Automatic extraction of OWL ontologies from UML class diagrams: A semantics-preserving approach

Article *in* World Wide Web · September 2011

DOI: 10.1007/s11280-011-0147-z

CITATIONS
24

READS
1,054

5 authors, including:

Zhuoming Xu
Hohai University
50 PUBLICATIONS **440** CITATIONS
SEE PROFILE

Lili Lin
Hohai University
8 PUBLICATIONS **96** CITATIONS
SEE PROFILE

Qin Yan
Hohai University
50 PUBLICATIONS **411** CITATIONS
SEE PROFILE

World Wide Web (2012) 15:517–545
DOI 10.1007/s11280-011-0147-z

Automatic extraction of OWL ontologies from UML class diagrams: a semantics-preserving approach

Zhuoming Xu · Yuyan Ni · Wenjie He · Lili Lin · Qin Yan

Received: 1 February 2011 / Revised: 29 August 2011
Accepted: 18 October 2011 / Published online: 22 November 2011

Abstract Full implementation of the Semantic Web requires widespread availability of OWL ontologies. Manual ontology development using current OWL editors remains a tedious and cumbersome task that requires significant understanding of the new ontology language and can easily result in a knowledge acquisition bottleneck. On the other hand, abundant domain knowledge has been specified by existing database schemata such as UML class diagrams. Thus developing an automatic tool for extracting OWL ontologies from existing UML class diagrams is helpful to Web ontology development. In this paper we propose an automatic, semantics-preserving approach for extracting OWL ontologies from existing UML class diagrams. This approach establishes a precise conceptual correspondence between UML and OWL through a semantics-preserving schema translation algorithm. The experiments with our implemented prototype tool, UML2OWL, show that the proposed approach is effective and a fully automatic ontology extraction is achievable. The proposed approach and tool will facilitate the development of Web ontologies and the realization of semantic interoperations between existing Web database applications and the Semantic Web.

Keywords Web ontology development · schema translation · semantics preservation · UML class diagram · OWL DL ontology · Semantic Web

Z. Xu (✉) · Y. Ni · W. He · L. Lin · Q. Yan
College of Computer and Information, Hohai University, Nanjing 210098 Jiangsu Province, People's Republic of China
e-mail: zmxu@hhu.edu.cn

Y. Ni
e-mail: yyni_83@hhu.edu.cn

W. He
e-mail: wenjiehe@hhu.edu.cn

L. Lin
e-mail: linlili@hhu.edu.cn

Q. Yan
e-mail: yan_qin@hhu.edu.cn

1 Introduction

The success and proliferation of the Semantic Web [3] depends heavily on quick and cheap construction of Web ontologies [17]. The importance of ontologies to the Semantic Web has prompted the establishment of the normative Web Ontology Language (OWL) [8] and the development of various OWL-aware ontology tools. Although the tools have matured over the years, manual ontology development using current OWL ontology editors, such as Protégé, remains a tedious and cumbersome task that requires significant understanding of the new ontology language and can easily result in a knowledge acquisition bottleneck. So it is necessary to develop approaches and tools that allow reducing the effort and adapting ontologies in an automatic or semi-automatic fashion using existing knowledge sources, being this the goal of ontology learning [18]. TANGO [28] is such a semi-automatic method that can process the format and content of human-created tables and serve to incrementally build a reusable ontology. On the other hand, the OMG's Unified Modeling Language (UML) [22, 23] has become a de facto industry standard for modeling applications in data and software engineering communities. Today, many data sources have been modeled in UML class diagrams, in which abundant domain knowledge has been specified. Therefore, investigating the approaches for extracting OWL ontologies from existing UML class diagrams is valuable to the development of Web ontologies.

