A Semantics-preserving Approach for Extracting RDF Knowledge from Object-oriented Databases

Jing Shan[1], Jiawen Lu[1,†], Xu Chen[2], Li Yan[1] and Zongmin Ma[1,3,*]

[1]*Nanjing University of Aeronautics and Astronautics, China*
[2]*North Minzu University, China*
[3]*Collaborative Innovation Center of Novel Software Technology and Industrialization, China*
E-mail: zongminma@nuaa.edu.cn
[*]*Corresponding Author*
[†]*Co-first Author*

Received 14 February 2022; Accepted 24 February 2023;
Publication 16 June 2023

Abstract

The Resource Description Framework and RDF Schema recommended by the World Wide Web Consortium provide a flexible model for semantically representing information about resources on the Web, which are playing an increasingly important role in intelligent processing of large-scale data. With the widespread acceptance and applications of RDF(S), construction of RDF(S) is of increasing importance. Automatic construction of RDF(S) with diverse data has attracted more attention. In this paper, we propose a novel approach for constructing an RDF(S) with object-oriented databases that are suitable for non-traditional applications. We propose the formal rules of mapping an object-oriented database model into a RDF(S) model based on the formal definitions of these two models. We develop a tool named OODB2RDF to verify our approach.

Keywords: RDF(S), knowledge extraction, object-oriented databases, db4o, mapping.

Journal of Web Engineering, Vol. 22_2, 197–220.
doi: 10.13052/jwe1540-9589.2221

1 Introduction

The Semantic Web [1] is a framework that provides data with semantic meaning (through metadata) and enables machines to consume, understand, and reason about the structure and purpose of data. The core of the Semantic Web is built on an RDF (Resource Description Framework)[1] and OWL (Web Ontology Language).[2] RDF and RDF Schema (RDF(S)) provide a flexible and concise model for representing metadata of resources on the Web. An RDF with several encoding formats (e.g., Turtle, N-Triples, JSON, N-Quod RDF) can represent structured and unstructured data, and has been the *de-facto* standard for representation and exchange of information. Nowadays, the RDF model is finding increasing use in a wide range beyond data management [2]. Typically, knowledge graphs, which are widely investigated and applied in various domains, adopt an RDF model as their infrastructure for the semantic and intelligent processing of large-scale data [3]. In this context, one of the most crucial issues is to automatically construct RDF(S) from various data resources [6].

There have been many efforts in constructing RDF(S) from diverse data resources (e.g., text [13], semi-structured documents [5], UML class diagram model [7], XML model [4], spreadsheet [8], and relational databases [9, 10]). Most of the existing Web sites derive their data from relational databases (RDBs) [11] and the legacy data in the RDBs are still in a preponderant position. Note that the RDBs are not enabled for data modeling in many non-traditional applications (e.g., CAD/CAM and knowledge-based systems) due to their flat two-dimensional table structure. As a result, object-oriented databases (OODBs) and object relational databases (ORDBs) have been proposed [12].

The OODB model (OODM) integrates the powerful modeling capabilities of the object-oriented paradigm into a database model. Compared with the RDB model, the OODM provides a powerful object-oriented modeling capability and OODBs have been applied in many complex application domains (e.g., industrial application [23]). In the context of knowledge engineering, OODBs have been applied to automatically construct ontologies [21] and store ontologies [24], respectively. With the utilization of OODBs and the availability of data in the OODBs, constructing the RDF(S) with the OODBs becomes important. Currently, many efforts have been devoted to the construction of RDF(S) with diverse data models. Unfortunately, the work on

[1] https://www.w3.org/RDF/

[2] http://www.w3.org/2004/OWL/

the construction of RDF(S) with OODBs is still scarce. In [25], a formal approach for mapping an OODB into an RDF(S) is presented and several simple mapping rules are developed. However, the presented approach is not implemented and verified. More importantly, the approach presented in [25] cannot cover a complete mapping of OODBs to RDF(S).

In this paper, we introduce the formal definitions for the OODB model and RDF(S) model. On this basis, we propose complete mapping rules to convert the OODM to RDF(S). Furthermore, we develop a tool named OODB2RDF to implement our approach based on db4o [20], an open-source object-oriented database engine. We verify our approach with experiments. The experimental results show that our approach can convert object-oriented data into RDF(S) and preserve maximum semantics in the OODBs.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 introduces the formal definitions of the OODB model and the RDF(S) model. Section 4 proposes the mapping rules that convert the OODBs into the RDF(S). We implement and verify our approach in Section 5. Section 6 concludes this paper.

2 Related Work

Automatic construction of RDF(S) has been extensively investigated in the context of RDBs. The W3C RDB2RDF Working Group proposes two standards, which are *direct mapping*[3] and *R2RML*,[4] to standardize the implementation of RDB2RDF. The direct mapping is an approach for directly mapping RDBs to an RDF, where relational tables are mapped to classes in an RDF vocabulary and the attributes of the tables are mapped to properties in the vocabulary. A specification inspired by the direct mapping is proposed in [10]. The specification uses the OWL2 vocabulary to map the RDBs directly to RDF graph and mainly focuses on four aspects: *monotonicity*, *information preservation*, *query preservation* and *semantics preservation*. R2RML is a standard language to describe RDB2RDF mapping files. Unlike direct mapping, R2RML is highly customizable and flexible, which allows calculation, processing, filtering, clarity, and integration of data in the RDB before the RDF is generated. Also, in [10], existing mapping languages are compared and divided into four categories, and recommendations for selecting a mapping language provided.

[3] https://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/
[4] http://www.w3.org/TR/2012/REC-r2rml-20120927/

There are some prototype systems or tools that can convert RDBs to an RDF, such as *Triplify* [14], RDB2OWL [15] and D2RQ [16]. D2RQ is a mapping language and platform for publishing relational databases as virtual RDF graphs. It uses SPARQL to access non-RDF database content as relational data over the Web, and then converts the RDB content into the RDF format. The SPARQL/Update, an extension of SPARQL to support update over RDF graphs, was introduced in [16]. Extending *D2RQ* to support SPARQL/Update statements is an important step towards the creation of a real-write Semantic Web.

