Discover Semantics from XML

Li Luochen (HT090442R)

Supervised by Professor *Ling Tok Wang*

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2012

Discover Semantics from XML

Li Luochen

luochen@comp.nus.edu.sg

Abstract

For years, the ER model and the semantic concepts carried by it such as entity type, relationship type, attribute and etc., have constituted invaluable leverage for improving the effectiveness and efficiency of different database applications, including query processing, keyword search as well as schema integration and data integration. For XML database, a similar semantics-rich data model ORA-SS has also been proposed to capture the corresponding semantic concepts, including the object class, relationship type, object/relationship attribute, etc. Given these semantic concepts in XML database, semantics-based approaches for query processing and keyword search are also proposed to achieve higher efficiency or accuracy. However, these semantic concepts which are named as ORA-semantics in this thesis are not always available for inputs. Therefore, we need an automatic semantics discovery approach for the XML database.

Currently, only a few studies realize the importance of discovering the ORA-semantics (e.g., object class, object identifier, relationship type, etc.) from XML database. Even those studies on discovering semantic information from XML database only focus on the object class in schema level and the object instance in data level. However, the existing approaches for semantics discovery miss many other important semantic concepts such as relationship type, object attribute, relationship attribute, etc. Without identifying these semantic concepts, XML keyword search approaches may return meaningless results because of not knowing the relationships between object instances; XML schema integration approaches may wrongly integrate object classes which have different relationship types with the same object class together; etc.

In this thesis, we present a novel rule-based approach to automatically discover the ORA-semantics with XML schema and XML data as inputs. The ORA-semantics contains following semantic concepts: object class, OID, object attribute, relationship type, relationship attribute, dependent object class, IDD relationship type, role name, composite attribute and aggregational node, which aggregates multiple nodes with identical or similar meaning. For each above semantic concept, our rule-based approach has the corresponding classification rules or algorithm to identify it from the XML schema tree with the information provided by the XML data.

There are mainly three steps in our rule-based approach: (1) internal node classification, which classifies the internal nodes of an XML schema tree into object class, role name, explicit relationship type, aggregational node or composite attribute; (2) leaf node classification, which classifies the leaf nodes of an XML schema tree into OID, object attribute or relationship attribute; (3) implicit relationship type discovery, which identifies the implicit relationship type which is not explicitly shown as a node in the XML schema tree. Furthermore, the dependent object class and IDD relationship type can also be discovered in this step.

As pre-processing, we identify the properties of each kind of semantic concept in the ORA-semantics. These properties are the necessary conditions, describing hierarchical structures of how it is designed in XML schema tree and constraints imposed on it by XML data. We also identify a sufficient (but not necessary) condition of how object classes and their OIDs are designed in XML schema tree, and this sufficient condition can be directly used as a classification rule for identifying object class and its OID.

In order to identify different semantic concepts, our rule-based approach combines the features of each semantic concept, which can be used to differentiate it from other semantic concepts, and form classification rules for identifying each particular semantic concept. For those semantic concepts which cannot be distinguished from each other only by their properties, we also proposed the related heuristics to help differentiate them. These heuristics are proposed based on our observations of the hierarchical structures and linguistic features about how designers usually design different semantic concepts in XML database. Although these heuristics are neither necessary conditions nor sufficient conditions of the corresponding semantic concept, and they do not guarantee to be 100% correct, we conduct experiments to show they can help to increase the accuracy of identifying the corresponding semantic concepts.

After all, we empirically and comparatively evaluate the effectiveness of our rule-based approach. Extensive experiments have been conducted to show the ORA-semantics discovered by our approach has high accuracy especially for object class (i.e., above 95% of precision and recall) with functional/multi-valued dependency being verified by users.

Furthermore, a demonstration system based on our rule-based approach has been built to discover the ORA-semantics given an XML data with or without its corresponding XML schema.

ACKNOWLEDGEMENT

During these years of study in School of Computing, National University of Singapore, I have met many challenges in both research and life. I am grateful that I also met many people who have helped me and supported me on my journey for the master degree. Without their help and support, this thesis would not be possible.

First of all, I would like to express my sincerest gratitude to my supervisor, Professor Ling Tok Wang. During my journey of study as his student, he has taught me not only a wealth of knowledge in my research direction, but also invaluable knowledge about how to do research. As my supervisor, his insights, knowledge and experience in database domain as well as his patient guidance in our discussion all benefit me a lot, and even for my whole life.

I would like to thank Prof. Stephane Bressan and Prof. Lee Mong Li for being the examiners of my graduate research paper. They have shown me many useful comments for my research work, and these comments help me to revise my research work and make my thesis better.

I would also like to show my gratitude to my friends in my lab, especially Wu Huayu, Bao Zhifeng, Zeng Yong and Thuy Ngoc Le. They have helped me a lot in my research, and I have come up with many good ideas during the discussion with them.

Last but not least, I would like to thank my family, especially my wife Ruby. It is their continuous support and love give me strength to overcome every difficulty I met during my journey in Singapore.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background on XML database

1.1.1 XML

XML (eXtensible Markup Language) [12] has become a frequently used standard for information exchange and data storage in the real world. 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 terms of efficiency, and generally hard to be understood 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.

In Fig. 1.1, we show a portion of an XML Document example for storing data about a department and the information of the courses, students and professors in this department. From the XML document we can see the data is stored between a starting tag and an ending tag, and these tags are not only organized in a hierarchical structure, but also explicitly store meaningful information in their tag names. Because of the hierarchical structure of the XML document, it can also be represented as an order tree. Fig. 1.2 shows the tree structure of the corresponding XML document in Fig. 1.1. There are two kinds of nodes in any tree structure, internal node (i.e., the node with at least one child node) and leaf node (the node without any child node). In the XML data tree (we call the tree structure of XML document as XML data tree), the internal nodes represent the elements and attributes in the corresponding XML document, and the leaf nodes are the data values of these elements and attributes. Furthermore, the edges in the XML data tree can represent an element-subelement relationship, an element-attribute relationship, an element-data value relationship or an attribute-data value relationship.

1.1.2 XML schema

XML schema is designed to capture the schema for an XML document. There are two frequently used XML schemas for XML documents, Document Type Definition (DTD) [13] and XML Schema (XSD) [61]. In Fig. 1.3, we show the DTD and XSD of the corresponding XML document in Fig. 1.1. Both DTD and XSD can also be represented as tree structures. In Fig. 1.4, we show the tree structure of the corresponding XML schema in Fig. 1.3, and we call this tree structure as XML schema tree. XML schema tree also contains two kinds of nodes, internal nodes

```
<Department>
  <D_Name>NUS</D_Name>
  <Address>Kent Ridge</Address>
  <Course>
    <Code>CS5201</Code>
    <C_Name>AI</C_Name>
    <Lecturer>John</Lecturer>
    <Student>
      <Matric#>HT001</Matric#>
      <S_Name>
        <S_FirstName>Bill</S_FirstName>
        <S_LastName>Smith</S_LastName>
      </S_Name>
      <Grade>A</Grade>
    </Student>
    <Student>
      <Matric#>HT002</Matric#>
      <S_Name>
        <S_FirstName>Bob</S_FirstName>
        <S_LastName>Hanks</S_LastName>
      </S_Name>
      <Grade>C</Grade>
    </Student>
  <Course>
  <Course>
    <Code>CS5208</Code>
    <C_Name>Database</C_Name>
    <Lecturer>Tan</Lecturer>
    <Student>
      <Matric#>HT001</Matric#>
      <S_Name>
        <S_FirstName>Bill</S_FirstName>
        <S_LastName>Smith</S_LastName>
      </S_Name>
      <Grade>B</Grade>
    </Student>
  <Course>
  <Professor>
   …
  </Professor>
</Department>
```

Figure 1.1: A portion an XML document

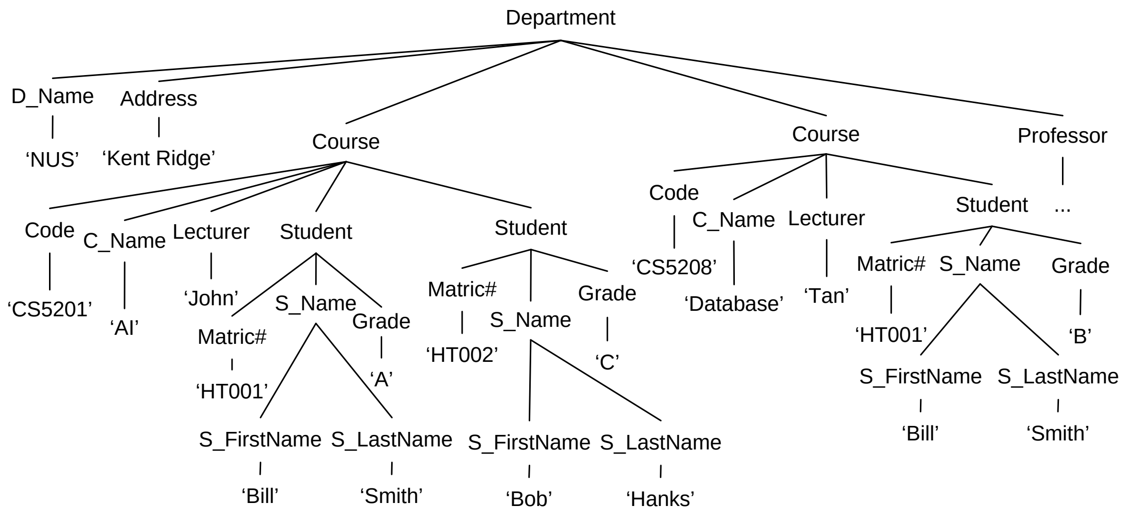


Figure 1.2: The tree structure of the XML document in Fig. 1.1

and leaf nodes. Internal nodes represent elements of an XML schema, while leaf nodes can be elements or attributes of an XML schema. Attribute nodes in an XML schema are represented as leaf nodes starting with a symbol '@' in the XML schema tree to be distinguished from elements. Furthermore, edges in an XML schema tree can represent element-subelement relationships or element-attribute relationships. For any two nodes connected by a single edge, we call them as parent and child of each other, and there is a parent-child (PC) relationship between them. Similarly, for any two nodes connected by more than one edge in the same path, we call them as ancestor and descendant of each other, and there is an ancestor-descendant (AD) relationship between them.

There are many useful semantic concepts, such as *object class*, *OID* (*object identifier*)[1], *relationship type*, *object attribute*, *relationship attribute* and *Composite Attribute*, cannot be captured and represented by DTD and XSD. In the following, we use the example in Fig. 1.4 to illustrate the semantic concepts that DTD and XSD cannot capture and represent.

[1]DTD and XSD can only capture and represent object identifier of a portion of object classes.

```
<!ELEMENT root (Department*)>

<!ELEMENT Department
(Address,Course*,Professor*)>
<!ATTLIST Department D_Name ID #REQUIRED>

<!ELEMENT Address (#PCDATA)>

<!ELEMENT Course
(Code,C_Name,Lecturer,Student*)>

<!ELEMENT Code (#PCDATA)>
<!ELEMENT C_Name (#PCDATA)>
<!ELEMENT Lecturer (#PCDATA)>

<!ELEMENT Student (Matric#,S_Name,Grade)>
<!ELEMENT S_Name (S_FirstName,S_LastName)>

<!ELEMENT Matric# (#PCDATA)>
<!ELEMENT S_FirstName (#PCDATA)>
<!ELEMENT S_LastName (#PCDATA)>
<!ELEMENT Grade (#PCDATA)>

<!ELEMENT Professor
(P_Name,Book*,Office,Email*)>

<!ELEMENT P_Name (#PCDATA)>
<!ELEMENT Office (#PCDATA)>
<!ELEMENT Email (#PCDATA)>

<!ELEMENT Book (ISBN, Title, BorrowDate)>

<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT BorrowDate (#PCDATA)>
```

