Knowledge-Based Systems 76 (2015) 240–255

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Storing OWL ontologies in object-oriented databases

Fu Zhang, Z.M. Ma *, Weijun Li

College of Information Science and Engineering, Northeastern University, Shenyang 110819, China

ARTICLE INFO

Article history:
Received 21 July 2014
Received in revised form 15 December 2014
Accepted 17 December 2014
Available online 2 January 2015

Keywords:
OWL
Ontology
Object-oriented database
Storage
Query

ABSTRACT

The Semantic Web uses ontological descriptions, in particular Web Ontology Language OWL, as a universal medium to formally describe and exchange knowledge of various domains. Currently, many OWL ontologies for different domains come into being successively. Therefore, how to store OWL ontologies becomes one of ordinary needs of the Semantic Web. Based on the efficient storage mechanism of object-oriented databases, they may be used to store OWL ontologies for realizing the management of large amounts of knowledge in the Semantic Web.

To this end, the main objective of this paper is to investigate how to store OWL ontologies in object-oriented databases, and we propose a formal approach and develop a prototype tool for storing OWL ontologies in object-oriented databases. *Firstly*, after giving a complete formal definition of OWL ontologies, we propose an overall architecture of storing OWL ontologies in object-oriented databases. Based on the architecture, we *further* give storage rules and explain how to store OWL ontologies in object-oriented databases with a running example in detail. The correctness and quality of the storage approach are proved and analyzed. *Finally*, we implement a prototype tool which can store OWL ontologies in a widely used open source object database db4o. Also, a query interface is developed in the prototype tool for querying the stored OWL ontologies. The storage and query examples are provided to show that the approach is feasible and the tool is efficient.

1. Introduction

Ontologies, a cornerstone of the Semantic Web, can enable shared, explicit and formal descriptions of knowledge [7,9]. Currently, ontologies are increasingly used in many application domains such as information systems, schema integration, and the Semantic Web [29]. Lots of ontologies have been created and real ontologies tend to become very large to huge (e.g., in the life sciences there are some very large ontologies such as FMA, AERO and NBO ontologies) [1,27]. Therefore, one problem is considered that has arisen from practical needs: namely, efficient storage of ontologies.

In general, there are several possible approaches to store ontologies [1,3,13,36]. One is to use *file systems* to store ontologies, while the problem with this approach is that the file systems do not provide scalability or query facility. Moreover, the database research community has successfully developed a wide theory corpus and a mature and efficient technology to deal with large and persistent amounts of information. In this case, the well-known relational, object or object-relational *databases* may be used to store ontologies.

Currently, there are some proposals for storing ontologies in *relational databases* [1,3,13,17,28,34,35,40]. Moreover, in [6], ontologies are stored in *object-relational databases*. Notice that, the ontologies mentioned in these approaches are relatively lightweight and are represented in OWL 1. OWL 1 is the Web Ontology Language developed by the W3C Web Ontology Working Group and published in 2004 (referred to hereafter as "OWL 1") [23]. However, a practical experience with OWL 1 has shown that it lacks several constructs for modeling complex domains [14]. The improvements of OWL 1, initially performed by some group of its users, have led to more expressive OWL 2 that is still allows for complete and decidable computing [14,27]. Currently, many ontologies are represented in OWL 2 [14,27,36]. Accordingly, efficient storage of OWL 2 ontologies is necessary for the Semantic Web.

Although there are some approaches for storing OWL 1 ontologies in relational and object-relational databases as mentioned above, and also there is an approach for storing ontologies described in OWL 2 metamodels to relational database schemas [33,36]. *The ways for storing OWL 2 ontologies in object-oriented databases still do not exist*. As mentioned in [6], storing ontologies in relational databases is less straightforward than storing

* Corresponding author. Tel./fax: +86 24 83681582.
E-mail address: mazongmin@ise.neu.edu.cn (Z.M. Ma).

http://dx.doi.org/10.1016/j.knosys.2014.12.020
0950-7051/

ontologies in object databases, because relational databases do not provide support for many constructs of ontologies, e.g., class inheritance, object properties, and cardinalities. In particular, object-oriented databases are designed to model complex objects and relationships in real-world applications [10,25]. Therefore, object-oriented databases may be used to store OWL 2 ontologies for realizing the efficient management of large amounts of knowledge in the Semantic Web.

To this end, the main objective of this paper is to investigate how to store OWL 2 ontologies in object-oriented databases, and we ***propose a complete approach and develop a prototype tool to store OWL 2 ontologies in object-oriented databases***, including:

- After giving a complete formal definition of OWL 2 ontologies, we ***propose an overall architecture*** of storing OWL 2 ontologies in object-oriented databases.
- Based on the architecture, we further ***propose storage rules*** and ***explain how to store OWL 2 ontologies in object-oriented databases with a running example*** in detail. Also, ***the correctness and quality of the storage approach are proved and analyzed***.
- Finally, we ***implement a prototype tool*** which can store OWL 2 ontologies in object-oriented database *db4o* [26], which is a widely used open source object database recommended by Object-oriented Database Management Group (*ODMG*) [10]. Also, ***a query interface is developed*** in the prototype tool for querying the stored OWL 2 ontologies. The storage and query examples are provided to show that the approach is feasible and the tool is efficient.

The remainder of this paper is organized as follows: Section 2 introduces some preliminaries, and proposes a formal definition of OWL 2 ontologies. Section 3 proposes a formal approach for storing OWL 2 ontologies in object-oriented databases and develops a prototype storage tool. Section 4 introduces the related work. Section 5 shows the conclusion.

2. OWL 2 ontologies and object-oriented databases

In this section, some preliminaries on OWL 2 ontologies and object-oriented databases are introduced. The characteristics of OWL 2 are summarized and a formal definition of OWL 2 ontologies is presented.

2.1. OWL 2 ontologies

Ontologies are formalized vocabularies of terms, often covering a specific domain and shared by a community of users [9]. Ontologies can be defined by ontology languages such as RDFS, DAML + OIL, or OWL [16]. The *Web Ontology language OWL* [23], which is developed by W3C Web Ontology Working Group and published in 2004 (referred to hereafter as "*OWL 1*"), is to be de facto standard for ontologies. OWL 1 was mainly focused on constructs for expressing information about classes and individuals, and exhibited some weakness regarding expressiveness for properties. *OWL 2* is an extension and revision of OWL 1 [14,27]. OWL 2 adds several new features to OWL 1, some of the *new features* are *syntactic sugar* (e.g., disjoint union of classes) while others offer new expressivity, including: *increased expressive power for properties*, *simple metamodeling capabilities*, *extended support for datatypes*, *extended annotation capabilities*, and *other innovations and minor features*.

Table 1 gives the *syntax* of OWL 1. OWL 2 inherits OWL 1 language features and adds several new features to OWL 1. Table 2 further provides a summary of the main new features with examples. Here, we show some examples and comments to reconcile an easy understandable illustration for each feature.

Moreover, the *semantics* for OWL 1 and OWL 2 are given based on Description Logics [8,27] (Description logics, which are a family of knowledge representation languages that are widely used in ontological modeling, are the logical underpinnings of OWL 1 and OWL 2). The semantics allow humans and computer systems to exchange ontologies without ambiguity as to their meaning, and also make it possible to use logical deduction to infer additional information from the facts stated explicitly in an ontology. The detailed syntax and semantics for OWL 1 and OWL 2 can be found at [23,27].

An *ontology* formulated in *OWL 2* language is called *OWL 2 ontology*. In the following we present a formal definition of OWL 2 ontologies by summarizing the characteristics of OWL 2 ontologies.

Definition 1. (*OWL 2 ontologies*). An OWL 2 ontology can be formally defined as a tuple $\mathcal{O} = \{\mathcal{I}, \mathcal{P}, \mathcal{X}, \mathcal{D}, \mathcal{A}\}$:

- $\mathcal{I}$ is a set of *individuals*; Each individual is an instance of a class, and it may be an abstract individual or a concrete individual as mentioned in Tables 1 and 2;
- $\mathcal{P}$ is a set of *properties*; A property can be classified into two kinds of properties: *object properties* $\mathcal{P}$ and *datatype properties* $\mathcal{T}$, the former link individuals to individuals and the later link individuals to data values;
- $\mathcal{X}$ is a set of *classes*; $\mathcal{X}, \mathcal{P}$ and $\mathcal{I}$ form the primitive terms of an ontology, e.g., a class *Person* can be used to represent the set of all people, the object property *parentOf* can be used to represent the parent–child relationship, and the individual *Peter* can be used to represent a particular person called *Peter*;
- $\mathcal{P}$ is a set of *data range identifier s*; Each data range identifier is a predefined XML Schema datatype;
- $\mathcal{A}$ is a set of axioms defined over $\mathcal{I} \sqcup \mathcal{P} \sqcup \mathcal{X} \sqcup \mathcal{D}$ as shown in Tables 1 and 2.

The *semantics* for OWL 2 ontologies can be given based on the interpretations of Description Logics [8,23,27]. An OWL 2 ontology $\mathcal{O}$ is satisfied in an interpretation if all axioms in $\mathcal{O}$ are satisfied in the interpretation, and in this case we say that the interpretation is a model of $\mathcal{O}$.

The Definition 1 summarizes the main notions of OWL 2 ontologies. However, it should be noted that we do not expect this definition to become a universal and standard definition, because we understand that a universal ontology definition is difficult since the different application requirements. All of the notions of OWL 2 ontologies as mentioned in Definition 1 (i.e., Tables 1 and 2) will be stored in object-oriented databases in the subsequent sections.

2.2. Object-oriented databases

As mentioned in [10,25], an *object-oriented database* (OODB) is basically a set of declarations of classes. A class declaration depicts several parts of objects: *structure* (*attribute*, and *relationship* to other objects like association), *behavior* (a set of operations) and *inheritance*. A class, called subclass, is produced from another class, called superclass by means of inheriting all attributes and methods of the superclass, overriding some attributes and methods of the superclass, and defining some new attributes and methods. Any object belonging to the subclass must belong to the superclass.

Here we begin with an example in [25] to further explain the class declarations in OODBs.

Table 1
OWL 1 syntax and the corresponding Description Logic (DL) syntax.