Another motivation of our work concerns the development of Semantic Web enabled applications. To build Semantic Web sites [29] and portals [27] or publish "Linked Data" on the Web [4], existing Web data and documents must be "upgraded" to Semantic Web contents which are modeled in the Resource Description Framework (RDF) (http://www.w3.org/RDF/) data model and semantically annotated with OWL ontologies. It is widely believed that the majority of current Web contents are dynamic contents powered by relational databases (RDB) [13]. Thus a critical requirement to realize the Semantic Web vision of "Web of Data" is the inclusion of the vast quantities of data stored in relational databases. The mapping of these vast quantities of data from RDB to RDF has been the focus of a large body of research work in diverse domains [25]. Investigations, such as [25], have shown that the incorporation of domain semantics through the use of application-specific domain ontology in mapping RDB to RDF is an important aspect to fully leverage the expressivity of the RDF model—the standard data model of the Semantic Web. Therefore it is necessary to develop approaches for extracting domain knowledge from database schemata (e.g., UML class diagrams) and mapping the knowledge to OWL ontologies.

Here we concentrate on UML class diagrams for the conceptual perspective. UML class diagrams allow for modeling, in a declarative way, the static structure of an application domain, in terms of concepts and relations between them. Several studies [1, 7, 9, 11] have investigated the mapping between UML class diagrams and Web ontologies. But most existing approaches have focused on extending UML and providing a visual method for modeling Web ontologies. As a result, users have to model UML class diagrams in an unfamiliar way and cannot extract the existing knowledge from legacy UML class diagrams. A few researches [10, 14, 20] have addressed the problem of reusing knowledge in existing UML class diagrams to develop Web ontologies. However, to the best of our knowledge, these approaches tend to extract limited knowledge from the UML model, without placing emphasis on the semantics-preservation (i.e., knowledge-preservation) of UML-to-OWL translation, and often produce target ontologies in a non-normative language or fail to develop an easy-to-use tool.

In this paper we investigate a more formal approach to automatic, semantics-preserving translation from a standard UML 2 class diagram to an OWL DL ontology. Our goal is to

extract domain knowledge as much as possible from the UML class diagram and semantics-preservingly translate the knowledge into the OWL ontology. The main idea of our approach is to establish a precise conceptual correspondence between the UML class diagram and the OWL ontology through a semantics-preserving UML-to-OWL translation algorithm. We also examine the feasibility of our proposed algorithm through the development of a prototype tool called UML2OWL in Java programming language, and test the algorithmic efficiency via case study experiments. The implementary and experimental study shows that our proposed approach is effective and a fully automatic ontology extraction tool is achievable.

A short paper of this work [30] was originally presented at the 2009 International Conference on Database Theory and Application (DTA 2009). The present paper is a substantial extension and enhancement to the previously published conference paper; the significant extensions include: (i) theoretical analysis of the time complexity of UML-to-OWL translation algorithm (Section 3.5), together with additional algorithmic efficiency tests and discussion (Section 5.2); (ii) formal proof for the semantics preservation of our UML-to-OWL translation approach, including the formalization of the semantics of UML class diagram and the semantics of OWL DL ontology (Section 4); (iii) extended related work analysis and discussion (Section 2).

The remainder of the paper is organized as follows. Section 2 reviews related work. After providing an example of UML class diagram to highlight the problems, we explicate our UML-to-OWL translation algorithm and then analyze the time complexity of the algorithm in Section 3. The formal proof for the semantics preservation of the translation is given in Section 4. Section 5 presents our prototype translation tool, UML2OWL, followed by our experimental results. We conclude our work in the last section.

2 Related work

OMG's UML is a general-purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of a software system. The two complementary specifications, Infrastructure [22] and Superstructure [23], constitute a complete specification for the UML 2 modeling language. A UML class diagram is a type of structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes. The syntax of the UML is precisely defined in the specifications, but the semantics of the UML are informally described in terms of natural language (English) and Object Constraint Language (OCL). Berardi et al. [2] thus described the semantics of each construct of a UML class diagram in terms of the First Order Logic (FOL). Earlier research by Calvanese et al. [5] proposed to unify class-based representation formalisms (including semantic data models such as the Entity-Relationship (ER) model) in description logics (DLs). Our work adopts these ideas to specify our formalization of a UML class diagram.