In addition to the RDBs, there are some efforts in handling RDF(S) by object-oriented techniques. In [17], a mapping approach embeds Semantic Web data into object-oriented languages. The mismatch between object-oriented programming languages and the Semantic Web data is discussed and an object-oriented API for managing RDF data is proposed. Actually, much work focuses on combining object-oriented technology with OWL. In [18], a production-rule-based system O-DEVICE is proposed, which can handle OWL semantics following an object-oriented approach. In [19], the authors aim to infer and materialize in advance as many properties for OWL instances as possible under the semantics of OWL constructors. The issues of the construction and storage of OWL ontologies with object-oriented databases are investigated in [21] and [24], respectively. Without implementation and verification, several simple mapping rules for converting the OODBs into the RDF(S) are developed in [25]. So far, few works carry out automatic construction of the RDF(S) with OODBs. Being different from [25], in this paper, we propose a complete approach for constructing the RDF(S) with OODBs and implements a tool to verify the proposed approach with experiments.

3 Formal Descriptions of OODBs and RDF(S)

3.1 Formal Definition of the OODB Model

The OODBs defined in [12] are databases that integrate an object orientation with database capabilities. An OODB model consists of objects, classes, attributes, methods, inheritance relationships and association relationships. A real-world entity can be represented by only one modeling concept. An object defined uniquely by a system-defined identifier (OID) includes object presentation, attributes, methods, and relationships with other objects. Property values, methods, and relationships for an object can change, but

identifiers are unique. A class is a collection of objects with the same characteristics and methods [21]. The concept of class provides a uniform abstract description for all objects which belong to this class. Methods are the operations that are defined on an object, which are part of the definition of the object. The attributes form the data structure of an object and the range of values for an attribute can be any class or specific value.

Classes can be organized in class inheritance relationships. A class can have any number of subclasses. These subclasses inherit all the attributes and methods of this class (i.e., superclass) without having to rewrite the original class and defining some new attributes and methods. An object belonging to a subclass must belong to its superclass. In some systems, a class may have more than one superclass, while in others it is restricted to be only one superclass. An aggregation is used to represent a part–whole structural relationship among different classes. It connects relatively independent classes together. Unlike the inheritance relationships, the aggregation relationships can be represented by the properties of classes. When there is a fixed correspondence relationship between two classes, there is a correlation between them.

According to the formal definition in [21], we present a formal representation of an OODB model as follows.

Definition 1: An OODB model is a finite set of class declaration, and each class declaration describes a set of objects sharing a collection of features. An OODB model is a six-tuple $S = (D_S, C_S, A_S, I_S, AG_S, O_S)$.

(1) D_S: a collection of attribute type names (e. g. *String* and *Integer*).
(2) C_S: a collection of class names.
(3) A_S: a collection of attribute names. Symbol *A* is applied for attributes. $A_S = A_{SS} \cup A_{SC}$ represents that an attribute can be either a simple attribute A_{SS} when its domain is a type $D \in D_S$ or a complex attribute A_{SC} when its domain is a class $C \in C_S$.
(4) I_S: a finite set of inheritance relationship between classes. I_{C1} represents the inheritance relationship of C_1 $I_{C1} = \{x | x$ is the superclass of $C_1\}$, where $C_1 \in C_S, x \in C_S I_{c1} \in I_S$.
(5) AG_S: a finite set of aggregation relationship. AG_{C1} represents the aggregation relationship of C_1. $AG_{C1} = \{(x, y) | x$ is the class which has aggregation relationship with C_1, y is the name of the aggregation relationship$\}$ $C_1 \in C_S$, $x \in C_S$, $AG_{C1} \in AG_{CS}$.
(6) O_S: the collection of all instances stored in the OODBs. Each instance has a unique object identifier (OID) and the value of OID is determined by the OODM.

S = (D_S, C_S, A_S, I_S, AG_S, O_S), where $A_S = A_{SS} \cup A_{SC}$
D_S ={*String, Integer*};
C_S ={*Person, Teacher, Professor, Student, AssProfessor, Course, Postgraduate, Undergraduate*};
A_{SS} ={*name, age, email, title, grade, major, college, location, department*};
A_{SC} ={manage, enroll, teach};
I_S={
$I_{Teacher}$={*Person*}, $I_{Student}$={*Person*}, $I_{Postgraduate}$={*Student*}, $I_{Undergraduate}$={*Student*},
$I_{Professor}$={*Teacher*}, $I_{AssProfessor}$={*Teacher*}
}
AG_S ={
$AG_{Professor}$ = {(*Postgraduate, manage*)}, AG_{Course} = {(*Student, enroll*)},
$AG_{AssProfessor}$ = {(*Course, teacher*)}
}

Figure 1 An OODM S_1.

Let us look at an example. Suppose that we have eight classes: *Person* {name: String, age: Integer, email: String}; *Course* {enroll: Student, location: String}; *Teacher* is the subclass of *Person* {title: String}; *Student* is the subclass of *Person* {grade: String}; *Postgraduate* is the subclass of *Student* {major: String}; *Undergraduate* is the subclass of *Student* {college: String}; *Professor* is the subclass of *Teacher* {manage: Postgraduate}; *AssProfessor* is the subclass of *Teacher* {department: String, teach: Course}. Suppose that we have an OODB model S_1 as shown in Figure 1.

(1) There are eight classes stored in the OODBs and they are expressed in the OODM as $C_S = \{$*Person*, *Teacher*, *Professor*, *Student*, *AssProfessor*, *Course*, *Postgraduate*, *Undergraduate*$\}$. Among them, there are six inheritance relationships (e.g., "*class Teacher is the subclass of Person*"). The inheritance relationship is expressed as $I_{Teacher} = \{$*Person*$\}$. The class *Person* has is an attribute "name" with type *String*.
(2) A_{SC} represents the collection of complex attributes. For example, attribute "manage" of the *Professor* class is a complex attribute because its type is the *Postgraduate* class. We represent this relationship in OODM as $AG_{Professor} = \{(Postgraduate, manage)\}$.
(3) A_{SS} represents the set of simple attributes. For example, attributes "*name*" and "*age*" of the Person class are two simple attributes.

The instances of the OODB model S_1 are shown in Figure 2.