OWL 1 syntax	DL syntax	Examples and comments
Class descriptions		
C, which is a URIref of a class	C	*Student* //*Student* is defined as a class
owl:Thing	$\top$	// It is a top class
owl:Nothing	$\bot$	// It is a bottom class
intersectionOf $(C_1 \ldots C_n)$	$C_1 \sqcap \ldots \sqcap C_n$	intersectionOf (*Student Person*) $\Longleftrightarrow$ $Student \sqcap Person$
unionOf $(C_1 \ldots C_n)$	$C_1 \sqcup \ldots \sqcup C_n$	unionOf (*Student Teacher*) $\Longleftrightarrow$ $Student \sqcup Teacher$
complementOf (C)	$\neg C$	complementOf (*Student*) $\Longleftrightarrow$ $\neg Student$
oneOf $(w_1 \ldots w_n)$	$\{w_1 \ldots w_n\}$	oneOf (*Monday Tuesday* ...)
restriction (P someValuesFrom (E))	$\exists P.E$	restriction (*hasChild* someValuesFrom (*Man*))
restriction (P allValuesFrom (E))	$\forall P.E$	restriction (*hasChild* allValuesFrom (*Woman*))
restriction (P hasValue (E))	$\exists P.\{w\}$	restriction (*hasChild* hasValue (*John*))
restriction (P minCardinality (n))	$\geqslant n\ P$	restriction (*hasChild* minCardinality (1))
restriction (P maxCardinality (n))	$\leqslant n\ P$	restriction (*hasChild* maxCardinality (3))
restriction (P Cardinality (n))	$= n\ P$	restriction (*hasChild* cardinality (2))
ObjectProperty (R)	R	*isProducedBy*
DatatypeProperty (T)	T	*hasAge*
AbstractIndividual (o)	o	s_1 // if s_1 is defined as a student
DatatypeIndividual (v)	v	5^^xsd:integer
Datatype (D)	D	xsd:integer, xsd:string, and so on
Facets (F_i) $P \in \{R, T\}$ $E \in \{C, D\}$ $w \in \{o, v\}$	F_i	F_i are the constraining facets, e.g., *minInclusive* and *minInclusive*
Class axioms		
Class (C partial $C_1 \ldots C_n$)	$C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	Class (*hpPrinter* partial *hpProduct Printer*)
Class (C complete $C_1 \ldots C_n$)	$C \equiv C_1 \sqcap \ldots \sqcap C_n$	Class (*Woman* complete *Person Female*)
SubClassOf $(C_1 C_2)$	$C_1 \sqsubseteq C_2$	SubClassOf (*Woman Person*)
EquivalentClasses $(C_1 \ldots C_n)$	$C_1 \equiv \ldots \equiv C_n$	EquivalentClasses (*Faculty AcademicStaffMember*)
EnumeratedClass $(Cw_1 \ldots w_n)$	$C \equiv \{w_1 \ldots w_n\}$	EnumeratedClass (*DaysOfTheWeek Sunday Monday* ...)
Property axioms		
DatatypeProperty (T domain (C_1)...domain (C_n) range (D_1)...range (D_n) [Functional])	$\geqslant 1T \sqsubseteq C_i \top \sqsubseteq \forall T.D_i \top \sqsubseteq \leqslant 1T$	DatatypeProperty (*hasAge* domain (*Student*) range (*xsd:integer*) [Functional])
ObjectProperty (R		ObjectProperty (*hasSameGradeAs*
domain (C_1)...domain (C_n)	$\geqslant 1R \sqsubseteq C_i$	domain (*student*)
range (C_1)...range (C_n)	$\top \sqsubseteq \forall R.\ C_i$	range (*student*)
[Functional]	$\top \sqsubseteq \leqslant 1R$	[Symmetric]
[inverseOf(R_0)]	$R = (R_0)^-$	[Transitive]
[Symmetric]	$R = R^-$	...)
[InverseFunctional]	$\top \sqsubseteq \leqslant 1R^-$	
[Transitive])	Trans (R)	
SubPropertyOf (E_1, E_2)	$E_1 \sqsubseteq E_2$	SubPropertyOf (*isTaughtBy involves*)
EquivalentProperties $(E_1 \ldots E_n)$	$E_1 \equiv \ldots \equiv E_n$	EquivalentProperties (*lecturesIn teaches*)
Individual axioms		
Individual (o type (C_1)...type (C_n))	$o \in C_i$	Individual (s_1 type (*Student*))
value (R_1, o_1)...value (R_n, o_n)	$(o, o_i) \in R_i$	value (*hasSameGradeAs*, s_2)
value (T_1, v_1)...value (T_n, v_n))	$(o, v_i) \in T_i$	value (*hasAge*, *20^^xsd:integer*))
SameIndividual $(o_1 \ldots o_n)$	$o_1 = \ldots = o_n$	SameIndividual (s_1*John*)
DifferentIndividuals $(o_1 \ldots o_n)$	$o_i \neq o_j i \neq j$	DifferentIndividuals $(s_1 s_2)$

The Fig. 1 is the diagram of an OODB. The declarations of several classes are defined as follows:

Class Book {
title: String;
ISBN: Int;
publishedBy: Publisher *inverse* publish;
writtenBy: Author *inverse* write;
}
...
Class ArtBook *isa* Book {
style: String;
}

Class Author {
name: String;
authorNo: Int;
write: Book *inverse* writtenBy;
}

In the *Book* example we have attributes *title*, *ISBN*, *publishedBy*, and *writtenBy*. *Book* and *Publisher* is connected with *1:N* association. An *ArtBook* is a *Book*, therefore the *ArtBook* class is a subclass of *Book* class. Currently, the OODBs can be stored in db4o database [26], which is a widely used open source object database recommended by Object-oriented Database Management Group (ODMG) [10].

For a comprehensive review of OODBs, please refer to [10,25].

3. Storage of OWL 2 ontologies in object-oriented databases

In this section, we propose an approach and develop a tool to store OWL 2 ontologies in object-oriented databases, including:

(i) We *propose an overall architecture* of storage approach, and illustrate the architecture (see Section 3.1);
(ii) Based on the architecture, we further *give storage rules* in detail and begin with an example to *explain* how to store OWL 2 ontologies in object-oriented databases (see Section 3.2); Also, we *prove the correctness of the approach* (see Section 3.3);
(iii) We *implement a prototype tool* called *OWL2OODB* which can automatically store OWL 2 ontologies in object-oriented databases, and also we *develop a query interface* for querying the stored OWL 2 ontologies in the prototype tool (see Section 3.4);
(iv) Finally, we make a discussion and analysis about the approach and tool, including the storage in the semantics level and other issues (see Section 3.5).

Table 2
The new features of OWL 2 syntax and the corresponding Description Logic (DL) syntax.

OWL 2 new features	DL syntax	Examples and comments
Syntactic sugar		
DisjointUnion ($C\ C_1 \ldots C_n$)	$C = C_1 \sqcup \ldots \sqcup C_n$ $C_i \sqcap C_j \sqsubseteq \perp$ $i \neq j\ i,j \in \{1,\ldots,n\}$	DisjointUnion (*CarDoor FrontDoor RearDoor TrunkDoor*) // A *CarDoor* is exclusively either a *FrontDoor*, a *RearDoor* or a *TrunkDoor* and not more than one of them
DisjointClasses ($C_1 \ldots C_n$)	$C_i \sqcap C_j \sqsubseteq \perp$ $i \neq j\ i,j \in \{1,\ldots,n\}$	DisjointClasses (*LeftLung RightLung*) // Nothing can be both a *LeftLung* and a *RightLung*
NegativeObjectPropertyAssertion ($R\ o_1\ o_2$)	$(o_1, o_2) \notin R$	NegativeObjectPropertyAssertion (*livesIn ThisPatient IleDeFrance*) // *ThisPatient* does not *live in* the *IleDeFrance* region
NegativeDataPropertyAssertion ($T\ o\ v$)	$(o, v) \notin T$	NegativeDataPropertyAssertion (*hasAge ThisPatient* 5^^xsd:integer) // *ThisPatient* is not five years old
New constructs for properties		
ObjectHasSelf (R)	$\{o \mid (o,o) \in R\}$	SubClassOf (*AutoRegulatingProcess* ObjectHasSelf (*regulate*)) //*Auto-regulating processes regulate* themselves // The feature is called *self restriction* or *local reflexivity*, which denotes: classes of objects that are related to themselves by a given property
ObjectMinCardinality ($n\ R[C]$)	$\geqslant n\ R.\ C$	ObjectMinCardinality (1 *hasPart* [*Door*]) // Class of objects having at least 1 *Door*
ObjectMaxCardinality ($n\ R[C]$)	$\leqslant n\ R.\ C$	ObjectMaxCardinality (5 *hasPart* [*Door*]) // Class of objects having at most 5 *Door*
ObjectExactCardinality ($n\ R[C]$)	$= n\ R.C$	ObjectExactCardinality (2 *hasPart* [*RearDoor*]) // Class of objects having exactly 2 *RearDoor*
DataMinCardinality ($n\ T[D]$)	$\geqslant n\ T.D$	DataMinCardinality (1 *hasName* [xsd:string]) // Each individual has at least one *Name*
DataMaxCardinality ($n\ T[D]$)	$\leqslant n\ T.D$	DataMaxCardinality (1 *hasSSN* [xsd:string]) // Each individual has at most one *Social Security Number*
DataExactCardinality ($n\ T[D]$)	$= n\ T.D$	DataExactCardinality (1 *hasNumber* [xsd:string]) // Each individual has exactly one *Number*

ReflexiveObjectProperty (R)	$\forall o \rightarrow (o,o) \in R$	ReflexiveObjectProperty (*sameBloodGroup*)
		// Everything has the same blood group as itself
		// The feature is called *global reflexivity*, which denotes: the object property holds for all individuals

IrreflexiveObjectProperty (R)	$\forall o \rightarrow (o,o) \notin R$	IrreflexiveObjectProperty (*flowsInto*) // Nothing can flow into itself
AsymmetricObjectProperty (R)	$R \neq R^-$	AsymmetricObjectProperty (*proper_part_of*) // The property *proper_part_of* is asymmetric
SubPropertyOf (ObjectPropertyChain ($R_1 \ldots R_n$)R)	$R_1(o, o_1) \circ \ldots \circ R_n(o_{n-1}, o_n) \rightarrow R(o, o_n)$	SubPropertyOf (ObjectPropertyChain (*locatedIn partOf*) *locatedIn*) // If *x* is located in *y* and *y* is part of *z* then *x* is located in *z*, for example a disease located in a part is located in the whole
DisjointObjectProperties ($R_1 \ldots R_n$)	$R_i \sqcap R_j \sqsubseteq \perp$ $i \neq j\ i,j \in \{1,\ldots,n\}$	DisjointObjectProperties (*connectedTo contiguousWith*) // *connectedTo* and *contiguousWith* are exclusive properties
DisjointDataProperties ($T_1 \ldots T_n$)	$T_i \sqcap T_j \sqsubseteq \perp$ $i \neq j i,j \in \{1,\ldots,n\}$	DisjointDataProperties (*startTime endTime*) // *Start time* of something, e.g., surgery, must be different from its *end time*

HasKey ($C(R_1 \ldots R_n) \mid (T_1 \ldots T_n)$)	$\forall o_1, o_2, o \in C : R_i(o_1, o)$ and $R_i(o_2, o) \rightarrow o_1 = o_2$; or $\forall o_1, o_2 \in C : T_i(o_1, v)$ and $T_i(o_2, v) \rightarrow o_1 = o_2$	HasKey (*RegisteredPatient hasWaitingListN*)
		// Each *registered patient* is uniquely identified by his *waiting list number*
		HasKey (*Transplantation donorId recipientId ofOrgan*)
		// Each *Transplantation* is uniquely identified by a *donor*, a *recipient*, and an *organ*

Extended datatype capabilities		
DatatypeRestriction ($DF_1d_1 \ldots F_nd_n$)	$D \sqcap (F_1\ v_1) \sqcap \ldots \sqcap (F_n v_n)$	DatatypeRestriction (*xsd:integer minInclusive* 18) // new datatype with a lower bound of 18 on the XML Schema datatype xsd:integer
DatatypeDefinition (D_iD_j)	$D_i = D_j$	DatatypeDefinition (*adultAge* DatatypeRestriction (*xsd:integer minInclusive* 18)) // An *adult age* is defined by using a lower bound of 18 with the XML Schema datatype xsd:integer
DataIntersectionOf ($D_1 \ldots D_n$)	$D_1 \sqcap \ldots \sqcap D_n$	DataIntersectionOf (*xsd:nonNegativeInteger xsd:nonPositiveInteger*)
DataUnionOf ($D_1 \ldots D_n$)	$D_1 \sqcup \ldots \sqcup D_n$	DataUnionOf (*xsd:string xsd:integer*)
DataComplementOf (D)	$\neg D$	DataComplementOf (*xsd:positiveInteger*)
Enhanced capabilities and other innovations and features		

Other features such as *punning* and *annotations*, *declarations*		// *Punning* allows different uses of the same term
		// OWL 2 allows for *annotations*
		// A declaration signals that an entity is part of the vocabulary of an ontology
		// Note that, all these types have no semantic meaning in OWL 2. The use of them is left to the applications that use OWL 2

(*continued on next page*)

Table 2 (*continued*)

OWL 2 new features	DL syntax	Examples and comments
owl:topObjectProperty	$\top_R$	// all pairs of individuals are connected by owl:topObjectProperty
owl:bottomObjectProperty	$\bot_R$	// no individuals are connected by owl:bottomObjectProperty
owl:topDataProperty	$\top_T$	// all possible individuals are connected with all literals by owl:topDataProperty
owl:bottomDataProperty	$\bot_T$	// no individual is connected by owl:bottomDataProperty to a literal
ObjectInverseOf (R)	R^-	ObjectInverseOf (*partOf*) // this expression represents the inverse property of *partOf*
InverseObjectProperties (R_1, R_2)	$R_1 = R_2^-$	InverseObjectProperties (*hasPart partOf*) // *hasPart* and *partOf* are inverse properties

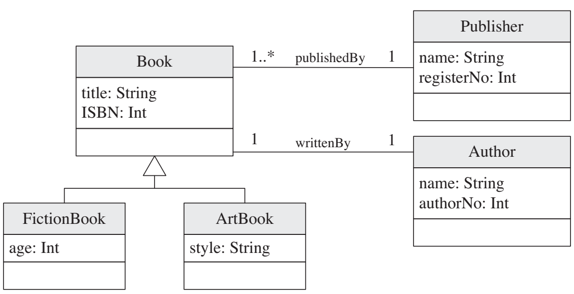


Fig. 1. The diagram of an OODB.