(A) DTD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Department" minOccurs="1" maxOccurs="1">
 <xs:key name="DepartmentKey">
   <xs:selector xpath="/Department"/>
   <xs:field xpath="/Department/@D_Name"/>
  </xs:key>
  <xs:complexType>
  <xs:sequence>
   <xs:element name="Address" type="xs:string"/>
   <xs:element name="Course" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Code" type="xs:positiveInteger"/>
      <xs:element name="C_Name" type="xs:string"/>
      <xs:element name="Lecturer" type="xs:string"/>
      <xs:element name="Student" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Matric#" type="xs:positiveInteger"/>
         <xs:element name="S_Name" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="S_FirstName" type="xs:string"/>
            <xs:element name="S_LasttName" type="xs:string"/>
           </xs:sequence>
          </xs:complexType>
         <xs:element name="Grade" type="xs:string"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="Professor" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="P_Name" type="xs:string"/>
      <xs:element name="Office" type="xs:string"/>
      <xs:element name="Email" maxOccurs="unbounded" type="xs:positiveInteger"/>
      <xs:element name="Book">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="ISBN" type="xs:string"/>
          <xs:element name="Title" type="xs:string"/>
         <xs:element name="BorrowDate"type="xs:string"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="D_Name" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
</xs:schema>
```

(B) XSD

Figure 1.3: DTD and XSD of the XML document in Fig. 1.1

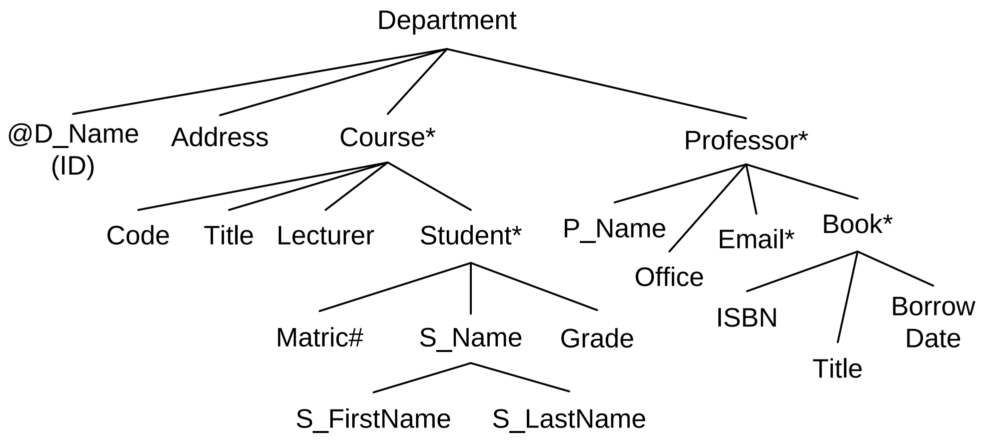


Figure 1.4: The tree structure of the XML schema in Fig. 1.3

For consistency, in the rest of this thesis, object class and relationship type are used in schema level, and object/object instance and relationship are used in data level with respect to the semantic concepts.

Object Class

Both DTD and XSD cannot capture and represent the semantic concept object class, because there is no way they can distinguish between the object classes and composite attributes. In DTD and XSD, both of them are represented as elements having more than one component as their child nodes in their corresponding XML schema trees.

For example, in Fig. 1.4 we cannot distinguish the internal node *Student* with the internal node *S_Name* which are object class and composite attribute respectively.

OID

In DTD, we can specify the identifier for an element by designing an ID attribute as its child node. If the element represents an object class, then the ID attribute should be the OID[2] of the object class. However, the value of an ID attribute is required to be unique within the XML document, and this makes it impossible for some object classes to have ID attributes being specified in their XML schemas. Furthermore, as ID attribute is a single attribute, which makes it impossible for an object class to have combined attributes as its OID.

For example, as shown in Fig. 1.3 (A), because the element *Department* has an ID attribute in its DTD as its child node, we know *Name* is the OID of the object class *Department*. However, some other element such as *Student*, cannot have ID attribute as its child node, because the same student can take more than one course, and thus, there will be more than one student element with the same value of *Matric#* as shown in Fig. 2.1. Furthermore, composite attribute *S_Name* cannot be designed as the ID attribute of *Student* neither, because ID attribute can only be a single attribute.

In XSD, there is a kind of element node named key element, which is designed for the same purpose as the ID attribute in DTD. Key element contains a selector element and a field element. The selector element contains an XPath expression specifying the set of elements across which the values specified by the XPath expression of the field element must be unique. The field element of a key element is similar to the ID attribute in DTD. Although the selector element limits the scope in which the field element must be unique, it is still impossible to capture of the OID of some object classes because of the uniqueness constraint.

For example, in Fig. 1.3 (B), there is a key element named *DepartmentKey*

[2]Note OID is a semantic concept meaning the identifier of an object class, which is different from the ID attribute which is one of the attribute types in DTD.

with its selector element as `xpath='/Department'`, and field element as `xpath='/Department/@D_Name'`, which means the value of attribute *D_Name* must be unique among all instances of *Department*. However, for element *Student*, we cannot specify a key element for it even with the selector element as `xpath='/Department/Course/Student'` and the field element as `xpath='/Department/Course/Student/@Matric#'`, because the scenario that the same student taking more than one course will violate the uniqueness constraint of attribute *Matric#*.

Relationship Type

The hierarchical structure of an XML schema can be captured and represented in a tree structure by both DTD and XSD, such as the XML schema tree in Fig. 1.4. Although the tree structure captures the parent-child (PC) relationships, without identifying the object classes, there is no way it can identify whether a PC relationship represent a relationship type between two object classes. Furthermore, it cannot capture and represent the ternary and n-nary relationship types without losing semantic information.

For example, in Fig. 1.4, there is a PC relationship between internal nodes *Course* and *Student*, and there is another between internal nodes *Student* and *S_Name*. Without further information, there is no way to know there is a binary relationship type between object classes *Course* and *Student*, while the object class *Student* has a composite attribute *S_Name*.

Object Attribute & Relationship Attribute

In both DTD and XSD, attributes are represented as simple element or attribute, which are represented as leaf nodes in the XML schema tree. Thus, it is impossible for them to distinguish object attributes with relationship attributes.

For example, in Fig. 1.4, under the internal node *Book*, it is impossible to know that *BorrowData* is a relationship attribute of the binary relationship type between *Professor* and *Book*, while *Title* is just an object attribute of *Book*.

1.2 Research Problem: Semantics Discovery from XML

In order to improve the conceptual quality, one needs to discover the intended semantics in the logical XML schemas and data. This requires discovering semantic information such as object classes, relationship types, OIDs, object attributes and relationship attributes, as presented in conceptual models for semi-structured data such as ORA-SS proposed in [22]. We refer to this semantic 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 the effectiveness or efficiency of different applications. In the following, we use the XML data tree in Fig. 1.2 and examples to illustrate how the availability of such semantics positively impacts different applications including XML query processing, XML keyword search and XML schema and data integration.

XML query processing

To process an XPath query, e.g.*//Student[Matric# =‘HT001’]/Name* in Fig. 1.2, most of the existing approaches match the query pattern to the XML data to find all matching occurrences. However, if we have the semantics that *Matric#* is the OID of object class *Student* and *Matric#* functionally determines object attribute *Name*, after we get the first matching occurrence, and find a name of the student with his/her matric# being *'HT001'*, we can stop searching the rest of the data. This is reasonable because each *Student* instance should only have one name, and the same student instance may occur many times in the XML data. With the correct and reasonable functional dependency between *Matric#* and *Name*, we can guarantee that even we scan the whole data and return all value of *Name* for that student, all of them will be the same. Other situations that semantics improves XML query processing are shown in [67].

XML keyword search

Semantics-based XML keyword search approaches have been proposed to improve search efficiency and quality, such as in [66, 4]. However, the use of semantics in current studies is still shallow (only on object level, without touching the relationship). For some queries, e.g., {CS5201, CS5208} in Fig. 1.2, its intuitive meaning is to find the common information of two courses, most existing keyword search approaches in XML domain will return the LCA [27] (or enhanced LCA such as SLCA [68]) of the two nodes in their XML data tree. Only by discovering that there is a relationship type between the object classes *Student* and *Course*, one can infer that 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 [69, 59, 68]. More details about how semantics can be used to increase the accuracy of XML keyword search are discussed in [31].

Schema/data integration

For XML schema integration, most of the existing approaches (e.g., [3, 35]) match among elements in XML schema and integrate them based on their structural and linguistic similarities. However, this information is still coarse-grained, as they do not realize the relationship types between the object classes, and do not distinguish object attribute with relationship attribute. For example, in Fig. 1.4, leaf node *Grade* is a relationship attribute of the relationship type between object classes *Course* and *Student*. Without this semantics, when we integrate this XML schema with another similar XML schema, in which object class *Student* has an object attribute *Grade* which means the year of his/her study in school, existing approaches may wrongly match and integrate these two different attributes which have the same attribute name *Grade* and the same parent object class *Student*, because of their high structural and linguistic similarities.

Furthermore, without identifying the ORA-semantics during the XML schema integration, existing approaches may encounter many conflicts including structural conflicts and constraint conflicts. For example, in Fig. 1.4 the object class *Student* is a child node of the object class *Course*, and has a relationship attribute *Grade* as its direct child node. In another similar XML schema tree, the corresponding objet class *Student* may be designed as parent node of object class *Course*, and the corresponding relationship attribute *Grade* may be designed as the direct child node of object class *Course*. In this case, there is an ancestor-descendant conflict among these two XML schema tree, and the attribute *Grade* may be replicated without knowing that it is a relationship attribute. More structural conflicts are discussed in [70].

Although semantics-based XML processing is attracting more and more research attention, unfortunately, most practical applications are still semantics-less.

The main issue is the availability of such semantic information. As mentioned before, most existing XML schema languages used by applications, e.g., DTD and XSD, cannot fully represent the useful 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, which is proposed just to capture the semantic information rather than discovering them, such model still requires manual provision of semantic information (such as specifying the object classes and relationship types among them, etc.) from the initial design or during model transformation from other semantics-less models. 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 in XML database[16, 41], which only identify object instances in XML data or object classes in XML schema, we consider a more comprehensive set of semantic concepts. In particular, we also discover OID, relationship type, composite attribute and distinguish between object attribute and relationship attribute. In our work, we define all these semantic concepts as the ORA-semantics (formal definition will be given in Chapter 2), and propose a novel step-by-step approach to discover the ORA-semantics with the following 4 steps (including the pre-processing step):

1. (Pre-processing) We discover the properties of each semantic concept and propose some related heuristics (if any) based on their characteristics in XML schema (e.g. DTD and XSD) and XML data. Most of these properties are captured and represented in the semantic model for XML data, ORA-SS.

2. We use the properties of each semantic concept to distinguish object class

from other semantic concepts such as role name, composite attribute, aggregational node (which aggregates multiple nodes with identical or similar meaning by an internal node) and explicit relationship type (which explicitly represents the relationship type between object classes as an internal node). In order to distinguish among role name, aggregational node and explicit relationship type, we also make use of the related heuristics together with statistic information of them.

3. Together with properties and heuristics discovered in the pre-processing step, we utilize the statistic information with data mining techniques to identify the OID for each identified object class. After that, we use the identified OIDs to distinguish between object attributes and relationship attributes by the functional/multi-valued dependency extracted from the XML data;

4. We discover the implicit relationship types including IDD relationship types, and dependent object classes which can only be discovered after knowing the OIDs of all object classes. We discover them based on the results from the previous steps and functional/multi-valued dependencies extracted from the XML data.

1.3 Our Contributions

The main contributions of our work include:

1. We discover and summarize the properties and related heuristics for different semantic concepts (e.g. object class, OID, object attribute, composite attribute, role name, aggregational node, relationship type, relationship attribute and dependent object class) of XML database based on their charac-

teristics when they are designed in XML database, including their hierarchical structures, constraints and linguistic features. These properties and related heuristics will be used in our automatic semantics discovery approach.

2. We propose a novel approach to automatically discover semantics from XML schema and XML data. Different from the existing semantics inference approaches mentioned in [16, 41], which only identify object instances in XML data or object classes in XML schema with high recall but low precision, 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.

3. To validate our automatic semantic discovery approach, we conduct experiments over 15 real world data-centric XML datasets, 18 synthetic XML datasets and 5 XML datasets translated from real relational datasets by PhD students doing research in XML. The results show that the ORA-semantics (including object classes, OIDs, object attributes, composite attributes, role names, aggregational nodes, relationship types, relationship attributes and dependent object classes) discovered by our approach has high precision and recall (i.e., above 90% of precision and recall for internal node classification, leaf node classification and implicit relationship type identification).

4. We also develop a demonstration system to show the discovered ORA-semantics by our automatic semantic discovery approach, given an XML data with/without its corresponding XML schema (e.g. DTD or XSD). Furthermore, users can also interact with our demonstration system to verify the discovered results, which will help the revision of other results.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In chapter 2, we explain the ambiguous concept 'semantics' in different domains, and formally define the semantics (i.e., ORA-semantics) which is going to be discovered from XML database in this thesis, as well as how the ORA-semantics can be captured by different semantics data models. In chapter 3, we compare the semantics represented by Ontology model with the ORA-semantics. We also review different studies about discovering semantics in both relational database and XML database. In chapter 4, we present our rule-based semantics discovery approach, and its performance studies are shown in chapter 5. In chapter 6, we introduce a demonstration system based on our approach introduced in Chapter 4. Our future work is briefly introduced in Chapter 7. Conclusion is shown in chapter 8.

CHAPTER 2

PRELIMINARY

2.1 What is Semantics ?

Semantics is the study of meaning. The word semantics itself denotes a range of meanings for different domains, from linguistics, to computer science, and even psychology. Semantics is easily confused with another word, syntax which is the study of correct combination of the building blocks (words, phrases, or symbols) of a language. For example, in [49] the author defined syntax as the study of the principles and processes by which sentences are constructed in particular languages. In the following, we will briefly introduce the semantics in linguistics domain, semantics in ontology domain, semantics in relational/XML database domain and semantics in programming language domain:

Semantics in Linguistics Domain

For semantics in linguistics domain, it is the study and interpretation of the meanings of words and phrases, which are used to understand human expression through language. Furthermore, it also studies about the relationship between different linguistic units such as synonym, homonym, antonym, hypernym, hyponym and so on. Usually, the meaning of a word or a phrase is built by its contextual relations, which means the meaning of it is usually highly dependent on and reflected by its context.

Semantics in Ontology Domain

Different semantic models (e.g., Resource Description Framework (RDF) [30], Resource Description Framework Schema (RDFS) [14], Web Ontology Language (OWL) [7], and Semantic Web [8]) have been designed by the World Wide Web Consortium (W3C) to describe the correct meanings of words or phrases, as well as to connect those words or phrases with the same or similar meaning. Usually, these semantic data model are represented by directed graphs in which its nodes represent real world concepts or entities and the edges represent relationships between them. Furthermore, relationships such as generalization and specialization between concepts or entities are also represented by the hierarchical structure of the directed graph.

For example, the Semantic Web is designed based on the World Wide Web and becomes an extension of it by capturing meanings of component metadata of the web using semantic data model such as RDF, OWL, etc., so that the data can be interpreted and processed by machines.

Semantics in Database Domain

The term semantics can also be used in conceptual modeling in database domain. For example, the ER model is a conceptual model which can capture the semantics such as entities, relationships among entity as well as attributes of entities and attributes of relationships. Similarly, ORA-SS is proposed for semi-structured data to capture the semantics such as object classes, relationship types among object classes as well as object attributes and relationship attributes. To be more specific, semantics here are used to describe some conceptual concepts in schema level rather than data level, which are designed to represent extra information of a metadata besides its linguistic meanings, so that they can help increase efficient/effectiveness of different applications, such as keyword search, query processing, etc.

For example, the ORA-semantics we are going to discover in this thesis refers to different types of element nodes in XML schema.

Semantics in Programming Language Domain

In programming language domain, there is another understanding of semantics. For this kind of semantics, it is defined by the designers of programming languages to represent the meaning of an expression as the computational result the expression contains when it is executed on machines. To be more specific, different programming languages may represent the same semantics with different syntaxes. For example, for the same semantics that stores result of variable ‘x’ and ‘y’ in the variable ‘x’, different syntaxes are needed for different programming languages, such as ‘x += y’ or ‘x = x + y’ in ‘C++’ and ‘Java’, ‘ADD x, y’ in Assembly languages, ‘ADD Y TO X GIVING X’ in ‘COBOL’ and so on.

As the semantics in linguistics domain and in programming language domain are out of the scope of our work, we only consider different semantics in ontology domain and in database domain. More details about the semantics in these two domains will be discussed in Chapter 3. In the following, we will formally define the ORA-semantics, which is the scope of semantic concepts being considered in this thesis, and the results we are going to discover from XML by our approach introduced in this thesis.

2.2 ORA-semantics

In this section, we will describe the semantic concepts used in this thesis. We refer to the tree structure derived from an XML schema as an *XML schema tree*, which not only captures the data but also their hierarchical structures in the XML schema. In the XML schema tree, there are two types of node: *internal node* which has at least one node as its child node, and *leaf node* which does not have any node as its child node. In the following we will introduce 11 semantic concepts in the XML schema tree: *object class*, *object identifier (OID)*, *object attribute*, *explicit relationship type*, *implicit relationship type*, *relationship attribute*, *identifier dependency (IDD) relationship type*, *dependent object class*, *role name*, *composite attribute* and *aggregational node*.

We define the above semantic concepts as the *ORA-Semantics*, which is the scope of the semantic concepts we consider in this thesis. The ORA-semantics is based on the semantic concepts captured by the ER model [15] and ORA-SS [22], especially ORA-SS, which is a semantically rich data model for semi-structured data. In the next section we will discussed how the ORA-semantics can be captured and represented by them and other data models.

Concept 1. ***ORA-semantics (Object-Relationship-Attribute-semantics)*** *In an XML schema tree, the ORA-semantics is the identification of object class with its object identifier (OID) and object attributes, explicit/implicit relationship type with their relationship attributes, dependent object class with its related IDD relationship type, role name, composite attribute and aggregational node. Each*

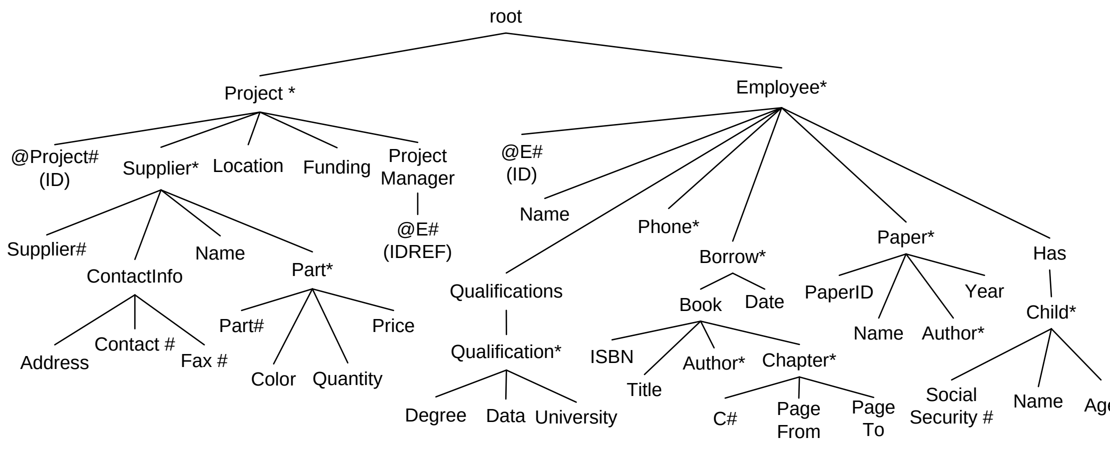


Figure 2.1: An XML schema tree

particular semantic concept in ORA-semantics is called an ORA-semantic concept.

In the following we will explain each ORA-semantic concept in the ORA-semantics, and illustrate them using the XML schema tree in Fig. 2.1 which is derived from a DTD. Among the XML schema tree, each node with a '$*$' as its superscript means it is a repeatable node, which means it can occur multiple times with the same XPath in its corresponding XML data; each node with a '@' as the first letter in its tag name means it is defined as an attribute node[1] in its DTD; each node with an ID or $IDREF$ in the last part of its tag name means it is defined as an ID attribute or IDREF attribute in its DTD.

Object Class, OID & Object Attribute

In the XML schema tree, *object class* is an internal node which represents a real world entity or concept. Along with the object class, there are at least two *object attributes* as its child nodes or descendant nodes which are designed to store and describe the information of the object class. Among the object attributes of each object class, there is an *object identifier (OID)*

[1]Recall that in DTD, a node can either be defined as an element node or an attribute node.

which is designed to identify the object class, which means the value of the OID can uniquely identify each object instance of the object class.

For example, in Fig. 2.1 the internal node *Project* is an object class with its OID *Project#* which is defined as an ID attribute in its DTD, and another two leaf nodes *Location* and *Funding* are the object attributes of the object class *Project*.

Explicit Relationship Type & Implicit Relationship Type

Two or more object classes may be related to each other through a *relationship type*. In the XML schema tree, the relationship type can be divided into two categories based on their structures: *explicit relationship type* and *implicit relationship type*.

- **Explicit relationship type** is explicitly designed as an internal node in the XML schema tree between the object classes, which participate in this explicit relationship type.

  For example, in Fig. 2.1 the internal node *Borrow* is an explicit relationship type between the object classes *Employee* and *Book*, which describes the relationship that an employee borrowing book(s).

- **Implicit relationship type** is not explicitly designed as any node, and it is represented as one or more successive edge(s) in hierarchical order in the XML schema tree among the object classes, which participate in this implicit relationship type. However, not every edge in the XML schema tree represents an implicit relationship type, and we only consider those edges between the object classes.

  For example, among the object classes *Project*, *Supplier* and *Part*, there is an implicit relationship type describing the relationship type that a supplier supplies part(s) for project(s). However, the edge between *Project* and its child node *Location*, just represents the *Location* is an object attribute of the object class *Project*.

For both explicit relationship type and implicit relationship type, their degrees are the number of object classes which participate in them.

Relationship Attribute

Similar to object class and object attribute, a relationship type (both explicit relationship type and implicit relationship type) may have one or more attribute(s) to store and describe its information. We call these attributes as *relationship attributes*. However, relationship attribute is not necessary for a relationship type.

For explicit relationship type, its relationship attributes are designed as its child nodes or descendant nodes. On the other hand, for implicit relationship type, its relationship attributes locates in the lowest level among all the object classes which participate in the implicit relationship type.

For example, the leaf node *Data* in Fig. 2.1 is a relationship attribute of the explicit relationship type *Borrow*, which is its parent node. For example, in Fig. 2.1, given the implicit relationship type among the object classes *Project*, *Supplier* and *Part*, its relationship attributes (if any) will be designed as the child nodes or descendant nodes of the object class *Part*, such as the leaf node *Quantity*.

Dependent Object Class & IDD Relationship Type

Similar to the relationship between entity and weak entity in the ER model, *dependent object class* cannot be identified by its own attributes alone, but has to be identified by its relationship with its parent/ancestor object class(es), and this relationship is called *identifier dependency (IDD) relationship type*. In order to uniquely identify each instance of a dependent object class, the dependent object class needs the OID(s) of its parent/ancestor object class(es) which participates in this IDD relationship type together with its own attributes.

For example, in Fig. 2.1, given the object class *Book*, the internal node *Chapter* is a dependent object class and there is an IDD relationship type between the dependent object class *Chapter* and object class *Book*, because without specifying a book by its ISBN, there is no way we can uniquely identify a chapter by its attribute alone, such as chapter number. Thus, the OID of the dependent object class *Chapter* is the combination of *C#* and *ISBN*, which is the OID of object class *Book*.

Role Name

Role name is a specialization of an object class with an alias or a more specific name of that object class. In the XML schema tree, role name uses the IDREF(S) attribute of DTD or XSD as its child node to reference the original object class which stores its full information.

For example, in Fig. 2.1 the internal node *ProjectManager* is a role name of the object class *Employee* by using its child node *E#*, which is an IDREF attribute of DTD or XSD to reference the OID of the object class *Employee*, which is an ID attribute of DTD or XSD.

Composite Attribute

Composite attribute is a combination of more than one related attributes, which can be object attributes, relationship attributes. Because of this, in the

XML schema tree a composite attribute must contain multiple components as its child nodes, each of which can be a single attribute or another composite attribute.

For example, in Fig. 2.1 the internal nodes *ContactInfo* is a composite attributes which combines three object attributes *Address*, *Contact#* and *Fax#* for its parent object class *Supplier*.

Aggregational Node

In the XML schema tree, an *aggregational node* is an internal node which aggregates all its child nodes with identical or similar meaning, and the aggregational node serves as a structural node in the XML schema tree without extra semantics information besides the semantics of its child nodes. An aggregational node can only aggregate a single kind of semantic concept at a time, and these semantic concepts can be object class, explicit relationship type or composite attribute.

For example, in Fig. 2.1 the internal node *Qualifications* is an aggregational node which aggregates the composite attribute *Qualification*.

2.3 ORA-semantics in Other Models

Recall that the ORA-semantics is proposed based on the semantic concepts captured by the ORA-SS and ER model. In order to let readers have a better understanding of the ORA-semantics, in this section we will discuss whether and how these ORA-semantic concepts can be captured and represented by the ORA-SS, ER model, and other data models.

2.3.1 ORA-SS

Essentially, all ORA-semantic concepts (except role name) introduced in Section 2.2 comes from the semantic concepts captured in **ORA-SS** (Object-Relationship-Attribute Model for Semi-Structured Data), which is a semantically rich XML model. ORA-SS is proposed to capture and represent the semantics in XML data and XML schema. The main idea of ORA-SS is capturing and representing three main semantic concepts: *object classes*, *relationship types* and *attributes*, as highlighted in its name. The ORA-SS model consists of 4 kinds of diagrams, including ORA-SS schema diagram, functional dependency diagram, ORA-SS instance diagram and ORA-SS inheritance diagram. Here we only focus on the ORA-SS schema diagram, which is closely related to the ORA-semantics and capable of capturing the following semantic concepts:

Object Class

- Attributes of object classes;

Relationship Type

- Attributes of relationship types;
- Degree of n-nary relationship types;

Attribute

- Identifier attribute;
- Composite attributes;

In ORA-SS, an *object class* is represented as a labeled rectangle in the ORA-SS schema diagram. An object class has a name and a set of *attributes* to describe and store the information of this object class. Attributes of an object class are represented as labeled circles, and there are directed edges pointing from the corresponding object class to them. Among these attributes, one attribute or a combination of more than one attribute is chosen to be the *OID* of the corresponding object class, and the OID is expressed as filled circle.

Two object classes are connected and related to each other through a binary *relationship type* with degree of 2. In ORA-SS schema diagram, a binary relationship type is expressed as a labeled diamond which is optional and it is assumed on any directed edge from a parent object class to a child object class, which both participate in the relationship type. The relationship type is labeled with its name, its degree and the participation constraints of both two object classes. For a ternary relationship type with degree of 3, there exist a binary relationship type between two object classes and another relationship type between the other object class and the binary relationship type. Similar to the attribute of an object class, an attribute can also belong to a relationship type and be used to store information of it. The difference between object attribute and relationship attribute is the directed edge pointing to the relationship attribute (also expressed as a labeled circle) will be labeled with the name of the corresponding relationship type.

IDD relationship type and *dependent object class* can also be captured by ORA-SS. An IDD relationship types is represented in an ORA-SS schema diagram by a diamond labeled with IDD, and the child node of the diamond is the corresponding dependent object class. Furthermore, in an ORA-SS schema diagram, a *composite attribute* is represented as a labeled circle with directed edge pointing to its component attributes. Although the ORA-SS model can capture and express a lot of other information and constraints, such as cardinality of attributes, ordering of object classes and attributes, etc., we only focus on the ORA-semantics in this thesis. For more details about ORA-SS, please refer to [22, 39].

Example 2.1: In Fig. 2.2 we show the ORA-SS schema diagram of the XML schema tree in Fig. 2.1. There is a binary relationship type *SP* between object

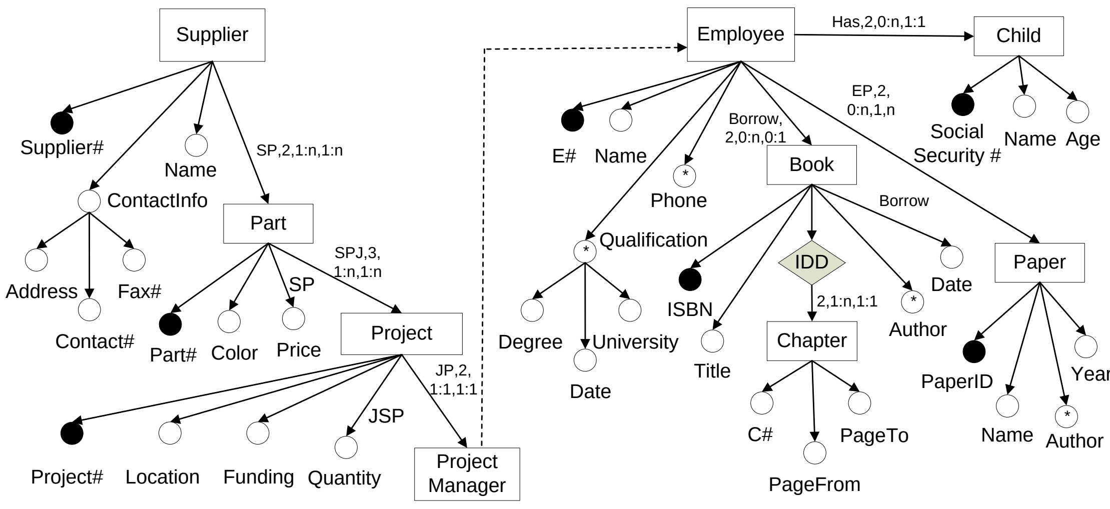


Figure 2.2: An ORA-SS schema diagram of SPJ (Project-Supplier-Part)

classes *Supplier* and *Part* with relationship attribute *Price*; a ternary relationship type *SPJ* among object classes *Supplier*, *Part* and *Project* with relationship attribute *Quantity*. In Fig. 2.2, it also captures the IDD relationship type between object class *Book* and dependent object class *Chapter* by a diamond labeled with 'IDD' between them. □

2.3.2 Relational Model & ER Model

The **relational model** [17] is a popular used data model for commercial data management. In the relational model, data is represented and stored in different relations. Different constraints including primary key constraint, foreign key constraint and domain constrain can be captured in the relational model. However, the relational model does not capture semantic information such as object class, relationship type and other ORA-semantic concepts introduced in Section 2.2.

On the other hand, **ER (Entity-Relationship) model** can be an abstract and conceptual representation of database, which is also designed to capture and represent the semantics in terms of object classes and the relationship types among them, underlies the data. In an ER diagram, an object class is represented as an *entity* with a set of *attributes* which describes the property of the entity. Among the attributes of an entity, there is an attribute or a combination of attributes, which is specified as the *primary key* of the entity, and the value of this attribute can uniquely identify any instance of this entity. The primary key of an entity corresponds to the OID of an object class in the ORA-semantics. In an ER diagram, *relationship types* are explicit designed as diamond having edges connecting to entities participating to this relationship type. Thus, by attaching the attributes to the corresponding entities or relationship types, the ER model can distinguish attributes of entity and attributes of relationship type. Furthermore, *n-nary relationship types*, *weak entities* with its *partial key* and *ID relationship types* can also be captured and represented by the ER model.

ER model can capture and represent most of the ORA-semantic concepts (except aggregational node) just with different terms. Recall that ORA-SS can also capture and represent most of the ORA-semantic concepts. The main difference between ER model and ORA-SS is ER model does not capture the hierarchical structure, while ORA-SS does. This is because ORA-SS is proposed for semi-structured data, especially for XML, which is represented in hierarchical structure.

Example 2.2:

In Fig. 2.3 we show the corresponding ER diagram of the XML schema tree in Fig. 2.1. Entities *Supplier*, *Part*, *Project*, *Employ*, *Book* and *Child* are expressed by rectangles with their primary keys expressed by underlined eclipses in the ER diagram. Weak entity *Chapter* and its partial key are expressed by double-lined rectangle and dot-lined eclipse respectively. Furthermore, the ER diagram also captures the relationship types by diamonds having edges pointing to their relationship attributes, such as binary relationship type *SP*, ternary relationship type

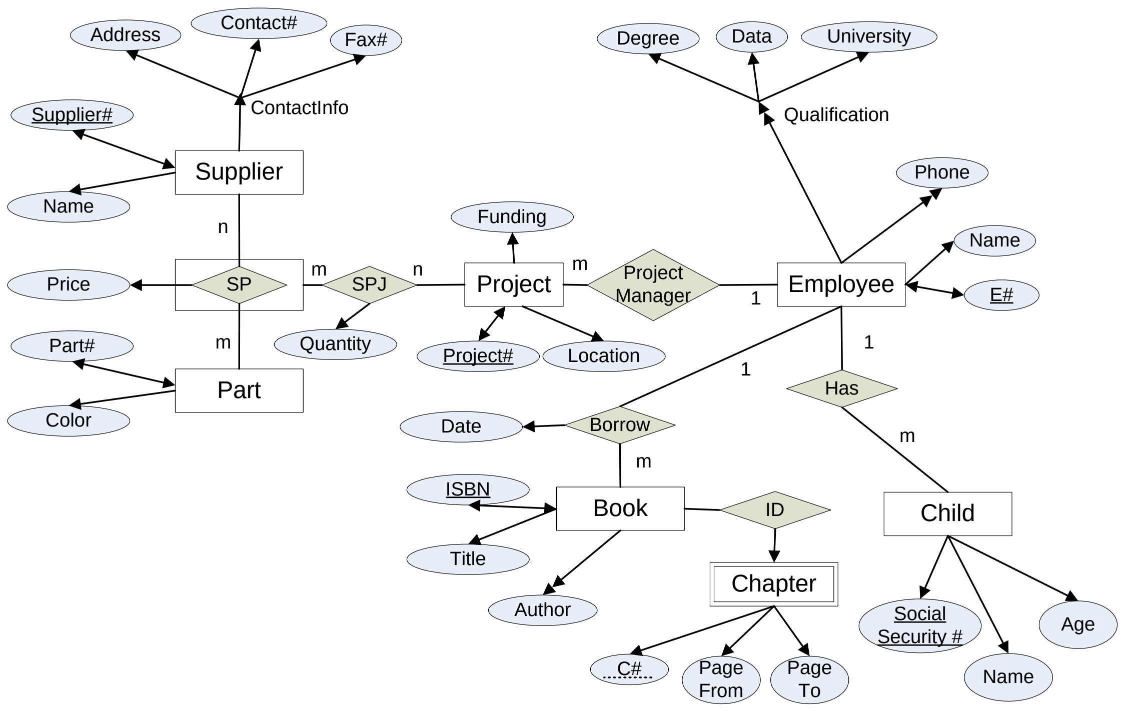


Figure 2.3: An ER diagram of SPJ (Project-Supplier-Part)

SPJ and ID relationship type between entity *Book* and weak entity *Chapter*.

Although this ER diagram cannot capture the aggregational node *Qualifications* as in Fig. 2.1, actually it does not loss any semantics information. It is because *Qualifications* in Fig. 2.1 is just a structural node representing there can be more than one qualification for an employee, which can also be well captured by ER model. □

2.3.3 A Semantic Network-Based Model for XML

In order to capture and represent the semantics that underlies the XML database, another **semantic networked-based model** for XML is proposed in [24]. (Latter we will show many semantics concepts such as ternary/n-nary relationship type, relationship attribute, etc., actually cannot be captured.) This model represents the XML database with the following 4 major components in a directed diagram:

- A series of *basic nodes* and *complex nodes*, which represent the *attributes* and real-world *object classes* respectively;
- A series of directed edges, which represent the *relationship types* between the attribute and object class, and the *relationship type* between object classes;
- A series of labels, which specify each relationship type as one of the following 4 relationship type: *generalization*, *association*, *aggregation* and *of-property*;
- A series of *constraints* defined to restrict the object classes/relationship types.

In the directed diagram, each object class is represented as a complex node having directed edges pointing to a set of basic nodes, which represent the attributes of this object class. Although uniqueness constraints can be applied to the attributes of an object class, it is not enough to capture the *OIDs* of object classes in this model, especially when there are more than one attribute of an object class restricted by the uniqueness constraint, or when the OIDs is comprised by more than one attribute of the object class.

Each directed edge in the semantic network-based model represents a relationship type between the two nodes, which are linked by this directed edge (i.e., from the starting node pointing to the ending node). However, all these relationship types only capture the connection between two nodes, which means they can only express binary relationship type. Furthermore, there is no way it can distinguish the attributes of object class with the attributes of relationship type, as all attributes are expressed as basic node pointed by an object class; *IDD relationship type* and *dependent object class* cannot be captured neither.

Example 2.3: In Fig. 2.4 we show the corresponding semantic network-based diagram of the XML schema tree in Fig. 2.1. Although this diagram captures the object classes *Project*, *Supplier*, etc., and the binary relationship types between

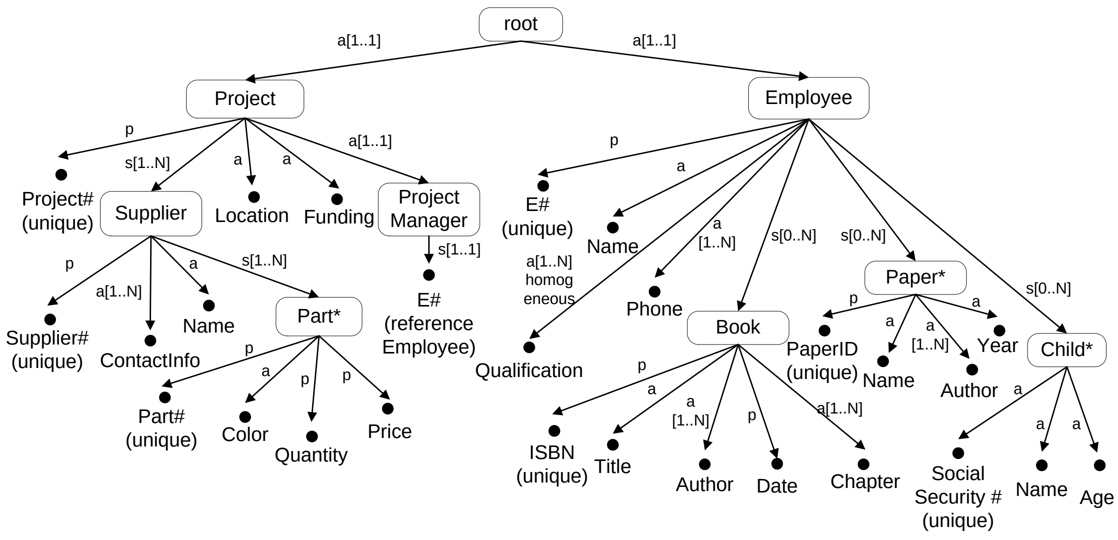


Figure 2.4: An semantic network-based diagram of SPJ (Project-Supplier-Part)

them, it cannot specify their OIDs by using uniqueness constraint alone. Also, it cannot capture the ternary relationship type among *Project*, *Supplier* as well as *Part*, and cannot distinguish between attribute of object class *Color* and attribute of relationship type *Quantity*. Furthermore, the dependent object class *Chapter* as well as the IDD relationship type between it and its parent object class *Book* are lost by this model. □

2.3.4 Object Exchange Model & DataGuide

The **Object Exchange Model (OEM)** [45] is proposed to describe an XML data with a labeled directed graph. In the OEM, *objects* are represented as vertices and *relationships* are represented as edges. There are two kinds of object in the OEM: *atomic object* and *complex object*. Atomic object does not have any outgoing edge, but it contains a value from one of the basic atomic data types such as string, integer, etc.; while complex object has at least one outing edge pointing to other complex objects or atomic object. The OEM only represents an instance of an XML document, and a **DataGuide** [26] model can be used to represent the schema of an OEM instance graph. For an OEM instance graph, the label path of every object in it is only appear exactly once in its corresponding DataGuide.

Given an OEM instance graph with its corresponding DataGuide, let us discuss how they can capture the semantic concepts in the ORA-semantics. Because there are only two kinds of node in OEM and DataGuide: atomic object and complex object, although the semantic concept object class in our ORA-semantics can be represented by the complex object in DataGuide, it still cannot distinguish the object class with some other semantic concepts such as composite attribute, aggregational node and dependent object class in the ORA-semantics. Similarly, all attributes are represented as the atomic object in OEM and DataGuide, which make it not possible to distinguish object attribute and relationship attribute in the ORA-semantics. For relationship type, only the binary relationship type between two complex objects can be captured by OEM and DataGuide, while the ternary and n-nary relationship type, as well as the OID and IDD relationship type are not possible to be captured by OEM and DataGuide.

Example 2.4: In Fig. 2.5 we show the corresponding DataGuide of the XML schema tree in Fig. 2.1. The DataGuide captures the hierarchical structure of the XML schema. However, as mentioned above, it cannot distinguish among the object class *Supplier*, composite attribute *ContactInfo*, aggregational node *Qualifications* and the dependent object class *Chapter*; it cannot identify the ternary relationship type among *Project*, *Supplier* and *Part* as well as the IDD relationship type between object classes *Book* and dependent object class *Chapter*; it cannot distinguish between the object attribute *Color* with relationship attributes *Quantity* and *Price.* □

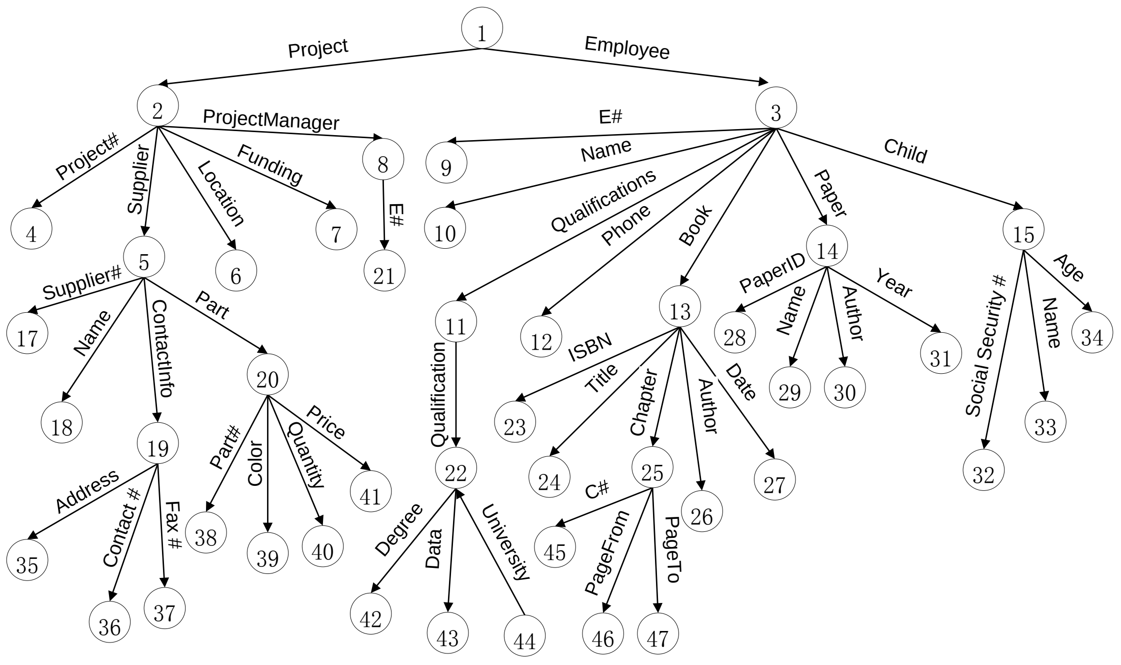


Figure 2.5: An DataGuide diagram of SPJ (Project-Supplier-Part)

2.4 Chapter Summary

In this chapter, after clarifying the ambiguous world ‘semantics’ in different domains, we introduce 11 semantic concepts which will be used in this thesis: object class, object identifier (OID), object attribute, explicit relationship type, implicit relationship type, relationship attribute, identifier dependency (IDD) relationship type, dependent object class, role name, composite attribute and aggregational node. We define them as the ORA-semantics, and call each single semantic concept included in ORA-semantics as an ORA-semantic concept.

The ORA-semantics is proposed based on the semantics concepts captured and represented in ER model and ORA-SS. Different from the ER model, ORA-SS can also capture the hierarchical structure of the underlying data. Another semantic network-based model for XML is proposed to capture the semantics that underlies the XML data and XML schema, many ORA-semantic concepts cannot captured by this model, such as OID, ternary/n-nary relationship type, relationship attribute, etc. Object exchange model and DataGuide are proposed to capture the hierarchical structure XML data and XML schema. However, they only distinguish two kinds of nodes, atomic object and complex object. It is far from enough to distinguish between object class and composite attribute, and DataGuide also cannot capture the ternary/n-nary relationship type as well as relationship attribute.

To summarize the above discussion, we list ORA-semantic concepts and show whether they can be captured and represented by the above 4 models. (i.e., ER model, ORA-SS, semantic network-based model, and DataGuide) in Table 2.1.

Table 2.1: ORA-semantic concepts supported in different models

	ER Model	ORA-SS	Semantic Network-based Model	DataGuide
Object Class	✓	✓	✓✘	✓✘
OID	✓	✓	✘	✘
Object Attribute	✓	✓	✓✘	✓✘
Binary Relationship Type	✓	✓	✓✘	✓✘
N-nary Relationship Type	✓	✓	✘	✘
Relationship Attribute	✓	✓	✘	✘
IDD Relationship Type	✓	✓	✘	✘
Dependent Object Class	✓	✓	✘	✘
Role Name	✓	✘	✘	✘
Composite Attribute	✓	✓	✓	✘
Aggregational Node	✘	✓	✘	✘

Note that '✓✘' means although the corresponding ORA-semantic concept can be represented in this model, but it also represents many other ORA-semantic concepts with the same notation. For example, although object class can be represented as a *complex object* in OEM & DataGuide, other ORA-semantic concept such as composite attribute is also represented as a complex object in this model.

CHAPTER 3

RELATED WORK

As discussed in Chapter 2, in this thesis we only consider the semantics in ontology domain as well as the semantics in database domain. In this chapter, we will review the semantics captured and represented in different ontology languages (i.e., RDF, RDFS and OWL), and compare them with the semantics captured by our ORA-semantics in XML database.

Furthermore, because of the structural flexibility of XML, discovering semantics in XML is very challenging and attracts little attention from researchers. We will also review the related works about discovering semantics from relational database and related works about discovering semantics from XML database in this chapter.

3.1 Ontology

3.1.1 Semantics in Ontology

In order to make information more easily understood and accessed by machines, different ontology languages (such as RDF, RDFS, OWL, etc.) have been proposed to model the real world concepts using the corresponding ontologies with their semantics being explicitly captured in the corresponding semantic models. For example, the World Wide Web is constructed and organized by billions of unstructured documents, which makes it can be easily understood by human being but cannot be understood and accessed by machines. In order to solve this problem, the Semantic Web has been proposed by the W3C to organize the information in the World Wide Web with a more machine-friendly way, by building the web on the RDF, which is a framework for describing Web resources.

Usually, related ontologies are designed in a huge knowledge base in a particular domain, and they must be designed by some domain experts to ensure their accuracy. For example, the Gene Ontology Project [1] is a huge knowledge base of bioinformatics and it is proposed and designed to standardize the representation of genes of different species; the TGDdataset [2] is a knowledge base about the association among traditional Chinese medicine, genes and diseases.

In the following we will introduce 3 frequently used ontology languages: RDF, RDFS and OWL.

3.1.2 Resource Description Framework & RDF Schema

Resource Description Framework (RDF) [30] is a W3C standard proposed to describe Web resources, such as the color, year, and type of a wine. RDF was proposed

[1]http://www.geneontology.org/

[2]http://code.google.com/p/junsbriefcase/wiki/TGDdataset

to provide a way for describing information for computer applications to understand and process, rather than for human being. RDF documents are published in XML format so that they can be exchanged between different applications easily.

RDF uses a *triple* to describe any resource, and the triple contains a *resource*, a *property* and a *property value.*

- A **Resource** is a thing which can have a URI (Uniform Resource Identifier), such as 'http://www.w3.org/RDF/'.
- A **Property** is a thing which is used to describe the resource, such as the name of a resource.
- A **Property Value** is the value of a property.

Example 3.1: In Fig. 3.1, we show a simple RDF document describing a wind. Recall that RDF documents are written in XML, in Fig. 3.1, `<rdf:RDF>` is considered as the root node of this RDF document, and `xmlns:rdf` and `xmlns:food` specify the namespaces for elements with prefixes `rdf` and `food` respectively. The element `<rdf:Description>` describes the resource *Grape* identified by the attribute `rdf:about` and elements `<food:color>` and `<food:year>` describe the properties *color* and *year* with their corresponding property values of the resource. □

With the triple (resource, property, property value), the ontology language RDF can be used to describe any resources in the real world. Although RDF can capture the binary relationship type between resources by indicating the relationship type and another resource using the property and property value respectively, it cannot capture the ternary or n-nary relationship types. Furthermore, RDF can only describes resources in data instance level, and cannot represent these resources in a higher level, the schema level, so that it can be designed and understood more easily.

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:food=""http://www.w3.org/TR/2003/owl/food#">

  <rdf:Description rdf:about="resource="http://www.w3.org/TR/2003/owl/wine">
    <food:color>Red</food:color>
    <food:year>1998</food:year>
  </rdf:Description>

</rdf:RDF>
```

Figure 3.1: An RDF document describing the resource Grape.