It is shown in Figure 1 that the *Student* class is the superclass of the *Undergraduate* and *Postgraduate* class. Then instance O_1 of the *Undergraduate* class and instance O_1'' of the *Postgraduate* class are both the instances of the *Student* class. The *Professor* class is the subclass of the *Teacher* class,

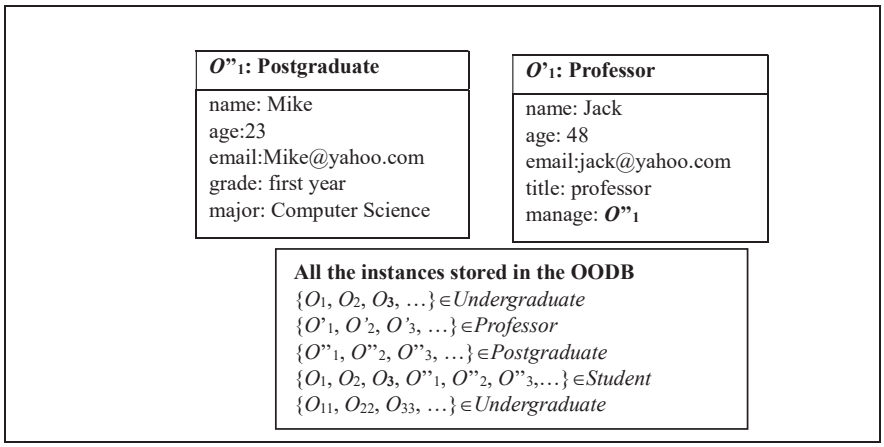

Figure 2 Instances of the OODM S_1.

which is the subclass of the *Person* class as well. So, an instance O'_1 of the *Professor* class has all attributes of classes *Person* and *Teacher*. The *Professor* class has a complex attribute "*manage*" with attribute type of the *Postgraduate* class. So, we can refer that the value of the attribute "*manage*" is an instance of the *Postgraduate* class.

3.2 Formal Definition of RDF(S)

Being the basic composition unit of RDF model, an RDF statement is represented by a triple, which is composed of *subject*, *predicate* and *object*. A subject is a declaration of a resource, a predicate is used to describe various properties of subjects, and an object is the value of a property. Note that a property value can be a literal or another resource. When the value of property is a resource, the property can be viewed as a relationship between two resources. Everything that is identified as a uniform resource identifier (URI) on the Web can be regarded as a resource, even if it cannot be retrieved directly from the Web. Literal is the value of a string or data type. The description of resource is to declare its property and value, which enables the data machine-understandable.

Note that the RDF only provides simple descriptions about resources and their values. To depict the classes of resources or specific properties of these resources, we need to apply RDF Schema (RDF(S) for short). RDF(S) is a semantic extension of the RDF. Based on the RDF, the RDF(S) is used to describe simple models of classes and relationships between classes.

The important components of an RDF(S) are *rdf:type*, *rdfs:subclassof* and *rdfs:subpropertyof*. Here *rdf:type* can describe the relationships between properties and classes. *rdfs:subclassof* can depict the inheritance relationships. In addition, *rdfs:range* can represent the property value, which is an instance of a given class. *rdfs:domain* indicates that a property is the attribute of a given class. We present a formal definition of the RDF(s) model as follows:

Definition 2: RDF(S) model is a five-tuple $R = \{RC, RD, RA, RI, RAxiom\}$.

(1) *RC* is a finite set of class identifiers. Each element in the set is a reference to the URI that identifies the class.
(2) *RD* is a finite set of identifiers of attributes datatype. Each element in the set is a reference to the URI that identifies the datatype.
(3) *RA* is a finite set of attribute identifiers. Each element in the set is a reference to the URI that identifies the attribute.
(4) *RI* is a set of instances.
(5) *RAxiom* is a finite set of axioms and *RAxiom* = *CAxiom* $\cup$ *PAxiom*, *CAxiom* is the set of class axioms. $CAxiom_{C1} = \{x|x$ is the superclass of C_1, $x \in RC$, $C_1 \in RC\}$. $CAxiom_{C1}$ represents a finite set of all superclass of class C_1. *PAxiom* is the set of property axioms and *PAxiom* = *Dxiom* $\cup$ *Rxiom*.

We define $Dxiom_A = \{x|x$ is the domain of the property $A\}$, where $x \in RC$, $A \in RA$. The domain of the property *A* is a class resource. We define $Rxiom_A = \{x|x$ is the range of the property $A\}$, where $A \in RA$ and $x \in R \cup RD$. $Rxiom_A$ represents a finite set of the range resource of property *A*. The range of a property can be a class resource or an attribute datatype.

Let us look at an example of the RDF file in Figure 3. Following Definition 2, we have an RDF(S) model R_S in Figure 4.

(1) There are five classes in the RDF file: *Person, Course, Teacher, Assistant* and *Student* class, which are expressed in RDF(S) as $RC_S = \{$*Person*, *Course*, *Teacher*, *Assistant*, *Student*$\}$. Some classes have inheritance relationship. For example, $CAxiom_{Teacher} = \{$*Person*$\}$ means the *Person* class is the superclass of the *Teacher* class.
(2) *RAxiom* = *CAxiom* $\cup$ *PAxiom* means *RAxiom* is a finite set of class axioms and attribute axioms.
(3) *PAxiom* = *Dxiom* $\cup$ *Rxiom*. $Dxiom_A$ is a finite set of the domain resource of the property *A*. For example, $Dxiom_{phone} = \{$*Assistant*$\}$ represents the

```
<rdfs: Class rdf: ID=”Person”/>
<rdfs: Class rdf: ID=”Course”/>
<rdfs: Class rdf: ID=”Teacher”>
    <rdfs: subClassOf rdf:
resource="#Person"/>
</rdfs: Class>
<rdfs: Class rdf: ID="Assistant">
     <rdfs: subClassOf rdf:
resource="#Teacher"/>
</rdfs: Class>
<rdfs: Class rdf: ID="Student">
     <rdfs: subClassOf rdf:
resource="#Person"/>
</rdfs: Class>
<rdf: Property rdf: ID="age">
     <rdfs: domain rdf: resource="#Person"/>
     <rdfs: range rdf: resource="&xsd:
integer"/>
</rdf: Property>
```

```
<rdf: Property rdf: ID="phone">
     <rdfs: domain rdf:
resource="#Assistant"/>
     <rdfs: range rdf: resource="&xsd:
string"/>
</rdf: Property>
<rdf: Property rdf: ID="title">
     <rdfs: domain rdf:
resource="#Assistant"/>
     <rdfs: domain rdf: resource="#Teacher"/>
     <rdfs: range rdf: resource="&xsd:
string"/>
</rdf: Property>
<rdf: Property rdf: ID ="courseName">
     <rdfs: domain rdf: resource="#Course"/>
     <rdfs: range rdf: resource="&xsd:
string"/>
</rdf: Property>
<rdf: Property rdf: ID="enroll">
     <rdfs: domain rdf: resource="#Student"/>
     <rdfs: range rdf: resource="#Course"/>
</rdf: Property>
```

Figure 3 An example of the RDF(S) file.