3.1. The overall architecture of storage approach

In the following we propose an overall architecture of storage approach, which is helpful to well understand the storage process of OWL 2 ontologies in object-oriented databases. Fig. 2 shows the overall architecture of storage approach. Furthermore, Table 3 further explains the architecture in detail.

In Fig. 2, all the elements of an OWL 2 ontology in Definition 1 (i.e., Tables 1 and 2) are considered and stored in an object-oriented database. The details can be found in Fig. 2, Table 3, and the later Section 3.2. *In brief*, all the *symbols* of classes, properties, individuals, and datatypes in an OWL 2 ontology are stored in a resource class (i.e., Resource class shown in Fig. 2). Then, all the *axioms* defined over the symbols in the OWL 2 ontology (i.e., class axioms, property axioms, and individual axioms) are stored in different classes as shown in Fig. 2.

Here, it should be noted that we assign a new ID to each anonymous class occurring in complex OWL 2 class axioms. Taking the class axiom "*Mother* $\equiv$ *Woman* $\sqcap\exists$*hasChild.Person*" as an example, we first define a new class ID (e.g., *c_1*) for the anonymous class "$\exists$*hasChild.Person*", and then store the new ID *c_1* and its property restriction "$\exists$*hasChild.Person*" in the Resource class and Property_Restriction class shown in Fig. 2, respectively. As a result, the class axiom "*Mother* $\equiv$ *Woman* $\sqcap\exists$*hasChild.Person*" is replaced with the class axiom "*Mother* $\equiv$ *Woman* $\sqcap$ *c_1*", which is stored in the Class_Operation class shown in Fig. 2. Such a design is motivated by making the semantics of the complex class axioms explicit.

3.2. The detailed procedures of storing OWL 2 ontologies in object-oriented databases

Based on the storage architecture in Section 3.1, in this section, we use an example throughout the subsections to demonstrate and introduce the idea of the storage architecture. This example can show how to map each construct in an OWL 2 ontology to a corresponding construct in an object-oriented database.

Fig. 3 shows an OWL 2 ontology $\mathcal{O}_{uni}$ modeling parts of the reality at a university, which includes the structure information and the instance information. Further, for ease of understanding, the OWL 2 ontology $\mathcal{O}_{uni}$ is also represented as a graph in Fig. 4.

In the following two subsections, on the basis of the storage architecture proposed in Section 3.1, both of the structure and instance information of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3 will be stored in an object-oriented database in detail.

3.2.1. Storing the structure information of OWL 2 ontology in object-oriented database

The *structure information* of the OWL 2 ontology $\mathcal{O} = \{\mathcal{I}, \mathcal{P}, \mathcal{X}, \mathcal{D}, \mathcal{A}\}$ includes the sets of properties $\mathcal{P}$, classes $\mathcal{X}$, data range identifiers $\mathcal{D}$, and axioms $\mathcal{A}$. Based on the architecture proposed in Section 3.1, the following procedures will store the *structure information* of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3 in an object-oriented database.

(1) Storing the resources
As mentioned in Section 3.1, all *resources* in an OWL 2 ontology (including properties $\mathcal{P}$, classes $\mathcal{X}$, data range identifiers $\mathcal{D}$, and individuals $\mathcal{I}$) will be stored in a *Resource class* (including 5 fields, i.e., *OntoName*, *ID*, *namespace*, *localname*, and *type*).
Fig. 5 shows the class *Resource* and its *objects* in a target object-oriented database, which stores all resources of the OWL 2 ontology $\mathcal{O}_{\text{uni}}$ in Fig. 3. Here, *OntoName* is the ontology name "$\mathcal{O}$*_1*"; *ID* uniquely identifies a resource; *namespace* and *localname* describe the URIref of a resource; and *type* describes the type of a resource.

(2) Storing the relationships among classes
As mentioned in Section 3.1, the *relationships among classes* in an OWL 2 ontology (including *partial*, *complete*, *SubClassOf*, and *DisjointUnion*) will be stored in a *Class_Relation class* (including fields *Vector<ClassID>* and *relationship*).
Fig. 6 shows the class *Class_Relation* and its *objects* in a target object-oriented database, which stores the relationships among classes in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. In detail, the classes axioms SubClassOf (*Professor Staff*), SubClassOf (*Associate_Prof Staff*), DisjointUnion (*Student Undergraduate Postgraduate*), Class (*Staff* partial restriction (*work_in* allValuesFrom(*College*))), and so on, are stored in Fig. 6.
Note that, as has been indicated at the beginning of Section 3.1, for storing the class axioms such as "Class (*Staff* partial restriction (*work_in* allValuesFrom(*College*)))", the following several steps need to be done:

(i) we need to assign a new ID (e.g., *new_c_10*) to the anonymous class "restriction (*work_in* allValuesFrom (*College*))". In this case, the original class axiom above can be replaced with two new axioms:
"Class (*Staff* partial *new_c_10*)" and
"*new_c_10* $\equiv$ restriction (*work_in* allValuesFrom (*College*))";

(ii) Finally, the new ID "*new_c_10*" is stored in the *Resource* class as shown in Fig. 5;

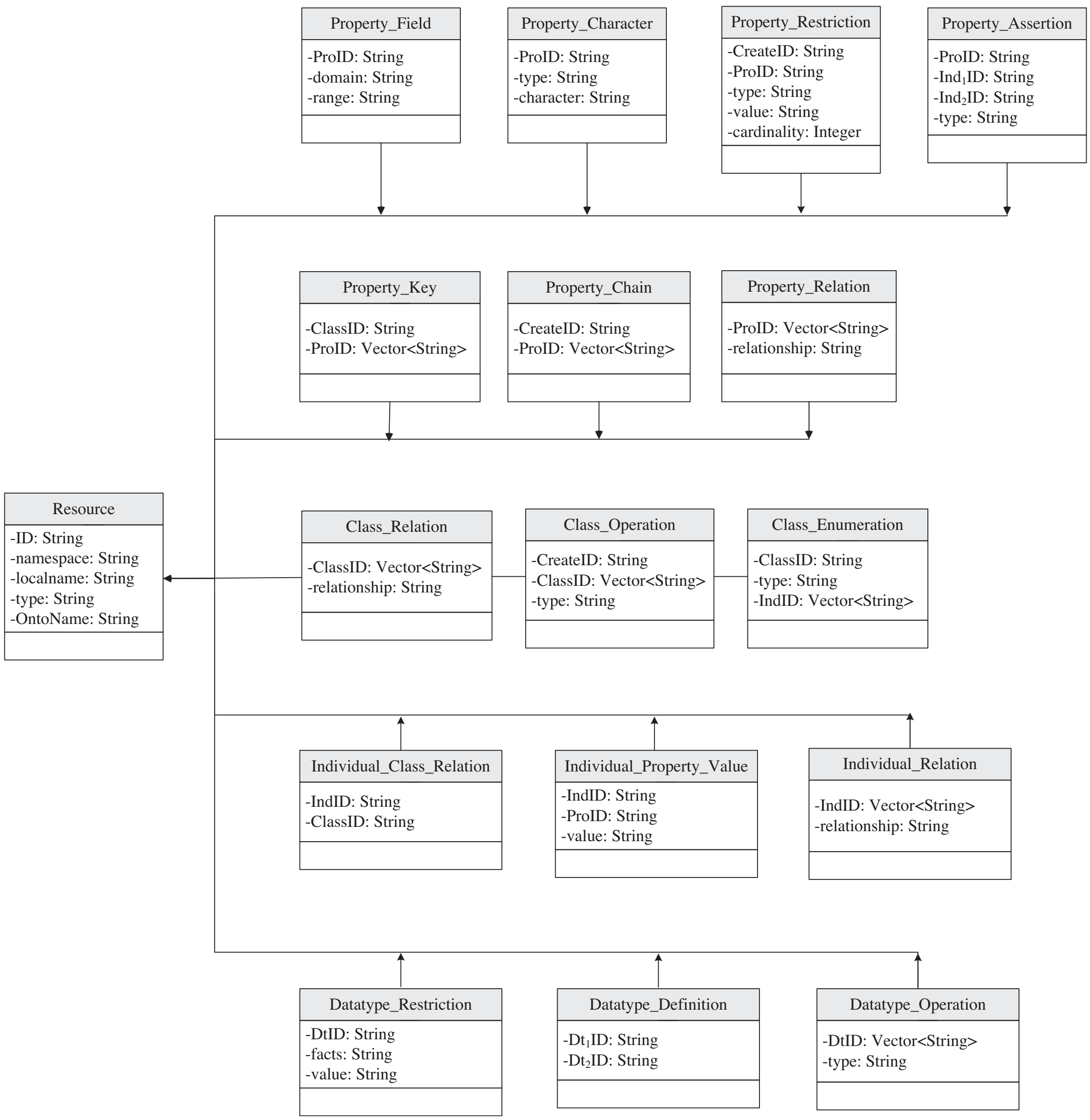


Fig. 2. The overall architecture of storage approach.

(iii) The first new axiom "Class (*Staff* partial *new_c_10*)" is stored in the *Class_Relation* class as shown in Fig. 6; and the second one "*new_c_10* ≡ restriction (*work_in* allValuesFrom(*College*))" will be stored in the *Property_Restriction* class as will be introduced in the later step (6).

(3) Storing the domains and ranges of properties

As mentioned in Section 3.1, the *domains* and *ranges of properties* in an OWL 2 ontology will be stored in a *Property_Field class* (including 3 fields, i.e., *ProID*, *domain*, and *range*).

Fig. 7 shows *Property_Field class* and its *objects* in a target object-oriented database, which stores the domains and ranges of properties in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, for the object property *work_in* (*p_2*), its domain is *Staff* (*c_7*) and range is *College* (*c_2*); for the datatype property *Name* (*p_11*), its domain is *Student* (*c_3*) and range is *xsd:String*.

(4) Storing the key properties

As mentioned in Section 3.1, the *key properties* in an OWL 2 ontology will be stored in a *Property_Key class* (including fields *ClassID* and *ProID*).

Fig. 8 shows the class *Property_Key* and its *objects* in a target object-oriented database, which stores the key properties in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, the property *StuNo* (*p_9*) is the key property of the class *Student* (*c_3*), which is stored in Fig. 8.