The OWL language is an ontology language for the Semantic Web with formally defined meaning. OWL ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web documents. The most important influences on the design of OWL came, via its predecessors OIL, DAML and DAML+OIL, from description logics, from the frames paradigm, and from RDF [16]. Since becoming W3C recommendation in February 2004 [8, 24], the OWL language has been successfully applied to many application domains. The language was extended with a small but useful set of features [6] and revised by the W3C OWL Working Group. The new version OWL 2 [15] became a W3C

recommendation in October 2009. The original version of OWL ("OWL 1") is a family of three sub-languages (species) of increasing expressive power: OWL Lite, OWL DL, and OWL Full. OWL 2 maintains the OWL Full and OWL DL "duality", but OWL Lite has been replaced by "profiles" (sub-languages) including OWL 2 EL, OWL 2 QL, and OWL 2 RL. Each of the profiles is more restrictive than OWL DL and trades off different aspects of OWL's expressive power in return for different computational and/or implementational benefits [15]. OWL has two normative syntaxes: the abstract syntax [24] and the exchange syntax (i.e., the RDF/XML syntax) [8], but there are various syntaxes available for OWL 2 which serve various purposes. The Direct Model-theoretic Semantics [19, 24] and the RDF-based Semantics [24, 26] provide two alternative ways of assigning meaning to OWL 2 ontologies, with a correspondence theorem providing a link between the two. OWL 2 is fully backwards compatible with OWL, both syntactically (the RDF/XML syntax) and semantically, i.e., every OWL ontology is a valid OWL 2 ontology; every OWL 2 ontology not using new features is a valid OWL ontology [12, 15].

Given the fact that manual ontology development using current OWL editors remains a tedious and cumbersome task, several studies have investigated the approaches for mapping UML class diagrams into Web ontologies. The existing approaches fall into two categories according to their focuses.

The first category focuses on *visual modeling*, i.e., providing a UML-based visual environment for modeling Web ontology. Cranefield [7] is the first one who put forward this idea. In his method, domain experts create an ontology in a UML tool initially and then save it as an XMI-coded file. Finally an XSLT stylesheet translates the XMI-coded file into the corresponding RDF Schema (RDFS) ontology. Baclawski, et al. [1] proposed a method for extending UML to support ontology engineering in the Semantic Web. They summarized the similarities and differences between UML and DAML and proposed an extension of UML to resolve the differences. A mapping from the extended UML model to DAML ontologies was then defined in their work. The Components for Ontology Driven Information Push (CODIP) program (http://codip.projects.semwebcentral.org/) developed a tool called Duet that provides a UML-based environment for ontology development. The latest version of Duet can carry out the conversion between UML and OWL. As the standard UML cannot completely express all OWL concepts, Djurić et al. [9, 11] introduced Ontology Definition Metamodel (ODM) in their research and defined an OWL-based UML profile for integrating software engineering into ontology development. Summing up these studies, Cranefield's solution used RDFS as the language of the target ontology. However, this method has inherent drawbacks because RDFS does not have enough expressive power to capture the knowledge and constraints of a UML class diagram. All the other solutions used UML stereotype to extend the standard UML for bridging the gap between UML and Web ontology languages (DAML or OWL). As a result, the mapping sources will no longer be the standard UML model and each solution has its specific rules to create UML class diagrams – not the common and familiar way in which people model UML class diagrams today. Furthermore, ontology developers cannot extract the domain knowledge from legacy UML class diagrams using these methods and tools.

The second category focuses on *reusing knowledge*, i.e., extracting domain knowledge from existing UML class diagrams to facilitate the development of Web ontologies. Our work belongs to this category. Along this line, the representative work is [10, 14, 20]. Falkovych et al. [10] proposed the transformation from standard UML class diagrams to DAML+OIL ontologies. This method converts UML classes into DAML+OIL classes, attributes of UML class into DAML+OIL datatype properties, UML binary-associations into DAML+OIL object properties (taking no account of UML associations' attributes).