R = {*RC, RD, RA, RI, RAxiom*}, where $RAxiom = CAxiom \cup PAxiom$
RC_S = {*Person*, *Course*, *Teacher*, *Assistant*, *Student*}
RD_S = {*String*, *Integer*}
RA_S = {*age, phone, title, courseName, enroll*}
$CAxiom_S$ = {
  $CAxiom_{Teacher}$ = {*Person*}, $CAxiom_{title}$ = {*Teacher*, *Person*},
}
$PAxiom = Dxiom \cup Rxiom$
$Dxiom_S$ = {
  $Dxiom_{phone}$ = {*Assistant*}, $Dxiom_{title}$ = {*Teacher*, *Assistant*}, $Dxiom_{courseName}$ = {*Course*},
  $Dxiom_{enroll}$ = {*Student*}, $Dxiom_{age}$ = {*Person*}
}
$Rxiom_S$ = {
  $Rxiom_{phone}$ = {*String*}, $Rxiom_{title}$ = {*String*}, $Rxiom_{courseName}$ = {*String*},
  $Rxiom_{enroll}$ = {*Course*}, $Rxiom_{age}$ = {*Integer*}
}

Figure 4 An RDF(S) model R_S.

domain of the property “*phone*” is the *Assistant* class. Correspondingly, $Rxiom_A$ is a finite set of the range resource of the property *A*. The range resource of a property can be a class or an attribute datatype. For example: $Rxiom_{enroll} = \{Course\}$ represents the range resource of the

```
<rdf: Description rdf: about="161">
      <rdf: type rdf: resource="#Person"/>
      <uni: age rdf: datatype="&xsd: integer">27</uni: age>
</rdf: Description>
<rdf: Description rdf: about="162">
      <rdf: type rdf: resource="#Course"/>
      <uni: courseName rdf: datatype="&xsd: string">Math</uni: courseName>
</rdf: Description>
<rdf: Description rdf: about="163">
      <rdf: type rdf: resource="#Teacher"/>
      <uni: age rdf: datatype="&xsd: integer">35</uni: age>
      <uni: title rdf: datatype="&xsd: string">professor</uni:title>
</rdf: Description>
<rdf: Description rdf: about="164">
      <rdf: type rdf: resource="#Assistant"/>
      <uni: age rdf: datatype="&xsd: integer">30</uni: age>
      <uni: title rdf: datatype="&xsd: string">assistant</uni: title>
      <uni: phone rdf: datatype="&xsd: string">13578</uni: phone>
</rdf: Description>
<rdf: Description rdf: about="165">
      <rdf: type rdf: resource="#Student"/>
      <uni: enroll rdf: resource="162">
</rdf: Description>
```

Figure 5 Instances of the RDF(S) model R_S.

property "*enroll*" is the *Course* class. And $Rxiom_{title} = \{String\}$ means the range resource of the property "*title*" is the "*String*" datatype.

The RDF(S) instances are shown in Figure 5. Each property in the instance should conform to the definition of the RDF(S). For example, the description of "164", an instance of the *Assistant* class, has three attribute "age", "title" and "phone". Because the *Assistant* class is the subclass of the *Teacher* class and *Person* class, it inherits attributes of the superclass. Another example is the description of "165". The value of the attribute "enroll" is an instance, which signifies the relationship between two instances.

4 Mapping the OODB Model into the RDF(S) Model

With the formal definitions of the RDF(S) and the OODM, we propose some rules of mapping the OODBs to the RDF(S). From Definitions 1 and 2, it can be seen that there are some inevitable corresponding relationships between the OODBs and the RDF(S). On the premise of keeping the semantics of RDF(S) as far as possible, we summarize the corresponding relationships between elements in the two models in Figure 6.

Elements in OODM	Elements in RDF(S)
class	class
subclass	subclass
aggregation relationship	attribute value is resource
attribute	attribute value is RDF Literal
instances	instances
OID	none
none	namespace

Figure 6 Corresponding relationships between elements in the RDF(S) model and the OODM.

Generally speaking, converting the OODB to the RDF(S) can be carried out at two levels. The first level is to map the class hierarchy in the OODBs and the second level is to map the instances in the OODBs.

4.1 Mapping Class Hierarchy of Object-oriented Databases

We first define several symbols to better illustrate the mapping relationships. Let $S = (D_S, C_S, A_S, I_S, AG_S, O_S)$ be an OODB model derived from Definition 1 and $R = \{RC, RD, RA, RI, RAxiom\}$ be an RDF(S) model derived from Definition 2. Let φ be an operator and $\varphi(S)$ represents the mapping result from an OODM to RDF(S). Symbol "$\Rightarrow$" is used to represent the process of mapping. Then we propose some mapping rules as follows.

Rule 1: $(\forall c \in C_S) \Rightarrow (\varphi(c) \in RC)$.

For a class in the OODM, it can be converted directly to a class in the RDF(S). Suppose that we have a class named *"Student"* in the OODM. With *Rule 1*, we can create a class in RDF(S) with the same name as follows.

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

Rule 2: $(\forall c_1 \in C_S) \cap (\forall c_2 \in C_S) \cap (I_{C2} = \{c_1\}) \Rightarrow (\varphi(c_1) \in RC) \cap (\varphi(c_2) \in RC) \cap (CAxiom_{\varphi(c_2)} = \{\varphi(c_1)\})$.