(5) Storing the characters of properties

As mentioned in Section 3.1, the *characters of properties* (including *Functional*, *Symmetric*, *InverseFunctional*, *Transitive*, *ObjectHasSelf*, *ReflexiveObjectProperty*, *IrreflexiveObjectProperty*, *AsymmetricObjectProperty*, and *ObjectInverseOf* as introduced in Table 3) in an OWL 2 ontology will be stored in *Property_Character* class.

Fig. 9 shows *Property_Character class* and its *objects* in a target object-oriented database, which stores the characters of properties in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, the functional object properties *study_in* (*p_3*) and *supervise_by* (*p_8*) is stored in Fig. 9.

(6) Storing the restrictions of properties

As mentioned in Section 3.1, the *restrictions of properties* (including *allValuesFrom*, *someValuesFrom*, *hasValue*, *minCardinality*, *maxCardinality*, *Cardinality*, *ObjectMin-*

Table 3
The detailed explanations of the storage architecture in Fig. 2.

Classes	Structure and description
Resource class	***Storing resources in an ontology***, i.e., classes, properties, individuals, and datatypes. This class includes 5 attributes, i.e., ID, namespace, localname, type, and OntoName: – *ID* identifies uniquely a resource in an ontology; – *namespace* and *localname* describe a URIref of any resource; – *type* describes the type of a resource, which may be a class, a property, an individual, or a datatype; – *OntoName* describes the stored ontology name; In the following, ProID e ID denotes a property, ClassID e ID denotes a class, IndID e ID denotes an individual, and DtID e ID denotes a datatype.
Property_Field class	***Storing the domain and range of a property***. – *ProID* identifies uniquely a property in the ontology; – *domain* and*range* store the domain and range of the property.
Property_Character class	***Storing the characters of a property***. – *ProID* denotes a property; – *type*, which describes the type of a property, may be a*datatype* property or an*object* property; – *characters*, which describes the characters of a property, may be *Functional, Symmetric, InverseFunctional, Transitive, ObjectHasSelf, ReflexiveObjectProperty, IrreflexiveObjectProperty, AsymmetricObjectProperty*, or *ObjectInverseOf* (see Tables 1 and 2 in detail).
Property_Restriction class	***Storing the restrictions of a property***. – *CreateID* is a created ID, which can uniquely identify a property restriction; – *ProID* is a property; – *type*, which describes *restrictions* as shown in Tables 1 and 2, may be*allValuesFrom, someValuesFrom, hasValue, minCardinality, maxCardinality, Cardinality, ObjectMinCardinality, ObjectMaxCardinality, ObjectExactCardinality, DataMinCardinality, DataMaxCardinality, DataExactCardinality*; – *value* denotes the value of restriction of a property, where: *value* = ID (i.e., ID in the Resource class) if type = allValuesFrom | someValuesFrom | hasValue | ObjectMinCardinality | ObjectMaxCardinality | ObjectExactCardinality | DataMinCardinality | DataMaxCardinality | DataExactCardinality; otherwise, value is NULL. *cardinality* is the cardinality constraints.
Property_Assertion_Class	***Storing the property assertions***. – *ProID* may be a *datatype* property or an *object* property; – *IndjID* and *Ind2ID* are two individuals; – *type* denotes the type of the assertions, may be *NegativeObjectPropertyAssertion* or*NegativeDataPropertyAssertion*.
Property_Key class	***Storing the key properties***. – *ClassID* is a class; – *ProID* are object or datatype key properties of the class ClassID.
Property_Chain class	***Storing the property chains***. – *CreateID* is a created ID, which can uniquely identify a property chian; – *ProID* are object properties.
Property_Relation class	***Storing the relationships among properties***. – *ProID* are properties; – *relationship* may be *SubPropertyOf, EquivalentProperties, inverseOf, ObjectPropertyChain, InverseObjectProperties, DisjointObjectProperties*, or*DisjointDataProperties*. When the*relationship* is *SubPropertyOf*, the Pro_iID may be the CreateID in the Property_Chain class.
Class_Relation class	***Storing the relationships among classes***. – *ClassID* are classes, which may be the ID in the Resource class, CreateID in the Property_Restriction class, or CreateID in the following Class_Operation class; – *relationship* may be*partial, complete, SubClassOf*, or*DisjointUnion*.
Class_Operation class	***Storing the operations among classes***. – *CreateID* is a created ID, which can uniquely identify a class operation; – *ClassID* are Classes; – *type* is the operation of classes such as*intersectionOf, unionOf, complementOf, EquivlentClasses*, and*DisjointClasses*.
Class_Enumeration class	***Storing the enumerated class***. – *ClassID* is a class. – *IndID* are indivisuals; – *type* is the operation*oneOf* or*EnumeratedClass*.
Individual_Class _Relation class	***Storing the relationships among classes and individuals***. – *IndID* is an individual; – *ClassID* is a class.
Individual_Property_value class	***Storing the values of properties***. – *IndID* is an individual; – *ProID* is an object or datatype property; – *value* is the value of the property.

Table 3 (*continued*)

Classes	Structure and description
Individual_Relation class	***Storing the relationships among individuals***. – *IndID* are individuals; – *relationship* denotes the relationships among individuals such as*SameIndividual* or*DifferentIndividuals*.
Datatype_Restriction class	***Storing the restrictions of datatypes***. – *DtID* denotes a datatype; – *facts* is a constraining facet in XML Schema datatypes, e.g.,*minInclusive* and*minInclusive;* – *value* is the datatype individual as mentioned in Table 1.
Datatype_Definition class	***Storing the definitions of the datatypes***. – *DtiID* is the defined datatype; – *Dt2ID* is the existing datatype.
Datatype_Operation class	***Storing the operations of datatypes***. – *DtID* are*datatypes;* – *type* denotes an operation of datatypes such *as DataIntersectionOf, DataUnionOf*, or*DataComplementOf.*

An OWL 2 ontology O_{uni} = {I, Π, X, Δ, A}:

I = {*Stu_0810351*, *Prof.Mike*, *Information Science*, *Northeastern University*, *0810351*, *21*, *John*}; // I is a set of individuals.

Π = P ∫ T = {*belong_to*, *work_in*, *study_in*, *teach*, *s_choose*, *u_choose*, *supervise*, *supervise_by*, *StuNo*, *Age*, *Name*}; // Π is a set of properties, including object properties P and datatype properties T.

X = {*University*, *College*, *Staff*, *Professor*, *Associate_Prof*, *Course*, *Student*, *Undergraduate*, *Postgraduate*}; // X is a set of classes.

Δ = {*xsd:Integer*, *xsd:String*} // Δ is a set of data range identifiers.

A = { SubClassOf (*Professor* *Staff*),
SubClassOf (*Associate_Prof* *Staff*),
DisjointUnion (*Student* *Undergraduate* *Postgraduate*),
ObjectProperty (*study_in* domain(*Student*) range(*College*) [*Functional*]),
ObjectProperty (*belong_to* domain(*College*) range(*University*)),
InverseObjectProperties (*supervise* *supervise_by*),
HasKey (*Student* *StuNo*),
DatatypeProperty (*age* domain(*Student*) range(*xsd:Integer*)),
Individual (*Stu_0810351* type (*Student*) value (*StuNo*, *0810351^^xsd:String*)
value (*Age*, *21^^xsd:Integer*) value (*Name*, *John^^xsd:String*)),
Individual (*Prof.Mike* type (*Professor*)),
…} // A is a set of axioms defined over I ∫ Π ∫ X ∫ Δ.

Fig. 3. An OWL 2 ontology $\mathcal{O}_{uni}$ modeling parts of the reality at a university.

Cardinality, *ObjectMaxCardinality*, *ObjectExactCardinality*, *DataMinCardinality*, *Data- MaxCardinality*, *DataExactCardinality* as introduced in Table 3) in an OWL 2 ontology will be stored in *Property_Restriction* class.
Fig. 10 shows *Property_Restriction class* and its *objects* in a target object-oriented database, which stores the restrictions of properties in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, the restriction of the property*work_in* (*p_2*) "restriction (*work_in* allValuesFrom(*College*))", the restrictions of the property*s_choose* (*p_4*) "ObjectMinCardinality (3 *s_choose* [*Course*])" and "ObjectMaxCardinality (12 *s_choose* [*Course*])", and so on, are stored in Fig. 10.

(7) Storing the relationships among properties
As mentioned in Section 3.1, the *relationships among properties* in an OWL 2 ontology (including *SubPropertyOf*, *EquivalentProperties*, *inverseOf*, *ObjectPropertyChain*, *InverseObjectProperties*, *DisjointObjectProperties*, or *DisjointDataProperties* in Table 3) will be stored in a *Property_Relation* class (including fields*Vector<ProID>* and *relationship*).
Fig. 11 shows *Property_Relation class* and its *objects* in a target object-oriented database, which stores the relationships among properties in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, SubPropertyOf (*u_choose* *s_choose*) and Inverse ObjectProperties (*supervise supervise_by*) are stored in Fig. 11.

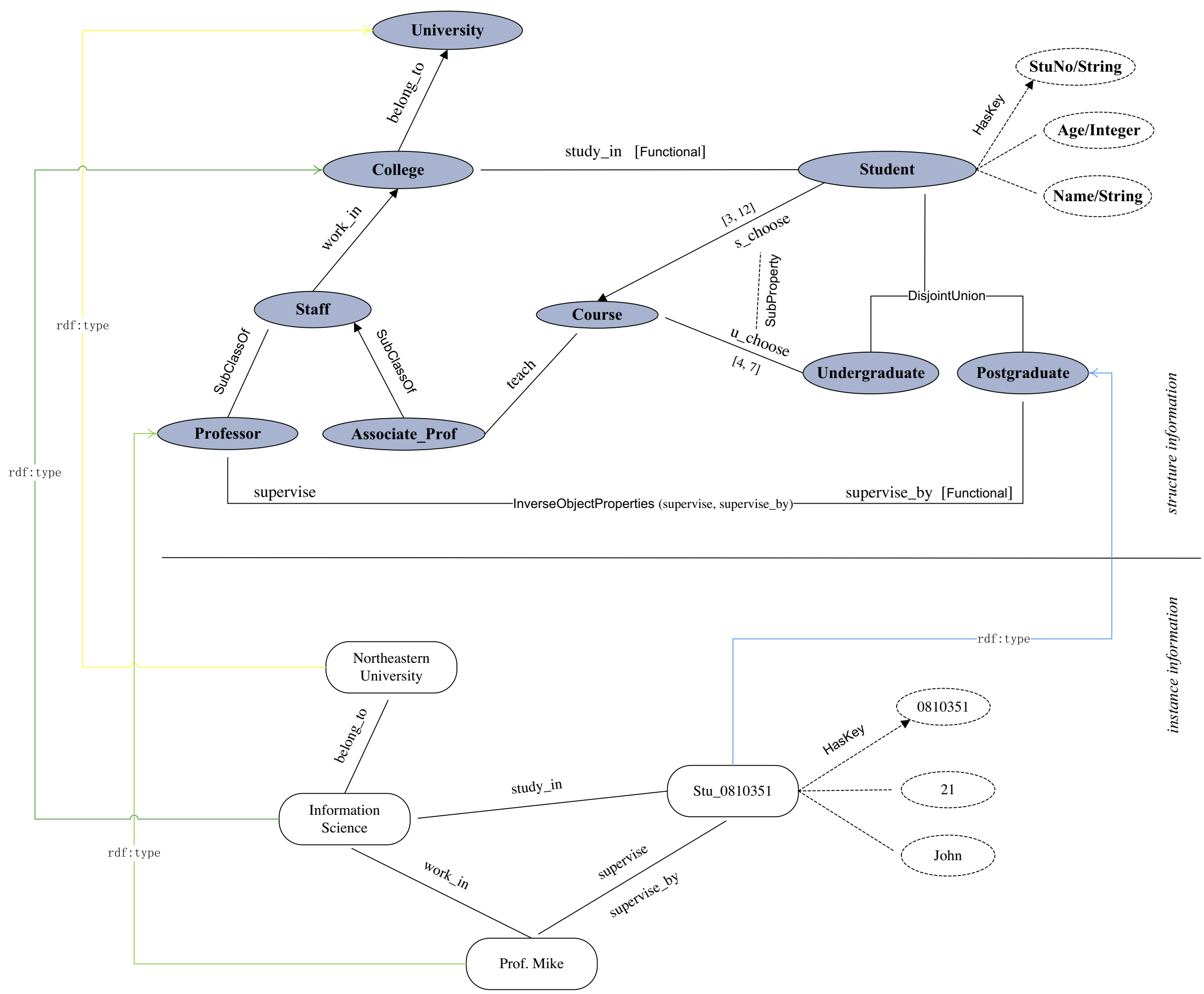


Fig. 4. The diagram of the OWL 2 ontology $\mathcal{O}_{\text{uni}}$ in Fig. 3.