When converting UML associations, they first define four special object properties in DAML+OIL ("Binary", "Unidirectional", "Aggregation", and "Composition") according to the features of binary associations, and then map each UML association to a sub-property of one of these object properties. This solution has not considered *n*-ary associations. Besides, it cannot use the ontology language to naturally model the conceptual entities and relationships which the UML descriptions are attempting to capture. Na, et al. [20] proposed a method for domain ontology building by extracting ontological elements from the UML model designed previously. They compared the UML model elements with the OWL ones and derived transformation rules between the corresponding model elements. However, some important constraints in UML class diagrams, such as the disjointness constraints and covering constraints enforced on a UML generalization set, were ignored in the transformation. Furthermore, a formal algorithm and an easy-to-use tool have not been introduced in their work. In addition, the implementation of UML-to-OWL conversion relies on XSLT transformation which is quite cumbersome to use when one wants to express more complex mappings. Hermida, et al. [14] presented a method for reuse of existing knowledge in the UML model to build domain ontologies represented in OWL DL. The UML-to-OWL transformation includes name transformation and content transformation. The name transformation resolves the naming gap using "URI Generation Rule". The content transformation turns the UML elements into OWL elements according to a set of rules, without consideration of disjointness and covering constraints enforced on the generalization set in the UML model.

Recently, OMG's Ontology Definition Metamodel (ODM) specification [21] provided an informative comparison between UML 2 and OWL Full, looking at the features the two have in common and reviewing the features that are prevalent in one but not the other. It also described the mapping from UML 2 to OWL DL in the MOF Query/Views/Transformations (QVT). However, this specification neither gives the formal proof for semantics preservation of the UML-to-OWL mapping, nor involves tool implementation.

Despite of their weakness, the existing approaches set up a foundation for the proposed work and are of valuable reference to our in-depth research. Compared with the existing solutions, our approach can capture richer knowledge of common UML constructs and constraints, use normative Web Ontology Language (OWL DL) as the target ontology language, and develop a more formal, semantics-preserving algorithm and automatic tool for extracting OWL ontologies from (legacy) UML class diagrams.

3 The UML-to-OWL translation algorithm

This section first provides an example of UML class diagram to highlight the problems more concretely, and then proposes our UML-to-OWL translation algorithm after giving the formalization of UML class diagram and the definition of OWL DL ontology. Finally we analyze the time complexity of the algorithm.

3.1 UML example and problem statement

Existing UML class diagrams are usually created by using current UML modeling tools that support OMG's UML 2 model and XML Metadata Interchange (XMI) 2.0. For example, No Magic, Inc.'s MagicDraw UML (http://www.magicdraw.com/) is such a UML tool that can visually model a UML class diagram and export an XMI 2.0-coded UML class diagram file. Figure 1 shows an example UML class diagram, `Dept`, created with MagicDraw UML

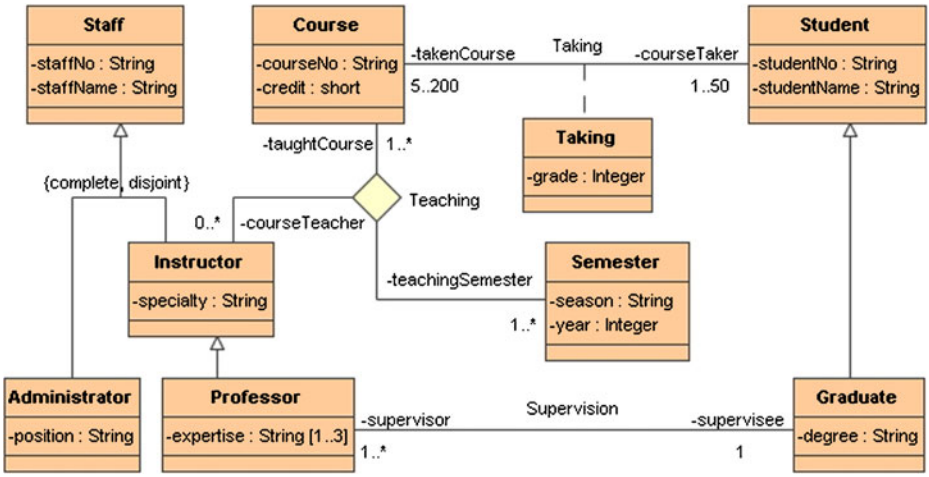

Figure 1 UML class diagram `Dept` modeled with the MagicDraw UML tool, where "*" in the multiplicities denotes "∞".