Let class c_1 be a superclass of class c_2 in an OODM. Then we create two classes $\varphi(c_1)$ and $\varphi(c_2)$ in RDF(S) model with the same name, in which $\varphi(c_1)$ is the superclass of $\varphi(c_2)$.

Rule 3: $(\forall c_1 \in C_S) \cap (\forall a_1 \in A_{SS}) \cap (\forall d_1 \in D_S) \Rightarrow (\varphi(c_1) \in RC) \cap (\varphi(a_1) \in RA) \cap (\varphi(d_1) \in RD) \cap (DAxiom_{\varphi(a_1)} = \{\varphi(c_1)\}) \cap (RAxiom_{\varphi(a_1)} = \{\varphi(d_1)\})$.

DataType	OODM	RDF(S)
Date and time type	Date	xsd:date
	Time	xsd:time
	Datetime	xsd:datetime
Boolean type	Boolean	xsd:boolean
Character type	Byte	xsd:byte
	Character	xsd:character
	String	xsd:string
Basic numerical type	Decimal	xsd:decimal
	Integer	xsd:integer
	Short	xsd:short
	Long	xsd:long
	Float	xsd:float
	Double	xsd:double
Enum type	Enum	xsd:enum

Figure 7 Datatype mapping between OODM and RDF(S).

Let attribute a_1 with simple datatype d_1 be the attribute of class c_1 in the OODM. Then, $\varphi(c_1)$ is a class of the RDF(S) model and $\varphi(a_1)$ is a property of $\varphi(c_1)$. So, the domain resource of $\varphi(a_1)$ is $\varphi(c_1)$ and range resource of $\varphi(a_1)$ is $\varphi(d_1)$. Note that RDF(S) does not have built-in datatype. So, RDF(S) uses XSD (XML Schema Datatype). The relationships between OODM datatype to RDF(S) datatype are shown in Figure 7.

Rule 4: $(\forall c_1 \in C_S) \cap (\forall a_1 \in A_{SC}) \cap (AG_{C1} = \{(c_2, a_1)\}) \cap (\forall c_2 \in C_S) \Rightarrow (\varphi(c_1) \in RC) \cap (\varphi(c_2) \in RC) \cap (\varphi(a_1) \in RA) \cap (DAxiom_{\varphi(a_1)} = \{\varphi(c_1)\}) \cap (RAxiom_{\varphi(a_1)} = \{\varphi(c_2)\})$.

Let attribute a_1 with resource datatype c_2 be an attribute of class c_1 in the OODM. Then, $\varphi(c_1)$ is a class of the RDF(S) model and $\varphi(a_1)$ is a property of $\varphi(c_1)$. The created property $\varphi(a_1)$ has domain $\varphi(c_1)$ and range $\varphi(c_2)$.

4.2 Mapping Instances of Object-oriented Databases

A class contains some attributes and its instances are described by the corresponding attribute values. The mapping of OODB instances is to map the instances of classes in the OODBs to the instances of the RDF(S). This mapping is carried out after the mapping of schema in the OODBs.

Rule 5: $(\forall \mathit{OID} \in O_S) \Rightarrow (\varphi(\mathit{OID}) \in \mathit{RI})$.

Each instance in the OODBs has a unique object identifier (OID). The value of the OID does not have a practical meaning and is just applied to distinguish different instances. The RDF(S) doesn't have an OID property.

In the RDF(S), namespace mechanism is used for disambiguation purpose and expected to be RDF(S) documents defining resources. In the process of mapping, we discard the value of the OID and generate a random value for the namespace. For the next instance, we set the namespace as this random number plus one. This ensures that each RDF(S) instance has a unique namespace.

Rule 6: $(\forall o_1 \in O_{SC1}) \cap (c_1 \in C_S) \cap (\forall a_{c1} \in A_{SC1}) \Rightarrow (\varphi(o_1) \in RI) \cap (\varphi(a_{c1}) \in RA))$.

The value of an instance in the OODBs can be directly converted into an RDF(S) structure. If the value of an attribute is null in the OODBs, it cannot be displayed in the generated RDF(S). The instances in the OODBs are transformed based on the class relationship.

Rule 7: $(\forall o_1 \in O_{SC1}) \cap (c_1 \in C_S) \cap (I_{C1} = \{c_2\}) \cap (c_2 \in C_S) \cap (\forall a_{c1} \in A_{SC1}) \cap (\forall a_{c2} \in A_{SC2}) \Rightarrow (\varphi(o_1) \in RI) \cap (\varphi(a_{c1}) \in RA_{c1}) \cap (\varphi(a_{c2}) \in RA_{c2})$.

The instances in the OODBs are transformed based on class relationship. Let us look at an instance of the class Student: "56379" = [name: Mike, age: 45, email: Mike@yahoo.com degree: master]. The class has attribute "degree" and has a parent class Person". Then we need to determine if attributes "name", "age" and "email" belong to the Person class. If yes, we construct RDF(S) as follows.

<rdf:Descriptin rdf:ID="12367">
<degree rdf:datatype="&xsd:string">master</degree>
<name rdf:datatype="&xsd:string">Mike</name>
<age rdf:datatype="&xsd:integer">45</age>
<Email rdf:datatype="&xsd:string">Mike@yahoo.com </Email>
</rdf:Description>

Rule 8: $(\forall o_1 \in O_{SC1}) \cap (c_1 \in C_S) \cap (\forall a_{c1} \in A_{SC1}) \Rightarrow (\varphi(o_1) \in RI) \cap (\varphi(a_{c1}) \in RI))$.

An instance aggregation relationship in the OODBs is mapped to a parent-child nested attribute in the created RDF(S), in which the corresponding attribute value should be added to the generated RDFS attribute.

Discussion. For the mapping from relational databases (RDBs) to OWL/RDF, *information preservation* and *query preservation* of mapping rules are introduced in [10] to formally verify the mapping rules. Here information preservation means that the information about the transformed object instance is

not lost in the mapping process; query preservation refers to using RDF data obtained by mapping rules to query all instances of each class. Following the step of [10], it can be formally verified that our proposed rules of mapping OODBs to RDF(S) are both information and query preservations, and thereby semantics preservation.

5 Prototype Implementation and Validation

We designed an automatic construction tool for OODM to RDF(S) transformation named OODM2RDF. In this section, we introduce the system structure of the tool and verify the correctness and validity of the transformation results.