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdfs:Class rdf:ID="Wine" />

<rdfs:Class rdf:ID="WineGrape">
  <rdfs:subClassOf rdf:resource="#Wine"/>
</rdfs:Class>

</rdf:RDF>
```

Figure 3.2: An RDFS document describing the resources Grape and WineGrape.

RDF schema (RDFS) [14] is proposed as an extension to RDF for capturing the semantics in schema level and let designer design a set of classes with the same properties in any particular application domain. Classes in RDFS are similar to the classes in object oriented programming languages and XML schemas in XML database. With the ontology language RDFS, resources can be defined as instances of classes, as well as subclass/superclass of other classes.

Example 3.2: In Fig. 3.2, we show a RDFS document describing 2 classes *Grape* and *WineGrape* with element `<rdfs:Class>`, and uses element `<rdfs:subClassOf>` as a child node of class *WineGrape* to define it as a subclass of class *Grape*. □

Besides defining classes and subclasses, RDFS language can also define subproperty of a defined property with element `<rdfs:subPropertyOf>`, as well as specifying the domain and value range of a property with `<rdfs:domain>` and `<rdfs:range>` respectively. With these elements, RDFS can represent the hierarchical structure of different classes, similar to DTD or XSD capturing the hierarchical structure of XML schema. However, although the hierarchical structure of classes can be captured by RDFS, it still cannot fully capture the relationship types between classes, especially for ternary and n-nary relationship types. The only relationship types can be captured are those directly represented in the hierarchical structure. To be more precise, the hierarchical structure only captures the binary relationship type such as class-subclass relationship type between two classes which are directly connected with each other in the corresponding hierarchical structure of the RDFS document.

3.1.3 Ontology Web Language

Ontology Web Language (OWL) [7] is proposed based on RDF/RDFS and to extend the description ability of RDF/RDFS to better describe the ontologies. In order to be compatible with RDF/RDFS, OWL is also written in XML format, and it inherits many predefined elements in RDF/RDFS such as `<rdf:resource>`, `<rdfs:subClassOf>`, `<rdfs:subPropertyOf>`, `<rdfs:domain>` and `<rdfs:range>`, etc.

Different from the triples in RDF, OWL model uses two basic components *class* and *property* to capture the semantics in schema level. Correspondingly, OWL uses *instances of class* and *relationship* between these instances to capture and represent the semantics in the real world in data level. Similar to RDF and RDFS, the gold of OWL is to make the data easier for machine to understand. In the following, we will introduce the two basic components of OWL language, *class* and *property*, and corresponding examples will be given based on the OWL document shown in

Fig. 3.3[3]. Furthermore, in Fig 3.4 we show an undirected graph which captures the hierarchical structure of the OWL document in Fig. 3.3.

Class

For a real world concept or entity, it is defined and represented as *class* by the element `<owl:Class>` in OWL. OWL also inherits the element `<rdfs:subClassOf>` from RDFS to capture the superclass-subclass relationship types among classes. Similar to the object instances of an object class in ORA-semantics, in OWL a class can also has its instances, which are called individuals in OWL.

Example 3.3: In Fig. 3.3, the OWL document defines the classes *Food*, *Wine*, *Grape*, *WineGrape*, *VintageYear*, *WineDescriptor* and *WineColor* in line 1, 2, 5, 8, 12, 24 and 25. Furthermore, as is shown in Fig. 3.4, it also defines class *WineGrape* as a subclass of class *Grape*, which is also a subclass of class *Food*, and class *WineColor* as a subclass of class *WineDescriptor*. □

Property

In OWL, there are two kinds of properties, *datatype property* and *object property*. For a datatype property, defined by an element `<owl:DatatypeProperty>` in OWL, it describes the relation between a class and a RDF literals/an XSD datatype. The corresponding class is specified by a subelement of datatype property `<rdfs:domain>`, while the RDF literals/XSD datatype, which contains built-in simple datatypes such as string, boolean, integer, etc., is specified by another subelement `<rdfs:range>`. Essentially, the datatype property is designed to represent the attribute-value relation, similar to the *attribute* in our ORA-semantics.

[3]For simplicity, we ignore the document header and namespace specification.

```
<owl:Class rdf:ID="Food"/>
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&Food"/>
</owl:Class>
<owl:Class rdf:ID="Grape"/>
  <rdfs:subClassOf rdf:resource="&Food" />
</owl:Class>
<owl:Class rdf:ID="WineGrape">
  <rdfs:subClassOf rdf:resource="&Grape" />
</owl:Class>
<WineGrape rdf:ID="CabernetSauvignonGrape" />
<owl:Class rdf:ID="VintageYear" />
<owl:DatatypeProperty rdf:ID="yearValue">
  <rdfs:domain rdf:resource="#VintageYear" />
  <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
<VintageYear rdf:ID="Year1998">
  <yearValue rdf:datatype="&xsd;positiveInteger">1998</yearValue>
</VintageYear>
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
<owl:Class rdf:ID="WineDescriptor" />
<owl:Class rdf:ID="WineColor">
  <rdfs:subClassOf rdf:resource="#WineDescriptor" />
</owl:Class>
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasColor">
  <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
  <rdfs:range rdf:resource="#WineColor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdfs:domain rdf:resource="#WineGrape" />
  <rdfs:range rdf:resource="#VintageYear" />
</owl:ObjectProperty>
```

Figure 3.3: An OWL document describing the ontology of Wine.

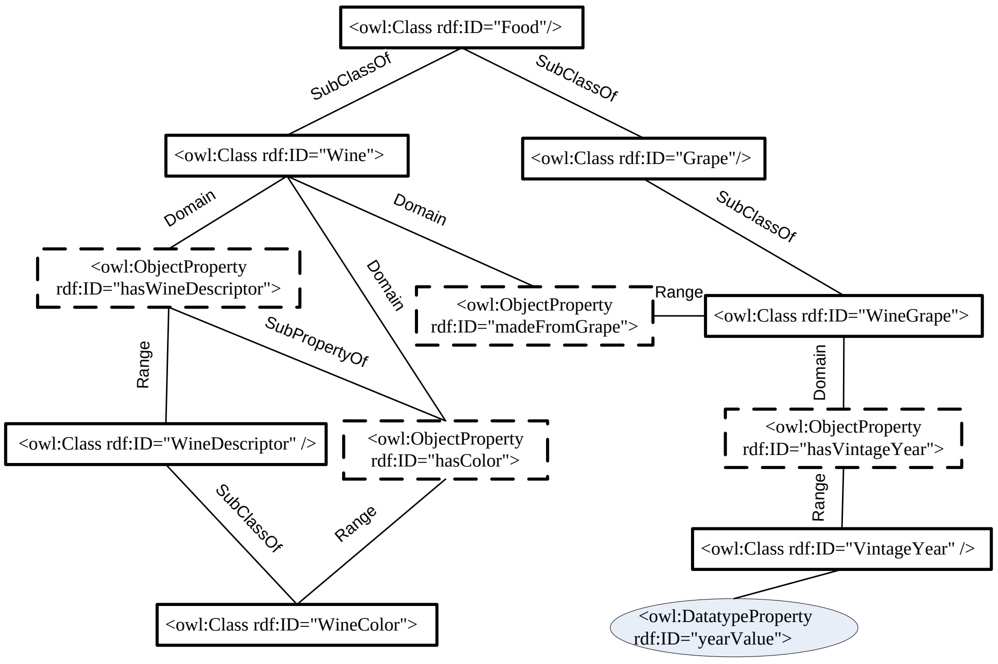

<owl:Class rdf:ID="Food"/>
SubClassOf
SubClassOf
<owl:Class rdf:ID="Wine">
<owl:Class rdf:ID="Grape"/>
Domain
Domain
Domain
SubClassOf
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
<owl:ObjectProperty rdf:ID="madeFromGrape">
Range
<owl:Class rdf:ID="WineGrape">
Range
SubPropertyOf
Domain
<owl:Class rdf:ID="WineDescriptor" />
<owl:ObjectProperty rdf:ID="hasColor">
<owl:ObjectProperty rdf:ID="hasVintageYear">
SubClassOf
Range
Range
<owl:Class rdf:ID="VintageYear" />
<owl:Class rdf:ID="WineColor">
<owl:DatatypeProperty rdf:ID="yearValue">

Figure 3.4: A undirected graph capturing the hierarchical structure of the OWL document in Fig. 3.3

Example 3.4: In Fig. 3.3, given a class *VintageYear* being defined in the OWL document, in line 13-16 a datatype property *yearValue* is being defined with its domain as class *VintageYear* and its range as *positiveInteger*, which is a basic XSD datatype. This datatype captures the semantics that class *VintageYear* has a property *yearValue* with its value as a positive integer. The same semantics can be captured using our ORA-semantics by defining an object class with an object attribute, whose value is a positive integer. □

Another property in OWL is called *object property*, represented as an element `<owl:ObjectProperty>` in OWL. Object property is proposed to describes the relationship type between two classes, which are specified by the elements `<rdfs:domain>` and `<rdfs:range>`. To be more precise, the domain element of an object property specifies the class that can have this kind of object property, and the range of an object property specifies the class that having a relationship type with the class specified by the domain element. Similar to the class in OWL, OWL also inherits the element `<rdfs:subPropertyOf>` from RDFS to define a sub-property of an object property. If the sub-property does not redefine its domain and range, it will inherit the domain and range of its super-property.

Example 3.5: In Fig. 3.3, line 20-23 define an object property named *madeFromGrape* with its domain being defined as *Wine* and range being defined as *WineGrape*, it means there is a binary relationship type between class *Wine* and class *WineGrape* with the semantics that wine is make from wine grape, as is shown in Fig. 3.4. Line 32-35 define an object property *hasColor* which is a sub-property of another object property *hasWineDescriptor*. Because *hasColor* only redefines its range as class *WineColor*, it will inherit the domain *Wine* from its parent object property *hasWineDescriptor*, and represent a binary relationship type between class *Wine* and class *WineColor*. □

Although OWL uses object properties to capture the binary relationship type between classes, there is impossible for it to capture the ternary and n-nary relationship types between classes, as the object property can only connect to two classes by its subelement domain and range. Furthermore, OWL also cannot capture the relationship attribute, because attributes are represented by datatype properties in OWL, while these datatype properties can only connect to classes. This means in OWL the object attributes and relationship attributes are mixed together, and there is no way it can distinguish them.

3.1.4 Differences: Semantic in Ontology vs. Semantics in XML

From the previous introduction about ontology languages, especially OWL, it seems that the semantics captured by ontology model, which is the semantic model built with ontology languages (e.g. OWL), is similar to the semantics in our ORA-semantics (such as object class and relationship type). This makes the semantic model built by ontology languages seems be more informative and can be more applicable for different applications. However, the XML data model and the ORA-semantics hidden in XML cannot be replaced by them for the following two reasons:

The design purposes of ontology model and XML model are different

Ontology model is frequently used to design knowledge bases in different domains. First of all, because of the design purpose of an ontology knowledge base is to create an authoritative, organized, and huge collection of data in a particular domain for different people or applications to share and utilize. Thus, it is usually designed to be as complete as possible, trying to capture as much concepts and relationships as possible in the corresponding domain, which makes the corresponding ontology model complex and huge. The existing ontology knowledge bases are also domain specified, such as the Gene Ontology Project and TGDdatase mentioned in Section 3.1.1. Furthermore, there is a high accuracy requirement for ontology knowledge base, and in order to ensure the accuracy of an ontology knowledge base, it is usually manually designed by the domain experts, which makes creating an ontology knowledge base becomes very costly. Because of all the above features of ontology models, there are much less ontology documents than XML documents in the real world, and ontologies are not always available in any domain.

Most of the famous frequently used ontology documents are built on some specific domains, such as the domain of gene information, the domain of medicines and diseases, etc. In these domains, with a well-defined ontology model designed by the domain experts, the machine can work on these ontologies to access and discover knowledge (such as meaningful sequences of particular gene pairs, identifying which kinds of medicine is suitable for a particular kind of disease, identifying diseases based on particular symptoms, etc.) by machine reasoning. However, for XML model, it is much more frequently used in the real world and especially for exchanging information in the Internet. Most of the time, the XML document does not need to contain complete information of a whole domain, and it only need to contain information that user cares about. Furthermore, an XML document is not domain specified.

The semantics in ontology model and ORA-semantics are different

Object class and relationship type among object classes are two important concepts in our ORA-semantics. Although both ontology model and XML model can define object class (class in OWL model) and relationship type among object classes (object property in OWL model, which only capture binary relationship type), they are not exactly the same. In the following, we use OWL as a representative of ontology model to compare the object classes, relationship types and attributes captured by OWL and our ORA-semantics.

(I) Object Class in ORA-Semantics & Class in OWL

In ontology model, class is a more general concept than the object class in our ORA-semantics. In OWL, any real world concept can be defined as a class, and any defined class is a sub-class of the class `<owl:Thing>`, which means any concept can be defined as a class in OWL, even some concepts (such as name, age, etc.), which should be defined as attributes of a class to describe its properties, are also defined as classes in OWL.

For example, in Fig. 3.3, both *Wine* and *VintageYear* are defined as classes. However, for the object class in our ORA-semantics, it is defined as a real world entity or concept which has a set of attributes describing the properties of this entity or concept. Based on this, *Wine* is reasonable to be defined as an object class if it also has a set of attributes to describe properties of the wine, such as *Brand*, *Type*, etc. However, for *VintageYear*, it should not be defined as an object class in XML model, and it is more reasonable to be defined as an attribute of object class *Wine* to describe the year on which the wine is vintaged.

(II) Relationship Type in ORA-Semantics & Object Property in OWL

OWL uses object properties to represent the relationship type between two classes. As mentioned above, some object attributes in ORA-semantics (e.g. *VintageYear*) are also defined as classes in OWL. In consequence, the object property in OWL also captures the relationship types between a class and some of its object attributes.

For example, the *hasVintageYear* represents a binary relationship type between classes *Wine* and *VintageYear* in OWL, which will not be considered as relationship type in ORA-semantics, but the object class *Wine* having an object attribute *VintageYear*.

The object property in OWL can only capture the binary relationship type between two classes, while with ORA-semantics we can capture ternary and n-nary relationship types among more than two object classes. Furthermore, OWL also cannot capture the relationship attribute.

(III) Attribute in ORA-Semantics & Data Property in OWL

Another difference between the semantics captured by OWL and ORA-semantics is that ORA-semantics distinguishes between object attribute and relationship attribute in XML, while OWL cannot. This is because an attribute is captured by a dataproperty in OWL, which is associated with a defined class, and a class does not distinguish whether this attribute is describing the property of it or describing the relationship type it participates in.

3.2 Semantics Discovery in Relational Database

Currently, relational database has been popularly deployed and used for commercial data storage and data management. However, an ER diagram which captures the corresponding semantics information such as entities (object classes), relationship types, and distinction between attributes of entity and attributes of relationship type, of a relational database is not always available. With these semantic information, the meaning of the database can be better comprehended, which largely facilitate data processing and data management, even further database maintenance will be much easier.

In order to discover semantic information from relational database, database reverse engineering (DBRE) has been proposed to reconstruct a higher level of abstraction (such as conceptual schema) of a database in form of an entity relationship diagram in ER model or an extended entity relationship diagram in EER model, or other extension of ER models. In [28] the authors proposed a framework for DBRE, which contains a two-step process containing:

1. **Data structure extracting**, which extracts the DBMS-dependent data structure;

2. **Data structure conceptualization**, which discovers the semantics from the data structure and expresses it with a high level data model.

In the following, we introduce different approaches on DBRE for extracting an ER model or EER model from a relational database.

3.2.1 DBRE with Integrity Constraints

In the early stage, many approaches [48, 44, 55] for transforming a relational database to a conceptual model usually assume that the correct functional/multi-valued dependencies, inclusion dependencies, even primary keys and foreign keys are provided at the beginning of the process, which is not always the case.

In [2], the authors proposed an approach for extracting the conceptual model (ERC+ data model [56], which is an extension of ER model by capturing multi-valued, complex objects and multi-instantiation) schema given a relational database. This approach extracts the semantic information such as entities and relationships. The authors in [2] analyze the results of join clauses from user queries to identify those semantically connected attributes in different relations by links. To be more precise, given a relational database, equal-joins are used to discover those attributes that represent references between relations. From these equal-join conditions, keys of different relations can also be inferred by some heuristics. For example, those involved attributes in a cyclic join, which means attributes join with themselves, cannot be the key. With the analysis of these equal-join clauses, a connected diagram can be built with relations as nodes and joint attributes as edges/links between them. Functional dependencies and inclusion dependencies can also be extracted from the data. The authors in [2] use id-independency properties of different relations, which are determined by analyzing the correspondences of references and keys of relations, to identify the entity types, relationship types and even weak entity types and form the corresponding conceptual model.

In [2], the authors require relational schemas in third normal form (3NF), which makes sure that each relation corresponds to a unique entity/object class. However, because of the implementation and maintenance concern, some commercial databases are directly built in 1NF or 2NF, or denormalized for efficiency requirement. Furthermore, this approach also heavily depends on the links discovered between attributes of different relations, and these links may be built between the attributes with accidentally unique values. Thus, user interaction is still needed for verifying the intermediate result (such as the discovered functional dependencies and inclusion dependencies) during the process.

In [50], the authors proposed a approach to deal with the database reverse engineering with denormalized relational schemas, which means the relational schemas are necessary to be in 3NF beforehand. The first step of this approach is to decompose a denormalized relational schema into 3NF schemas by functional dependencies, which has been studied in [9]. After this, each relation in 3NF maps one entity/object class. Similar to [2], this approach also discovers the key constraints, referential integrity constraints, and the relationship (links) between entities by analyzing the equal-join queries. Nevertheless, as the dependencies (including functional dependencies, and inclusion dependencies) discovered from the data may not be meaningful because of the small data problem (i.e. as dependencies are merely constraints imposed in the current data, whenever the dataset is small, meaningless dependencies may also be discovered.), user interaction is still necessary in this approach to validate the presumptions on the discovered dependencies.

3.2.2 DBRE with Semantic Dependencies

Although above approaches try to identify relationships between relations mainly through integrity constraints and dependency constraints with the flat underlying data, they cannot identify ternary relationships and n-nary relationships as well as the corresponding relationship attributes. Furthermore, many other semantics concepts hidden in the underlying data such as multi-valued attributes, ISA relationships, and recursive relationships, etc., are also not considered in these approaches.

In [36, 37], the authors claimed that the functional/multi-valued dependencies and inclusion dependencies are only constraints to enforce database integrity, and cannot imply any semantic relationship between two sets of attributes. Based on this, the authors proposed a concept named semantic dependency to specify a semantic relationship between two sets of attributes. With the semantic dependency, in [37] they proposed an approach to convert the relational schema to the corresponding ER schema in two steps. Firstly, they use a semi-automatized approach to semantically enrich the relational schema (i.e., identifying semantic dependencies in the relational schema). The second step is converting the relational schema to the corresponding ER schema with the previous semantic dependencies.

However, although with the semantic dependencies we can better discovery the semantic relationship underlies the data. Necessary user input/interaction is still necessary during the semantically enrich step.

3.3 Semantics Discovery in XML Database

As discussed in Chapter 1, because of the limitations of XML schema language (e.g. DTD and XSD), many important semantic concepts (such as object class, relationship type, etc.) cannot be captured and represented by them. Because of this, the ORA-SS is proposed to capture and represent different semantic concepts for semi-structured data, especially for XML. However, as discussed in Chapter 2, ORA-SS is proposed to capture the semantic concepts rather than discovering them, and it still requires manual provision of semantic information (such as specifying the object classes and relationship types among them, etc.) from the initial design or during model transformation from other semantics-less models. In consequence, the semantics discovery approach for XML we are going to introduce in this thesis can be used as an important component for building an semantic model (such as ORA-SS) for a given data-centric XML document, or transforming it from a semantics-less XML model to a semantics-rich model.

Currently, because of the structural flexibility of XML document, discovering semantics in XML is still very challenging and attracts little attention from researchers. Most of the existing related works only focus on the topic of object identification [41, 74, 16]. The problem of object identification in XML has been understood differently by different researchers. The one which is closely related to our semantics discovery approach is identifying the object class in XML schema level or identifying object instance in XML data level. 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, such as object class, relationship type and relationship attribute, etc.

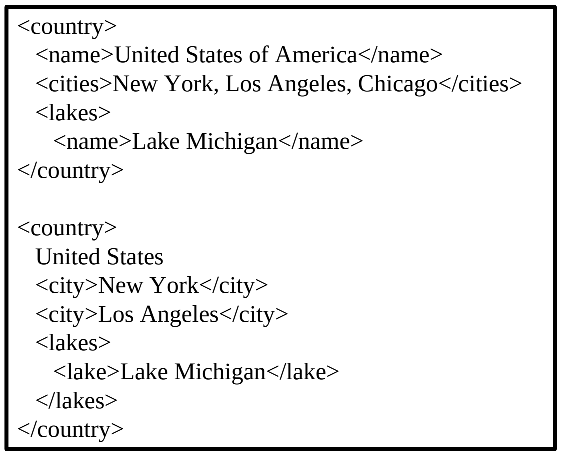
3.3.1 Object Class Identification in XML Schema Level

In some XML applications, researchers try to identify the object classes from XML schema to serve their applications. For example, in the context of view design, all internal nodes in the XML schema tree are considered as object classes in [16].

Obviously it is not always correct since the internal nodes can also be semantic concepts such as relationship type, composite attribute or aggregational node as well. XSeek [41] and MaxMatch [74] infer semantics from the XML schema to identify return nodes/relevant matches. They identify all repeatable nodes as object classes, and recall that repeatable nodes are nodes which can occur more than once with the same XPath in their XML data tree. However, other semantic concepts such as composite attributes can also be designed as repeatable nodes in their XML schema.

All the above approaches try to identify the object class based on some particular structural properties of object class about how it is designed by the designer. However, as we can see, these are far from enough to identify all object class with high accuracy in term of both precision and recall. Although they can achieve high recalls, which means most of the object classes can be identified by them. These approaches also suffer from low precisions, because many elements belonging to other semantic concepts rather than object class are also identified as object classes by them.

Furthermore, these approaches only focus on the identification of object class, while the ORA-semantics we are going to discover in this thesis also contains OID, object attributes, explicit/implicit relationship type, relationship attributes, dependent object class, IDD relationship type, role name, composite attribute and aggregational node. All these ORA-semantic concepts also contain much semantic information, which can also help increasing efficiency or effectiveness for different applications such as XML keyword search, XML query processing, etc. as shown in Chapter 1.