All the constructors of OWL 2 ontologies mentioned in Section 2.1 can be stored in object-oriented databases following the similar procedures given above. The following section will further store the instance information of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3 in the object-oriented database.

3.2.2. Storing the instance information of OWL 2 ontology in object-oriented database

The *instance information* of the OWL 2 ontology $\mathcal{O} = \{\mathcal{I}, \mathcal{P}, \mathcal{X}, \mathcal{D}, \mathcal{A}\}$ includes the sets of individuals $\mathcal{I}$ and axioms $\mathcal{A}$. Based on the architecture proposed in Section 3.1, the following procedures will store the *instance information* of the OWL 2 ontology $\mathcal{O}_{\text{uni}}$ in Fig. 3 in an object-oriented database.

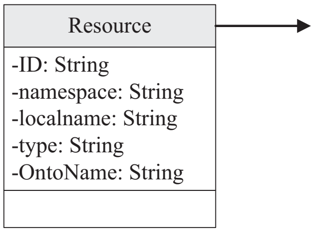
The *instance information* of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3 includes:

- several individuals: $\mathcal{I}$ = {*Northeastern University* (*i_1*), *Information Science* (*i_2*), *Prof.Mike* (*i_3*), *Stu_0810351* (*i_4*)};
- several individual axioms: $\mathcal{A}$ = {Individual (*Stu_0810351* type (*Student*) value (*StuNo*, 0810351 ^^*xsd:String*) value (*Age*, *21*^^*xsd:Integer*) value (*Name*, *John*^^*xsd:String*)), Individual (*Prof.Mike* type (*Professor*)), ..., DifferentIndividuals (*Northeastern University*, *Information Science*, *Prof.Mike*, *Stu_0810351*)}. Here, some axioms are omitted.

(8) Storing the relationships of individuals/classes
As mentioned in Section 3.1, the *relationships of individuals/classes* in an OWL 2 ontology will be stored in *Individual_Class_Relation* class (including fields*IndID* and *ClassID*).
Fig. 12 shows *Individual_Class_Relation class* and its *objects* in a target object-oriented database, which stores the relationships of individuals/classes in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, *Northeastern University* (*i_1*) is an instance of the class *University* (*c_1*), *Stu_0810351* (*i_4*) is an instance of the class *Student* (*c_3*), and so on, which are stored in Fig. 12.

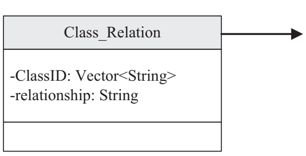
(9) Storing the values of properties of individuals
As mentioned in Section 3.1, the *values of properties of individuals* in an OWL 2 ontology will be stored in *Individual_Property_Value* class (including fields *IndID*, *ProID*, and *value*).
Fig. 13 shows *Individual_Property_Value class* and its *objects* in a target object-oriented database, which stores the values of properties in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, the values of properties of individual *Stu_0810351* (*i_4*) are stored in Fig. 13.

(10) Storing the relationships among individuals
As mentioned in Section 3.1, the *relationships among individuals* in an OWL 2 ontology will be stored in *Individual_Relation* class (including fields *Vector*<*IndID*> and *relationship*).
Fig. 14 shows *Individual_Relation class* and its *objects* in a target object-oriented database, which stores the relationships



ID	namespace	localname	type	OntoName
c_1	http://www...	University	class	O_1
c_2	http://www...	College	class	O_1
c_3	http://www...	Student	class	O_1
c_4	http://www...	Undergraduate	class	O_1
c_5	http://www...	Postgraduate	class	O_1
c_6	http://www...	Course	class	O_1
c_7	http://www...	Staff	class	O_1
c_8	http://www...	Professor	class	O_1
c_9	http://www...	Associate_Prof	class	O_1
new_c_10	http://www...	restriction (work_in ...	class	O_1
new_c_11	http://www...	restriction (s_choose ...	class	O_1
...	...	...	class	O_1
i_1	http://www...	Northeastern University	individual	O_1
i_2	http://www...	Information Science	individual	O_1
i_3	http://www...	Prof. Mike	individual	O_1
i_4	http://www...	Stu_0810351	individual	O_1
...	...	...	individual	O_1
d_1	http://www...	xsd:String	datatype	O_1
d_2	http://www...	xsd:Integer	datatype	O_1
p_1	http://www...	belong_to	ObjectProperty	O_1
p_2	http://www...	work_in	ObjectProperty	O_1
p_3	http://www...	study_in	ObjectProperty	O_1
p_4	http://www...	s_choose	ObjectProperty	O_1
p_5	http://www...	u_choose	ObjectProperty	O_1
p_6	http://www...	teach	ObjectProperty	O_1
p_7	http://www...	supervise	ObjectProperty	O_1
p_8	http://www...	supervise_by	ObjectProperty	O_1
p_9	http://www...	StuNo	DatatypeProperty	O_1
p_10	http://www...	Age	DatatypeProperty	O_1
p_11	http://www...	Name	DatatypeProperty	O_1

Fig. 5. The class Resource and its objects in a target object-oriented database for storing the resources of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.



<Vector>ClassID	relationship
<c_8, c_7>	SubClassOf
<c_9, c_7>	SubClassOf
<c_3, c_4, c_5>	DisjointUnion
<c_7, new_c_10>	partial
...	...

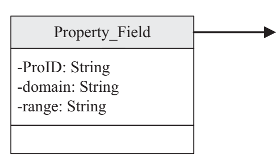
Fig. 6. The class Class_Relation and its objects in a target object-oriented database for storing the relationships among classes of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

among individuals in the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3. For example, the *relationships among several individuals* "DifferentIndividuals (*Northeastern University*, *Information Science*, *Prof.-Mike*, *Stu_0810351*)" are stored in Fig. 14.

According to the previous sections, an OWL 2 ontology, including classes, properties, individuals, and axioms, can be stored in an object-oriented database. In the following section we prove the correctness of the approach.

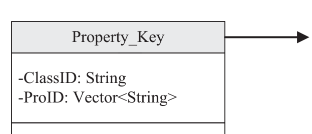
3.3. Correctness of the storage approach

In the following we prove the correctness of the storage approach. As mentioned in the literature (e.g., [3,13]), when storing domain knowledge, one should ensure that the result of the



ProID	domain	range
p_1	c_2	c_1
p_2	c_7	c_2
p_3	c_3	c_2
p_4	c_3	c_6
p_5	c_4	c_6
p_6	c_9	c_6
p_7	c_8	c_5
p_8	c_5	c_8
p_9	c_3	d_1
p_10	c_3	d_2
p_11	c_3	d_1

Fig. 7. The class Property_Field and its objects in a target object-oriented database for storing the domains and ranges of properties of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.



ClassID	ProID
c_3	p_9

Fig. 8. The class Property_Key and its objects in a target object-oriented database for storing the key properties of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

Property_Character
-ProID: String -type: String -character: String

ProID	type	character
p_3	ObjectProperty	Functional
p_8	ObjectProperty	Functional

Fig. 9. The class Property_Character and its objects in a target object-oriented database storing the characters of properties of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

storage can describe the original information. From the storage procedures in the previous sections, it shows that our storage approach can be seen as a transformation. The evaluation of transforming, mapping or storing information is an important and central issue, but it is also a difficult task. Currently there are still no any standard frameworks/metrics for evaluation, and many works used the standard information retrieval metrics to evaluate their approaches (e.g., [19,22,31]). On this basis, we also evaluate our approach using the relative *information capacities* of the source and target resources.

Based on the notion of *information capacity* [19,22], we give the formal proof of the correctness of our storage approach (i.e., Theorem 1).

Theorem 1. *Given an OWL 2 ontology $\mathcal{O}$, the storage procedure from $\mathcal{O}$ to an object-oriented database OODB in the previous sections is an information capacity preserving storage.*

Proof. Let $I(\mathcal{O})$ and $I(OODB)$ be consistent instances of schemata $\mathcal{O}$ and *OODB*. A mapping from $I(\mathcal{O})$ to $I(OODB)$, i.e., $\lambda : I(\mathcal{O}) \to I(OODB)$ can be established as follows: assuming that $\mathfrak{I} \in I(\mathcal{O})$ is an instance of $\mathcal{O}$, then $\lambda(\mathfrak{I}) \in I(OODB)$ is an instance of *OODB* derived according to the storage approach in Sections 3.1 and 3.2. Formally, the mapping λ can be defined as (taking the Individual_Property_Value class for example):

For $i = 1$ to m	// m classes
$c_i = \{p_i^1, p_i^2, \ldots, p_i^n\}$	// each class c_i has n properties
For $k = 1$ to s	// each class has s instances
For $j = 1$ to n	
$\lambda(\mathfrak{I})\left[IndID_i^k\right] \leftarrow id_i^k$	// *IndID* is the individual identifier in Fig. 2 and Table 3, and id_i^k is the identifier of the kth instance of the class c_i and is stored in the field $IndID_i^k$.
$\lambda(\mathfrak{I})\left[ProID_i^j\right] \leftarrow p_i^j$	//*ProID* is the property identifier in Fig. 2 and Table 3 property p_i^j is stored in the field $ProID_i^j$.
$\lambda(\mathfrak{I})\left[value_i^j\right] \leftarrow \mathfrak{I}\left[p_i^j\right]$	//*value* is the property value in Fig. 2 and Table 3, and $\mathfrak{I}\left[p_i^j\right]$ denotes a component value of p_i^j of $\mathfrak{I}$ and is stored in the field $value_i^j$.