5.1 System Architecture

The architecture of the OODM2RDF is shown in Figure 8, which consists of six main modules as follows.

(1) Database processing module. It reads database files, extracts the semantics which are useful for transforming. Here we define two data structures: *InfoClass* and *RelationClass*. The data extracted from the database are stored in the data structures in Figure 9. The *InfoClass* stores the information about a class, including its class name, superclass names, attributes and attribute values. The *RelationClass* stores class names and the relationship name between two classes. The input and output of this module are a database file and the instances of *InfoClass* and *RelationClass* respectively.

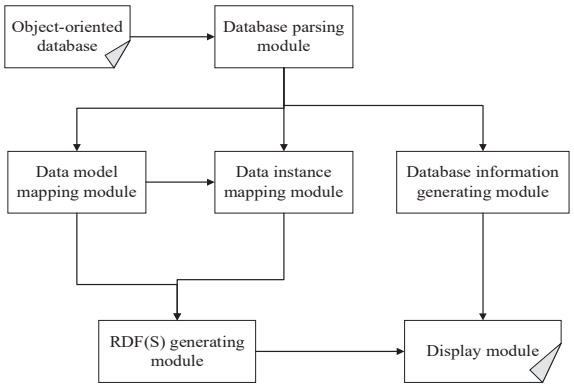

Figure 8 Architecture of the OODM2RDF.

InfoClass data structure	RelationClass data structure

public class InfoClass{
private String className;
private ArrayList<String> superclass;
private Map<String, String>attribute;
private Map<String, String>value;
…}
	public class RelationClass{
	private String name;
	private String domainclassname;
	private String rangeclassname;
	……
	}

Figure 9 InfoClass and RelationClass structures.

```
Algorithm:generating RDFS
Input:info:an obejct of InfoClass, relation:an object of RelationClass
Output:RDFS file
Begin
        for i←0 to info
                e=CreateResource(info[i].getclassname,RDFS.Class)
                if (info[i].getparentclass!=null)then
                        pa=CreateResource(parentclassname,RDFS.Class)
                        Create(e,RDFS.subClassOf,pa)
                for each property in info[i]
                        p=CreateResource(propertyname,RDF.Property)
                        if(propertyname is in relation)then
                                co=CreateResource(propertyname,RDFS.Class)
                                Create(co,RDFS.range,p)
                        else
                                d=CreateResourse(propertyname,RDFS.Datatype)
                                Create(d,RDFS.range,p)
                        Create(e,RDFS.domain,p)
END
```

Figure 10 The algorithm for generating RDF Schema.

(2) Data model mapping module. Its function is to transform the semantic information in the OODM according to the transformation rules presented in Section 4.1. Its input is the semantic information obtained from the database processing module. Figure 10 presents the algorithm of generating RDFS, which includes two main functions *Create*() and *CreateResource*(). *Create*() creates the relationship between class and property and *CreateResource()* creates a resource node in RDFS. The generated RDFS is stored in a data structure called model.

(3) Data instance mapping module. Its function is to convert instances in the OODM according to the transformation rules in Section 4.2. Its input is the data structures obtained from the database processing module. To process the instance mapping, the module needs to check if the corresponding data schema of the instance has completely converted in the data model mapping module. If the data model does not exist, the instance is not mapped.

```
Algorithm: generating RDF
Input: info: an object of InfoClass, relation: an object of RelationClass
Output: RDF file
Begin
        for each element i in info[]
                create ID for each i and store ID in allobejct[]
        for each element i in info[]
                e=CreateResource(allobject[i])
                p=info[i].getclass
                while(p!=null)
                        Create(e, RDF.type, classname)
                        for each property pr in p then
                                pr=CreateProperty(e, pr.name)
                                if(pr.value!=null) then
                                        if(pr.value is a resource) then
                                                Go through the allobject[] and find the corresponding ID
                                                CreateValue(pr, pr.name, ID)
                                        else CreateValue(pr, pr.name, pr.value)
                        p=p.getparentclass

END
```

Figure 11 The algorithm for generating RDF instances.

Figure 11 presents the algorithm of generating RDF instances, which also includes two main functions *Create*() and *CreateResource*(). *CreateProperty*() can create a property for a resource node and *CreateValue*() can determine the value of the property. The result of the instances mapping is stored in the model structure also.

(4) RDF(S) generating module. Its input is the model structures obtained in the data model mapping module and data instance mapping module. Then we use the API of Jena to generate the RDF(S) document. The document will be transmitted to the display module.

(5) Database generating module. Its input is the data structure obtained from the database processing module. It writes information stored in the data model into a document and then outputs the generated document to the display module.

(6) Display module. Its input is the documents generated by the RDF(S) generating module and the database generating module. It presents the content of the document in an appropriate form on the user interface.

5.2 System Implementation

We implemented a tool named OODB2RDF to construct the RDF(S) with the OODBs. We apply db4o [20] as the OODBs for RDF(S) construction.

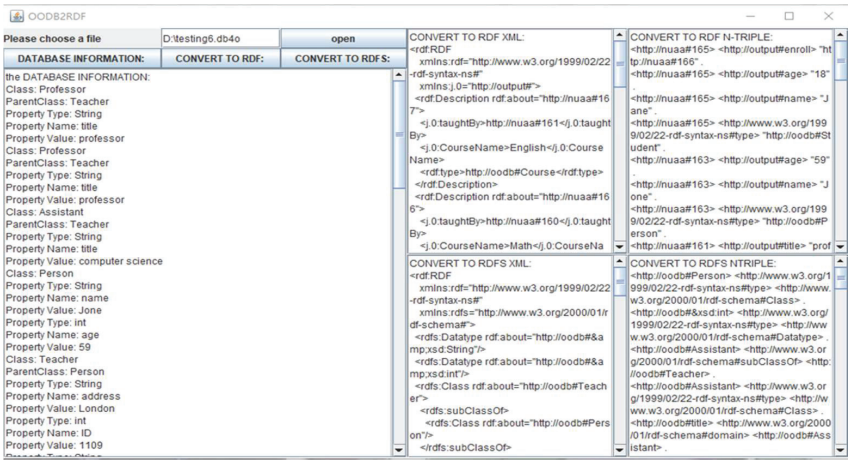

Figure 12 A screen snapshot of the OODB2RDF.

db4o is an open-source OODB engine, which can be operated by using Java and Java.net. In db4o, data are stored in .db4o files. We also used Jena, an ontology toolkit for Java language development to generate the RDF(S) from the OODBs.