```
<country>
  <name>United States of America</name>
  <cities>New York, Los Angeles, Chicago</cities>
  <lakes>
    <name>Lake Michigan</name>
</country>

<country>
  United States
  <city>New York</city>
  <city>Los Angeles</city>
  <lakes>
    <lake>Lake Michigan</lake>
  </lakes>
</country>
```

Figure 3.5: Two objects with different structures but representing the same real world entity.

3.3.2 Object Identification in XML Data Level

Another understanding of object identification is inherited from the corresponding topic in relational database, entity identification. Entity identification in relational database is also named record linkage and record matching, which has been well-studied for years. Some approaches [63, 64] have been proposed to solve the corresponding problem for semi-structured data. In these approaches, their goals are to compare objects/records for identifying those objects/records with different representations (such as with different names, with different hierarchical structures or with different attributes describing the objects), but with the same meaning/semantics and representing the same real world entity.

For example, in Fig. 3.5, both two country elements are describing the same real work entity, Unite State of America, but with different hierarchical structures in the XML data.

The above approaches focus on comparing among objects, but ignore the difficulty of identifying these objects. They consider XML elements with subelements or attributes as objects. In [64], the authors propose a framework named *DOG-*

MATIX, which extracts data from XML document and stores them in form of relations, which is called object descriptions. Each object description contains many tuples to describe different aspects of this object, which is similar to the object attributes, and different tuples may be identified as similar or contradictory with each other based on their properties (such as their string edit distances, their value range, and other constraints). Finally, *DOGMATIX* uses the number of contradictory or similar tuples to determine the similarity of different object descriptions.

In [51], the authors improve the *DOGMATIX* approach by also considering nested objects and naming them as descendants. To be more precise, these nested objects are captured by the PC (parent-child) and AD (ancestor-descendant) relationship by XML hierarchical structure. They use bottom-up approach to compare objects level by level. In this way, the similarity of objects in lower levels can also be considered as a factor when comparing objects in upper levels. To be more precise, the similarity of two objects is affected by whether their descendants are also similar to each other.

However, although these approaches capture the binary relationship between objects by their hierarchical structures (ternary/n-nary relationship cannot be captured), these is not enough. All relationships they captured do not carry any semantics. They only realize that two objects are related to each other, but never with how they are related and they are related to each other by which relationship. Furthermore, these approaches also do not realize the OID of objects, which is designed by designer to uniquely identify the objects. With OIDs of objects being identifying, identical objects can be discovered easily.

In [53], the authors introduce an approach to identify object with approximate joins. In [63], XML objects are compared using string comparison functions to detect duplicate objects. Semantics discovered in these works is very limited. First, as mentioned, these approaches only discover objects, rather than other ORA-semantic concepts, such as object identifier, relationship type, object attribute, relationship attribute, etc. Second, these approaches heavily depend only on XML data they have, which may cause wrong discoveries when the datasets are small.

Besides object identification, in many web-based applications, inference of different semantic concepts is actually done by user corroboration, i.e., manual effort [21, 58]. 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.

3.4 Chapter Summary

In this chapter, we introduced the ontology languages (including RDF, RDFS and OWL) and the semantics captured by them. We show that the semantics captured and represented by ontology languages is essentially different from our ORA-semantics. Ontology model formed by ontology languages are usually proposed to construct and represent knowledge base in a particular domain, while XML documents are domain independent and are frequently used for data information exchange. Furthermore, ontology models at most can only capture and represent binary relationship type, and cannot capture the relationship attribute.

We also review different research works about discovering semantics in both relational database and XML database.

For relational database, DBRE (database reverse engineering) has been well studies for decades. Many approaches [2, 50] discover the object classes by the assumption that the input relational schemas are in 3NF so that each relation corresponds to a unique object class. Relationship type between object classes are discovered by the primary key/foreign key constraints and functional/inclusion dependencies. As the above constraints are only for enforcing database integrity and cannot imply any semantic relationship between two sets of attributes, those approaches cannot guarantee discovering the semantics information with high accuracy, especially when the dataset is small. Other approaches [36, 37] have also been proposed for discovering semantic dependencies to specify the semantic relationships underlies the data.

Although above DBRE approaches claims that they can discover most of the important semantic information underlies the data, such as object class, OID, relationship types, relationship attribute, etc., user supports/interactions for the necessary information is still needed.

On the other hand, some research works [41, 74, 16] have also been done on XML database to discovery semantic information. However, these approaches mainly focus on discovery the object classes in schema level and object instances in data level. Furthermore, they can only identify the binary relationship types according the hierarchical structure of the XML schema and XML data. The discovery of many other semantic concepts such as ternary/n-nary relationship, relationship attribute, dependent object class as well as IDD relationship type are ignored by these approaches.

For more details, after we introduce our rule-based semantics discovery approach for XML which will be introduced in Chapter 4, we will compare it with the above semantics discovery approaches in Section 5.3.

CHAPTER 4

DISCOVER SEMANTICS FROM XML

4.1 Introduction

In this chapter, we propose an automatic rule-based approach for discovering ORA-semantics from XML data and XML schema. 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 rule-based approach conform to the design of the corresponding ORA-SS, and the heuristics are summarized based on the characteristics and our observations of different ORA-semantic concepts.

The inputs of our automatic rule-based approach for semantics discovery are the XML schema and XML data. For the XML schema, what we need are the hierarchical structural information and the tag name information in the XML schema. However, the corresponding XML schema is not always available with each XML data. To resolve this issue, XML schema summarization/extraction has

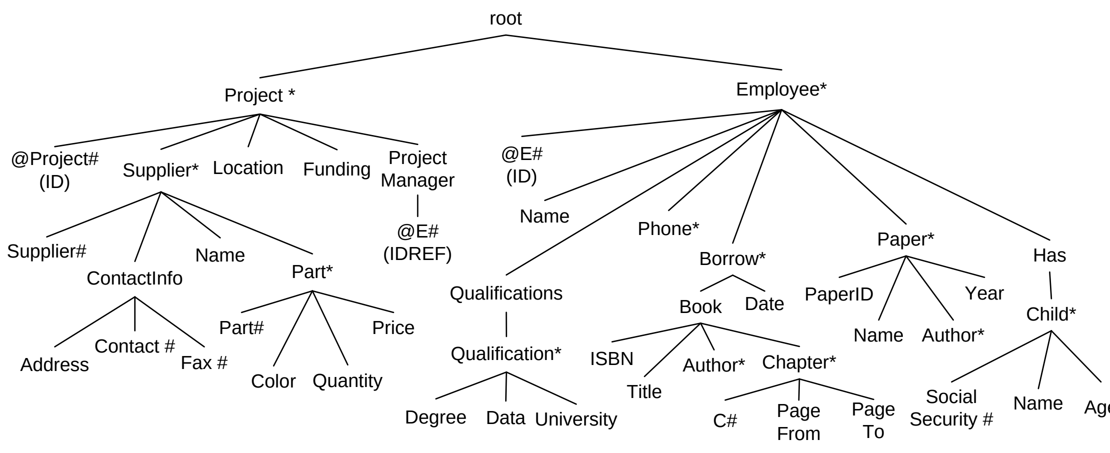


Figure 4.1: An XML schema tree as our running example

been studied in [72, 29, 19, 43], in case an XML schema is not available alongside the corresponding XML data. For the XML data, what we need are values of different element/attribute nodes and meaningful functional/multi-valued dependencies (FDs/MVDs) being imposed in the data.

The FD and MVD in XML data, defined in [34], are not exactly the same as the FD/MVD in relational data. In XML data, a FD/MVD is valid only under a header path, which is a path expression starting at the root and ending at the common ancestor node of all nodes involved in the FD/MVD, which means all nodes involved in the FD/MVD share the same prefix, which is the header path. To discover FDs/MVDs in XML data, many approaches have been proposed [54, 71, 73, 23]. For simplicity, in the rest of this thesis, given a FD/MVD in an XML schema tree, we use the XPath of the lowest common ancestor node of all nodes involved in the FD/MVD as its header path, and do not show it explicitly.

Example 4.1: For example, in Fig. 4.1[1], given a header path */root/Project/Supplier*, if we have the following FD:

- $\{Supplier\#\} \rightarrow \{Name\}$;

[1]For consistency and ease of reference, this figure duplicates the XML schema tree in Fig. 2.1

It means among all the *Supplier* nodes with the XPath */root/Project/Supplier* in the corresponding XML data, the value of *Supplier#* node can uniquely determine the values of *Name* node, which are both descendant nodes of the same *Supplier* node. However, for other nodes whose XPaths do not have */root/Project/Supplier* as their prefixes will not be considered, such as the *Name* under *Employee*.

Given a FD/MVD involving a set of nodes such as *Supplier#*, *Part#* and *Price*, we can infer the header path is the XPath of their lowest common ancestor in the XML schema tree (i.e., */root/Project/Supplier*). □

However, as FDs/MVDs are only constraints imposed in the data, and they do not carry semantic information with them, many meaningless FDs/MVDs (such as the *Contact#* node may be able to functionally determine the *Address* node in Fig. 4.1) will be discovered from the XML data, especially when the XML data is small. (For more details about semantic dependency, please refer to [38].) Because only those meaningful FDs/MVDs are useful to our rule-based approach, our rule-based approach uses the feedbacks and verifications from users/designers to prune out those meaningless FDs/MVs. Furthermore, based on the structural features of the nodes in FDs/MVDs, we also propose different heuristics from our observations and statistic information to rank different FDs/MVs and utilize the FDs/MVDs with the highest rank. More details about how we utilize FDs/MVDs in our rule-based approach will be discussed in Section 4.4.1.

The general process of our rule-based approach is shown in Fig. 4.2. As mentioned before, our rule-based approach can be separated into four steps:

1. **Pre-processing** It is a one-time effort, and we summarize the properties and heuristics for each ORA-semantic concept in this step.

2. **Internal node classification** We use bottom-up approach to classify all in-

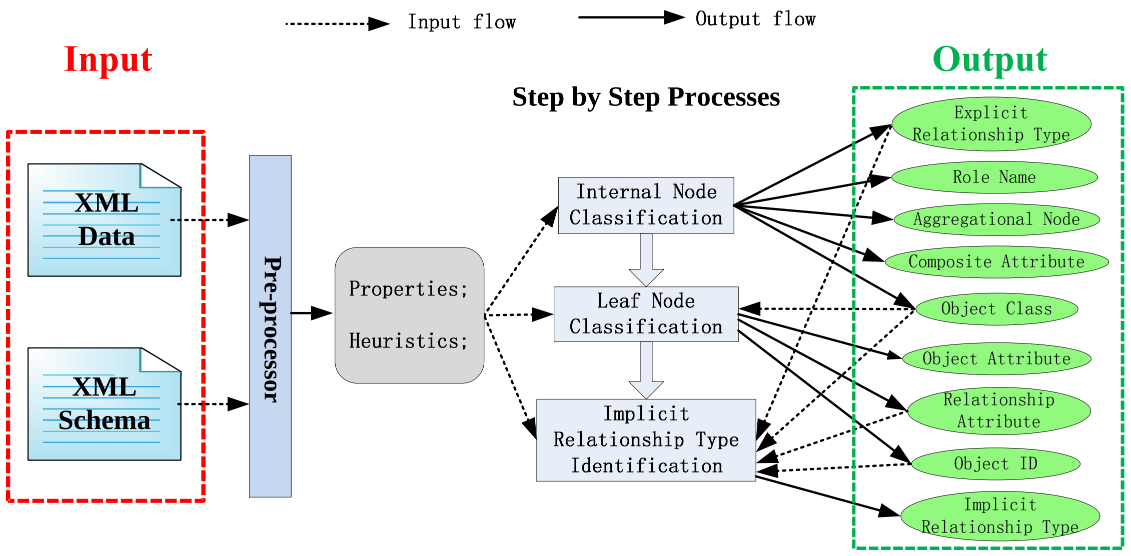


Figure 4.2: General process of our automatic rule-based semantics discovery approach

ternal nodes (i.e., from the lowest level internal nodes to their parent/ancestor nodes which are internal nodes) in the XML schema tree into one of the following categories of ORA-semantic concepts: object class, role name, composite attribute, aggregational node and explicit relationship type;

3. **Leaf node classification** We use a top-down approach to identify OID for each object class (i.e. identifying the OID for highest level object classes first, and then their child/descendant nodes which are object classes), and then distinguish between object attributes and relationship attributes using the identified OIDs;

4. **Implicit relationship type identification** We identify implicit relationship types, which are not represented in XML schema with those ORA-semantic concepts being discovered in internal node classification and leaf node classification.

4.2 Pre-processing

4.2.1 Properties (Necessary Conditions) of ORA-semantic Concepts

In this section, we extract and summarize the properties for each ORA-semantic concept from its XML schema and XML data, and these properties contains hierarchical structures of how different ORA-semantic concepts are designed in their XML schema tree and constraints imposed on them by the XML data. Properties of an ORA-semantic concept are also necessary conditions of it, which means given an identified ORA-semantic concept, it must satisfy its properties. Recall the semantic model ORA-SS, which is proposed and designed to capture and represent the semantic concepts in XML database. As discussed in Chapter 2, most of the ORA-semantic concepts can also be captured by ORA-SS. In consequence, the properties of those ORA-semantic concepts which can be captured in ORA-SS, are also satisfied in ORA-SS.

Example 4.2: For ORA-semantic concept object class, it has a property that *'It is an internal node in its XML schema tree'*, because as discussed in Chapter 2, object class in our ORA-semantics is a real world entity or concept with at least two object attributes as its child nodes or descendant nodes to store and describe its information, such as the object class *Part* is an internal node in Fig. 4.1. This property is also a necessary condition of object class, which means each object class must be an internal node in its XML schema tree. Furthermore, this property also conforms to the design of object class in ORA-SS model. □

4.2.2 Sufficient Conditions of ORA-semantic Concepts

Besides the properties (necessary conditions) of each ORA-semantic concept, we also propose the sufficient conditions for ORA-semantic concepts, which means by satisfying this sufficient condition, we can surely identify an input element as a particular ORA-semantic concept.

Example 4.3: For ORA-semantic concept object class, it has a sufficient condition that *'It has an ID attribute specified in its XML schema (XML DTD in particular) as its child node in its XML schema tree.'*, because ID attribute in DTD is designed to specify the identifier of its parent node, and we can identify this parent node as an object class by having an ID attribute as its child node, such as the internal node *Project* which has an ID attribute *Project#* as its child node in Fig. 4.1 can be directly identified as an object class. □

With the corresponding sufficient conditions of some ORA-semantic concepts, they can be easily identified from their XML schema, but not all ORA-semantic concepts have sufficient conditions. Furthermore, even we have sufficient condition for a certain ORA-semantic concept, we cannot guarantee that all occurrences of this ORA-semantic concept can be identified, because it is not a sufficient and necessary condition, e.g., we can use the sufficient condition in Example 4.2.2 to identify some object classes from their XML schema, but not all of them.

Based on the logic, we can easily identify that an input element must not be a particular ORA-semantic concept by violating at least one property (necessary condition) of that ORA-semantic concept.

4.2.3 Heuristics of ORA-semantic Concepts

In this section, we will also propose some heuristics related to different ORA-semantic concepts. All of these heuristics are summarized and proposed based on the characteristics (such as linguistic feature, etc.) and our observations of these ORA-semantic concepts from the datasets in the real world. 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 techniques and statistic information.

Example 4.4: For ORA-semantic concept object identifier (OID), it has a heuristic that *'It only contains one element/attribute in its XML schema tree.'*, which means the OID is a single attribute rather than a combined attribute, such as the *Project#* in Fig. 4.1 which is the OID of object class *Project*. Although this heuristic cannot guarantee 100% correctness, it is correct for OIDs of most of the object classes. It is because OIDs are designed to uniquely identify any instance of object classes, and when the designer designs the OID for an object class, most of the time he/she will design it as a single attribute for easily access, processing and management. □

4.2.4 ORA-semantic Concept: Internal Node vs. Leaf Node

In Table 4.1[2] and Table 4.2, for each ORA-semantic concept considered in our rule-based approach, we list its properties (necessary conditions), sufficient conditions (if any), related heuristics (if any) and example instances. Some properties may be shared by more than one ORA-semantic concept. All properties/sufficient conditions/heuristics related to the structure mean the structure in the corresponding

[2]The EDLN inside the table means Exclusive Descendant Leaf Node, which will be formally defined in Section 4.3.

Table 4.1: Properties (necessary conditions) of different ORA-semantic concepts

ORA-semantics	Properties (Necessary Conditions)	Examples
Object Class	**P_O1)** It is an internal node in its XML schema tree;	*Supplier*
	P_O2) It has more than one child node in its XML schema tree;	*Employee*
	P_O3) It has at least one FD/MVD among its EDLNs;	*Part*
	P_O4) Not all nodes in the LHS of each of its FDs/MVDs are IDREF attribute or role name;	*Book* *Paper*
Role Name	**P_R1)** It is an internal node in its XML schema tree;	*Landlord*
	P_R2) It has only one child node in its XML schema tree;	*Tenant*
	P_R3) Its only child node is not a repeatable node in its XML schema tree;	
	P_R4) Its only child node is an IDREF(S) attribute in its XML schema tree;	
Explicit Relationship Type	**P_E1)** It is an internal node in its XML schema tree;	*Borrow*
	P_E2) It has at least one object class, IDREF(S) attribute or role name as its descendant node in its XML schema tree;	*Has* *RentBy*
	P_E3) If it has at least one FD/MVD among its EDLN(s), then all nodes in the LHS of each of its FDs/MVDs are IDREF attributes or role names;	*Buy*
	P_E4) Its EDLN which is not an IDREF(S) attribute is a relationship attribute;	
Aggregational Node	**P_A1)** It is an internal node in its XML schema tree;	*Qualifications*
	P_A2) It has only one child node in its XML schema tree;	
	P_A3) Its only child node is a repeatable node in its XML schema tree;	
Composite Attribute	**P_C1)** It is an internal node in its XML schema tree;	*Qualification*
	P_C2) It has more than one child node in its XML schema tree;	
	P_C3) It does not have FD/MVD among its EDLNs;	
	P_C4) It does not has any object class, IDREF(S) attribute or role name as its descendant node in XML schema tree;	
OID of object class	**P_OID1)** It is a leaf node in its XML schema tree;	*Project#*
	P_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;	*Supplier#* *ISBN*
Object Attribute	**P_OA1)** It is a leaf node in its XML schema tree;	*Location*
	P_OA2) It can be functionally/ multi-valued determined by the OID of its lowest ancestor object class in its XML schema tree;	*Address* *Age*
	P_OA3) Its lowest ancestor object class is the object class it belongs to;	*Location*
Relationship Attribute	**P_RA1)** It is a leaf node in its XML schema tree;	*Quantity*
	P_RA2) It cannot be functionally/ multi-valued determined by the OID of its lowest ancestor object class in its XML schema tree;;	*Price*
	P_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;	
	P_RA4) It is an EDLN of an explicit relationship type or an EDLN of the lowest object class which involves in an implicit relationhsip type to which the relationship attribute belongs;	
IDREF(S) Attribute	**P_IDR1)** It is a leaf node in its XML schema tree;	*ProjectManager*
	P_IDR2) Its value range is a subset of the value range of the OID of the object class(es) which it references.	
Dependent Object Class	**P_W1)** It is an internal node in its XML schema tree;	*Chapter*
	P_W2) It has more than one child node in its XML schema tree;	
	P_W3) It does not has any object class, IDREF(s) attribute or role name as its descendant node in its XML schema tree;	
	P_W4) It has a child node which cannot functional/multi-valued determine other EDLN(s) of the dependent object class, but with the OID of its lowest ancestor object class, they can functional/ multi-valued determine all EDLN(s) of the weak object class;	

Table 4.2: Sufficient conditions and heuristics of different ORA-semantic concepts

	Sufficient Conditions	**Heuristics / Observations**	**Examples**
Object Class	**SC_1)** It has ID attribute / key element in its XML schema; (E.g. *Project*)	**H_O)** The number of the object classes without relationship attribute is more than the number of object classes with relationship attributes	*Supplier*
			Employee
			Part
Role Name		**H_R)** 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*
			Tenant
Explicit Relationship Type		**H_E1**) Its tag name can be a verb form.	*Borrow*
		H_E2) The number of binary relationship type is more than the number of ternary relationship type, and the number of ternary realtionship type is more than the number of 4-nary relationship type, and so on.	*Has*
			BoughtBy
			Buy
Aggregational Node		**H_A)** Its tag name is the plural form of the tag name of its only child node;	*Qualifications*
OID of object class	**SC_2)** It is specified as ID attribute in its XML schema; (E.g. *Project #*)	**H_OID1)** It is a single attribute;	*Project#*
		H_OID2) It contains the first child nodeof the corresponding object class;	*Supplier#*
			Part#
		H_OID3) It contains substring 'Identifier', 'Number', 'Key' or their abbreviations in its tag name;	*PaperID*
			E#
		H_OID4) It has numeric as (part of) its value, and the numerical part is in sequence;	*ISBN*
Relationship Attribute		**H_RA)** The number of relationship attribute of binary relationship type is more than the number of relationship attribute of ternary relationship type, and the number of relationship attribute of ternary realtionship type is more than the number of relationship attribute of 4-nary relationship type, and so on.	*Quantity*
			Price
IDREF(S) Attribute		**H_IDR)** Its tag name shares high linguistic similarity with or being a specialization of the tag name of the object class which it references, or the corresponding OID.	*ProjectManager*

XML schema tree, and the examples are from Fig. 4.1 and Fig. 4.4.

In an XML schema tree, each node must be either an internal node or a leaf node. Based on this property of each ORA-semantic concept (i.e. whether it is an internal node or a leaf node in its XML schema tree), all ORA-semantic concepts can be grouped into two groups:

- **Group 1**: It is an internal node in its XML schema tree, which includes object class, role name, composite attribute, aggregational node, explicit relationship type and dependent object class;

- **Group 2**: It is a leaf node in its XML schema tree, which includes OID, object attribute and relationship attribute.

We will identify these two groups of ORA-semantic concepts in Section 4.3 and Section 4.4 respectively. Furthermore, there is another ORA-semantic concept, implicit relationship type, which is not explicitly shown in the XML schema or XML schema tree, and we will identify it in Section 4.5.

4.3 Internal Node Classification

The inputs of this step are the internal nodes[3] in the XML schema tree, which can be easily obtained from any XML schema. Based on the design in ORA-SS model and ER model, we classify the internal nodes of 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.

Note that the dependent object class is also an internal node but cannot be identified in this step. This is because without knowing the OIDs of other object classes in XML schema, there is no way we can identify dependent object class. Furthermore, OIDs of object classes can only be identified after all object classes being identified in this step. More details about dependent object class will be given in Section 4.5.3.

In order to classify the internal nodes, we build a decision tree (shown in Fig. 4.3) using properties, sufficient conditions (if any) and related heuristics (if any) of each ORA-semantic concept listed in Table 4.1 and Table 4.2 as building blocks. In the decision tree, properties and sufficient conditions are represented by bold line rectangles; while related heuristics are represented by dotted line rectangles. As

[3]The root node will not be considered in the input.

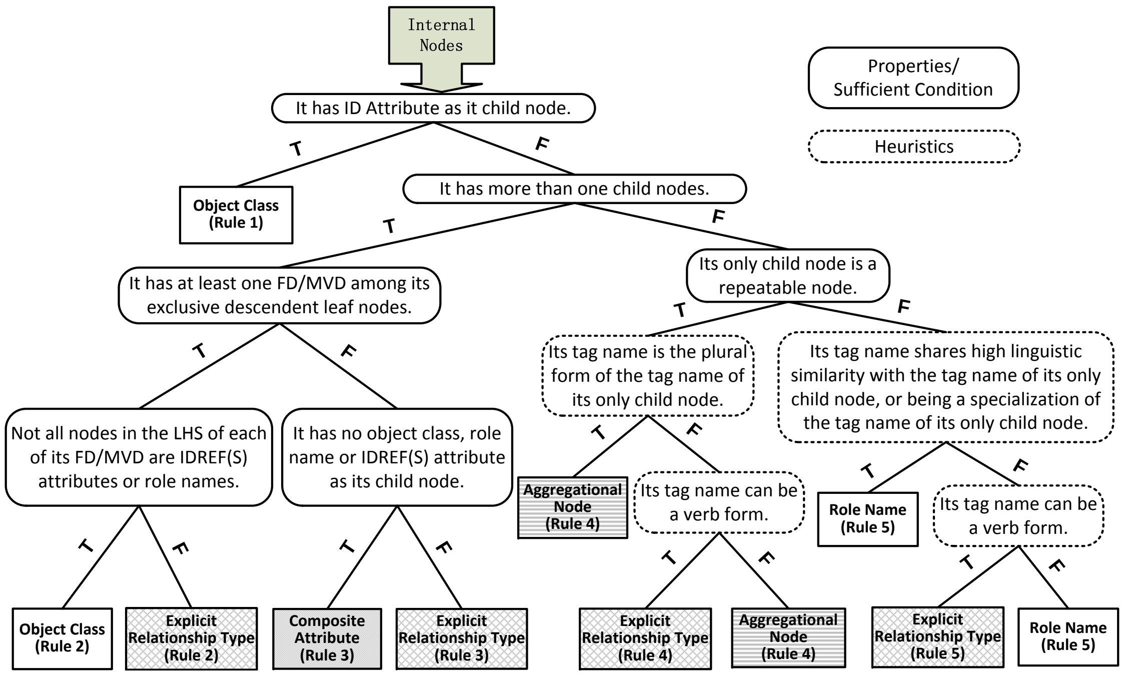

Internal Nodes
Properties/ Sufficient Condition
Heuristics
It has ID Attribute as it child node.
T
F
Object Class (Rule 1)
It has more than one child nodes.
T
F
It has at least one FD/MVD among its exclusive descendent leaf nodes.
Its only child node is a repeatable node.
T
F
Not all nodes in the LHS of each of its FD/MVD are IDREF(S) attributes or role names.
It has no object class, role name or IDREF(S) attribute as its child node.
Its tag name is the plural form of the tag name of its only child node.
Its tag name shares high linguistic similarity with the tag name of its only child node, or being a specialization of the tag name of its only child node.
Aggregational Node (Rule 4)
Its tag name can be a verb form.
Role Name (Rule 5)
Its tag name can be a verb form.
Object Class (Rule 2)
Explicit Relationship Type (Rule 2)
Composite Attribute (Rule 3)
Explicit Relationship Type (Rule 3)
Explicit Relationship Type (Rule 4)
Aggregational Node (Rule 4)
Explicit Relationship Type (Rule 5)
Role Name (Rule 5)

Figure 4.3: Decision Tree for Internal Node Classification

discussed before, the sufficient condition can be directly used as a classification rule for identifying the corresponding ORA-semantic concept. Thus, in our decision tree, we put the sufficient condition in the first level. The differences between properties and heuristics are:

1. Properties are necessary conditions for the corresponding ORA-semantic concepts, which can be used to filter out incorrect categories of ORA-semantic concept for each input internal node. For heuristics, they are just the characteristics of the corresponding ORA-semantic concepts or some naming convention summarized from observations, which can only be used to increase the probability of an internal node being correctly classified as an ORA-semantic concept when the properties are not enough to distinguish it from others.

2. Properties/sufficient conditions should be correct for the corresponding ORA-semantic concepts, while heuristics are not guaranteed to be 100% correct.

The pseudo code of our internal node classification is given in Algorithm 1. We use bottom-up approach (i.e., we classify lowest level internal nodes first then their parent internal nodes till the root node.) in our algorithm so that the category of an internal node can be used to help identifying the category of its parent internal node. Furthermore, the decision tree also shows that our rule-based approach for internal node classification is complete, which means all internal nodes in an XML schema tree can be classified into one of the above five categories. For ease of description, we represent and explain the decision tree using the following classification 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 an XML schema tree, except the root node;
       XML data;
Output: Identified object classes $\mathbb{O}$;
        Identified role name $\mathbb{R}$;
        Identified composite attributes $\mathbb{C}$;
        Identified aggregational nodes $\mathbb{A}$;
        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.
```

4.3.1 Object Class & OID

Rule 1. *[**Object Class**] Given an XML schema tree, if an internal node has an ID attribute[4] specified in its XML schema as its child node, then this internal node is classified as an object class, and the ID attribute is the OID of the object class.*

Rule 1 is a sufficient condition for object class, which means each internal node having an ID attribute specified in its corresponding XML schema as its child node

[4]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 simplicity, Rule 1 is illustrated using ID attribute, but key element also applies.

must be an object class. ID attribute in DTD is used to specify a unique identifier for an element. However, some element may not or cannot have ID attribute in the corresponding XML schema because of the limitation of XML schema language. The definitions of some XML schema languages (such as DTD) only allow single element to be specified as ID attribute. However, some designers may want to design an object class with composite OID. Furthermore, as discussed in Chapter 1, the value of an ID attribute is required to be unique for the corresponding object class in the whole data, which makes it impossible for some object classes to have ID attribute being specified in their XML schemas.

Example 4.5: In the XML schema tree shown in Fig. 4.1, internal node *Project* is classified as an object class because its child node *Project#* is specified as an ID attribute. *Project#* becomes the OID of object class *Project*. However, internal nodes *Supplier* and *Part* cannot have ID attributes as their child nodes. 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 more than once in the same XML data for the same object class. □

Besides DTD, XSD (XML Schema) is another kind of XML schema language which is also frequently used in XML applications. As discussed in Chapter 1, 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 in 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 class 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 classification rules to classify the rest of the internal nodes, including those object classes without ID attribute or key element being specified in their XML schemas.

4.3.2 Object Class vs. Explicit Relationship Type

Before we introduce the other classification rules, let us introduce a concept named Exclusive Descendant Leaf Node (EDLN):

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 another object class* o*, such that* o *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*.

Example 4.6: In Fig. 4.1, given object classes *Project* and *Supplier*, the EDLNs of *Project* include: *Location* and *Funding*. The leaf node *Name* is not an EDLN of *Project*, because there is another object class *Supplier* between *Name* and *Project*.

□

In order to identify the object class without any ID attribute being designed as its child node, we use of the properties of object class. By comparing them to the properties of other ORA-semantic concepts in Table 4.1, we have following two conflict statements:

(I) An object class has more than one child node in its XML schema tree ([`P_02`] in Table 4.1[5]); while both aggregational nodes and role names have only one child node in their XML schema tree ([P_A2] and [P_R2]).

(II) An object class has at least one FD/MVD among its EDLNs ([P_O3)]; while composite attributes do not have any FD/MVD among its EDLNs ([P_C3]).

With (I) and (II), we can distinguish object class from role name, aggregational node and composite attribute. However, we still cannot uniquely identify an object class, because some explicit relationship types also satisfy the properties of object class mentioned in (I) and (II).

Example 4.7:

In Fig. 4.1, internal node *Part* has 4 child nodes *Part#*, *Color*, *Quantity* and *Price*. Assume there is a FD under internal node *Part*: (Recall that we use the XPath of the lowest common ancestor node of all nodes involved in the FD/MVD as its header path, and do not show it explicitly)

- $\{Part\#\} \rightarrow \{Color\}$;

From the above information, we know internal node *Part* can never be an aggregational node, a role name or a composite attribute. However, in Fig. 4.4, internal node *Buy* also has more than one child node *C_ID*, *P_ID* as well as *Price*, and it has FD:

- $\{C\_ID, P\_ID\} \rightarrow \{Price\}$.

With above information, we still cannot identity *Part* as an object class and *Buy* as an explicit relationship type, if we do not know their semantic meanings. □

In order to distinguish object class and explicit relationship type, we note that the difference between object class and explicit relationship type is:

[5]For simplicity, we use property labels such as [P_O2] to indicate the corresponding properties in Table 4.1 in the rest of this thesis.

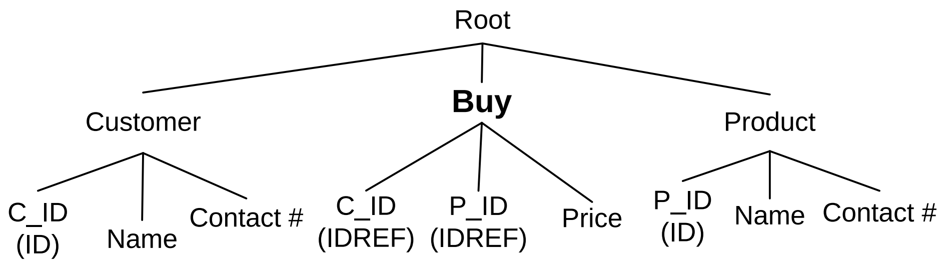


Figure 4.4: Explicit relationship type with FD among its child nodes

(III) Not all nodes in the left-hand-side (LHS) of each FD/MVD of an object class are IDREF attribute or role name ([P_O4)]; while all nodes in the LHS of each FD/MVD of an explicit relationship type are IDREF attributes or role names ([P_E3]).

