without the user verification, the precision and recall of our approach only drop slightly, because our approach for discovering object class also gets high precision and recall. The low precision of implicit relationship attribute is because its identification is heavily depended on the FDs and MVDs among the corresponding XML data, which may not be large enough to return all the correct FDs and MVDs.

4.3 Accuracy of Implicit Relationship Type Discovery

Last, we conduct our experiment on our approach for implicit relationship type discovery. Recall that all explicit relationship types have been discovered using our Rule 2, 3, 4, 5 in Section 3.2, and its accuracy has been shown in Fig. 6. Similar to leaf node classification, we also conduct two groups of experiment to evaluate the accuracy of our approach, one with user verification, which means all object classes with their OIDs, object attributes and relationship attributes have been correctly labeled in XML schema trees, and the other based on the results of our previous two steps without user verification.



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

Fig. 8. Precision, recall and F-measure of implicit relationship type discovery

Fig. 8 shows the precision, recall and F-measure for identifying regular implicit relationship types and IDD relationship type in XML schema trees.

Furthermore, we show the overall precision, recall and F-measure of our approach discovering all ORA-semantics in Fig. 9.

5 Related Work

To the best of our knowledge, few researchers have frontally addressed the problem of automatically discovering the implicit semantics embedded in XML schema and XML data. The authors of [6] try to find semantic annotations for XML schemas using ontologies. The intention is to discover the meaning of each XML schema node. In contract, our work focuses on a comprehensive set of semantic concepts grouped under the ORA-semantics. Previous works related

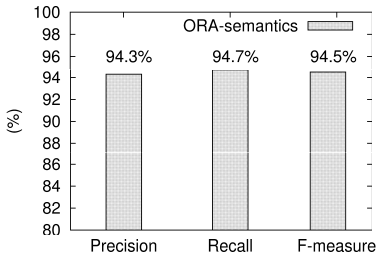


Fig. 9. Precision, recall and F-measure of discovering all ORA-semantics

to the ORA-semantics mainly focused on identifying objects in XML data. In [13], the authors introduce an approach to identify objects with approximate joins. In [18], XML objects are compared using string comparison functions to detect duplicate objects. The set of semantic concepts targeted in these works are limited. The works above are also strict to XML data and do not exploit the availability of XML schemas.

In [3], in the context of view design, all internal nodes in an XML schema tree are considered as object classes. This is not correct since internal nodes can be role names, relationship types, composite attributes or aggregational nodes as well. In the context of keyword search, XSeek [10] also infers semantics from XML schemas to identify return nodes. This work infers semantics of objects by using the repeatable node. In web application, the identification of semantic concepts is usually done by the manual effort from users, e.g.[15]. For the sake of conciseness, we do not continue this catalogue raisonné of works that address elements of the question but do not provide a global and satisfactory solution. The originality and significance of our work reside in that it addresses and solves the problem holistically and independently from any specific type of target processing.

6 Conclusion and Future Work

The availability of a conceptual schema or of elements of semantics constitute invaluable leverage for improving the effectiveness, and sometimes the efficiency, of many tasks including query processing, keyword search and schema and data integration. Yet XML data and XML schema are instances and schemas of a logical model that fails to explicitly represent the intended semantics.

In this paper we have presented and evaluated a set of techniques for the discovery of semantics implicitly embedded into XML data and schemas. Our target model is of the family of the Object-Relationship-Attribute for Semi-Structured data (ORA-SS) model. We therefore refer to the set of instances of these inferred semantic concepts as the ORA-semantics of the application. The techniques we devised and presented leverage a property-based decision tree in order to identify object classes, role name, explicit relationship types, aggregational nodes and composite attributes from internal nodes of an XML schema tree. The techniques use properties and statistic information to identify OID for

each object class and to distinguish object attributes and relationship attributes from leaf nodes of an XML schema tree. Finally, our approach is also able to discover the implicit relationship type among object classes. We have empirically evaluated the effectiveness of our approach using several established data sets and schemas. The experiments that we have conducted demonstrate that our approach can achieve almost 95% overall precision, recall and F-measure.

In our future work, we will use this discovered semantics information to facilitate different XML-related applications. Semantic-based XML query processing will be proposed to improve the efficiency of answer XML queries. We will also propose object-oriented XML keyword search and object-oriented XML schema/data integration. Besides these, we will also try to discovery this semantics information by using not only structure information but also ontology information to further increase our accuracy.

References

1. D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *SIGMOD Conference*, pages 906–908, 2005.
2. P. P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
3. Y. B. Chen, T. W. Ling, and M.-L. Lee. Designing valid XML views. In *ER*, pages 463–478, 2002.
4. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.
5. J. Hegewald, F. Naumann, and M. Weis. Xstruct: Efficient schema extraction from multiple and large XML documents. In *ICDE Workshops*, page 81, 2006.
6. J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic annotations for wsdl and XML schema. *IEEE Internet Computing*, 11(6):60–67, 2007.
7. M.-L. Lee, T. W. Ling, and W. L. Low. Designing functional dependencies for XML. In *EDBT*, pages 124–141, 2002.
8. M.-L. Lee, L. H. Yang, W. Hsu, and X. Yang. XClust: clustering XML schemas for effective integration. In *CIKM*, pages 292–299, 2002.
9. T. W. Ling, M. L. Lee, and G. Dobbie. Semistructured database design. Springer, 2005.
10. Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD Conference*, pages 329–340, 2007.
11. S. Mizuta and K. Hanya. Specifications of word set in linguistic approach for similarity estimation. In *BICoB*, pages 25–29, 2010.
12. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
13. L. Ribeiro and T. Härder. Entity identification in XML documents. In *Grundlagen von Datenbanken*, pages 130–134, 2006.

14. H. Shi, T. Amagasa, and H. Kitagawa. Fast detection of functional dependencies in XML data. In *XSym*, pages 113–127, 2010.
15. A. Spink. A user-centered approach to evaluating human interaction with web search engines: an exploratory study. *Inf. Process. Manage.*, 38(3):401–426, 2002.
16. C. Sun, C. Y. Chan, and A. K. Goenka. Multiway SLCA-based keyword search in XML data. In *WWW*, pages 1043–1052, 2007.
17. Y. Tang and J. Zheng. Linguistic modelling based on semantic similarity relation among linguistic labels. *Fuzzy Sets and Systems*, 157(12):1662–1673, 2006.
18. M. Weis and F. Naumann. Detecting duplicate objects in XML documents. In *IQIS*, pages 10–19, 2004.
19. D. Wu and J. M. Mendel. A vector similarity measure for linguistic approximation: Interval type-2 and type-1 fuzzy sets. *Inf. Sci.*, 178(2):381–402, 2008.
20. H. Wu and Z. Bao. Object-oriented xml keyword search. In *ER*, pages 402–410, 2011.
21. H. Wu, T. W. Ling, and B. Chen. VERT: A semantic approach for content search and content extraction in XML query processing. In *ER*, pages 534–549, 2007.
22. Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD Conference*, pages 537–538, 2005.
23. Y. Xu and Y. Papakonstantinou. Efficient lca based keyword search in XML data. In *EDBT*, pages 535–546, 2008.
24. C. Yu and H. V. Jagadish. Efficient discovery of XML data redundancies. In *VLDB*, pages 103–114, 2006.
25. C. Yu and H. V. Jagadish. XML schema refinement through redundancy detection and normalization. *VLDB J.*, 17(2):203–223, 2008.