Property_Relation
-ProID: Vector<String> -relationship: String

<Vector>ProID	relationship
<p_5, p_4>	SubPropertyOf
<p_7, p_8>	InverseObjectProperties

Fig. 11. The class Property_Relation and its objects in a target object-oriented database storing the relationships among properties of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

Individual_Class_Relation
-IndID: String -ClassID: String

IndID	ClassID
i_1	c_1
i_2	c_2
i_3	c_8
i_4	c_3

Fig. 12. The class Individual_Class_Relation and its objects in a target object-oriented database storing the relationships of individuals/classes of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

Individual_Property_Value
-IndID: String -ProID: String -value: String

IndID	ProID	value
i_4	p_9	0810351^^d_1
i_4	p_10	21^^d_2
i_4	p_11	John^^d_1

Fig. 13. The class Individual_Property_Value and its objects in a target object-oriented database storing the values of properties of individuals of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

Individual_Relation
-IndID: Vector<String> -relationship: String

<Vector>IndID	relationship
<i_1, i_2, i_3, i_4>	DifferentIndividuals

Fig. 14. The class Individual_Relation and its objects in a target object-oriented database for storing the relationships among individuals of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

The following proves the mapping λ is a function *firstly*. From the definition of λ above, each property value of an individual belonging to a class in the OWL 2 ontology $\mathcal{O}$ corresponds to an object in the Individual_Property_Value class. Since the object identifier of the class can ensure that $\lambda(\mathfrak{I})$ is an instance of *OODB* (an object), i.e., λ is a function from $\mathcal{O}$ to *OODB*; *Secondly*, we further prove that λ is an injective function. Let $\mathfrak{I}_1 = (\mathfrak{I}_1[p_i^1], \mathfrak{I}_1[p_i^2], \ldots, \mathfrak{I}_1[p_i^n])$ and $\mathfrak{I}_2 = (\mathfrak{I}_2[p_i^1], \mathfrak{I}_2[p_i^2], \ldots, \mathfrak{I}_2[p_i^n])$ be two different instances of the class c_i, then there is at least one $j \in \{1, \ldots, n\}$ such that $\mathfrak{I}_1\left[p_i^j\right] \neq \mathfrak{I}_2\left[p_i^j\right]$. According to the definition of λ above, there are objects $\lambda(\mathfrak{I}_1) = \left(\lambda(\mathfrak{I}_1)\left[IndID_i^1\right], \lambda(\mathfrak{I}_1)\left[ProID_i^j\right], \lambda(\mathfrak{I}_1)\left[value_i^j\right]\right)$ and $\lambda(\mathfrak{I}_2) = \left(\lambda(\mathfrak{I}_2)\left[IndID_i^2\right], \lambda(\mathfrak{I}_2)\left[ProID_i^j\right], \lambda(\mathfrak{I}_2)\left[value_i^j\right]\right)$ in the class,

Property_Restriction
-CreateID: String -ProID: String -type: String -value: String -cardinality: Integer

CreateID	ProID	type	value	cardinality
new_c_10	p_2	allValuesFrom	c_2	
new_c_11	p_4	ObjectMinCardinality	c_6	3
new_c_11	p_4	ObjectMaxCardinality	c_6	12
...	...	...	...	...

Fig. 10. The class Property_Restriction and its objects in a target object-oriented database storing the restrictions of properties of the OWL 2 ontology $\mathcal{O}_{uni}$ in Fig. 3.

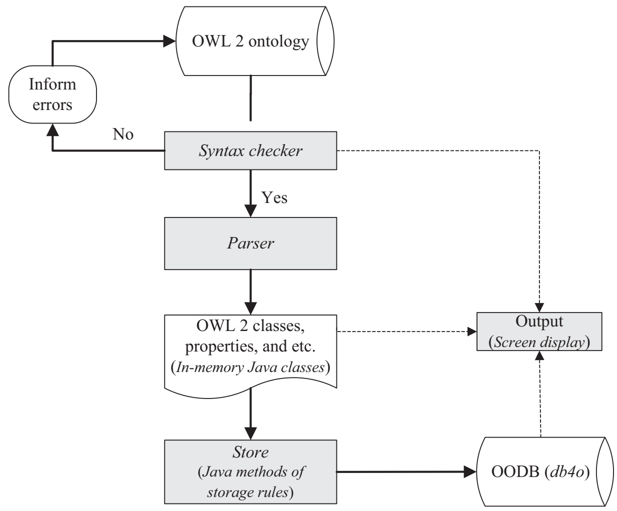


Fig. 15. The overall architecture of OWL2OODB.

where $j = 1, \ldots, n$. Furthermore, since there is at least one j such that $\Im_1\left[p_i^j\right] \neq \Im_2\left[p_i^j\right]$ as mentioned above, it follows that there is also at least one $j \in \{1, \ldots, n\}$ such that $\lambda(\Im_1)\left[value_i^j\right] \neq \lambda(\Im_2)\left[value_i^j\right]$, i.e., we have $\lambda(\Im_1) \neq \lambda(\Im_2)$, that is, λ is an injective function. □

Based on the previous sections and the Theorem 1, it is shown that the storage from an OWL 2 ontology to an object-oriented database is an information capacity preserving and correct storage. That is, given an OWL 2 ontology, the proposed approach can correctly and completely store the information of the OWL 2 ontology in an object-oriented database. Furthermore, in order to implement the automated storage of OWL 2 ontologies, in the following section we will develop a prototype storage tool.

3.4. Prototype storage tool

Following the proposed storage approach in the previous sections, we implement a prototype tool called *OWL2OODB* for storing OWL 2 ontologies in object-oriented databases. In the following, we briefly sketch the design and implementation of *OWL2OODB*.

The implementation of *OWL2OODB* is based on JAVA JDK 1.7.0 platform, and the Graphical User Interface is exploited by using the java.awt and javax.swing packages. The used object-oriented database *db4o* [26], which is a widely used and the most popular open source object database recommended by Object-oriented Database Management Group (*ODMG*) [10], enables Java and .NET developers to store and retrieve any application object. The database *db4o* provides several main packages including com.db4o, com.db4o.ext, com.db4o.config, and com.db4o.query. The detailed introduction about the object-oriented database *db4o* can be found in [26]. Fig. 15 shows the overall architecture of *OWL2OODB*.

OWL2OODB includes four main modules as shown in Fig. 15, i.e., *syntax checking module*, *parsing module*, *storage module*, and *output module*:

- *Syntax checking module*: The prototype tool opens an OWL 2 ontology file for reading, and then the *syntax checking module* checks the correctness of the syntax. If the syntax does not match OWL 2 notations (see Section 2.1), then the tool will inform errors.
  Here, we provide several examples represented in OWL 2 syntax that can be imported in the tool (the complete syntax can be found in Tables 1 and 2 of Section 2.1). For example: (*i*) the OWL 2 property axioms ObjectProperty (*study_in* domain(*Student*) range(*College*) [Functional]); InverseObjectProperties (*supervise supervise_by*); HasKey (*Student StuNo*); and etc; (*ii*) the OWL 2 class axioms SubClassOf (*Associate_Prof Staff*); DisjointUnion (*Student Undergraduate Postgraduate*); and etc; (*iii*) the OWL 2 individual axioms Individual (*Stu_0810351* type (*Student*)); Individual (*Prof.Mike* type (*Professor*)); and etc.
- *Parsing module*: If the syntax of the file matches OWL 2 notations, the *parsing module* parses the OWL 2 ontology file and stores the parsed results as Java classes.
- *Storage module*: The *storage module* stores the parsed OWL 2 ontology information in a target object-oriented database according to the proposed approach in Sections 3.1 and 3.2. Moreover, the tool *OWL2OODB* provides a graphical user interface to show the stored data in the target object-oriented database. In addition, a query interface is also provided in order that users can conveniently retrieve the data in databases.
- *Output module*: The *output module* finally displays the information including the source OWL 2 ontology information, the parsed results, and the target object-oriented database information on the tool screen (see Fig. 17).

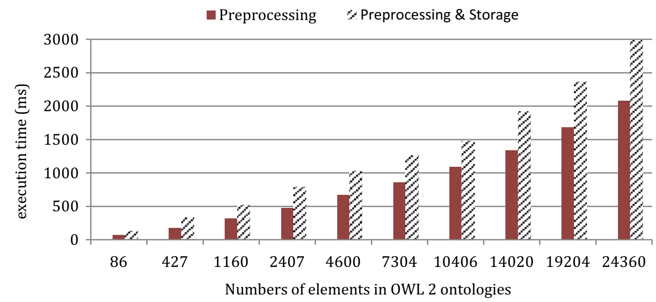


Fig. 16. The execution time of the tool OWL2OODB routine on several OWL 2 ontologies.

We carried out some storage experiments of OWL 2 ontologies using the implemented tool *OWL2OODB*, with a PC (Inter Core i7 CPU@3.40 GHz, RAM 8.0 GB and Windows 7 system). As we have known, currently there is no a widely accepted or standard dataset of OWL 2 ontologies. In this case, the OWL 2 ontologies used in our experiments are mainly from the following several parts:

- Some ones come from the existing common OWL 2 ontologies (e.g., the complete sample OWL 2 ontology,[1] the populated version of the Wine ontology that contains about 483 individuals and uses most of the OWL 2 DL constructs,[2] the OWL 2 ontology bro in Dumontier Lab,[3] the FMA_OWL 2 ontology,[4] the Chemistry ontology in [32], and some ontologies mentioned in Part 7[5]);
- Some ones are derived by importing some additional OWL 2 constructs into the existing familiar OWL 1 ontologies by means of the popular ontology editor Protégé[6] (e.g., the AERO and NBO ontologies in OBO Foundry[7], the Molecule and Pharmacogenomics ontologies[3], the LKIF Core ontology,[8] and some ontologies in the ELK project[9]);

[1] http://www.w3.org/TR/2012/REC-owl2-primer-20121211/.
[2] https://code.google.com/p/epistemicdl/source/browse/trunk/EQuIK/wine_1.owl.
[3] http://dumontierlab.com/?page=ontologies.
[4] http://gforge-lirmm.lirmm.fr/gf/download/docmanfileversion/211/743/FMA_owl2_noMTC_100417.zip.
[5] http://www.w3.org/TR/2012/REC-owl2-new-features-20121211/.
[6] http://protege.stanford.edu/.
[7] http://www.obofoundry.org/
[8] http://github.com/RinkeHoekstra/lkif-core.
[9] https://code.google.com/p/elk-reasoner/wiki/TestOntologies.

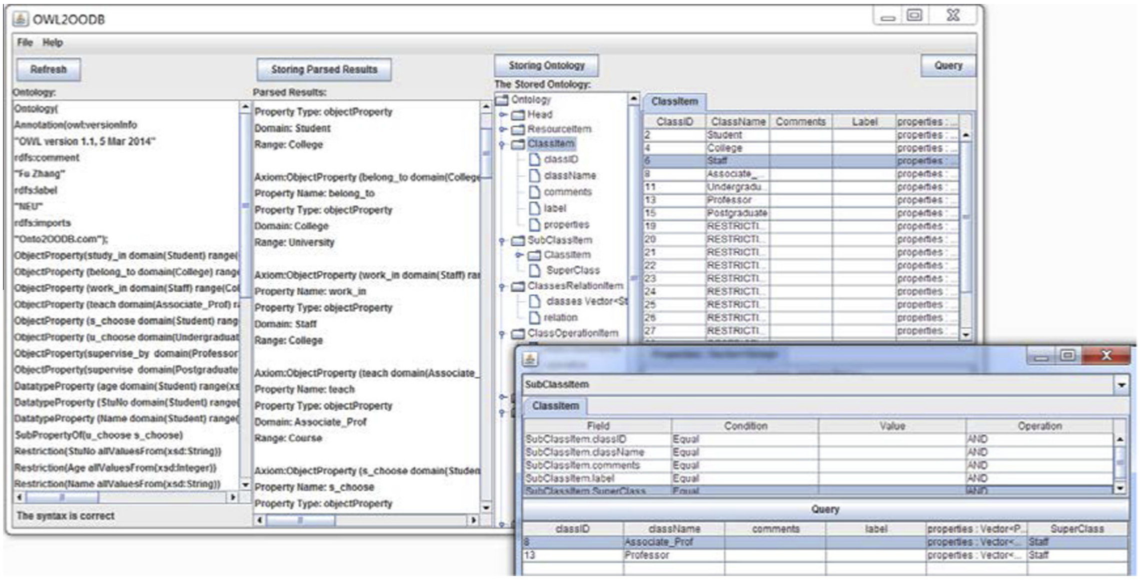


Fig. 17. The screen snapshot of OWL2OODB.