This is because the FD/MVD among EDLNs of an explicit relationship type must involve the relationship attribute, which is functionally/multi-valued determined by the OIDs of all participating 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 the FD/MVD among EDLNs of an object class, its LHS should contain the OID of this object class, which should not be an IDREF attribute or role name.

Recall the two FDs in Example 4.7, the LHS of the FD under *Part* is neither an IDREF attribute nor a role name, while the LHS of the FD under *Buy* are both IDREF attribute. Based on these, we can finally identity *Part* as an object class and *Buy* as an explicit relationship type.

With the above analysis, we have the following classification rule for object class and explicit relationship type using (I), (II) and (III):

Rule 2. *[**Object Class** vs. **Explicit Relationship**] Given an XML schema tree, let* i *be an internal node with more than one child node and there is at least one FD or MVD among its EDLNs; if not all left-hand-side (LHS) nodes of those FDs* *or MVDs are IDREF(S) attributes or role names, then* i *is classified as an object class, else* i *is classified as an explicit relationship type.*

4.3.3 Composite Attribute vs. Explicit Relationship Type

Besides object class and explicit relationship type, composite attribute is another ORA-semantic concept which should have more than one child node in its XML schema tree. By definition, composite attribute is a special kind of attribute which combines more than one attribute, and it should not have any FD/MVD among its exclusive descendant leaf nodes in its XML schema tree according to its property `P_C3`. In consequence, we can distinguish a composite attribute with other ORA-semantic concepts with the following conflict statement:

(IV) A composite attribute has more than one child node in its XML schema tree ([`P_C2`]); while both aggregational nodes and role names have only one child node in their XML schema tree ([`P_A2`] and [`P_R2`]).

Recall the (II) in Section 4.3.2, which can be used to distinguish composite attribute and object class by checking whether they have any FD/MVD among their EDLNs. Thus, with (II) and (IV) we can distinguish composite attribute from object class, role name and aggregational node.

Because of the flexible structure of explicit relationship type, some explicit relationship types also satisfy the properties of composite attribute in (II) and (IV). We distinguish composite attributes with explicit relationship types with the following conflict statement:

(V) A composite attribute should not has any object class, IDREF(S) attribute or role name as its descendant node in its XML schema tree ([`P_C4`]); while an explicit relationship type should has at least one object class, IDREF(S) attribute or role name as its descendant node in its XML schema tree ([`P_E2`]).

A composite attribute is still an attribute, and an attribute should not have any object class, IDREF(S) attribute or role name as its descendant node. On the other hand, an explicit relationship type needs object class, IDREF(S) attribute or role name as its descendant node to represent the object classes participating in it.

Example 4.8: In Fig. 4.1, the internal node *Qualification* has 3 child nodes *Degree*, *Data* and *University*, and assume there is no FD/MVD among these nodes under *Qualification*. From these information, we know that *Qualification* cannot be an aggregational node, a role name or an object class. Although internal node *Borrow* in Fig. 4.1 also has 2 child nodes *Book* and *Date*, and no FD/MVD among them. We can identify *Qualification* as a composite attribute by all its child nodes are not object class, IDREF(S) attribute or role name, while *Borrow* has an object class *Book* as its child node. (Recall we uses bottom-up approach to classify internal nodes, so that when we classify the internal node *Borrow*, its child node *Book* has already been classified as an object class.) □

Based on the above analysis, we construct the following classification rule for composite attribute and explicit relationship type using (II), (IV) and (V):

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

4.3.4 Aggregational Node vs. Explicit Relationship Type

Recall the (I) in Section 4.3.2 and (IV) in Section 4.3.3, they can be used to distinguish aggregational node with object class and composite attribute respectively. Based on the properties of aggregational node, we can use the following conflict statement to distinguish aggregational node with role name:

(VI) The only child node of an aggregational node is a repeatable node[6] ([P_A3]); while the only child node of a role name is not a repeatable node ([P_R3]).

However, with all properties of aggregational node, we still cannot distinguish between aggregational node and explicit relationship type. In consequence, we need related heuristics in Table 4.2 to help to distinguish between them.

By definition of aggregational node in Chapter 2, it is a structural node for aggregating its repeatable child nodes without extra semantics besides the semantics of its child nodes. It only appears in some XML document and it is not common in relational database and ER diagram. Based on this characteristic of aggregational node, it is reasonable to have the following heuristic [H_A] for aggregational node:

[H_A] *Its tag name should be the plural form[7] of the tag name of its child node.*

On the other hand, based on our observations, relationship types are usually designed with the purpose of representing certain action between more than one object classes, especially for explicit relationship type. In consequence, it is reasonable to have the following heuristic [H_E1] for explicit relationship type:

[H_E1] *Its tag name can be a verb form.*[8]

[6]Recall that repeatable node is the node which can occur multiple times with the same XPath in the corresponding XML data.

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

[8]We cannot say *'its tag name is a verb form'*, because many words can be both verb form and noun form, such as 'book'.

Example 4.9: In Fig. 4.1, both internal nodes *Qualifications* and *Has* has only one child node and both of the child nodes are repeatable nodes, which make them cannot be object classes, composite attributes or role names. However, the tag name of *Qualifications* is the plural form of the tag name of its only child node *Qualification*, while the tag name of *Has* is a verb form. With above information, we can identify *Qualifications* as an aggregational node, which aggregates its child node *Qualification*, and *Has* is an explicit relationship type. □

In order to determine whether we should test [H_A] or [H_E1] first in our classification rule/decision tree for classifying aggregational node and explicit relationship type, we adopt the measurement of *splitting-attribute selection* in ID3 [52] (i.e. choosing the splitting-attribute (i.e., [H_A] or [H_E1]) which has higher information gain). We collect a training data with 125 internal nodes from 18 XML schema trees with their ORA-semantic concepts being correctly specified by users. Results show that [H_A] returns higher information gain than [H_E1]. Thus, we put [H_A] as condition in a higher level than [H_E1] as condition in our decision tree as shown in Fig. 4.3, and test [H_A] first in our classification rule for aggregational node and explicit relationship type.

It is possible that there are some internal nodes which satisfy the properties of aggregational node in (I), (IV) and (IV), but satisfy neither [H_A] nor [H_E1] (i.e., An internal node having only one child node, and the only child node is a repeatable node, and its tag name is not the plural form of the tag name of its child node, and its tag name cannot be a verb form.). In this case, we classify them as aggregational node because among the training data which satisfy the above conditions, there are 66.7% of aggregational node and 33.3% of explicit relationship type.

Note that by considering heuristics of different ORA-semantic concepts for classifying internal nodes, our approach cannot guarantee to be 100% correct. In order to increase the accuracy of our classifications, if user feedback or user verification is available, we will ask user for verification of these internal nodes.

With the above analysis, we have the following classification rule for aggregational node and explicit relationship type using (I), (IV), (IV), [`H_A`] and [`H_E1`]:

Rule 4. *[**Aggregational Node** vs. **Explicit Relationship Type**] Given an XML schema tree, let* i *be an internal node, which has only one child node* c *and* c *is a repeatable node; if* i *is not the plural form of the tag name of* c*, and the tag name of* i *can be a verb form, then* i *is classified as an explicit relationship type, else* i *is classified as an aggregational node.*

4.3.5 Role Name vs. Explicit Relationship Type

Another kind of internal node we have not considered is role name. Similar to the others, we compare the properties of role name to the properties of other ORA-semantic concepts in Table 4.1. Recall the (I) in Section 4.3.2, (IV) in Section 4.3.3 and (VI) in Section 4.3.4, they can be used to distinguish role name with object class, composite attribute and aggregational node respectively.

However, the properties of role name listed in Table 4.1 are not enough to distinguish it from explicit relationship type, and we need to use related heuristics in Table 4.2 to distinguish between them.

For role name, 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 role name to have the following heuristic:

[H_R] *Tag name of a role name and tag name of the corresponding object class which the role name references share high linguistic similarity, or tag name of the role name is a specialization of tag name of the corresponding object class.*

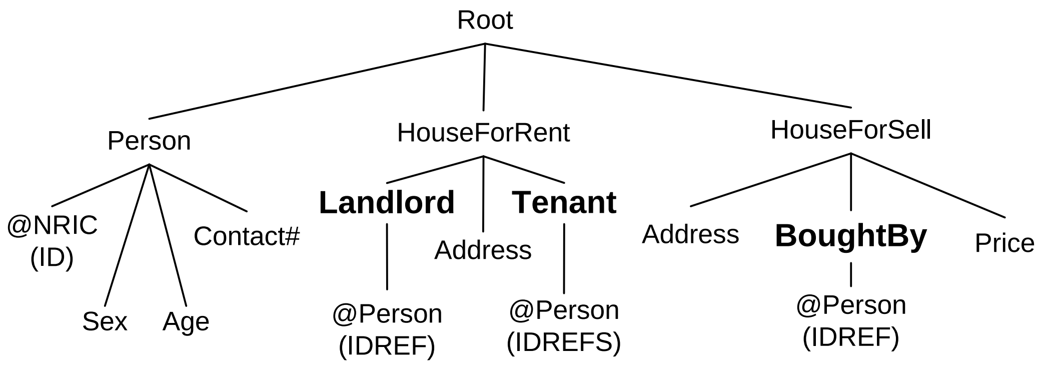


Figure 4.5: Role names and explicit relationship type with IDREF(S) as child nodes

Example 4.10: In Fig. 4.5, object class *Person* has an ID attribute *NRIC* being designed as its child node. *NRIC* is referenced by two IDREF attributes and one IDREFS attribute with the same tag name *Person*, which are child nodes of internal nodes *Landlord*, *Tenant* and *BoughtBy* respectively. Both *Landlord*, *Tenant* and *BoughtBy* have only one child node, and their child nodes are both not repeatable nodes. Because object class and composite attribute require having more than one child node and aggregational node requires its child node to be a repeatable node, both *Landlord*, *Tenant* and *BoughtBy* can only be role names or explicit relationship types.

Note that both words 'Landlord' and 'Tenant' are specialization of the word 'Person', while 'BoughtBy' contains a verb 'Bought'. With the heuristic [`H_R`] of role name and heuristic [`H_E1`] of explicit relationship type, we can identify *Landlord* and *Tenant* as role names and *BoughtBy* as explicit relationship type. □

In order to compare the similarity between two tag names, some researches [47, 65, 60] have already been done on comparing linguistic similarity between different words, such as using the information from WordNet[9] for linguistic and semantic comparisons. We can also check whether a given word can be a verb form from WordNet. In our rule-based approach, for simplicity, we calculate the edit

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

distance between two words for their linguistic similarity.

In order to identifying generalization and specialization, we maintain a database including pairs of general and specialized concepts such as ‘person’ and ‘customer’, ‘employee’ and ‘manager’, etc. This database will be incrementally updated with user feedbacks and verifications.

Similar to the process in Section 4.3.4, we use the same training data to calculate the information gain of [`H_R`] and [`H_E1`] respectively. Based on the results, we determine to put [`H_R`] as condition in a higher level than [`H_E1`] as condition in our decision tree as shown in Fig. 4.3, and test [`H_R`] first in our classification rule for role name and explicit relationship type.

It is also possible that there are some internal nodes which satisfy the properties of role name in (I), (IV), (VI), but satisfy neither [*H_R*] nor [*H_E1*]. In this case, we classify them as role name because among the training data which satisfy the above conditions, there are all role names.

In order to increase the accuracy of our classification, if user feedback or user verification is available, we will ask user for verification for these internal nodes.

With the above analysis, we have the following classification rule for role name and explicit relationship type using (I), (IV), (VI), [*H_R*] and [*H_E1*]:

Rule 5. *[**Role Name** vs. **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; if* i *does not share high linguistic similarity with the tag name of* c *or not being a specialization of the tag name of* c*, and the tag name of* i *can be a verb form, then* i *is identified as an explicit relationship type, else* i *is identified as a role name.*

4.4 Leaf Node Classification

4.4.1 OID Discovery

After processing internal nodes in the previous step, our next step is to identify OID for 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 to identify the ID attribute as OID of the corresponding object class. In this section, we consider the case that the single-attributed OID is not specified in XML schema (e.g., *ISBN* of object class *Book* in Fig. 4.1), or the OID contains multiple attributes.

In an XML schema tree, the attributes under an object class may be its object attributes or attributes of some relationship types in which it participates. Based on the definition of OID, only its object attributes can be functionally/multi-valued determined by its OID, while relationship attributes cannot. Our rule-based approach identifies OID for each identified object class based on this property.

Super OID

Before we explain how we identify OIDs, we first introduce another concept named *Super OID*, which can be used to discovered OID of an object class.

Concept 3. ***Super OID*** *A super OID of an object class o is a minimal set of nodes which contains a subset of the exclusive descendant leaf nodes (EDLN) of o and the OIDs of some ancestor object classes of o, such that this super OID functionally/multi-valued determines all EDLNs of o.*

Recall that 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.

Example 4.11: Assume in our internal node classification step, internal nodes *Part* in Fig. 4.1 has been classified as an object class, with its EDLN as {*Part#*, *Color*, *Quantity*, *Price*}. *Project* and *Supplier* are also classified as object classes with their OIDs as *Project#* and *Supplier#*. In the following we list the full FDs related to the EDLNs of *Part*:

- {*Part#*} → {*Color*};
- {*Supplier#, Part#*} → {*Price*};
- {*Project#, Supplier#, Part#*} → {*Quantity*};

With above information, we know {*Project#*, *Supplier#*, *Part#*}, {*Supplier#*, *Part#*, *Quantity#*} and {*Part#*, *Price#*, *Quantity#*} are both super OID of object class *Part*, because they both can functionally determine all EDLNs of *Part*. □

In an XML schema tree, given an object class o, its 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. [`P_OID2`], [`P_OA2`], [`P_RA2`] and [`P_RA3`] in Table. 4.1), a super OID of o should be able to 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.

Discover Candidate OIDs with Super OID

The rationale of our rule-based approach to identify OIDs is that given an object class o, there is an attribute set S formed by the OID of o and the OIDs of some of the ancestor object classes of o, and S 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, and the attribute set S should be a super OID of o, which is also a superset of the OID of o. Thus, this shows us a way to identify the OID of an object class through its super OIDs. Note that for an identified object class, its super OID may not be unique (as is shown in Example 4.11). By excluding all OID(s) of the ancestor object class(es) of an input object class from each of its super OID, and what is left should be one of its candidate OIDs.

Based on the above analysis, we proposed a top-down approach (shown in Algorithm 2) to identify candidate OIDs of each identified object class without ID attribute being specified in its XML schema. The reason why we use top-down approach is the OID of an ancestor object class is needed for identifying the OIDs of its descendant object classes.

Given an object class o, we create a set $SupEDLN_o$, which is a superset of its EDLNs, denoted as $EDLN(o)$. $SupEDLN_o$ also includes OIDs of all its ancestor object classes. In $SupEDLN_o$, we identify all super OIDs of o, which are minimal subsets of $SupEDLN_o$ that functionally/multi-valued determine all elements in $EDLN(o)$. As there may be more than one minimal subset of $SupEDLN_o$ that satisfies the above condition, there may be more than one super OID of o being generated by our Algorithm 2, and more than one candidate OID being returned.

Example 4.12: Here we continue our example in Example 4.11, and show how Algorithm 2 discovers the candidate OIDs for three identified object classes *Project*, *Supplier* and *Part* in Fig. 4.1.

Object Class Project

We identify *Project#* as OID of object class *Project* by Rule 1, because it is specified as an ID attribute of *Project*.

Algorithm 2: Candidate OID Discovery

```
Input: Identified object classes $\mathbb{O}$;
       Exclusive descendant leaf nodes $EDLN(o)$ for each identified object class $o \in \mathbb{O}$;
Output: Candidate OID $id_o$ for each identified object class $o \in \mathbb{O}$
foreach identified object class $o \in \mathbb{O}$ do
    $SupEDLN_o = EDLN(o)$;
    foreach $o_i \in \mathbb{O}$, which is ancestor object class of $o$ do
        $SupEDLN_o = SupEDLN_o \cup id_{oi}$;        //$id_{oi}$ is the OID of object class $o_i$
    foreach $SID_o \subset SupEDLN_o$ do
        if $\forall e \in EDLN(o)$, such that $SID_o \rightarrow e$ or $SID_o \twoheadrightarrow e$ then
            if $\nexists S \subset SID_o$, such that $\forall e \in EDLN(o)$, such that $S \rightarrow e$ or $S \twoheadrightarrow e$ then
                foreach $e \in SID_o$ do
                    if $e \in EDLN(o)$ then
                        $e \in id_o$;
                    return $id_o$ as a candidate OID of $o$.
```

Object Class Supplier

The EDLNs of object class *Supplier* are $\{$*Supplier#*, *Name*, *Address*, *Contact#*, *Fax#*$\}$. Assume we have the following full FDs:

- $\{Supplier\#\} \rightarrow \{Name\}$;
- $\{Supplier\#\} \rightarrow \{Address\}$;
- $\{Supplier\#\} \rightarrow \{Contact\#\}$;
- $\{Supplier\#\} \rightarrow \{Fax\#\}$;

As the single attribute *Supplier#* functionally determines all EDLNs of object class *Supplier*, we identify *Supplier#* as its OID, the same as its super OID.

Object Class Part

The EDLNs of object class *Part* are $\{$*Part#*, *Color*, *Quantity*, *Price*$\}$. Assume we have the following full FDs (same as the FDs in Example 4.11):

- $\{Part\#\} \rightarrow \{Color\}$;
- $\{Supplier\#,\ Part\#\} \rightarrow \{Price\}$;
- $\{Project\#,\ Supplier\#,\ Part\#\} \rightarrow \{Quantity\}$;

We combine the EDLNs of *Part* and OIDs of its ancestor object classes *Supplier* and *Project*, and discover the minimal subsets that functionally/multi-valued determine all its EDLNs to be its super OID, which are {Project#, Supplier#, Part#}, {Supplier#, Part#, Quantity} and {Part#, Quantity, Price}. In consequence, we get {Part#}, {Part#, Quantity} and {Part#, Quantity, Price} as candidate OIDs of object class *Part* by deleting those OIDs of ancestor object classes of *Part* from those super OIDs.

□

Determine OID from Candidate OIDs with Heuristics

As is shown in Example 4.12, Algorithm 2 may return more than one candidate OID. However, if we do not consider the linguistic meanings of those candidate OIDs from their tag names, we cannot determine which candidate OID is better. Different candidate OIDs show us different ORA-semantics.

Example 4.13: In this example, we consider those 3 candidate OIDs for object class *Part* mentioned in Example 4.12. In order to avoid the influence of linguistic meanings of those candidate OIDs from their tag names, we replace all their tag names with meaningless characters.

For object class O, Algorithm 2 returns 3 candidate OIDs for it, $\{A, B, C\}$, $\{A, B\}$ and $\{A\}$. If $\{A, B, C\}$ is the correct OID for object class O, it means object class O does not have any relationship attribute; If $\{A, B\}$ is the correct OID for object class O, it means object class O has one relationship attribute C; If $\{A\}$ is the correct OID for object class O, it means object class O has two relationship attributes B and C. □

In consequence, we use some related heuristics (shown in Table 4.2) summarized from our observations to help to identify the best OID from all candidate OIDs returned by Algorithm 2.

Observation 1. ***[OID]*** *In XML schema, 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', etc., 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 2. ***[Relationship Attribute]*** *In XML schema, we have the following heuristics/observations about relationship attributes:*
(1) the number of the object classes without relationship attribute is more than the number of object classes with relationship attribute;
(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.

Observation 1 is summarized and proposed based on our observation of the structural and linguistic characteristics of OIDs designed in the real world; while Observation 2 is summarized and proposed based on our observation of the statistic information of relationship types and relationship attributes. The reason for Observation 2 is in the real world, the probability of a relationship type being designed is less and less with more and more object classes participating in it. Furthermore, there are also a lot of relationship types being designed in the XML schema but without any corresponding relationship attribute. Thus, given an identified object class, the chance of having its super OID, being the same as its OID is higher than containing the OID of only one its ancestor object class, and the case of containing OID of only one its ancestor object class is higher than the case of containing OIDs of two or more its ancestor object classes.

Given an identified object class, we use a ranking model to rank all its candidate

OIDs. The ranking score is based on statistic data about the features of how the OID of an object class should look like in term of those characters mentioned in Observation 1 and Observation 2.

We collect 122 object classes, each of which has more than one candidate OIDs. We ask users to manually specify their OIDs from all their candidate OIDs, and uses all these candidate OIDs as our training data. We extract the statistic information covered in Observation 1 and Observation 2. Using such statistic information, we adopt the Bayes' Theorem to calculate the posterior probability of each kind of candidate OID to be the correct OID of its corresponding object class, with different degree of satisfactions of the heuristics in Observation 1 and Observation 2. Then we rank all candidate OIDs for each object class based on their posterior probabilities, and choose the highest one as its OID. More detail of the Bayes' Theorem is given in [25].

Example 4.14: Recall the candidate OIDs for object class *Part* discovered in Example 4.12 are {Part#}, {Part#, Quantity} and {Part#, Quantity, Price}. {Part#} gets the highest ranking with our Bayes' ranking model. One main reason of why {Part#} gets the highest ranking is the statistic data conforms with our heuristic that OID with single attribute occurs more often than OID with multiple attributes. Thus, {*Part#*} is identified as the OID of object class *Part*. □

4.4.2 Object Attribute vs. Relationship Attribute

After identifying the OID for each identified object class, our next step is to classify all leaf nodes except those being identified as OIDs in XML schema trees to be object attributes or relationship attributes.

For an explicit relationship type (discovered in Section 4.3), we identify its EDLNs (exclusive descendant leaf nodes) as its relationship attributes, because relationship attributes are designed to store and describe information about relationship types.

For implicit relationship type, its relationship attributes should appear as an EDLNs of the lowest object class which participates in the relationship type, 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 the OID of the object class it belongs to, while relationship attribute should not, to differentiate them.

Rule 6. *[**Object Attribute** vs. **Relationship Attribute**] Given an object class* o *and its OID, if an EDLN* e *of* o *is functionally/ multi-valued determined by the OID of* o*, then* e *is identified as an object attribute of* o*. Otherwise,* e *is identified as a relationship attribute of certain relationship type which* o *participates in.*

Example 4.15: In Figure 4.1, 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 implicit relationship types that *Part* participates in. We will determine the implicit relationship types in next section. □

4.5 Implicit Relationship Type Discovery

Recall that explicit relationship type can be identified by Rule 2, 3, 4 and 5 in Section 4.3. However, there are some implicit relationship types which are not explicitly represented as any node in its XML schema tree. In this section, we are going to present our approach to identify the implicit relationship types which fall into any of following 3 categories:

1. Implicit relationship type with at least one relationship attribute;
2. Implicit relationship type with IDREF(S) attribute;
3. IDentifier Dependency (IDD) Relationship Type [22].

Note that except for the IDD relationship type, the other implicit relationship types discovered by our approach can be binary relationship type, ternary relationship type and n-nary relationship type.

4.5.1 Implicit Relationship Type with Relationship Attribute

For each relationship attribute discovered in Section 4.4.1 (except those EDLNs of explicit relationship type), there must be an implicit relationship type it belongs to. Recall that relationship attribute should be functionally/multi-valued determined by the OIDs of all object classes participating in the implicit relationship type, to which the relationship attribute belongs. Therefore, for each relationship attribute identified in previous step, we use a bottom-up approach, Algorithm 3, to identify the corresponding implicit relationship type with its degree, and all participating object classes.

The bottom-up strategy of our approach for discovering implicit relationship type with relationship attribute means we process the relationship attribute in the lowest level of an XML schema tree first. Furthermore, for a given relationship attribute, we also use bottom-up approach to test whether an object class participates in the relationship type to which the relationship attribute belongs, and then test its parent/ancestor object classes.