Community Edition v11.5, which models a university department and specifies the domain knowledge (i.e., ontological knowledge) including:

- concepts such as classes `Instructor`, `Professor`, and `Graduate`; class `Graduate` has attribute `degree` whose data type is `String`;
- relations between concepts such as association `Supervision`(`Professor`, `Graduate`) and generalization `Professor` is a `Instructor`;
- constraints such as two classes `Administrator` and `Instructor` are disjoint.

Given a UML class diagram like `Dept` shown in Figure 1, where the domain knowledge has been specified both in graphic notation and in XMI 2.0 code, how to extract domain knowledge as much as possible from the UML class diagram and semantics-preservingly translate the knowledge into an OWL ontology? This overall problem can be broken down into the following concrete problems to be addressed in the present paper:

- How to formalize a UML class diagram to facilitate the accurate expression of domain knowledge specified in the diagram?
- How to define an OWL ontology with the W3C specifications to capture the domain knowledge extracted from the UML class diagram?
- How to establish a precise conceptual correspondence between UML and OWL and design an automatic translation algorithm from UML class diagram to OWL DL ontology?
- Is the translation algorithm knowledge-preserving (i.e., semantics-preserving)? What is the proof for the semantics preservation?
- Is the translation algorithm implementable in mainstream programming languages such as Java? What is the time-complexity of the algorithm?

3.2 Formalization of UML class diagram

The two models, UML class diagrams and OWL DL ontologies, share a set of core functionalities but despite this overlap, there are many features which can only be expressed in OWL, and others

which can only be expressed in UML [21]. Therefore, in our formalization of UML class diagrams, we restrict our attention to those key aspects that constitute the core of a UML class diagram and are necessary for expressing ontological knowledge. The considered UML constructs and features are class, association or association class among classes, attribute of class or association class, association end (i.e., role), multiplicity, generalization between classes (excluding association classes) and disjointness/covering constraints enforced on generalization [22, 23]. Other UML constructs and features, such as abstract classifier, complex object, and stereotypes, are ignored because essentially the target ontology language (i.e., OWL DL) does not have equivalent or relevant constructs to capture their semantics (i.e., knowledge).

A UML class diagram can be formalized in terms of the First Order Logic (FOL) or description logics (DLs) [2, 5]. We adopt these ideas to specify our formalization of a UML class diagram. In the following, Definition 1 is an accessorial definition; Definition 2 gives the formal syntax and informal semantics of the UML class diagram that includes the considered UML constructs and features as stated above. The formal semantics will be given in Section 4.1.

Definition 1 For two finite sets X and Y, we call a function from a subset of X to Y an *X-labeled tuple over Y*. The labeled tuple Z that maps $x_i \in X$ to $y_i \in Y$, for i=1,2,…,k ($k \geq 1$), is denoted $[x_1{:}y_1,\ldots, x_k{:}y_k]$. We also write $Z[x_i]$ to denote y_i.

Definition 2 A *UML class diagram* is a tuple $U = (L_U, att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U)$, where