The OODB2RDF was developed on the eclipse platform with Java and Java 2 JDK 1.9,[5] and run on a PC with an Intel Core i5 2.50 GHz CPU and 8 GB of RAM. The screen snapshot of OODB2RDF running one of the case studies is shown in Figure 12.

It is shown in Figure 12 that the graphical user interface mainly contains three displaying areas. Users can select the db4o file they want to convert. The information stored in the db4o file is displayed in the left area. The middle area presents the constructed RDFS and RDF with XML format. The constructed RDF(S) and RDF(S) with N-triple format are displayed in the right area.

5.3 Verification of Experimental Results

To verify if there is any semantic loss in the mapping process and if the mapping result is correct, we apply query experiments to compare the information stored in the database and the RDF(S) file generated by the OODB2RDF tool, respectively. We use the SPARQL language recommended by the W3C [22]

[5]https://jena.apache.org/documentation/rdf/index.html

Build Query | Query Results

cn.nuaa.maven.Algorithm22.Teacher

Row Id	address	ID	phone	name	age
1	New York	6023	256889	Michael	35
2	New York	6024	2567890	Mike	35
3	New York	6013	123456	Jack	30
4	London	1109	156789	Wang	30

Figure 13 Instances of the *Teacher* class stored in db4o.

URI	type	ID	age	name	phone	title	address
<http://nuaa#160>	"http://oodb#Professor"	"6023"	"35"	"Michael"	"256889"	"professor"	"New York"
<http://nuaa#161>	"http://oodb#Professor"	"6024"	"35"	"Mike"	"2567890"	"professor"	"New York"
<http://nuaa#162>	"http://oodb#Assistant"	"6013"	"30"	"Jack"	"123456"	"computer science"	"New York"

Figure 14 Instances of the *Teacher* class stored in the RDF document.

to query the RDF(S) file, and use the Object Manager Enterprise [20] tool provided by db4o. With the example in Section 5.3, we designed several experiments to verify the information preservation and query preservation.

(1) Validation of the inheritance relationship. The classes *Professor* and *Assistant* are subclasses of the *Teacher* class. The instances of the *Teacher* class stored in db4o are shown in Figure 13. There are four instances of the *Teacher* class, which have attributes *address, ID, phone, name* and *age*.

We use the SPARQL to query all instances of the *Teacher* class in the generated RDF document. The query result is shown in Figure 14. The "URI" column represents the uniform resource identifier of instances. The "type" column represents the class which the instance belongs to. The results in Figure 14 show that the OODB2RDF can map the corresponding attribute and its values accurately and complete the transformation of inheritance relationship.

(2) Validation of the aggregation relationship. The *Course* class has an aggregation relationship "taughtBy" with the *Professor* class. An instance of the *Course* class stored in db4o is shown in Figure 15. In Figure 15, the "Field" column represents the attributes, the "Value" column represents the attribute values, and the "Type" column represents the attribute types. The instance shown in Figure 15 has an attribute "CourseName" and its value is "Math". Similarly, the value of the "taughtBy" attribute is an instance of the *Professor* class.

Object 1

Field	Value	Type
cn.nuaa.maven.Algorithm22	(G) cn.nuaa.maven.Algorithm22.Course	cn.nuaa.maven.Algorithm22.Course
CourseName	Math	java.lang.String
taughtBy	(G) cn.nuaa.maven.Algorithm22.Professor	cn.nuaa.maven.Algorithm22.Professor
title	professor	java.lang.String
address	New York	java.lang.String
ID	6023	int
phone	256889	java.lang.String
name	Michael	java.lang.String
age	35	int

Figure 15 An instance of the *Course* class stored in db4o.

URI	type	CourseName	taughtBy
<http://nuaa#166>	"http://oodb#Course"	"Math"	"http://oodb#160"
<http://nuaa#167>	"http://oodb#Course"	"English"	"http://oodb#161"

Figure 16 Instances of the *Course* class stored in the RDF document.

URI	type	ID	age	name	phone	title	address
<http://nuaa#160>	"http://oodbProfessor"	"6023"	"35"	"Michael"	"256889"	"professor"	"New York"

Figure 17 The instance represented by "http://oodb#160".

We use SPARQL to query instances of the *Course* class in the generated RDF document. The query result is shown in Figure 16. In Figure 16, the "URI" column means the uniform resource identifier of instances, and the "type" column means the class which the instance belongs to. The "*CourseName*" and "*taughtBy*" columns are attributes of instances.

The value of the "*taughtBy*" attribute is an instance's URI. We query the generated RDF document for the instance represented by http://oodb#160 and obtain the result shown in Figure 17. Figure 17 shows the instance represented by "http://oodb#160", which is an instance of the *Professor* class. This result is consistent with the data stored in db4o. It is shown that OODM2RDF can preserve the semantic of aggregation relationship in the mapping process.

6 Conclusions

The RDF is one of the cornerstones of the Semantic Web. It is of great significance to build an RDF based on databases. This paper is devoted to

constructing RDF(S) from OODBs. We propose some mapping rules based on the formal definitions of OODBs and RDF(S). With the proposed mapping rules, the elements in the OODBs can be transformed into the corresponding elements in the RDF(S), while preserving the inheritance and aggregation semantic of database information. We develop a tool named OODB2RDF that can convert an object-oriented database db4o to RDF(S). We verify with experiments that our proposed method is both information preservation and query preservation.

Note that db4o cannot support some complex relationships in OODBs (e.g., multiple inheritance relation and methods of classes). In our future work, we will further improve the mapping rules so that more complex semantics in databases can be rightly converted into RDF(S).

Acknowledgments

The work was supported in part by the National Natural Science Foundation of China (62176121), the Basic Research Program of Jiangsu Province (BK20191274), and the Natural Science Foundation of Ningxia Province (2020AAC03212).