In Algorithm 3, 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 represented as $|\mathbb{C}|$, 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 the $SemID_{ra}$ functionally/multi-valued determines ra, we identify the implicit relationship type $r(\mathbb{C})$, to which ra belongs, and return the participating object classes as those object classes whose OIDs are in $SemID_{ra}$ and the degree of the implicit relationship type as the number of participating 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: Implicit relationship type $r(\mathbb{C})$ with its participating object classes $\mathbb{C}$ and degree $|\mathbb{C}|$, for each
        relationship attribute in $\mathbb{A}$
1 foreach 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 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 $SemID_{ra} \to ra$ or $SemID_{ra} \twoheadrightarrow ra$; then break;
9     return implicit relationship type $r(\mathbb{C})$ to which $ra$ belongs, with object classes in $\mathbb{C}$ as its
      participating object classes and $|\mathbb{C}|$ as its degree;
```

Example 4.16: Recall that in Example 4.12 and Example 4.15, we have 3 object classes *Project*, *Supplier* and *Part*, with their OIDs as *Project#*, *Supplier#* and *Part#* respectively in Fig. 4.1. We also discovery two relationship attributes *Price* and *Quantity* under object class *Part*, with following related full FDs:

1. $\{$*Supplier#, Part#*$\} \to \{$*Price*$\}$;
2. $\{$*Project#, Supplier#, Part#*$\} \to \{$*Quantity*$\}$;

Given above full FD (1), we identify that there is an implicit binary relationship type between object classes *Supplier* and *Part*, with relationship attribute *Price*.

For another relationship attribute *Quantity*, with above full FD (2), we identify an implicit ternary relationship type among object classes *Project*, *Supplier* and *Part*, with relationship attribute *Quantity*. □

4.5.2 Implicit relationship type with IDREF(S) Attribute

In XML schema, some designers may design an implicit relationship type by specifying an IDREF(S) attribute for an object class, which references the ID attribute of another object class. Based on this hint, if an object class has a child node being specified as an IDREF(S) attribute in XML schema, we identify there is an implicit relationship type between the object class having this IDREF(S) attribute and the object class having an ID attribute being referenced by the IDREF(S) attribute.

For some XML schema language (e.g., DTD), they do not specify which ID attribute an IDREF(S) attribute references. Thus, we propose a four-step method to discover the ID attribute which is referenced by an IDREF(S) attribute, and identify the corresponding implicit relationship type:

1. **Filter ID Attributes by Value Range**

   There is a necessary condition for IDREF(S) attribute and the ID attribute being referenced by it. That is the value range of an IDREF(S) attribute in its XML data must be a subset of the value range of the ID attribute it references. We use this necessary condition to filter out those ID attributes which is impossible to be referenced by a given IDREF(S) attribute.

2. **Identify Candidate ID Attribute by Tag Name Similarity**

   IDREF(S) attribute is designed to reference an object class with the same real world entity/concept, or reference a more general object class. Therefore, it is reasonable that an IDREF attribute satisfies any of the following conditions:

1. The tag name of an IDREF(S) attribute shares high linguistic similarity with the tag name of the ID attribute it referencing;
2. The tag name of an IDREF(S) attribute is a specialization of the tag name of the ID attribute it referencing;
3. The tag name of an IDREF(S) attribute shares high linguistic similarity with the tag name of the object class whose ID attribute it referencing;
4. The tag name of an IDREF(S) attribute is a specialization of the tag name of the object class whose ID attribute it referencing;

To compare the similarity between two tag names, or determine whether they are generalization/specialization of each other, we use the approach which we use to classify role name in Section 4.3.5 (i.e., using the lexical database, WordNet).

3. Determine ID attribute by User Verification

Our method cannot guarantee what we discover is 100% correct. Therefore, given an IDREF(S) attribute with XML data, we still need user to verified the ID attribute discovered by our approach.

4. Identify Corresponding Implicit Relationship Type

Given an IDREF(S) attribute with the corresponding ID attribute being discovered, we can identify an implicit relationship type between the object class having this IDREF(S) attribute and the object class having an ID attribute being referenced by the IDREF(S) attribute.

Example 4.17: Given two object classes *Department* and *Staff* with their ID attributes *Dept#* and *Staff#* respectively, there is an IDREF attribute under object class *Staff* with its tag name as *Dept#*. For the corresponding XML data, we assume the value range of IDREF attribute *Dept#* is a subset of the value range of the ID attribute *Dept#*. As the IDREF attribute shares the same tag name with the ID attribute of object class *Department*, we guest the IDREF attribute references the ID attribute of object class *Department*. Therefore, with user verification, we identify there is an implicit relationship type between *Department* and *Staff*; □

Note that if the designer does not specify the ID attribute and/or IDREF(S) attribute in the XML schema, we can still apply our four-step method to discover implicit relationship types. We use leaf nodes in XML schema tree as input, and each time we assume one of them as an IDREF attribute, and use our four-step method to identify the corresponding ID attribute. In this case, the user verification becomes more important. It is because without ID attribute and IDREF(S) attribute being specified in the XML schema, our method is heavily depend on the XML data, which may suffer from the limitations of small size and error data.

4.5.3 Identifier Dependency (IDD) Relationship Type

Recall that in Chapter 2 we introduce a special kind of relationship type called identifier dependency (IDD) relationship type, which means there is a dependent object class, which is a special kind of object class whose instance can only be identified together with an instance of its lowest ancestor object class or lowest ancestor dependent object class.

As discussed in 4.3, although dependent object classes are internal nodes in its XML schema tree, without knowing the OIDs of other object classes in XML schema, there is no way we can identify dependent object class. Furthermore, OIDs of object classes can only be determined after all object classes (excluding dependent object class) have been identified. Thus, we can only identify dependent object class after the first step of our rule-based approach, internal node classification and the second step leaf node classification.

In this section we use a top-down approach with the following identification rule to identify dependent object classes and IDD relationship types. Top-down approach means we examine each internal node with the following identification rule for dependent object class, and then examine its child/descendant internal node.

Rule 7. ***[Dependent Object Class*** *and* ***IDD Relationship Type]*** *In an XML schema tree, given an internal node* i *and OID* id_o *of its lowest ancestor object class or lowest ancestor dependent object class* o*, if* $\exists$ e, e *is an EDLN (exclusive descendant leaf node) of* i *such that:*
(1) i *has more than one child node;*
(2) e *cannot functionally/multi-valued determine any other EDLN of* i*;*
(3) e *and* id_o *together can fully functionally/multi-valued determine all EDLNs of* i*;*
Then i *is identified as a dependent object class and there is an IDD relationship type between object class/dependent object class* o *and dependent object class* i.

The intuitive meaning of Rule 7 is that, for a dependent object class i, there must be an EDLN of i which is the local OID of i, and the value of this local OID alone cannot uniquely identify each instance of this dependent object class (i.e., the local OID cannot functionally/multi-valued determine any other EDLNs of i). However, together with the OID of an object class or another dependent object class o, which is the lowest ancestor object class/descendant object class of i, the value of this local OID is able to uniquely identify each instance of this dependent object class (i.e., they together can functionally/multi-valued determine all EDLNs of i). Note that as we use top-down approach to identify dependent object classes, when we are examining whether internal node i is a dependent object class, all its ancestor dependent object classes (if any) have beed identified with their OIDs. Furthermore, there is an IDD relationship type between the identified dependent object class i and its lowest ancestor object class/dependent object class o.

Example 4.18:

In Fig. 4.1, given an identified object class *Book* with its OID *ISBN*, it has a child node *Chapter*, which is an internal node. *Chapter* is identified as a composite attribute based on Rule 3 in our step of internal node classification, because *Chapter* has more than one child node, *C#*, *PageFrom* and *PageTo*; there is no FD/MVD among its child node; there is no IDREF(S) attribute, object class, or role name as its child node. Assume we only have the following FDs related to the child nodes of *Chapter*:

1. $\{ISBN,\ C\#\} \rightarrow \{PageFrom\}$;
2. $\{ISBN,\ C\#\} \rightarrow \{PageTo\}$;

Based on Rule 7, we identify internal node *Chapter* as a dependent object class with its local OID as *C#*, and there is an IDD relationship type between object class *Book* and dependent object class *Chapter*. This is because *C#* does not functionally/multi-valued determine any other EDLNs of *Chapter*, but *C#* together with *ISBN*, which is OID of its parent object class *Book*, $\{C\#,\ ISBN\}$ can functionally determine all EDLNs of *Chapter*.

The intuition meaning of this example is given an instance of *Chapter*, even with its *C#*, there is no way we can uniquely identify a particular chapter without knowing to which *Book* it belongs. □

4.6 Chapter Summary

In this chapter, we proposed an automatic rule-based approach for discovering ORA-semantics from XML data and XML schema. Our rule-based approach is mainly based on the properties and heuristics of different ORA-semantic concepts, and uses them to distinguish among different ORA-semantic concepts. For some ORA-semantic concepts such as object class, composite attribute, explicit relationship type etc., we proposed classification rules to identify them based on their properties and heuristics; for some other ORA-semantic concepts such as OID, we proposed algorithms which also consider its properties, heuristics as well as statistic information to identify it.

CHAPTER 5

PERFORMANCE STUDY

5.1 Experiment

5.1.1 Introduction

In this section, we experimentally evaluate the proposed rule-based approach for discovering the ORA-semantics from XML. Both XML schema and XML data are the inputs for our rule-based approach. As the ORA-semantics is identified from XML schema, we assume that the corresponding XML schemas for all experiment datasets are available for our rule-based approach either provided as inputs or extracted and summarized from the corresponding XML data. Furthermore, the corresponding FDs and MVDs are also available either specified by the users/designers or being extracted from the corresponding XML data with user verifications.

As discussed in Chapter 3, there is no existing research work for XML database, which can discovery the ORA-semantic concepts as our rule-based semantics discovery approach does. The closest research works to our work are those object identification approaches [16, 41, 74], which try to identify object classes in XML schemas. However, these approaches only focus on one ORA-semantic concept, while our rule-based semantics discovery approach can identify 11 different ORA-semantic concepts.

Therefore, in this section, we will not compare the experimental results of our rule-based semantics discovery approach with other existing research works. Instead, we will compare the semantic concepts discovered by our rule-based semantics discovery approach with existing works related to semantics (including ontology model, DBRE (database reverse engineering) approaches and object identification approaches) in Section 5.3.

5.1.2 Experimental Datasets

Our experimental data contains 15 real world data-centric XML datasets (including their XML data[1] and XML schemas), including the auction dataset[2], the university courses datasets[3], the SIGMOD records dataset[4], the baseball 1998 s-tatistic dataset[5], the Mondial dataset[6], the Market Place dataset[7], the XMark dataset[8],the Purchase Order dateset[9], and the TPoX dataset[10]. Details about these XML datasets (including the height of XML schema trees, the number of internal node, and the number of leaf node in their corresponding XML schema

[1]For those XML datasets without XML data, users will proved the FDs and MVDs information for our rule-based approach as input.

[2]http://www.cs.washington.edu/research/xmldatasets/www/repository.html#auctions

[3]http://www.cs.washington.edu/research/xmldatasets/www/repository.html#courses

[4]http://www.cs.washington.edu/research/xmldatasets/www/repository.html#sigmod-record

[5]http://www.cafeconleche.org/examples/baseball/1998statistics.xml

[6]http://www.cs.washington.edu/research/xmldatasets/www/repository.html#mondial

[7]http://www.prosper.com/tools/DataExport.aspx

[8]http://www.xml-benchmark.org

[9]http://disi.unitn.it/ accord/Schemas/CIDX_Excel.xml

[10]http://tpox.sourceforge.net

trees) are listed in Table 5.1. We also asked 8 PhD students study XML to design 18 synthetic XML schemas[11] with related FDs and MVDs being specified.

Note 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. In order to enlarge our experimental data, we also collected 5 relational datasets including the IMDB data[12], the TPC-H data[13], the Basketball data[14], the Baseball data[15], and the Music Brainz data[16]. We asked 8 PhD students study XML to reasonably design the corresponding XML schemas based on those 5 relational datasets.

In a word, all our experimental dataset contains 15 real world data-centric XML datasets, 18 synthetic XML datasets and 5 data-centric datasets transformed from read world relational datasets. Among all XML datasets for our experiments, some of them come with complex structure such as XMark, which contains 145 internal nodes, 173 leaf nodes and a maximal depth of 8 in its XML schema tree.

5.1.3 Ground Truth

To evaluate the accuracy of our rule-based approach, we measure precision, recall and F-measure[17] against the ground truth provided by 8 evaluators, who are all PhD students study XML. In order to handle the case that different evaluators may have different understandings of what a node in an XML schema should be identified as, we adopt the probability theory and use the probability of a node being identified as each ORA-semantic concept as its ground truth rather than using only

[11]The XML schema trees of all synthetic XML schemas can be found in the following address: https://dl.dropbox.com/u/11334632/SyntheticXMLSchemas.vsd

[12]http://www.imdb.com/interfaces

[13]http://www.tpc.org/tpch

[14]http://www.basketballreference.com

[15]http://seanlahman.com/baseball-archive/statistics

[16]http://musicbrainz.org/doc/MusicBrainz_Database/Schema

[17]F-measure = 2 * precision * recall/(precision + recall)

Table 5.1: Statistics of 15 real world data-centric XML datasets

	Height of XML Schema Tree	Number of Internal Node	Number of Leaf Node
eBay	5	6	25
Course_Reed	4	3	12
SIGMOD	7	5	7
Mondial	6	29	109
NBA	6	5	10
Baseball	6	4	26
MarketPlace	4	14	108
BookStore	6	4	11
XMark	8	145	173
Course_UWM	5	4	15
Course_WSU	4	3	16
Accord_PO	4	10	37
Accord_RDB	3	13	65
Accord_Star	3	5	34
TpoX	5	12	42
Total	-	262	690
Average	5	18	46

one ORA-semantic concept as ground truth for each node being identified.

For example, the ground truth of an internal node may be having 75% to be an object class and 25% to be a composite attribute as a result of among 8 evaluators, 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 considered as the ground truth with different probabilities by using their expected values in our calculation of precision, recall and F-measure (i.e. score 0.75 for object class, and score 0.25 for composite attribute.).

Table 5.2: 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%

5.1.4 Accuracy of Internal Node Classification

Except the dependent object classes[18], there are totally 512 internal nodes in XM-L schemas of all our experimental datasets. We ask our 8 evaluators to label them with their ORA-semantics (i.e., object class, role name, explicit relationship type, aggregational node or composite attribute). Table 5.2 shows that using all above 512 internal nodes as inputs, the overall accuracy[19] of our classification rules achieves almost 95% of precision, recall and F-measure. The relative low precisions for role name and aggregational node as well as the relative low recall for explicit relationship type are because their corresponding classification rules contain some related heuristics and these heuristics are not as accurate as the properties being used in other classification rules.

As discussed in Section 4.3, properties of aggregational node and explicit relationship type are not enough to distinguish them with each other. Role name and explicit relationship type cannot be distinguished with each other by their properties either. Therefore, we use the following heuristics to identify them:

Aggregational Node The tag name of an aggregational node should be the plural form of the tag name of its child node;

Explicit Relationship Type The tag name of an explicit relationship type is

[18]Recall in Section 4.5.3, the dependent object class cannot be identified during this step.

[19]Overall accuracy means we do not distinguish among different ORA-semantic concepts for their precisions, recalls and F-measures.

likely to be a word with verb form;

Role Name Tag name of a role name share high linguistic similarity with or being a specialization of the tag name of the object class it being the role name of.

However, as heuristics are not 100% correct, there are still aggregational nodes, explicit relationship types and role name which do not satisfy the above heuristics.

Example 5.1: In XML schema of XMark dataset, there is an internal node *MailBox*, which has a child node *MailBox*. Apparently, *MailBox* is not a plural form of *Mail*, and our rule-based approach cannot identify it as an aggregational node using the above heuristic.

In XML schema of XMark dataset, another internal node *InCategory* also cannot be correctly identified as explicit relationship type by above heuristic because its tag name is combined by multiple words, which makes our rule-based approach cannot identify whether its tag name can be a verb form or not.

Internal node *Interest* in the schema of XMark dataset also cannot be identified as role name because its low linguistic similarity with its child node *Category*. □

Another reason for the low precision of role name is because of its small percentage among all the internal nodes, which makes a single misidentification of it affect its precision heavily. In Table 5.3, we show the number and percentage of each ORA-semantic concept in all 512 input internal nodes. Obviously, object class is one of the most important ORA-semantic concepts needed to be identified, and its identification helps many XML applications to increase their efficiencies or effectiveness as introduced in Chapter 1. There are 311 object classes among all 512 internal nodes, which take up around 60% of all the internal nodes. Thus, it is especially important for a semantics discovery approach to have a high accuracy for identifying object class. Our rule-based approach achieves more than 98% of both precision and recall for identifying object class. On the other hand, other

Table 5.3: Distribution of different ORA-semantic concepts

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

ORA-semantic concepts only take up a small percentage of all the internal nodes, especially for role name, which only takes up less than 4% of all the internal nodes.

5.1.5 Accuracy of Leaf Node Classification

As discussed in Section 4.4.1, Algorithm 2 in rule-based semantics discovery approach may return more than one candidate OID for each identified object class. Therefore, as discussed in Section 4.4.1, in order to choose the best OID from all candidate OIDs, we adopt the Bayes' Theorem [25] to calculate their probabilities of being the best OID, and rank them to return the highest ranked candidate as the OID of the corresponding object class.

We collect 122 correctly identified object classes, each of which has more than one candidate OIDs returned by the Algorithm 2 in Section 4.4.1. For all 122 object classes, we ask our evaluators to manually specify their OIDs from all their candidate OIDs, and use all these candidate OIDs as our training data. For each node in the training dataset, we extract the statistic information of the related features mentioned in Observation 1 and Observation 2, which are discussed in Section 4.4.1. We then adopt the Bayes' Theorem to calculate the probability of a candidate OID with particular feature being the best OID for its corresponding object class based on the statistic information. In Table 5.4, based on the features mentioned in Observation 1 and Observation 2 in Section 4.4.1, we show the top

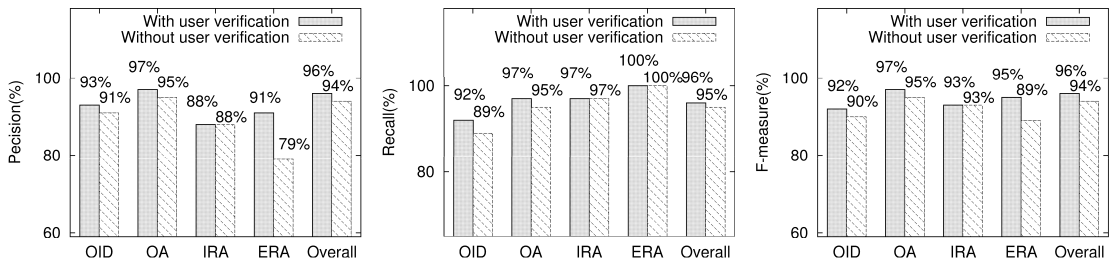
Table 5.4: Top 10 combined features of being OID

	Number of Involved Attributes	Number of Participating Object Classes	Contains First Child Node	Contains Certain Keyword Substring	Numerical Value with Pattern	Probability of Being OID
1	1	1	T	T	T	0.432
2	1	1	T	T	F	0.172
3	1	2	T	T	T	0.068
4	2	1	T	F	F	0.045
5	1	1	F	T	T	0.038
6	1	1	F	T	F	0.038
7	1	1	T	F	T	0.030
8	1	2	T	F	T	0.030
9	1	1	F	F	T	0.016
10	1	2	T	T	F	0.016

10 probabilities of the candidate OID with different features being the best OID of its corresponding object class.

Recall the general process (Fig. 4.2 in Section 7.2) of our rule-based semantics discovery approach, our approach is also a step by step approach. Therefore, some outputs of the previous step will work as the inputs for a latter step. E.g., object classes identified by internal node classification step will be inputs of leaf node classification step. In consequence, the accuracy of the latter step is affected by the accuracy of its previous step(s).

In order to show the accuracy of each step separately, we conduct two groups of experiments with all our experimental XML datasets to evaluate the precision, recall and F-measure of our rule-based semantics discovery approach for leaf node classification: one with user verification, which means all object classes have been correctly labeled in the corresponding XML schemas; and one without user verification, which means the object classes is the results from the step of internal node classification of our rule-based semantics discovery approach.



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

Figure 5.1: Precision, Recall and F-measure of Leaf Node Classification
(OA: Object Attribute;
ERA: Relationship Attribute of Explicit Relationship Type;
IRA: Relationship Attribute of Implicit Relationship Type.)

Fig. 5.1 shows that our rule-based semantics discovery approach for leaf node classification gets around 95% of overall precision, recall and F-measure[20] with or without user verifications. Even without user verification, the precision and recall of our rule-based approach only drop slightly (except for precision of relationship attribute of explicit relationship type, which will be explained later.).

The precision for identifying relationship attribute of implicit relationship type is a little bit lower than the precisions for identifying OID and object attribute. It is because given an implicit relationship type, if the OID of any participating object class is identified wrongly, many object attributes of participating object classes will be identified as relationship attributes of the implicit relationship type (because the incorrect OID may not be able to functionally/multi-valued determine those object attributes). Although the identification of object attribute is also affected by the identification of OID, as there are much more object attributes than relationship attributes in XML database, relationship attributes of implicit relationship type are more heavily affected by the identification of their corresponding OIDs.

For relationship attribute of explicit relationship type, the precision of its iden-

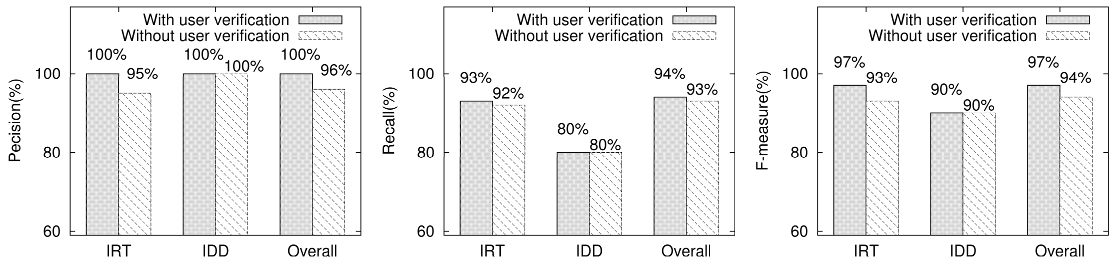
[20]Recall that overall means we do not distinguish among different ORA-semantic concepts for their precisions, recalls and F-measures.

tification is heavily depend on whether the corresponding explicit relationship type is correctly identified. It is because all EDLNs (exclusive descendant leaf nodes) of an explicit relationship type will be identified as its relationship attributes by our rule-based approach. Therefore, its low precision is because the misidentification of other ORA-semantic concepts as explicit relationship types which cause many object attributes being identified as relationship attributes. With user verification of the identification of explicit relationship types, its precision increases largely.

5.1.6 Accuracy of Implicit Relationship Type Identification

Last, we conduct our experiment on our rule-based approach for implicit relationship type discovery. Recall that all explicit relationship types have been discovered by our Rule 2, 3, 4, 5 in Section 4.3, and its accuracy has been shown in Table 5.2. Similar to leaf node classification, we also conduct two groups of experiments with all our experimental XML datasets to evaluate the accuracy of our rule-based approach for implicit relationship type identification, one with user verification, which means all object classes with their OIDs, object attributes and relationship attributes have been correctly labeled in the corresponding XML schemas, and one without user verification, which means the input our implicit relationship type i-dentification is the results of our previous two steps: internal node classification and leaf node classification.

Fig. 5.2 shows the precision, recall and F-measure for identifying implicit relationship types (including IDD relationship type) in XML schemas. For implicit relationship type, when there is no user verification, its misidentification is because of the wrongly identified relationship attribute from the previous step, leaf node classification. Recall that in our rule-based semantics discovery approach, for each identified relationship attribute, we will identify an implicit relationship type it



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

Figure 5.2: Precision, recall, F-measure of implicit relationship type identification (IRT: Implicit Relationship Type.)

belongs to (discussed in Section 4.5.1). Therefore, those object attributes, which have been wrongly identified as relationship attributes, make our approach generate wrong implicit relationship types.

On the other hand, our rule-based semantics discovery approach also identifies implicit relationship types by IDREF(S) attributes (discussed in 4.5.2). However, as discussed before, for some XML schemas (such as DTD), they do not specify the the object class or corresponding OID the IDREF(S) reference. Thus, some implicit relationship types may be lost.

5.2 Impact of Possible Misidentification for XML Applications

As our rule-based semantics discovery approach not uses only the properties of different ORA-semantic concepts but also their related heuristics to distinguish among them, which are not guaranteed to be 100% correct. This will reduce the accuracy of our rule-based approach. Moreover, the rule-based semantics discovery approach also uses the FDs and MVDs imposed in the corresponding XML data. However, in XML database even relational database, their FDs and MVDs are both constraints imposed in the corresponding data, and do not carry any semantic information with them. Thus, if there is no user verification or specification for the discovered FDs and MVDs, there may be some unexpected/meaningless FDs and MVDs returned by those FDs/MVDs summarization approaches, especially when the size of the dataset is small. These unexpected/meaningless FDs and MVDs will reduce the accuracy of our rule-based approach.

In this section we will discuss what kind of possible misidentification our rule-based semantics discovery approach may make, and if our rule-based semantics discovery approach returns wrong semantic information, how these wrongly identified semantics affects the effectiveness or efficiency of XML applications such as XML keyword search. Finally, we will show that even if our rule-based approach identifies some ORA-semantic concepts wrongly, the applications can still get their results no worse than those without any ORA-semantics being discovered.