- L_U is a finite *alphabet* partitioned into a set C_U of *class* (including *association class*) symbols, a set A_U of *attribute* symbols, a set S_U of *association* (including *association class*) symbols, a set R_U of *role* (i.e., association end) symbols, and a set T_U of *data type* (primitive type) symbols; each $T \in T_U$ has an associated predefined *basic domain* $D(T)$.
- att_U is a function that maps each class symbol in C_U to an A_U-labeled tuple over T_U. The function is used to model attributes of a class.
- ass_U is a function that maps each association symbol in S_U to an R_U-labeled tuple over C_U. We assume without loss of generality that: (i) each role is specific to exactly one association a class participates in; (ii) for each role $R \in R_U$, there is an association $S \in S_U$ and a class $C \in C_U$ such that $ass(S) = [\ldots, R{:}C, \ldots]$. The function associates a collection of roles to each association, determining implicitly also the *arity* of the association.
- $mult_U$ is a function from $C_U \times S_U \times R_U$ or $T_U \times C_U \times A_U$ to $N_0 \times (N_1 \cup \{\infty\})$ (where N_0 denotes non-negative integers, N_1 positive integers). The first component $minmult_U$ of $mult_U$ represents the lower bound and the second, $maxmult_U$, the upper bound. The function is used to specify *multiplicities*, i.e., constraints on the minimum and maximum number of times an instance of a class may participate in an association via some role, or on how many data values of a data type may fill an attribute of a class. If some multiplicity is stated as a single value in the class diagram, we consider both the lower and the upper bounds to be the same (i.e., equal to the value). If some multiplicity is not explicitly stated, then for roles, $minmult_U$ is assumed to be 0 and $maxmult_U$ be ∞; but for attributes, $minmult_U$ and $maxmult_U$ are all assumed to be 1.
- $gene_U \subseteq C_U \times C_U$ is an injective and acyclic binary relation over C_U that models a *generalization* relationship between a subclass (child class) and a superclass (parent class), meaning that each instance of the subclass is also an instance of the superclass. Several generalizations that have a common superclass can be grouped together to form a *generalization set*: $C_1\ gene_U\ C$, $C_2\ gene_U\ C$, …, $C_m\ gene_U\ C$, where $C_1, C_2, \ldots, C_m, C \in C_U, m \geq 1$.

- $dsjt_U \subseteq C_U^1 \times C_U^2 \times \ldots \times C_U^m$, where $C_U^i = C_U, i = 1, 2, \ldots, m, m \geq 2$, is a m-ary relation over C_U that models a *disjointness constraint* enforced on a generalization set, meaning that the collection of subclasses in the generalization set have no instance in common; that is, their intersection is empty.
- $covr_U \subseteq C_U^1 \times C_U^2 \times \ldots \times C_U^m \times C_U$, where $C_U^i = C_U, i = 1, 2, \ldots, m, m \geq 2$, is a ($m$+1)-ary relation over C_U that models a *covering constraint* enforced on a generalization set, meaning that the collection of subclasses are covering for the superclass, i.e., every instance of the superclass is also an instance of at least one of its subclasses.

Note that in UML an association class is both an association and a class. Logically, the association and the association class are the same semantic entity, though they are graphically distinct in the class diagram. The semantics of an association class is a combination of the semantics of an ordinary association and of an ordinary class. Therefore, for an association attached with an association class, the association symbol in S_U and the association class symbol in C_U denote the same symbol (i.e., a single name) and represent the same underlying model element.

According to the above definitions, UML class diagram `Dept` (Figure 1) is a tuple $\text{Dept} = \left(L_{\text{Dept}}, att_{\text{Dept}}, ass_{\text{Dept}}, mult_{\text{Dept}}, gene_{\text{Dept}}, dsjt_{\text{Dept}}, covr_{\text{Dept}}\right)$, where L_{Dept} is a finite alphabet partitioned into a set C_{Dept} of class (including association class) symbols, a set A_{Dept} of attribute symbols, a set S_{Dept} of association (including association class) symbols, a set R_{Dept} of role symbols, and a set T_{Dept} of data type symbols. The set of functions/relations over the alphabet includes $att_{\text{Dept}}()$, $ass_{\text{Dept}}()$, $mult_{\text{Dept}}()$, $gene_{\text{Dept}}()$, $dsjt_{\text{Dept}}()$, and $covr_{\text{Dept}}()$. The complete formal syntax of the class diagram is given in Figure 2.

3.3 Definition of OWL DL ontology

OWL DL supports users who want maximum expressiveness without losing computational completeness and decidability of reasoning systems. As mentioned earlier, it has two types of normative syntactic form: *exchange syntax*, i.e., the RDF/XML syntax that is used to publish and share ontology data over the Web, and the frame-like style *abstract syntax* that is abstracted from the exchange syntax for facilitating access to and evaluation of the ontologies (being this the reason for describing our formal approach using the abstract syntax in the paper). Regardless of using either syntax form, the formal meaning of an OWL DL ontology is solely determined by the same underlying RDF graph of the ontology, which is interpreted by description logics (DLs) based model-theoretic semantics [19, 24] for the language.