References

[1] M. Crasso, C. Mateos, A. Zunino, M. Campo, "A programming model for the Semantic Web." *Proceedings of the Second International Conference on Advances in New Technologies, Interactive Interfaces and Communicability,* 2012, pp. 208–218.

[2] M. Hert, G. Reif, H. C. Gall, "A comparison of RDB-to-RDF mapping languages." *Proceedings of the 7th International Conference on Semantic Systems,* 2011, pp. 25–32.

[3] Dezhao Song et al., "Building and querying an enterprise knowledge graph." *IEEE Transactions on Services Computing*, vol. 12, no. 3, pp. 356–369, 2019.

[4] S. Bischof et al., "Mapping between RDF and XML with XSPARQL." *Journal on Data Semantics*, vol. 1, no. 3, pp. 147–185, 2012.

[5] Amato F. et al., "Building RDF ontologies from semi-structured legal documents." *Proceedings of the Second International Conference on Complex, Intelligent and Software Intensive Systems*, 2008, pp. 997–1002.

[6] M. Kejriwal, "Domain-specific knowledge graph construction, *Springer Briefs in Computer Science*, 2019, pp. 1–87.
[7] Q. Tong, F. Zhang, J. Cheng, "Construction of RDF (S) from UML class diagrams." *Journal of Computing and Information Technology*, vol. 22, no. 4, pp. 237–250, 2014.
[8] L. Han et al., "RDF123: From spreadsheets to RDF." *Proceedings of the 7th International Conference on The Semantic Web*, 2008, pp. 451–466.
[9] P.T.T. Thuy et al., "RDB2RDF: Completed transformation from relational database into RDF ontology." *Proceedings of the 8th International Conference on Ubiquitous Information Management & Communication*, 2014, pp. 88:1–88:7.
[10] J. F. Sequeda, M. Arenas, D. P. Miranker, "On directly mapping relational databases to RDF and OWL." *Proceedings of the 21st International Conference on World Wide Web*, 2012, pp. 649–658.
[11] B. He, M. Patel, Z. Zhang, K. C. Chang, "Accessing the deep Web." *Communications of the ACM*, vol. 50, no. 5, 94–101, 2007.
[12] S. Bagui, "Achievements and weaknesses of object-oriented databases." *Journal of Object Technology*, vol. 2, no. 4, 29–41, 2003.
[13] J.-L. Martínez-Rodríguez, I. López-Arévalo, A. B. Rios-Alvarado, "OpenIE-based approach for knowledge graph construction from text." *Expert Systems with Applications*, vol. 113, pp. 339–355, 2018.
[14] S. Auer et al., "Triplify: Light-weight linked data publication from relational databases." *Proceeding of the 18th International Conference on World Wide Web,* 2009, pp. 621–630.
[15] G. Bûmans, K. Cerans, "RDB2OWL: A practical approach for transforming RDB data into RDF/OWL." *Proceedings of the 6th International Conference on Semantic Systems,* 2010, pp. 1–3.
[16] V. Eisenberg, Y. Kanza, "D2RQ/update: Updating relational data via virtual RDF." *Proceedings of the 21st International Conference on World Wide Web*, 2012, pp. 497–498.
[17] E. Oren, B. Heitmann, S. Decker, "ActiveRDF: Embedding semantic web data into object-oriented languages." *Journal of Web Semantics*, vol. 6, no. 3, pp. 191–202, 2008.
[18] G. Meditskos, N. Bassiliades, "A rule-based object-oriented OWL reasoner." *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 3, pp. 397–410, 2008.
[19] G. Meditskos, N. Bassiliades, "O-DEVICE: An object-oriented knowledge base system for OWL ontologies." *Proceedings of the 4th Helenic Conference on AI*, 2006, pp. 256–166.

[20] J. Paterson, S. Edlich, H. Horning, R. Hornig, *The Definitive Guide to db4o*, Apress Publisher, 2006.
[21] Fu Zhang, Z. M. Ma and Li Yan, "Construction of ontologies from object-oriented database models, *Integrated Computer-Aided Engineering*, vol. 18, no. 4, pp. 327–347, 2011.
[22] J. Pérez, M. Arenas, C. Gutierrez, "Semantics and complexity of SPARQL." *ACM Transactions on Database Systems*, vol. 34, no. 3, pp. 16:1–16:45, 2009.
[23] Songfei Wu et al., "Natural-language-based intelligent retrieval engine for BIM object database." *Computers in Industry*, vol. 108, pp. 73–88, 2019.
[24] Fu Zhang, Z. M. Ma and Weijun Li, "Storing OWL ontologies in object-oriented databases." *Knowledge-Based Systems*, vol. 76, pp. 240–255, 2015.
[25] Qiang Tong, "Mapping object-oriented database models into RDF(S)." *IEEE Access*, vol. 6, pp. 47125–47130, 2018.

Biographies

Jing Shan received her bachelor degree in information security in 2010 and her Master's degree in computer science in 2013 from Nanjing University of Aeronautics and Astronautics, China. She is a research associate at Nanjing University of Aeronautics and Astronautics, China, engaging in intelligent systems for task allocation, optimization algorithms and application of intelligent transportation systems.

Jiawen Lu received her bachelor degree in electronic information engineering in 2018 from the University of Electronic Science and Technology, China. She is currently pursuing her Master's degree at the College of Computer Science & Technology, Nanjing University of Aeronautics and Astronautics, China. Her research interests include RDF data management and knowledge graphs.

Xu Chen received his Ph.D. in computer application technology in 2018 from Northeastern University, China. He is currently a senior engineer at North Minzu University, China. His research interests include graph deep learning and cyber security.

Li Yan is currently a full professor at Nanjing University of Aeronautics and Astronautics, China. Her research interests mainly include big data

processing, knowledge graph, spatiotemporal data management, and fuzzy data modeling. She has published more than 50 papers on these topics. She is the author of three monographs published by Springer.

Zongmin Ma is currently a full professor at Nanjing University of Aeronautics and Astronautics, China. His research interests include big data and knowledge engineering, the Semantic Web, temporal/spatial information modeling and processing, deep learning, and knowledge representation and reasoning with a special focus on information uncertainty. He has published more than 200 papers in international journals and conferences on these topics. He has (co-)authored six monographs with Springer. He is a Fellow of the IFSA and a senior member of the IEEE.