In the following, for the first two subsections, we will focus on two possible misidentifications of our rule-based semantics discovery approach: object class vs. composite attribute, and dependent object class vs. object class. These two possible misidentifications are mainly caused by the unexpected FDs and MVDs due to the small size of datasets. Furthermore, possible misidentifications may also caused by the fact that some ORA-semantic concepts share some of their properties, and they may be difficult to be distinguished with each other, such as object class vs. explicit relationship type, composite attribute vs. explicit relationship type, aggregational node vs. explicit relationship type. In the last subsection, we will discuss that these misidentifications seldom occur in our rule-based semantics discovery approach, because we have at least one property to distinguish between them for each of the above misidentifications.

5.2.1 Possible Misidentification: Object Class vs. Composite Attribute

Recall the decision tree of our rule-based semantics discovery approach shown in Fig. 4.3 and the property table shown in Table 4.1, in XML schema, the object class and the composite attribute share a lot of properties in term of their structural features in their corresponding XML schema tree. Both object class and composite attribute are internal nodes with more than one child node in their XML schema trees. The only difference between object class and composite attribute is whether there is any FD/MVD among their exclusive descendant leaf nodes (EDLNs). However, as mentioned above, if the size of XML data is too small, unexpected FDs/MVDs will be discovered even for a composite attribute. Thus, it is easy for a composite attribute being identified as an object class by our rule-based approach.

In the following, we will show examples of XML keyword search by applying an existing LCA-based XML keyword search approach such as [59, 68], given an XML data with its XML schema, and keyword queries from users. With the ORA-semantics being discovered by our rule-based approach (even we wrongly identified some composite attributes as object classes), the query results are no worse than the results returned by applying the same XML keyword search approach without any ORA-semantics being identified.

In Fig. 5.3, our rule-based semantics discovery approach will wrongly identify composite attribute *ContactInfo* as an object class if we discover any FD/MVD among its EDLNs. Based on how the existing LCA-based keyword search approaches return the answers, we discuss the following two keyword queries by examples. Note that all processing are conducted on XML data in the rest of this section.

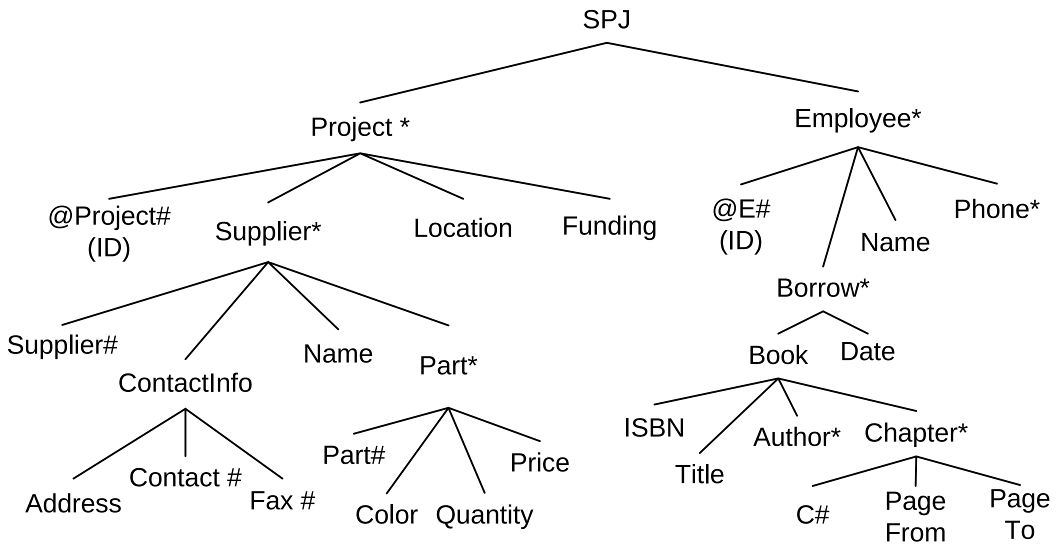


Figure 5.3: An XML schema tree

Example 5.2: [Keyword Query: {*Suppler, 10 Stanford Road*}]

Based on the XML schema tree in Fig. 5.3, the query keyword '*Supplier*' matches the internal node *Supplier* and assume the query keyword '*10 Stanford Road*' matches the value of leaf node *Address*. Without ORA-semantics, existing LCA-based approach will return the lowest common ancestor (LCA) of these two matching nodes and the subtree roots at this LCA node, which is an instance of the internal node *Supplier* with its *Address* as '*10 Stanford Road*'. Note that the subtree rooted at an instance of the internal node *Supplier* also contains all instances of *Part* which are under this particular *Supplier* instance. If this supplier instance supplies many parts, e.g. 1000, the query results will contain all information about these 1000 parts, which will overwhelm the users and may not be the query intention of the users.

On the other hand, for the same keyword query, our discovered ORA-semantics show that the internal node *Supplier* and *Part* are object classes and *Supplier#* and *Part#* are their OIDs respectively. Assume the composite attribute *ContactInfo* is misidentified as an object class by our rule-based approach. With above ORA-semantics, the keyword search approach will still return the subtree rooted at an object instance of *Supplier* with its *Address* as '*10 Stanford Road*'. By identifying *Part* as an object class, the query result will not show all the object instances of *Part*, but represent these object instances by their OID values only. For the instance of *ContactInfo*, it will be represented only by its 'fake' OID value similar to the object class *Part*, as it is misidentified as an object class. However, the user can always expand the instance of *ContactInfo* for more information. This also applies to other object instances such as object instances of *Part* as long as it is not the root node because all information about the root node has already been expanded and shown in the query result. □

In Example 5.2.1, we show that the results of keyword search approach with ORA-semantics are no worse than those without ORA-semantics. Furthermore, by representing object instances with their OIDs, the results with ORA-semantics give users a more clear and simple view of the results without overwhelming the users with unnecessary information.

Example 5.3: [Keyword Query: {*Contact#, 75862549*}]

Based on the XML schema tree in Fig. 5.3, the query keyword '*Contact#*' matches the leaf node *Contact#* and assume the query keyword '*75862549*' matches the value of *Contact#*. Without the ORA-semantics, existing LCA-based keyword search approach will return the LCA of the corresponding matching nodes and the subtree roots at it, which is an instance of the internal node *ContactInfo*. According to the keyword query, we can guess the user intention is to query the information whoever/whatever have its *Contact#* as '*75862549*'. In our XML data, it should be the supplier with its contact# as '*75862549*'. However, the subtree returned by the existing LCA-based keyword search approach only contains the information under an instance of *ContactInfo*, which is incomplete and meaningless.

On the other hand, for the same keyword query, assume the composite attribute *ContactInfo* is misidentified as an object class by our rule-based approach. With these ORA-semantics, the keyword search approach will also return the subtree roots at an instance of *ContactInfo* with its *Contact#* as '*75862549*' thinking it as an object instance. Although this answer is also incomplete, the result is no worse than the result without ORA-semantics being identified as discussed above. □

Therefore, we have shown that even if our rule-based semantics discovery approach wrongly identified composite attribute as object class, the returned results for XML keyword search are no worse than those without using any ORA-semantics.

5.2.2 Possible Misidentification: Dependent Object Class vs. Object Class

Similar to a composite attribute being misidentified as an object class, a dependent object class is also possibly misidentified as an object class when the XML data is not large enough to rule out those unexpected FDs and MVDs.

Recall that dependent object class is identifier depend on another object class, e.g., in Fig. 5.3, the dependent object class *Chapter* is identifier depend on object class *Book*. In order to uniquely identify an instance of dependent object class, an instance of its ancestor object class(es) is also necessary. For example, we can only uniquely identify an instance of *Chapter* by also providing an instance of *Book*. Thus, in the application of XML keyword search, similar to the composite attribute, it is meaningless to only return the instance of a dependent object class.

Example 5.4: [Keyword Query: {*Database Management, Chapter 01*}]

Based on the XML schema tree in Fig. 5.3, and assume the query keywords

'*Database Management*' and '*Chapter 01*' match the value of leaf nodes *Title* and *C#* respectively. Without ORA-semantics, existing LCA-based approach will return a *Book* instance whose title is '*Database Management*'. Note that the result also contains all *Chapter* instances under this *Book* instance with their attributes.

On the other hand, for the same keyword query, our discovered ORA-semantics show that the internal node *Book* is an object class with its OID *ISBN*. Assume the dependent object class *Chapter* is misidentified as an object class. With above ORA-semantics, the keyword search approach will still return a *Book* instance whose title is '*Database Management*'. For instances of *Chapter* under this *Book* instance, they will be represented only by their 'fake' OID value, as it is misidentified as an object class. However, the user can always expand the instance of *Chapter* for more information. □

Example 5.5: [Keyword Query: {*C#, Chapter 01*}]

Based on the XML schema tree in Fig. 5.3, the query keyword '*C#*' matches the leaf node *C#* and assume the query keyword '*Chapter 01*' matches the value of *C#*. For this keyword query, no matter whether the keyword search approach consider the ORA-semantics or not, the answer would be a *Chapter* instance whose *C#* is '*Chapter 01*'. As *Chapter* should be identified as a dependent object class, it is incomplete to return just the *Chapter* instance without showing which *Book* instance it belongs to. □

5.2.3 Possible Misidentifications: Explicit Relationship Type vs. Object Class/Composite Attribute/Aggregational Node

Recall that in our rule-based semantics discovery approach, we use a decision tree constructed by the properties and heuristics of different ORA-semantic concepts for classifying each internal node in XML schema tree into an ORA-semantic concept. As is mentioned in Section 4.3, the structural characteristics of explicit relationship type are flexible. Because of this, explicit relationship type may share lots of properties with other ORA-semantic concepts, including object class, composite attribute and aggregational node. In the following, we will show what is the key differences between them, and how our rule-based semantics discovery approach can distinguish between them.

Explicit Relationship Type vs. Object Class

For explicit relationship type and object class, their key difference is: there is no OID for any explicit relationship type, but there must be an OID for each object class. In XML schema, we can distinguish most of the explicit relationship types from object class by checking whether they have any FD/MVD **among their exclusive descendant leaf nodes(EDLNs)**. Recall 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. In the following, we will use an example to illustrate how to distinguish explicit relationship type and object class:

Example 5.6: In Fig. 5.4, we show the XML schema trees representing the borrow relationship between two object classes *Student* and *Book* by an explicit relationship type *Borrow* in 5 different designs. According to the definition of EDLN:

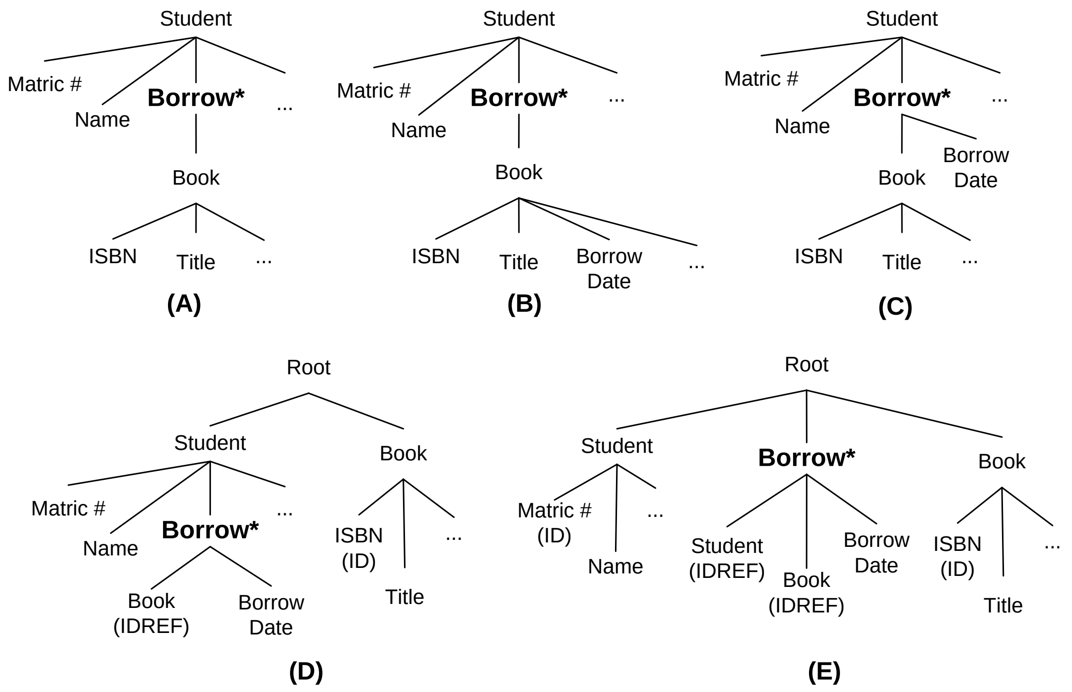


Figure 5.4: Explicit Relationship Types *Borrow* Represented by 5 Different Hierarchical Structures in XML Schema Trees

- The explicit relationship type *Borrow* in (A)[21] and (B) both do not have any EDLN;
- The explicit relationship type *Borrow* in (C) has only 1 EDLN: *BorrowData*;
- The explicit relationship type *Borrow* in (D) has 2 EDLNs: *Book* and *BorrowData*;
- The explicit relationship type *Borrow* in (E) has 3 EDLNs: *Student(IDREF)*, *Book(IDREF)* and *BorrowData*;

It is obviously that for the explicit relationship type *Borrow* in (A)(B)(C)(D), there is no FD/MVD among their EDLNs. However, for object classes such as *Student* and *Book*, because there must be an OID among its EDLNs for each object class, there should have at least one FD/MVD among their EDLNs, e.g.,:

[21]Recall that each node with a '$*$' as its superscript in the XML schema tree means it is a repeatable node, and repeatable node is the node which can occur multiple times with the same XPath in the corresponding XML data.

- $\{Matric\#\} \rightarrow \{Name\}$;
- $\{ISBN\} \rightarrow \{Title\}$;

On the other hand, as is shown in Fig. 5.4 (E), it is also possible for an explicit relationship type to have FD/MVD among its EDLNs (*Student(IDREF)* and *Book(IDREF)* are both IDREF attributes referring to OIDs of object classes *Student* and *Book*, i.e., *Matric#* and *ISBN*):

- $\{$*Student(IDREF)*, *Book(IDREF)*$\} \rightarrow \{$BorrowDate$\}$;

However, we find out that the left hand side (LHS) of the above FD/MVD are all IDREF attributes. Similarly, we can distinguish explicit relationship type with object class by checking whether the LHS of all FD/MVD among their EDLNs are IDREF attribute or role name. If the answer is yes, it should be an explicit relationship type. This is because for explicit relationship type with FD/MVD among its EDLN, the object class under it in the XML schema tree can only be represented as IDREF attribute or role name. Otherwise, its designs will be similar to Fig. 5.4 (A)(B)(C)(D), and there will not be any FD/MVD among its EDLNs. On the other hand, for object class, the LHS of the FD/MVD among its EDLNs should contain its own OID, which is not an IDREF attribute or a role name. □

Explicit Relationship Type vs. Composite Attribute

For explicit relationship type and composite attribute, both of them can have more than one child node in their XML schema tree, and it is possible that both of them do not have any FD/MVD among their child nodes (e.g., *Borrow* in Fig. 5.4 (A))). The key difference between them is there should be at least one object class, role name, or IDREF(S) attribute as the child node of an explicit relationship type (e.g., any explicit relationship type *Borrow* in Fig. 5.4), but there should never be any of them as the child node of a composite attribute.

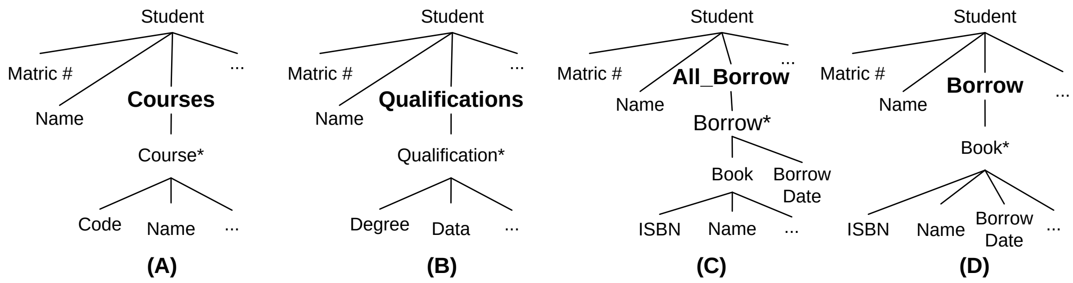


Figure 5.5: Aggregational Node (*Courses* in (A), *Qualifications* in (B), *All_Borrow* in (C) and *Borrow* in (D)) in XML Schema Trees

Explicit Relationship Type vs. Aggregational Node

The key difference between explicit relationship type and aggregational node in both XML schema and XML data is: for explicit relationship type, although its hierarchical structures may be different (e.g., explicit relationship types *Borrow* in Fig. 5.4), any explicit relationship type is representing a relationship between two or more object classes; while for aggregational node, it aggregates its child notes together for presentation purpose, so that user can have a easier and clearer understanding.

As we defined aggregational node in Section 2.2, it is an internal node in both XML schema tree and XML data tree which aggregates its child nodes with identical or similar meaning. Aggregational node can aggregates object class (e.g., aggregational node *Courses* in Fig. 5.5 (A)), composite attribute (e.g., aggregational node *Qualifications* in Fig. 5.5 (B)), and even relationship type, which can be explicit relationship type (e.g., aggregational node *All_Borrow* in Fig. 5.5 (C)) and implicit relationship type (e.g., aggregational node *Borrow* in Fig. 5.5 (D)). Notice that in Fig. 5.5 (D) there is an implicit relationship type between object classes *Student* and *Book* with a relationship attribute *BorrowData.*

We have shown that even if our rule-based semantics discovery approach wrongly identified dependent object class as usual object class, the returned results for XML keyword search are no worse than those without using any ORA-semantics.

Furthermore, as discussed in Chapter 1 and [31], whenever the ORA-semantics is correctly identified, this ORA-semantics can largely increase the accuracy for XML keyword query.

5.3 Comparisons with Existing Approaches

Recall in Chapter 3 we reviewed the semantics captured and represented by different ontology models, which are the semantic models built with ontology languages (such as RDF, RDFS and OWL). We also reviewed existing approaches for discovering semantics from relational database and existing approaches for discovering semantics from XML database.

In the following subsections, we will compare the semantics discovered by our rule-based approach with the semantics captured by ontology models and the semantics discovered by existing approaches for relational database (i.e., database reverse engineering (DBRE) approaches [2, 50, 36, 37]) and semantics discovered by existing approaches for XML database (i.e., object identification approaches [63, 64, 16, 41]). For ontology model, we use OWL [7] as its ontology language as it is proposed based on RDF [30] and RDFS [14] to capture more semantics, and other otology models share similar characteristics.

5.3.1 Comparisons with Ontology Models

For the ORA-semantics, it includes the semantic concepts such as object class, OID, object attribute, relationship type, relationship attribute, etc. However, for ontology model [7], as discussed in Chapter 3, although it can also represent the class class and relationship type in schema level, their underlying meanings are not exactly the same with the corresponding semantic concepts in ORA-semantics. To more more precise, objects in ontology model can be any 'thing' in the real world, even such as *VintageYear* with only one attribute *yearValue* as discussed in Example 3.3 of Chapter 3. However, in the ORA-semantics, object class is defined to represent a real world entity or concept with attributes to describe and store its information. For the above *VintageYear* and *yearValue*, they would be represented as an object attribute and its value in ORA-semantics.

Furthermore, ontology model can at most capture and represented binary relationship type. This is because ontology model can only capture relationship type using `ObjectProperty` with its `Domain` and `Range` which can only specifies two object classes. For ternary relationship type and n-nary relationship type, as well as relationship attribute, ontology model also cannot capture and represent them.

Example 5.7: For the XML schema tree shown in Fig. 5.3, the ontology model such as OWL must manually translate the XML data and schema into OWL document by domain experts, indicating the object class such as *Project*, *Supplier*, *Part*, *Employee*, *Book*, and binary relationship types such as the relationship type between object classes *Project* and *Supplier*, and the explicit relationship type *Borrow* between *Employee* and *Book*. However, as ontology model cannot capture and represent many other semantic concepts such as composite attribute, dependent object class, both *ContactInfo* and *Chapter* will be wrongly represented as object classes in ontology model. Furthermore, the ternary relationship type among the object classes *Project*, *Supplier* and *Part*, as well as the relationship attributes *Price*, *Quantity* and *Data* cannot be captured and represented in ontology model neither. □

In term of semantics discovery, the semantics such as object classes and relationship types in ontology model are usually designed based on the manually effort by domain experts, which will be very costly. Ontology model cannot take XML data or schema as input and automatically return the discovered semantics as we did in our rule-based semantics discovery approach.

5.3.2 Comparisons with DBRE Approaches for Relational Database

For DBRE approaches [2, 50], they are proposed to discover semantic information from an relational database and represent them in ER model or its variants. The gold of these approaches are similar with our rule-based approach, excepting that we are working on XML database. We both try to discover the semantic concepts such as object class, OID, object attribute, relationship type, relationship attribute, as well as dependent object class and IDD relationship type, etc. These semantic concepts can be well captured by ER model and ORA-SS for relational database and XML database respectively.

The difference between ORA-SS and ER model is the hierarchical structure which can be captured and represented by ORA-SS, but cannot by ER model. This difference also applies to relational database and XML database. Thus, for the DBRE approaches, they mainly use the constraints extracted and summarized from the relational data, such as primary key - foreign key constraints, functional dependencies, multi-valued dependencies, etc. However, as shown by the authors in [36, 37], that the functional dependencies and multi-valued dependencies are only constraints to enforce database integrity and does not contain semantic information.

Authors in [36, 37] has proposed the semantic dependency, which can capture the relationship, rather than integrity constraints between two sets of attributes. However, necessary user feedbacks and interactions are still necessary for these approaches during the relational to ER schema translation. However, there are still some semantic concepts which cannot be discovered by these approaches, such as aggregational node, which is not represented by ER model. This is because the aggregational node is just a structural node, which does not contain any semantic information, while ER model is proposed as a conceptual model which does not capture the hierarchical structure.

On the other hand, our rule-based semantics discovery approach considers not only the functional/multicalue dependencies, but also the hierarchical structure of the XML data and XML schema. To be more precise, we extract and summarize the properties (describing their hierarchical structures and functional/multicalue dependencies) of different ORA-semantic concepts, and use these properties to distinguish between different ORA-semantic concepts.

5.3.3 Comparisons with Object Identification Approaches for XML Database

For the object identification approaches, authors in [16, 41, 74] focus on discovering the object classes in schema level and object instances in data level; while authors in [63, 64] focus on identifying the identical object instances in XML data. Both of them ignore many other equivalently important semantic concepts such as OID, relationship type, relationship attribute as well as dependent object class and IDD relationship type, etc.

Furthermore, these approaches discover the corresponding semantics merely based on some basic linguistic information (such as tag name of element nodes, etc.) and structural information (such as whether an element node is an internal node or a leaf node; whether an element node is a repeatable node[22], etc.), which

[22]Recall that repeatable node is the node which can occur multiple times with the same XPath

are far from enough to identify all ORA-semantic concepts.

Example 5.8: Using the XML schema tree shown in Fig. 5.3 as input, in [41, 74] the authors identify all repeatable nodes as object classes. Although object classes *Project*, *Supplier*, *Part*, *Employee* and *Book* can be correctly identified, these approaches will also wrongly identify other semantic concepts such as composite attribute *ContactInfo*, explicit relationship type *Borrow* and dependent object class *Chapter* as object classes. Furthermore, many other semantic concepts such as relationship type, relationship attribute, etc. cannot be identified by these approaches.

□

On the other hand, as mentioned many times, our rule-based semantics discovery approach collects not only properties of different ORA-semantic concepts but also their heuristics summarized from statistics information from both XML schema and XML data. Among the properties and heuristics we collected for identifying different ORA-semantic concepts, they include not only the structural features, but also linguistics features as well as constraints such as functional dependency extracted from the XML data. Furthermore, our approach can discover much richer semantics including not only object class, but also OID, relationship type, object attribute, relationship attribute, etc.

5.3.4 Comparisons Summary

Similar to Chapter 2, to summarize the above discussion, we also list all ORA-semantic concepts and show whether they can be discovered by (in) ontology model, DBRE approaches and object identification approaches in Table 5.5.

Note that '✓' means the corresponding ORA-semantic concept can be discovered by (in) this approach (model) correctly, assuming there is no unexpected/

in the corresponding XML data.

Table 5.5: ORA-semantic concepts discovered by (in) existing approaches (model)

	Rule-based Semantics Discovery Approach	Ontology Model (OWL)	DBRE Approaches	Object Identification Approach
Object Class	✓	✓✘	✓	✓✘
OID	✓	✓✘	✓	✘
Object Attribute	✓	✓✘	✓	✓✘
Binary Relationship Type	✓	✓✘	✓	✓✘
N-nary Relationship Type	✓	✘	✓	✘
Relationship Attribute	✓	✘	✓	✘
IDD Relationship Type	✓	✘	✓	✘
Dependent Object Class	✓	✘	✓	✘
Role Name	✓	✘	✘	✘
Composite Attribute	✓	✘	✓	✘
Aggregational Node	✓	✘	✘	✘

meaningless FD/MVD because of the small size of input dataset; '✘' means the corresponding ORA-semantic concept cannot be discovered by (in) this approach (model), or it is not captured/represented in this model; while '✓✘' means the corresponding ORA-semantic concept can only be partially discovered by (in) this approach (model), or it may also wrongly discovers other ORA-semantic concepts as the corresponding ORA-semantic concept.

In ontology model, as discussed before, although it can capture object classes, it also represents many object attributes in ORA-semantics as object classes. For object identification approaches, although they can correctly identify object classes by identifying all internal nodes/repeatable nodes as object classes, they may also wrongly identify composite attributes, role names or aggregational nodes as object classes. For DBRE approaches, although they can identity most of the ORA-semantics concepts, these approaches also heavily depend on user interactions and some strong assumptions (such as relational schemas in 3NF, etc.).

5.4 Chapter Summary

Extensive experiments have been conducted in 15 real data-centric XML datasets to show that our rule-based approach gets high accuracy for discovering ORA-semantics (almost 95% of overall precision, recall and F-measure). However, FDs and MVDs are important factors for our rule-based approach. Meaningless and unexpected FDs/MVDs because of the small dataset will reduce the accuracy of our rule-based approach of identify the correct ORA-semantic concepts (such as composite attribute and dependent object class).

We discussed the impact for XML applications such as XML keyword search, when our rule-based approach misidentifies ORA-semantics. We showed that even our rule-based approach returns some incorrect answers (i.e., wrongly identify composite attributes and dependent object classes as object classes), the results for XML keyword search will not worse than those without any ORA-semantics being identified.

In this chapter, we also compared the ORA-semantics discovered by our rule-based approach with the semantics captured and represented in ontology model. We show that there many important ORA-semantic concepts which cannot be captured and represented by ontology model, such as ternary/n-nary relationship type, dependent object class, etc. Furthermore, we also compared our rule-based semantics discovery approach with other existing semantics discovery approaches for relational database (i.e., DBRE approaches) or for XML database (i.e., object identification approaches). We showed that DBRE approaches still need necessary user interactions and our rule-based approach can identify much more semantic concepts (such as OID, relationship attribute, etc.) than those object identification approaches.

CHAPTER 6

DEMONSTRATION SYSTEM

Based on the rule-based approach for discovering ORA-semantics we proposed in Chapter 4, we have built an ORA-semantics Discovery System. Our system takes XML files as input, and will discover the ORA-semantics automatically by making use of the XML data. DTD or XSD is also part of the input but it is optional. Because the schema information we need in the process can be derived from the XML data itself. Finally the semantics we discovered will be present in a user-friendly and interactive way.

In the following, we use the XML data (Fig. 1.1) and XML schema (Fig. 1.4) showed in Chapter 1 to demonstrate how the system offers a new and visual way to discover the ORA-semantics given an input, and how it greatly enhances user experience by:

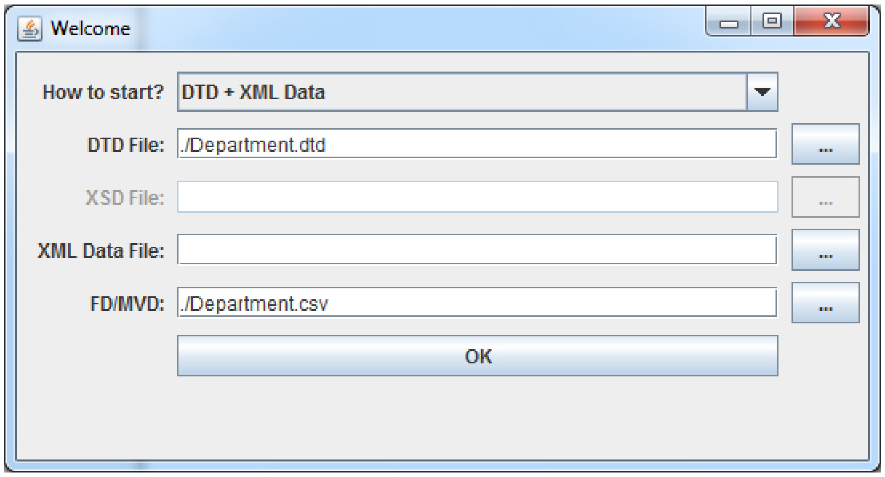
1. Showing the ORA-semantics in a graphical interface;
2. Providing an interactive way for users to explore and improve the ORA-semantics discovered.

6.1 System Input

When the system is launched, an open dialog is shown to the users, prompting them to specify the input files. The input includes:

1. An XML data file;
2. DTD or XSD (optional);
3. A file specifying functional dependencies (FDs) and multi-value dependencies (MVDs) imposed in the XML data (optional).

Fig. 6.1 shows the welcome dialog of our system. It provides three options for users to start the process, namely 'DTD + XML Data', 'XSD + XML Data' and 'XML Data'. This is because the schema information we need for the process can be derived from the XML data itself, which makes the DTD/XSD an optional input. Besides, FD/MVDs of the XML data is also option because users can also specify FD/MVD one by one later. As mentioned in section 7.2, there are many existing works on deriving FDs/MVDs from XML data. As it is not the main focus of our work, we let users provide this information.


Welcome

How to start? DTD + XML Data

DTD File: ./Department.dtd

XSD File:

XML Data File:

FD/MVD: ./Department.csv

Figure 6.1: Open dialog.