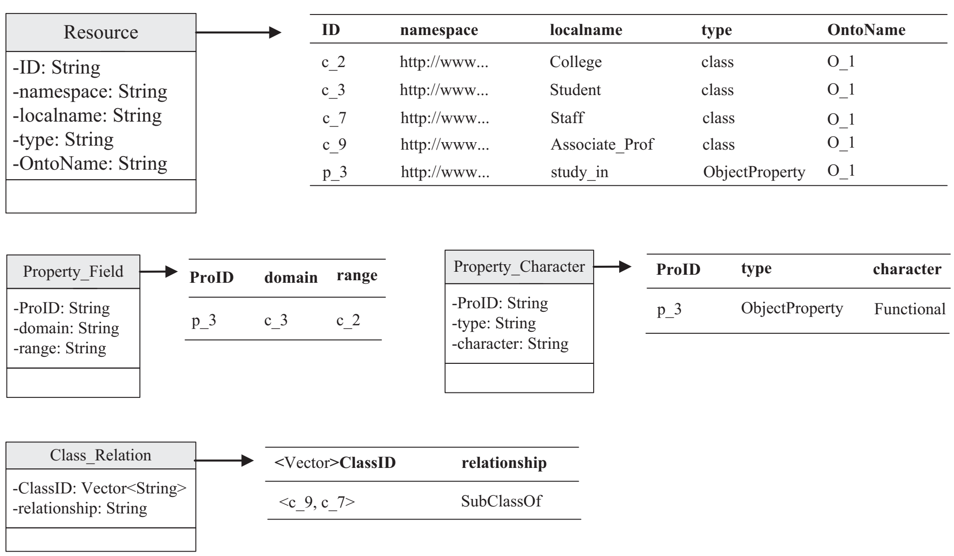


Fig. 18. A part of parsed results of the ontology in the database.

- Some others are created manually by us with the ontology editor Protégé (e.g., one of the OWL 2 ontologies mentioned in Section 3.2).

The sizes of the OWL 2 ontologies range about from 50 to 36,000 in our current tests. Here, the scale of an OWL 2 ontology denotes the numbers of classes, properties, individuals, and axioms in the OWL 2 ontology, and it can be measured after we parse the ontology in the parsing module as shown in Fig. 15. Fig. 16 shows the actual execution time routines in the *OWL2OODB* tool running several OWL 2 ontologies, where the *preprocessing* denotes the *operations of reading* and *parsing* the classes, properties, individuals, and axioms of the OWL 2 ontologies and *preparing* the data in computer memory for the usage in the storage procedure. Case studies show that our approach and prototype tool actually work. However, we also should be noted that, with the development of ontologies in various application domains, some larger scale OWL 2 ontologies may occur, and we will test them in our future work to further evaluate the tool.

In the following, we give the screen snapshot of *OWL2OODB*, and an example is provided to well show the running process of the tool *OWL2OODB*. Fig. 17 shows the screen snapshot of *OWL2OODB*, which displays the storage of an OWL 2 ontology (including the information of the OWL 2 ontology in Fig. 4) in an object-oriented database. In Fig. 17, the source OWL 2 ontology information, the parsed results, and the target object-oriented database information are displayed in the left, middle and right areas, respectively. Moreover, in order that users can conveniently retrieve the data in databases, we provide a query interface in the tool *OWL2OODB*. When users click the button “Query” in the graphical interface of *OWL2OODB* as shown in Fig. 17, a query window can be opened. Then the users can choose the classes that they want to query, and fill in the query conditions to execute the query. The screen snapshot of a brief query is also shown in the bottom right area of Fig. 17.

In order to well show the running process of the tool *OWL2OODB* in Fig. 17, here we provide an example in detail. *For example, after inputting* the OWL 2 ontology information (for simplicity here we only give two axioms of the OWL 2 ontology):

"ObjectProperty (*study_in* domain(*Student*) range(*College*) [Functional])"
"SubClassOf (*Associate_Prof Staff*)"

the tool will further *parse the axioms*, and the corresponding parsed results are:

Axiom: ObjectProperty (*study_in* domain(*Student*) range(*College*) [Functional])
Property Name: *study_in*
Property Type: *ObjectProperty*
Domain: *Student*
Range: *College*
Character: Functional
Axiom: SubClassOf (*Associate_Prof Staff*)
Type: SubClassOf
SubClass Name: *Associate_Prof*
SuperClass Name: *Staff*

finally, the tool stores the parsed results of the OWL 2 ontology in an object-oriented database according to the rules mentioned in Sections 3.1 and 3.2, and the corresponding classes "Resource", "Property_Field", "Property_Character", and "Class_Relation" are created and some objects belonging to these classes are inserted as follows (for visualization, the classes and objects in the object-oriented database are showed by the forms of diagrams and tables) (see Fig. 18).

After storing the OWL 2 ontology in the object-oriented database, users can access to the object-oriented database by using regular queries. The object-oriented database *db4o* provides the query engine so that users may access to the data in databases. For example, let us assume that a user wants to find the *domain and range* of a given property (e.g., the property *p_3* mentioned above), and the user can write the following query:

```
Query query = 𝒪.query();                        // creating a new query for
                                                querying the database 𝒪
query.constrain(Property_Field.class);          // querying the class Property_Field
                                                in 𝒪
Query pointQuery = query.descend                // the query condition is ProID =
   ("ProID").constrain("p_3");                  p_3
ObjectSet result =                              // executing the query
   query.execute();
listResult(result);                             // listing the query results
```

Also, a user may want to find all the *subclass/superclass* relationships in the original OWL 2 ontology, and the user can write the following query:

```
Query query = 𝒪.query();
query.constrain(Class_Relation.class);
Query pointQuery = query.descend("relationship").
   constrain("SubClassOf");
ObjectSet result = query.execute();
listResult(result).
```

Being similar to the queries shown above, users also may ask for the other queries, such as finding the disjoint classes with a given class, the super-properties or sub-properties of a given property, and so on. The detailed introduction about the query syntax of *db4o* can be found in [26]. Moreover, a query interface is provided in the prototype tool *OWL2OODB* as has been shown in Fig. 17.

3.5. Discussions

Until now, we propose an approach for storing OWL 2 ontologies in object-oriented databases, which is our main purpose of this paper. Further, the implemented prototype tool and some tests demonstrate that our approach actually works. However, with *the increasing of the scale* and *the richer of semantics expressiveness* of ontologies in the future real-world applications, it can be found that there are still some research issues on the storage of ontologies to be tackled, and our approach and tool may be further enhanced in the following several issues:

- *The issue of semantics preservation*: It should be noted that, in some cases the storage is not really lossless in the sense that there may be some *loss of semantics*, since it is well known that ontologies are semantically richer than databases [1,3,6,13,24,34–36,40]. Ontologies are primarily used for interoperability and they have richer capabilities to *represent semantics* and *infer implicitly knowledge* from the knowledge that is explicitly contained in the ontologies [29]. The primary use of databases (e.g., relational and object-oriented databases) is to structure, store and query a set of data. This differences impact on the preservation of semantics when storing OWL 2 ontologies in object-oriented databases, which can be further explained with the following examples:
  For example, given a *symmetric property R* in an OWL 2 ontology, i.e., *R*[*Symmetric*], and two individuals o_1, o_2 such that $(o_1, o_2) \in R$, all of these can be fully stored in an object-oriented database according to the storage approach proposed in the previous sections. In the OWL 2 ontology, since *R* is a symmetric property, it implies that $(o_2, o_1) \in R$. But, the implicit knowledge $(o_2, o_1) \in R$ is not stored in the object-oriented database. Similarly, for a *transitive property S* in an OWL 2 ontology, i.e., *S*[*Transitive*], and there are several individuals o_1, o_2, o_3 such that $(o_1, o_2) \in S, (o_2, o_3) \in S$, all of these can also be fully stored in an object-oriented database. In the OWL 2 ontology, since *S* is a transitive property, it implies that $(o_1, o_3) \in S$, while the implicit knowledge $(o_1, o_3) \in S$ is not stored in the object-oriented database.
  Extending an existing database system with reasoning capabilities for supporting the reasoning of large ontologies stored in databases may solve the problem of the loss of semantics, which will be investigated in our near future work. But it should be noted that, although some of the semantics captured in ontologies are inevitably lost when ontologies are stored in databases (e.g., relational databases [3,13,35]; object-relational databases [6]; and object-oriented databases proposed in the paper), the information of the OWL 2 ontologies can be fully stored in object-oriented databases based on our work in the previous sections. That is to say, by retrieving the object-oriented databases, the original explicit information of the OWL 2 ontologies can be found and further used in the Semantic Web as well as some applications using ontologies. The information capacity preserving and correct storage of OWL 2 ontologies in object-oriented databases proposed in our work makes it possible to manage the knowledge of ontologies in databases.
- *The issue of scale of ontologies*: As we have mentioned in Section 3.4, currently there is no a widely accepted or standard dataset of OWL 2 ontologies. Therefore, in our current experiments we search some existing OWL 2 ontologies and create some OWL 2 ontologies by means of the ontology editor Protégé, and we further provide the execution time of the storage running some scale OWL 2 ontologies. The results show that our approach and tool actually work. However, we also should be noted that, with the development of ontologies in the future various application domains, some super large scale OWL 2 ontologies may

occur, and we will test them in our future work to further evaluate the tool.

- *The issue of query inference*: It should be noted that, unlike ontologies or knowledge bases, databases store and retrieve explicit data well, but they generally do not perform inference. Currently, after storing OWL 2 ontologies in object-oriented databases according to our approach and tool, some queries may be done by means of the query interface in our tool or by using regular *db4o* queries, while reasoning is not performed while processing queries. In our near future work, on the basis of the approach and tool for storing OWL 2 ontologies proposed in the previous sections, we will further study the effective integration of storing, querying and reasoning, and test the querying and reasoning efficiency in depth.

In summary, our current work develops an approach and a prototype tool to fulfill a complete storage of OWL 2 ontologies in object-oriented databases, which is our main purpose of this paper. Some scale experiments show that our approach and tool actually work. The work may be useful for realizing the efficient management of knowledge in ontologies, and also may act as a gap-bridge between the existing object-oriented database applications and the Semantic Web. Moreover, we provide a query interface for querying OWL 2 ontologies stored in object-oriented databases. In addition, we make a discussion about the approach in several aspects of semantics, scale and inference, and also several directions for future researches are pointed out.

4. Related work

Relating Semantic Web ontologies with databases becomes a topical problem as ontologies provide richer capabilities for representing and reasoning on knowledge and databases may store and retrieve explicit data well. The growing number of methodologies and tools are considering this problem and they are looking from the point of view of ontologies or from the point of databases. A comprehensive review about *round-tripping between databases and ontologies* can be found in [18,24,30,41]. Table 4 briefly summarizes some proposals of relating ontologies with databases.

As shown in Table 4, several categories of approaches are related to our work according to their focuses.

The first category focuses on transforming databases into ontologies, which has increasingly attracted considerable attention because much knowledge and data is still stored in databases and is not published as an open Web of inter-referring resources. Currently, there are some approaches for transforming relational databases into ontologies [4,5,12,15,21]. Moreover, in [11], a rule-based approach for transforming object-relational databases into ontologies was proposed. In addition, more recently, the formal approaches and tools for transforming XML and object-oriented databases into ontologies was developed in [38,39], respectively. A comprehensive review about database-to-ontology mapping can be found in [30,41]. *The motivations and approaches of these researches are different comparing to this paper, and our work aims at storing ontologies in object-oriented databases.*

The second category focuses on storing ontologies in databases, which is closely related to our work. There are today several proposals for *storing OWL 1 ontologies in relational databases* [1,3,13,17,20,28,34,35,40]. A first attempt to store ontologies in relational databases was made in [7], where an algorithm was developed to map OWL 1 to relational schema. From then on, some proposals were proposed to store OWL 1 ontologies. An approach for storing OWL 1 ontologies in SQL relational databases was proposed in [3]. In [9], a relational database storage and inference system for OWL 1 ontologies called Minerva was developed. Further, taking the Minerva system as an example, a more in-depth discussion about OWL 1 ontology storage in database was done in [17]. More recently, an OntoMinD approach was proposed in [1] for reasoning with vary large ontologies by storing DL-Lite(R,∩) ontologies in relational databases. In [37], the authors dealt with the need for manage fuzzy data in the Semantic Web and provided several rules for storing fuzzy OWL 1 ontologies in fuzzy relational databases, but our work in this paper focuses on crisp ontologies and object-oriented databases. The other efforts on storing OWL 1 ontologies in relational databases were made in [17,20,28,34,35]. Moreover, how to *store OWL 1 ontologies in object-relational databases* was investigated in [6], where several brief rules were given to transform classes, class inheritances, cardinalities, and object properties of ontologies into object-relational databases. Furthermore, with the appearance of the ontology language OWL 2 (OWL 2 is an extension and revision of OWL 1), how to store ontologies described in OWL 2 metamodels in relational database schemas was discussed in [33,36].