When the files are ready, users can click the 'OK' button at the bottom of Fig. 6.1 to start discovering the ORA-semantics from input files. After the discovery process is completed, results will be displayed to the users.

6.2 ORA-semantics Display

When the system finish the discovery process, the ORA-semantics discovered will be displayed in a window. Nodes are classified into different categories and shown in a tree structure. Fig. 6.2 shows the ORA-semantics discovered by our system. An XML schema tree is displayed in the window. Each node is represented as a rectangle. The ORA-semantic concepts discovered are shown in rectangle of each node. Information of different node categories is shown at the upper right corner of the window.

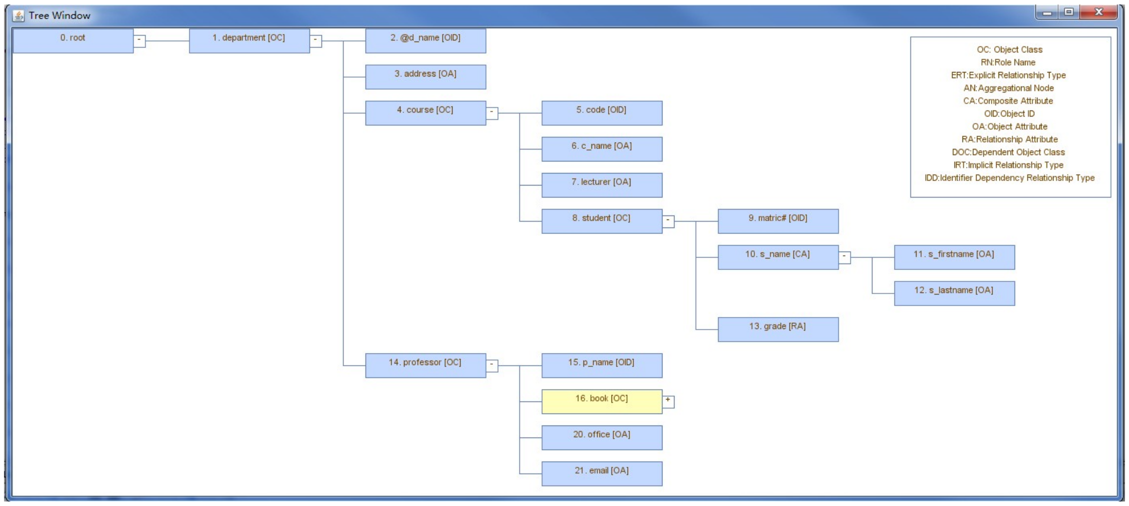
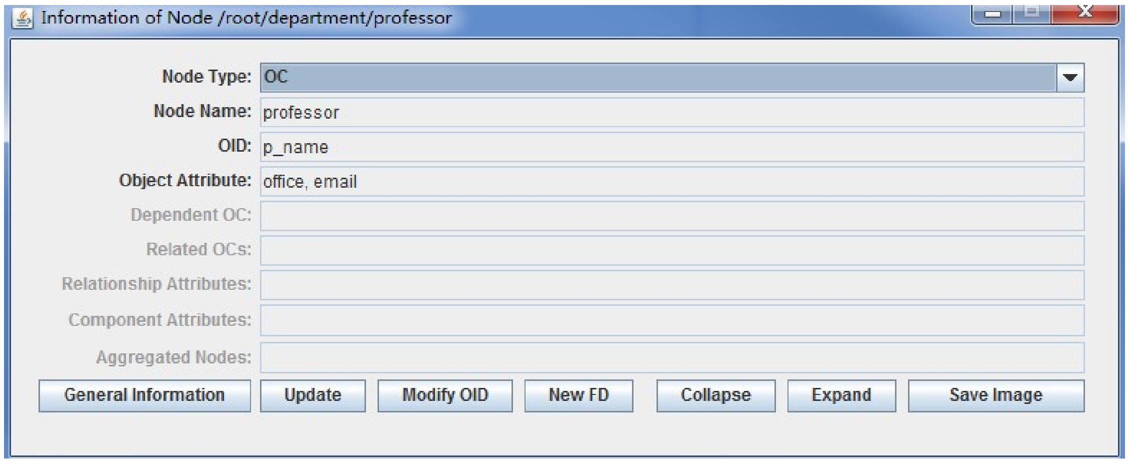


Figure 6.2: ORA-semantics discovered by the system.

Each node in the schema tree is foldable. Users can click on the '+' ('-') sign beside each node to unfold (fold) the subtree. Folded nodes are represented as yellow rectangles while unfolded nodes are represented as blue nodes. So the ORA-semantics can be managed in a neat way even when the schema tree is huge.

6.3 User Interaction

Apart from the XML schema tree view which displays the discovered ORA-semantics, the system also provides an interactive way for users to explore the semantic information and improve the ORA-semantics discovered by the system.


Information of Node /root/department/professor

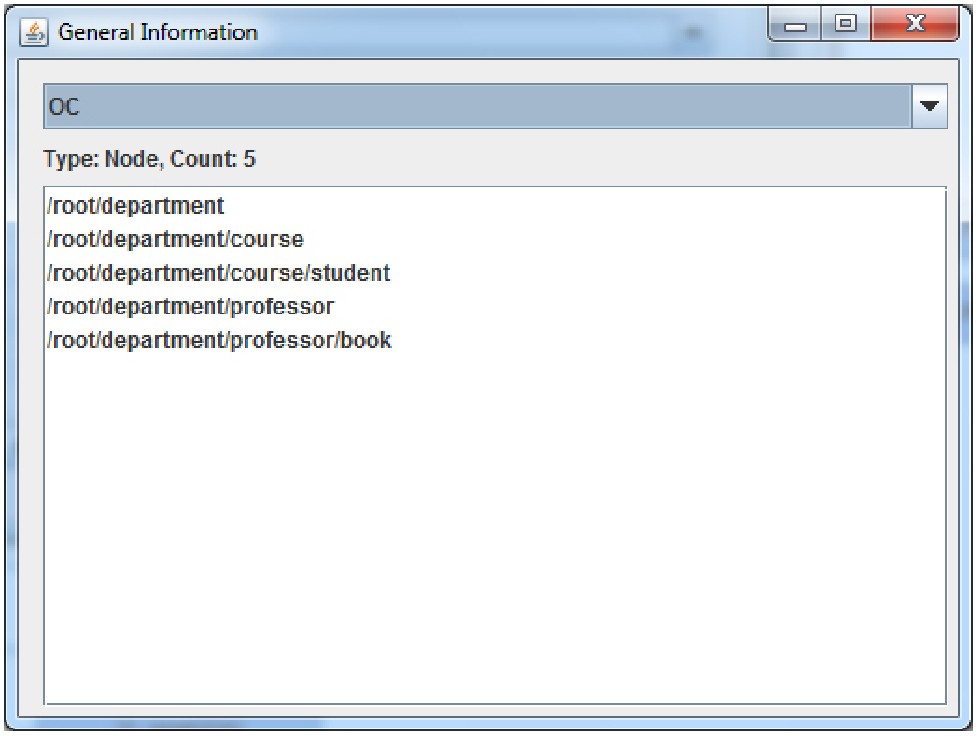
Node Type:	OC
Node Name:	professor
OID:	p_name
Object Attribute:	office, email
Dependent OC:	
Related OCs:	
Relationship Attributes:	
Component Attributes:	
Aggregated Nodes:	

Figure 6.3: Node information.

When a user clicks on any node in Fig. 6.2, a new window showing the detailed information of the node will pop up, as shown in Fig. 6.3. In the figure, detailed information about the node is displayed, including node type, node name, OID (if any), Object Attribute (if any), etc. Note that some information is not available for a specific node, so the information will be grey.

Users can also change the node type discovered by the system if it is not properly identified. Users can easily do it by choosing another type from the drop down menu on the top of Fig. 6.3 and click the ‘Update’ button at the bottom. E.g., if a node is classified to be an OC (Object Class) by mistake, users can change it to some other node types, like Composite Attribute, etc.

Besides, users can also see all nodes being identified as a particular node type by clicking the ‘General Information’ button at the bottom of Fig. 6.3, then Fig. 6.4 will pop up. Other nodes of the same node type are shown in the window.

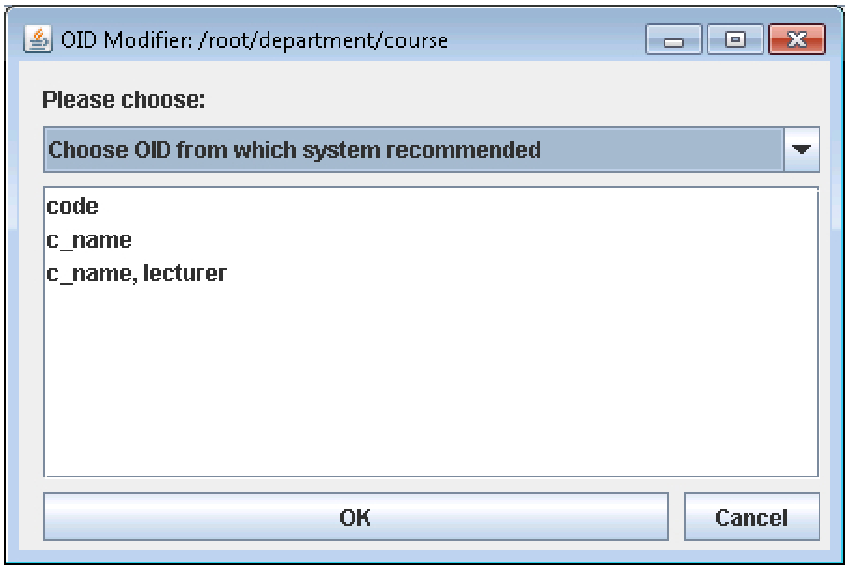

General Information

OC

Type: Node, Count: 5

/root/department
/root/department/course
/root/department/course/student
/root/department/professor
/root/department/professor/book

Figure 6.4: General information.


OID Modifier: /root/department/course

Please choose:

Choose OID from which system recommended

code
c_name
c_name, lecturer

Figure 6.5: Dialog for modifying Object ID.

Users can click the ‘Modify OID’ button in Fig. 6.3 for the case when an OID is wrongly identified. Then users can choose a new OID in Fig. 6.5. Users can choose one or multiple node from all EDLNs (exclusive descendant leaf nodes) of an object class. To further enhance user experience, we also provide all candidate OIDs which is discussed in Section 4.4.1 to the users, as shown in Fig. 6.5.

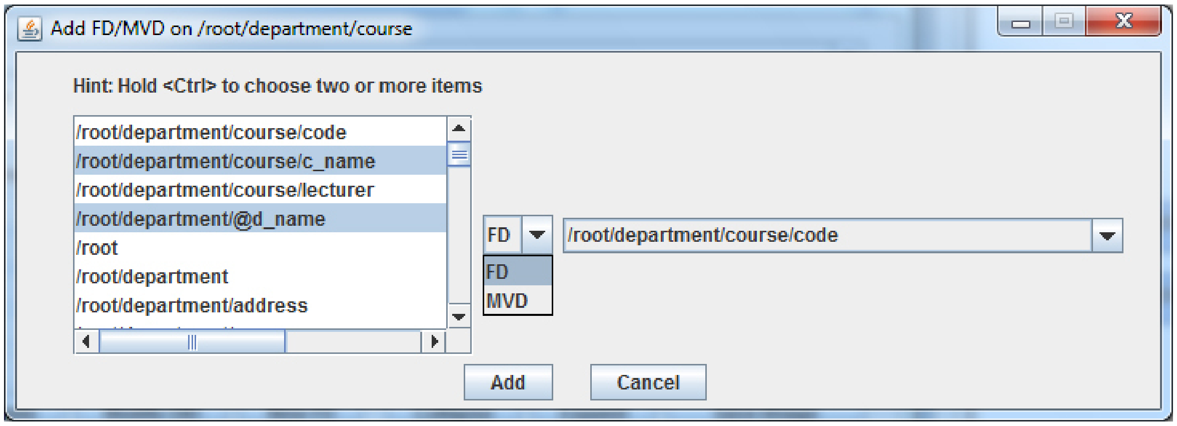

Add FD/MVD on /root/department/course
Hint: Hold <Ctrl> to choose two or more items
/root/department/course/code
/root/department/course/c_name
/root/department/course/lecturer
/root/department/@d_name
/root
/root/department
/root/department/address
FD
FD
MVD
/root/department/course/code
Add
Cancel

Figure 6.6: Dialog for adding new FD/MVD.

Sometimes users may want to add FDs/MVDs to the system. Users can simply click the ‘New FD’ button in Fig. 6.3 to add in new FDs/MVDs. Fig. 6.6 shows the dialog for adding FDs/MVDs. At the left side of the dialog, users choose the left-hand-side (LHS) nodes for a FD/MVD. To choose more than one node, users have to press the ‘Control’ button on the keyboard to select the second node onwards. In the middle of the window, users choose whether a FD or a MVD is added to the system. At the right side of the dialog, users can choose the right-hand-side (RHS) node for the FD/MVD. After that, users can click the ‘Add’ button to update the FD/MVD into the system.

6.4 Chapter Summary

In this chapter, we introduced a user-friendly system for discovering ORA-semantics for XML. This system is built based on our rule-based semantics discovery approach for XML discussed in Chapter 4. We showed a step-by-step tutorial to illustrate how users input a XML data file with/without its corresponding XML schema file (in DTD or XSD); how the discovered ORA-semantics is displayed to users; and how users can interact with our system to improve our accuracy of ORA-semantics discovery from XML.

CHAPTER 7

FUTURE WORK: SEMANTIC-BASED XML SCHEMA INTEGRATION AND DATA INTEGRATION

7.1 Introduction

XML document has been frequently created and exchanged by business and enterprise, and there is an increasing need for accurate and efficient XML schema integration and data integration. Moreover, with more and more XML documents being generated, heterogeneous data sources may need to be integrated for centralized management. With a general unified query interface for all heterogeneous data sources, users can easily access the information with information from different data sources and with redundancies being processed and removed. All these make XML schema integration and data integration become an important topic.

In the following, after reviewing some existing works on schema integration and data integration for both relational database and XML database. We proposed the framework of a step-by-step semantics-based approach for XML schema integration and data integration, which fully considers the ORA-semantics discovered in this thesis (Chapter 4) and uses them to help increasing the efficiency or effectiveness of the XML schema integration and data integration.

7.2 Existing Works

Schema integration has been well studied in the last decade. For relational database, a detail and comprehensive survey [6] has been done to analyze the methodologies for database schema integration. In [32, 33], the authors pointed out that many existing approaches such as [5, 57] lose valuable semantics information or FDs (functional dependencies) during their integrations because of the structural conflicts and constraint conflict they will encounter during the integration. To be more precise, an entity type in one schema may be modeled as a relationship type in another schema. If they integrate them together by transforming the relationship type into an entity type, lot of information will be lost because they split a relationship type into two or more. The semantics, cardinalities, identifier of these new relationship types will be uncertain, and even some FDs imposed in the previous relationship type will be lost. Thus, conflict resolution is an important step, which should be considered by any schema integration and data integration approach.

For XML database, many XML schema matching and XML schema integration approaches have also been proposed. Some of them use linguistic and hierarchical structural information to match and integrate among element nodes in XML schemas [35, 46]; some of them also use information from XML data [10, 1]; some of them use Natural Language Processing (NLP) techniques to extract useful information [42, 11]; some of them match and integrate element nodes by representing them using ontology model [18, 62].

However, none of these approaches consider the ORA-semantics discovered in Chapter 4, especial the ORA-semantic concepts such as object class, relationship type, object attribute and relationship attribute. Without considering these ORA-semantics, they may wrongly match and integrate different element nodes together with high linguistic similarity/hierarchical structural similarity, but actually with totally different semantics, such as object attribute and relationship attribute.

7.3 Semantics-based XML Schema Integration and Data Integration

The general process of our approach, and the ORA-semantics needed in each step, are shown in Fig.7.1. In the following, we will briefly describe each step separately.

7.3.1 Schema Fragmentation & Object Classes Matching

Firstly, we fragment all local XML schemas into substructures using the object classes discovered in Chapter 4. We match among these substructures, each of which represents an object class. In order to match among the object classes, we can use the structural information (e.g., parent/ancestor/sibling/child/descendant nodes, etc.), linguistic information (e.g., tag name similarity, synonyms, homonyms, etc.) and ontology information (i.e., the meaning of the tag names) of each internal node representing an object class. Also many matchers proposed in some existing schema matching approaches [3, 20] can be adopted here.

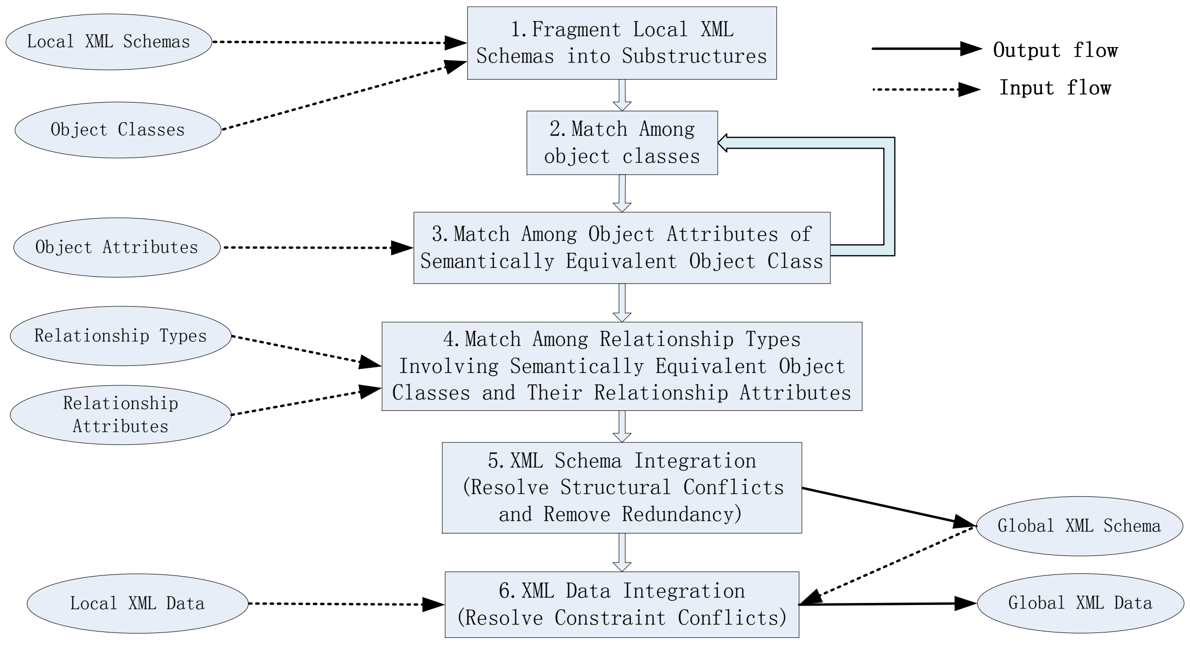

Local XML Schemas
Object Classes
Object Attributes
Relationship Types
Relationship Attributes
Local XML Data
1.Fragment Local XML Schemas into Substructures
2.Match Among object classes
3.Match Among Object Attributes of Semantically Equivalent Object Class
4.Match Among Relationship Types Involving Semantically Equivalent Object Classes and Their Relationship Attributes
5.XML Schema Integration (Resolve Structural Conflicts and Remove Redundancy)
6.XML Data Integration (Resolve Constraint Conflicts)
Output flow
Input flow
Global XML Schema
Global XML Data

Figure 7.1: General process of for XML schema integration and data integration

Furthermore, we need a score function to calculate similarities among object classes and a threshold, which can be adjusted by users, to determine whether the quality of a matching is good enough. Only those object classes with higher similarity than the threshold will be considered matched.

7.3.2 Object Attribute Matching

Next step is to match among object attributes of those semantically equivalent[1] object classes, which have been matched in Section 7.3.1. An object attribute *a* of object class *O* can be matched with object attribute *a'* of object class *O'* only when *O* and *O'* are matched in the previous step. Thus, given *O* matches with *O'*, for each object attribute of *O*, we only compare it with object attributes of *O'*, rather than each node in the local XML schema as they do in other approaches.

This makes our approach more efficient than the others. However, when object

[1]Two ORA-semantic concepts are semantically equivalent means they are representing the same thing/concept and should be matched and integrated together.

classes are wrongly matched together, all their object attributes will be matched wrongly. To resolve this problem, as shown in Fig. 7.1, we can use the matchings results of object attributes to refine the matchings of object classes. To be more precise, given two sets of object attributes, if they can hardly matched together (i.e., with low similarities), we know the object classes they belong to may be wrongly matched.

7.3.3 Relationship Type Matching & Relationship Attribute Matching

In Chapter 4, we identified the relationship types among object classes with their degrees, participating object classes, and their relationship attributes. To match among relationship types, we also need to know their semantic meanings when two or more relationship types involve semantically equivalent object classes. To identify whether they should be matched together, we consider the similarity between their relationship attributes. In case of both relationship types have relationship attributes, there are three kinds of correlation between them: (1) positive correlated; (2) negative correlated; (3) not correlated. If the relationship attributes are positively correlated, the corresponding relationship types should be matched together, while if the relationship attributes are negatively correlated or not correlated, the corresponding relationship types should not be matched together.

For example, If two relationship types with their participating object classes having been matched with each other, and they both have a relationship attribute *BorrowData* and *PurchaseData* respectively. By analyzing the correlation between these two relationship attributes, we know their corresponding relationship types represent different semantic meanings and should not be matched together. In order to identify the correlations information, we can use some existing lexical databases, such as WordNet mentioned in Chapter 4.

7.3.4 Schema Integration & Structural Conflict Resolution

After matching object classes, relationship types and their attributes, we merge them together and form a global integrated schema. During the integration, we may encounter different structural conflicts, such as:

Object Attribute vs. Object Class Conflict The same concept may be modeled as an object attribute in one schema, and be modeled as an object class in another schema. This conflict can be resolved by transforming the object attribute to an object class.

Generalizations vs. Specializations Conflict It happens when an object class in one schema is a general concept of an object class in another schema. This conflict can be resolve by including the generalization *ISA* hierarchy in the global integrated schema.

There are many other structural conflicts such as ancestor-descendant conflict, relationship type conflict (i.e., conflict among different relationship types with different degree), etc. These conflicts have been well studied and resolved in [70].

7.3.5 Data Integration & Constraint Conflict Resolution

During data integration, we may also encounter many constraint conflicts, such as domain constraint conflicts (domain mismatch), cardinality conflicts, etc.

Domain Mismatch It means the domains of two matched element nodes are not equivalent, i.e., their domain are with set relations: *SUBSET*, *OVERLAP* or *DISJOINT*, rather than *EQUAL*. With different set relations, we need to handle it differently to avoid or reduce losing information.

Cardinality Conflicts It means the cardinalities of matched element nodes are not consistent, an attribute as single value attribute in one local schema and as a multi-valued attribute in another local schema.

There are also many other constraint conflicts caused by partial/inconsistent information, caused by local OID or local FDs/MVDs. For more details about these constraints, please refer to those research works on resolving constraint conflicts for relational database [32, 33, 40]. They can be adopted for XML database in our semantic-based approach for XML schema and XML data integration.

7.4 Chapter Summary

In this chapter, we proposed a step-by-step semantic-based approach for XML schema integration and data integration. Our approach fully considers the ORA-semantics discovered in this thesis (Chapter 4) and uses them to help increasing the efficiency and effectiveness of integration process. Based on the object classes identified from local schemas, we fragment the local schemas into subtrees, each of which represents an object class. We match these object classes based on their linguistic/structural information, and then match their object attributes. The matching results for object attributes can also be used to refine the matching of the object classes they belong to. Furthermore, we only match the relationship types whose participating object classes have been matched together correspondingly. By using object class as basic unit for comparison and matching, our schema integration approach largely reduce the searching space for matching process.

As discussed before, there are still many challenges we may encounter during the integration process, such as how to determine whether two object classes is correctly matched together. If we match two object classes wrongly, we will also wrongly match their object attributes and relationship types they participate in. That is why we introduce the feedback/refinement process so that if most of the object attributes of two object classes can only be matched together with low similarity, we use this information to refine the matching of the corresponding object classes.

Furthermore, during the schema integration and data integration, we also need to consider the structural conflicts and constraints conflicts we may encounter. The resolutions for these conflicts can be adopt and modified from the corresponding resolutions for schema integration and data integration for relational database.

CHAPTER 8

CONCLUSION

The availability of a conceptual XML schema for a given XML database constitute invaluable leverage for improving the effectiveness or efficiency of many XML applications including XML query processing, XML keyword search as well as XML schema integration and data integration. However, XML data and XML schema are instances and schemas of a logical model that fail to explicitly represent the intended semantics.

In order to capture and discover the semantic information underlies the XML schema and XML data, we formally defined them as ORA-semantics. We also define each semantic concept included in ORA-semantics as an ORA-semantic concept (i.e., *object class, object identifier (OID), object attribute, explicit relationship type, implicit relationship type, relationship attribute, aggregational node, role name, composite attribute, dependent object class and identifier dependent (IDD) relationship type*). We have shown that many ORA-semantic concepts such as object class, OID, (implicit/explicit) relationship type, relationship attribute, etc., are proposed based on the corresponding semantic concepts captured and represented in ER model and ORA-SS. However, no existing approach can discover all ORA-semantics directly from XML data and XML schema.

In this thesis we have presented a rule-based semantics discovery approach to discover ORA-semantics implicitly embedded in XML. The input of our rule-based approach is XML data with/without XML schema, which can be extracted from XML data. Our rule-based approach leverages a set of classification rules based on properties and heuristics of different ORA-semantic concepts to identify object classes, role name, explicit relationship types, aggregational nodes and composite attributes from internal nodes of an XML schema tree. Our rule-based approach uses properties, heuristics and statistic information based on our observations from existing XML data to identify OID for each object class. Identified OIDs are used to distinguish between object attributes and relationship attributes from leaf nodes of an XML schema tree. Our rule-based approach can also discover the implicit relationship types among object classes.

We have empirically evaluated the effectiveness of our rule-based approach using 15 real world data-centric XML datasets. The experiments showed that our rule-based approach can achieve almost 95% overall precision, recall and F-measure. We also showed that in the application of XML keyword search, even with some ORA-semantic concepts being wrongly identified (such as dependent object class or composite attribute being wrongly identified as object class), the XML keyword search approach can still work no worse than those approaches without considering any ORA-semantics.

Based on our rule-based semantics discovery approach, we have developed a demonstration system. Given a XML data with/without its XML schema, our system can discovery the ORA-semantics and present it to users in a user-friendly way. Users can also interact with the system through feedback to achieve higher accuracy for discovering the ORA-semantics.

BIBLIOGRAPHY

[1] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang Chiew Tan. Designing and refining schema mappings via data examples. In *SIGMOD Conference*, pages 133–144, 2011.

[2] Martin Andersson. Extracting an entity relationship schema from a relational database through reverse engineering. In *ER*, pages 403–419, 1994.

[3] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *SIGMOD Conference*, pages 906–908, 2005.

[4] Zhifeng Bao, Jiaheng Lu, Tok Wang Ling, Liang Xu, and Huayu Wu. An effective object-level xml keyword search. In *DASFAA (1)*, pages 93–109, 2010.

[5] Carlo Batini and Maurizio Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Trans. Software Eng.*, 10(6):650–664, 1984.

[6] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[7] Sean Bechhofer. Web ontology language (owl) reference version 1.0. *W3C. Technical report*, 2004.

[8] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *VLDB J.*, pages 34–43, 2001.

[9] Philip A. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst.*, 1(4):277–298, 1976.

[10] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In *ICDE*, pages 69–80, 2005.

[11] Philip Bohannon, Eiman Elnahrawy, Wenfei Fan, and Michael Flaster. Putting context into schema matching. In *VLDB*, pages 307–318, 2006.

[12] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (xml) 1.0. *W3C. Technical report*, 1997.

[13] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (xml) 1.0. 2nd edition. *W3C. Technical report*, 2000.

[14] Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0: Rdf schema. *W3C. Technical report*, 2004.

[15] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.

[16] Ya Bing Chen, Tok Wang Ling, and Mong-Li Lee. Designing valid XML views. In *ER*, pages 463–478, 2002.

[17] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[18] Isabel F. Cruz, Huiyong Xiao, and Feihong Hsu. An ontology-based framework for xml semantic integration. In *IDEAS*, pages 217–226, 2004.

[19] Alin Deutsch, Mary F. Fernández, and Dan Suciu. Storing semistructured data with stored. In *SIGMOD Conference*, pages 431–442, 1999.

[20] Robin Dhamankar, Yoonkyong Lee, Anhai Doan, Alon Halevy, and Pedro Domingos. iMAP: discovering complex semantic matches between database schemas. In *in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, ACM.* Press, 2004.

[21] AnHai Doan, Raghu Ramakrishnan, Fei Chen 0002, Pedro DeRose, Yoonkyong Lee, Robert McCann, Mayssam Sayyadian, and Warren Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1):64–72, 2006.

[22] Gillian Dobbie, Xiaoying Wu, Tok Wang Ling, and Mong Li Lee. ORA-SS: An object-relationship-attribute model for semistructured data. In *TR21/00 Technique Report*, 2000.

[23] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. Discovering conditional functional dependencies. *IEEE Trans. Knowl. Data Eng.*, 23(5):683–698, 2011.

[24] Ling Feng, Elizabeth Chang, and Tharam S. Dillon. A semantic network-based design methodology for xml documents. *ACM Trans. Inf. Syst.*, 20(4):390–421, 2002.

[25] Nir Friedman, Dan Geiger, and Moisés Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.

[26] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, pages 436–445, 1997.

[27] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD Conference*, pages 16–27, 2003.

[28] Jean-Luc Hainaut. Database reverse engineering: Models, techniques, and strategies. In *ER*, pages 729–741, 1991.

[29] Jan Hegewald, Felix Naumann, and Melanie Weis. Xstruct: Efficient schema extraction from multiple and large XML documents. In *ICDE Workshops*, page 81, 2006.

[30] Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. *W3C. Technical report*, 2004.

[31] Thuy Ngoc Le, Huayu Wu, Tok Wang Ling, and Luochen Li. From revisiting the LCA-based approach to a new semantics-based approach for XML keyword search. *Technical Report TRB5/11, School of Computing, National University of Singapore, May 2011*, 2011.

[32] Mong-Li Lee and Tok Wang Ling. Resolving structural conflicts in the integration of entity relationship schemas. In *OOER*, pages 424–433, 1995.

[33] Mong-Li Lee and Tok Wang Ling. Resolving constraint conflicts in the integration of entity-relationship schemas. In *ER*, pages 394–407, 1997.

[34] Mong-Li Lee, Tok Wang Ling, and Wai Lup Low. Designing functional dependencies for XML. In *EDBT*, pages 124–141, 2002.

[35] Mong-Li Lee, Liang Huai Yang, Wynne Hsu, and Xia Yang. XClust: clustering XML schemas for effective integration. In *CIKM*, pages 292–299, 2002.

[36] Tok Wang Ling and Mong-Li Lee. Relational to er schema translation using semantic dependencies and inclusion dependencies. In *Journal of Integrated Computer-Aided Engineering*, 1994.

[37] Tok Wang Ling and Mong-Li Lee. Semantic dependencies in data modelling and database reverse engineering. In *Int. Symp. on Advanced Database Technologies and their Integration*, pages 47–54, 1994.

[38] Tok Wang Ling and Mong Li Lee. Relational to entity-relationship schema translation using semantic and inclusion dependencies. *Integr. Comput.-Aided Eng.*, 2(2):125–145, June 1995.

[39] Tok Wang Ling, Mong Li Lee, and Gillian Dobbie. *Semistructured Database Design*. Springer, 2005.

[40] Mengchi Liu and Tok Wang Ling. A data model for semistructured data with partial and inconsistent information. In *EDBT*, pages 317–331, 2000.

[41] Ziyang Liu and Yi Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD Conference*, pages 329–340, 2007.

[42] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Y. Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68, 2005.

[43] Jakub Marciniak. Xml schema and data summarization. In *ICAISC (2)*, pages 556–565, 2010.

[44] Victor M. Markowitz and Johann A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Trans. Software Eng.*, 16(8):777–790, 1990.

[45] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[46] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, pages 117–128, 2002.

[47] Satoshi Mizuta and Keishin Hanya. Specifications of word set in linguistic approach for similarity estimation. In *BICoB*, pages 25–29, 2010.

[48] Shamkant B. Navathe and A. M. Awong. Abstracting relational and hierarchical data with a semantic data model. In *ER*, pages 305–333, 1987.

[49] 2002 Noam Chomsky. *Syntactic Structures*. Springer, 2002.

[50] Jean-Marc Petit, Farouk Toumani, Jean-François Boulicaut, and Jacques K-ouloumdjian. Towards the reverse engineering of denormalized relational databases. In *ICDE*, pages 218–227, 1996.

[51] Sven Puhlmann, Melanie Weis, and Felix Naumann. XML duplicate detection using sorted neighborhoods. In *EDBT*, pages 773–791, 2006.

[52] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[53] Leonardo Ribeiro and Theo Härder. Entity identification in XML documents. In *Grundlagen von Datenbanken*, pages 130–134, 2006.

[54] Hang Shi, Toshiyuki Amagasa, and Hiroyuki Kitagawa. Fast detection of functional dependencies in XML data. In *XSym*, pages 113–127, 2010.

[55] Peretz Shoval and Nili Shreiber. Database reverse engineering: From the relational to the binary relationship model. *Data Knowl. Eng.*, 10:293–315, 1993.

[56] Stefano Spaccapietra, Ecole Polytechnique Fdrale, and Christine Parent. ER-C+: an object based entity relationship approach, 1992.

[57] Stefano Spaccapietra, Christine Parent, and Yann Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB J.*, 1(1):81–126, 1992.

[58] Amanda Spink. A user-centered approach to evaluating human interaction with web search engines: an exploratory study. *Inf. Process. Manage.*, 38(3):401–426, 2002.

[59] Chong Sun, Chee Yong Chan, and Amit K. Goenka. Multiway SLCA-based keyword search in XML data. In *WWW*, pages 1043–1052, 2007.

[60] Yongchuan Tang and Jiacheng Zheng. Linguistic modelling based on semantic similarity relation among linguistic labels. *Fuzzy Sets and Systems*, 157(12):1662–1673, 2006.

[61] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. Xml schema part 1: Structures. *W3C. Technical report*, 2001.

[62] Oguzhan Topsakal and Joachim Hammer. Schema matching with report analysis. In *VLDB*, 2005.

[63] Melanie Weis and Felix Naumann. Detecting duplicate objects in XML documents. In *IQIS*, pages 10–19, 2004.

[64] Melanie Weis and Felix Naumann. Dogmatix tracks down duplicates in xml. In *SIGMOD Conference*, pages 431–442, 2005.

[65] Dongrui Wu and Jerry M. Mendel. A vector similarity measure for linguistic approximation: Interval type-2 and type-1 fuzzy sets. *Inf. Sci.*, 178(2):381–402, 2008.

[66] Huayu Wu and Zhifeng Bao. Object-oriented XML keyword search. In *ER*, pages 402–410, 2011.

[67] Huayu Wu, Tok Wang Ling, and Bo Chen. VERT: A semantic approach for content search and content extraction in XML query processing. In *ER*, pages 534–549, 2007.

[68] Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD Conference*, pages 537–538, 2005.

[69] Yu Xu and Yannis Papakonstantinou. Efficient lca based keyword search in XML data. In *EDBT*, pages 535–546, 2008.

[70] Xia Yang, Mong-Li Lee, and Tok Wang Ling. Resolving structural conflicts in the integration of XML schemas: A semantic approach. In *ER*, pages 520–533, 2003.

[71] Cong Yu and H. V. Jagadish. Efficient discovery of XML data redundancies. In *VLDB*, pages 103–114, 2006.

[72] Cong Yu and H. V. Jagadish. Schema summarization. In *VLDB*, pages 319–330, 2006.

[73] Cong Yu and H. V. Jagadish. XML schema refinement through redundancy detection and normalization. *VLDB J.*, 17(2):203–223, 2008.

[74] Xiaohua Zhou, Xiaodan Zhang, and Xiaohua Hu. Maxmatcher: Biological concept extraction using approximate dictionary lookup. In *PRICAI*, pages 1145–1149, 2006.