In summary, these proposals mentioned above give us good hints for developing the approach in this paper, but they are different comparing to this paper: (i) the motivations and approaches of *the first category are opposite to our work*, and they focus on transforming databases into ontologies while our work aims at transforming ontologies into databases for storing ontologies; (ii) most of the approaches in *the second category* focus on storing OWL 1 ontologies in *relational* or *object-relational databases*, little research on storage of OWL 2 ontologies has been done. In particular, *there is no report on storing OWL 2 ontologies in object-oriented databases*. Storing OWL 2 ontologies in object-oriented databases may be useful for realizing the efficient management of large amounts of knowledge in the Semantic Web. Based on the observations mentioned above, to fulfill a complete storage of OWL 2 ontologies, an approach and a prototype tool for storing OWL 2 ontologies in object-oriented databases are developed in this paper.

5. Conclusions and future works

In this paper we investigated the storage of OWL 2 ontologies in object-oriented databases, and proposed a formal approach and developed a prototype tool for storing OWL 2 ontologies in

Table 4
A brief summarization of round-tripping between databases and ontologies.

Motivations	References	Main contributions
Database-to-ontology mapping	[5,12,15,21], etc. (see [30,41] for reviews)	Mapping relational databases to ontologies
	[11]	Mapping object-relational databases to ontologies
	[38]	Mapping object-oriented databases to ontologies
Ontology-to-database storing	[1,3,13,17,20,28,34,35,40]	Storing *OWL 1 ontologies* in *relational databases*
	[6]	Storing *OWL 1 ontologies* in *object-relational databases*
	[33,36]	Storing *OWL 2 ontologies* in *relational database schemas*
	There is no report on storing OWL 1 or OWL 2 ontologies in object-oriented databases	

object-oriented databases. An overall storage architecture and its detailed illustrations were proposed, and the correctness and quality of the storage approach were proved and analyzed. We achieve applicable object structure and avoid of losing the ontological information. A prototype tool, which could store OWL 2 ontologies in a widely used open source object database db4o, was implemented. Also, a query interface was developed in the prototype tool for querying the stored OWL 2 ontologies. The storage and query examples were provided to show that the approach is feasible and the tool is efficient.

The proposed method in this paper may be useful for some possible applications. Currently, ontologies are increasingly used in many application domains. For example, in [2], a novel ontology-supported case-based reasoning (OS-CBR) approach is proposed and implemented in the mobile-based response system (MERS) to support emergency decision makers to effectively respond to an emergency situation. In [7,29], ontologies are used to represent and manage knowledge in the Semantic Web. After storing ontologies in databases with our method, on one hand, this may to some extent solve the scalability issue raised by the real ontologies, and some mature and efficient database technologies (e.g., query, manipulation, and analysis [10,24,25]) may be useful for handling and managing the ontologies. On the other hand, some database users can access to the ontologies directly by querying the databases even though they do not know the details of the ontologies. In our near future work we will further investigate the possible applications of the proposed method in depth.

As far as future work, we realize that with the development of ontologies in various application domains, some super large scale OWL 2 ontologies may occur, and we will test them in our future work to further evaluate our approach and tool. Also, with the progressing of research works on storage of OWL 2 ontologies, we will create a rather larger real-world dataset to link and evaluate the experimentation with some works in databases and IR (information retrieval) communities. Moreover, we will comprehensively investigate and make further improvements in querying capabilities, and test the querying efficiency. In addition, extending an existing database system with reasoning capabilities for supporting the reasoning of large ontologies stored in databases is an important direction, which may solve the problem of the loss of semantics. Furthermore, we aim at studying the effective integration of storing, querying, and reasoning in depth.

Acknowledgments

The authors thank the anonymous referees for their valuable comments and suggestions, which improved the technical content and the presentation of the paper. The work is supported by the *National Natural Science Foundation of China* (61073139, 61202260, 61370075), the *Fundamental Research Funds for the Central Universities* (N120404005), and in part by the *Program for New Century Excellent Talents in University* (NCET-05-0288).

References

[1] L. Al-Jadir, C. Parent, S. Spaccapietra, Reasoning with large ontologies stored in relational databases: the OntoMinD approach, Data Knowl. Eng. 69 (11) (2010) 1158–1180.

[2] K. Amailef, J. Lu, Ontology-supported case-based reasoning approach for intelligent m-government emergency response services, Decis. Support Syst. 55 (1) (2013) 79–97.

[3] I. Astrova, N. Korda, A. Kalja, Storing OWL ontologies in SQL relational databases, in: Proc. of World Academy of Science Engineering and Technology, 2007, pp. 167–172.

[4] Y. An, A. Borgida, J. Mylopoulos, Refining semantic mappings from relational tables to ontologies, in: Proceedings of 2nd International workshop on Semantic Web and Databases (SWDB 2004), 2004, pp. 84–90.

[5] I. Astrova, Reverse engineering of relational database to ontologies, in: Proc. of the ESWC 2004, 2004, pp. 327–341.

[6] I. Astrova, A. Kalja, Storing OWL ontologies in SQL3 object-relational databases, in: Proc. of 8th WSEAS Int. Conf. on Applied Informatics and Communications (AIC'08), 2008, pp. 99–103.

[7] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific Am. 284 (5) (2001) 34–43.

[8] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, Cambridge, 2003.

[9] B. Chandrasekaran, R. Josephson John, V. Richard Benjamins, What are ontologies* and why do we need them?, IEEE Intell Syst. 14 (1) (1999) 20–26.

[10] R.G.G. Cattell, D.K. Barry, et al., The Object Data Standard: ODMG 3.0, springer, 2000.

[11] J. Chen, Y. Wu, Rules-based object-relational databases ontology construction, J. Syst. Eng. Electron. 20 (1) (2009) 211–215.

[12] C.P de Laborda, S. Conrad, Relational.OWL-A data and schema representation format based on OWL, in: Second Asia-Pacific Conference on Conceptual Modeling (APCCM2005), 2005, pp. 89–96.

[13] A. Gali, C.X. Chen, K.T. Claypool, R. Uceda-Sosa, From ontology to relational databases, in: Proc. of ER Workshops 2004, LNCS 3289, 2004, pp. 278–289.

[14] B.C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, U. Sattler, OWL 2: the next step for OWL, in: Web Semantics: Science, Services and Agents on the World Wide Web 6(4), 2008, pp. 309–322.

[15] R. Ghawi, N. Cullot, Database-to-ontology mapping generation for semantic interoperability, Third International Workshop on Database Interoperability (InterDB), 2007.

[16] I. Horrocks, P.F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: the making of a web ontology language, J. Web Semant. 1 (1) (2003) 7–26.

[17] S. Heymans et al., Ontology reasoning with large data repositories, in: M. Hepp, P. De Leenheer, A. de Moor, Y. Sure (Eds.), Ontology Management, Semantic Web, Semantic Web Services, and Business Applications, Springer, 2008, pp. 89–128.

[18] N. Konstantinou, D. M Spanos, M. Nikolas, Ontology and database mapping: a survey of current implementations and future directions, J. Web Eng. 7 (1) (2008) 1–24.

[19] I. Kwan, J. Fong, Schema integration methodology and its verification by use of information capacity, Inform. Syst. 24 (5) (1999) 355–376.

[20] A. Khalid, A.H. Shah, M.A. Qadir, OntRel: an ontology indexer to store OWL-DL ontologies and its instances, in: Proc. of Int. Conf. of Soft Computing and Pattern Recognition, 2009, pp. 478–483.

[21] L. Lubyte, S. Tessaris, Automatic extraction of ontologies wrapping relational data sources, in: DEXA 2009, 2009, pp. 128–142.

[22] R.J. Miller, Y.E. Ioannidis, R. Ramakrishnan, The use of information capacity in schema integration and translation, in: Proc. of the 19th VLDB Conference, 1993, pp. 120–133.

[23] OWL: Ontology Web Language. <http://www.w3.org/2004/OWL/>.

[24] C. Martinez-Cruz, I. Blanco, M. Vila, Ontologies versus relational databases: are they so different? a comparison, Artif. Intell. Rev. 38 (2012) 271–290.

[25] T.U. Muenchen, D. Maier, Object-Oriented Database Theory, 2001.

[26] Object-oriented database db4o: <http://www.db4o.com/>.

[27] OWL 2 Web Ontology Language Document Overview (Second Edition), <http://www.w3.org/TR/owl2-overview/>, W3C Recommendation 11 December 2012.

[28] Z. Pan, X. Zhang, J. Heflin, DLDB2: a scalable multi-perspective semantic web repository, in: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Sydney, NSW, 2008.

[29] S. Staab, R. Studer (Eds.), Handbook on Ontologies, second ed., Springer, 2009.

[30] D.E. Spanos, P. Stavrou, N. Mitrou, Bringing relational databases into the semantic web: a survey, Semant. Web 3 (2) (2012) 169–209.

[31] O. Udrea, L. Getoor, R.J. Miller, Leveraging data and structure in ontology integration, in: Proc. of the 27th ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 449–460.

[32] N. Villanueva-Rosales, M. Dumontier, Describing chemical functional groups in OWL-DL for the classification of chemical compounds, in: OWL: Experiences and Directions (OWLED 07), Innsbruck, Austria, 2007.

[33] E. Vyšniauskas, L. Nemuraitė, B. Paradauskas, Preserving semantics of Owl 2 ontologies in relational databases using hybrid approach, Inform. Technol. Control 41 (2) (2012) 103–115. ISSN: 1392-124X.

[34] E. Vysniauskas, L. Nemuraite, Mapping of OWL ontology concepts to RDB schemas, in: Information Technologies 2009: Proceedings of the 15th International Conference on Information and Software Technologies, IT 2009, Kaunas Lithuania, 2009, pp. 317–327.

[35] E. Vysniauskas, L. Nemuraite, Transforming ontology representation from OWL to relational database, Inform. Technol. Control 35 (3) (2006) 333–343.

[36] E. Vysniauskas, L. Nemuraite, A. Sukys, A hybrid approach for relating OWL 2 ontologies and relational databases, BIR 2010, LNBIP 64, 2010, pp. 86–101.

[37] F. Zhang, Z.M. Ma, L. Yan, J.W. Cheng, Storing fuzzy ontology in fuzzy relational database, in: Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA), 2011, pp. 447–455.

[38] F. Zhang, Z.M. Ma, L. Yan, Construction of ontologies from object-oriented database models, Integr. Comput.-Aid. Eng. 18 (4) (2011) 327–347.

[39] F. Zhang, Z.M. Ma, Representing and reasoning about XML with ontologies, Appl. Intell. 40 (2) (2014) 74–106.

[40] J. Zhou, L. Ma, Q. Liu, et al., Minerva: a scalable OWL ontology storage and inference system, in: Proc. of the ASWC 2006, LNCS 4185, 2006, pp. 429–443.

[41] S. Zhao, E. Chang, From database to semantic web ontology: an overview, in: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, 2007, pp. 1205–1214.