Typically, an OWL DL ontology consists of one optional ontology header, which may include a set of ontology properties, plus any number of axioms including *class axioms*, *property axioms*, and *individual axioms* (i.e., facts about individuals). In the following, Definition 3 gives a concise definition of the OWL DL ontology that is suitable for capturing the knowledge extracted from a UML class diagram.

Definition 3 An *OWL DL ontology* is a tuple $O = (ID_O, Axiom_O)$, where

- ID_O is a finite OWL *identifier* set partitioned into the following subsets that are pairwise disjoint:
  - a subset CID_O of *class* identifiers including user-defined classes and two predefined classes `owl:Thing` and `owl:Nothing`,

$\mathsf{Dept} = (L_{\mathsf{Dept}}, att_{\mathsf{Dept}}, ass_{\mathsf{Dept}}, mult_{\mathsf{Dept}}, gene_{\mathsf{Dept}}, dsjt_{\mathsf{Dept}}, covr_{\mathsf{Dept}})$, where alphabet L_{Dept} is partitioned into:

$C_{\mathsf{Dept}} = \{\mathsf{Staff}, \mathsf{Administrator}, \mathsf{Instructor}, \mathsf{Professor}, \mathsf{Course}, \mathsf{Semester}, \mathsf{Student}, \mathsf{Graduate}, \mathsf{Taking}\}$

$A_{\mathsf{Dept}} = \{\mathsf{staffNo}, \mathsf{staffName}, \mathsf{position}, \mathsf{specialty}, \mathsf{expertise}, \mathsf{courseNo}, \mathsf{credit}, \mathsf{season}, \mathsf{year}, \mathsf{studentNo}, \mathsf{studentName}, \mathsf{degree}, \mathsf{grade}\}$

$S_{\mathsf{Dept}} = \{\mathsf{Teaching}, \mathsf{Taking}, \mathsf{Supervision}\}$

$R_{\mathsf{Dept}} = \{\mathsf{courseTeacher}, \mathsf{taughtCourse}, \mathsf{takenCourse}, \mathsf{courseTaker}, \mathsf{supervisor}, \mathsf{supervisee}\}$

$T_{\mathsf{Dept}} = \{\mathsf{String}, \mathsf{Integer}\}$,

and the set of functions/relations over the alphabet consists of:

$att_{\mathsf{Dept}}(\mathsf{Staff}) = [\mathsf{staffNo} : \mathsf{String}, \mathsf{staffName} : \mathsf{String}]$

$att_{\mathsf{Dept}}(\mathsf{Administrator}) = [\mathsf{position} : \mathsf{String}]$

$att_{\mathsf{Dept}}(\mathsf{Instructor}) = [\mathsf{specialty} : \mathsf{String}]$

$att_{\mathsf{Dept}}(\mathsf{Professor}) = [\mathsf{expertise} : \mathsf{String}]$

$att_{\mathsf{Dept}}(\mathsf{Course}) = [\mathsf{courseNo} : \mathsf{String}, \mathsf{credit} : \mathsf{Integer}]$

$att_{\mathsf{Dept}}(\mathsf{Semester}) = [\mathsf{season} : \mathsf{String}, \mathsf{year} : \mathsf{Integer}]$

$att_{\mathsf{Dept}}(\mathsf{Student}) = [\mathsf{studentNo} : \mathsf{String}, \mathsf{studentName} : \mathsf{String}]$

$att_{\mathsf{Dept}}(\mathsf{Graduate}) = [\mathsf{degree} : \mathsf{String}]$

$att_{\mathsf{Dept}}(\mathsf{Taking}) = [\mathsf{grade} : \mathsf{String}]$

$ass_{\mathsf{Dept}}(\mathsf{Teaching}) = [\mathsf{courseTeacher} : \mathsf{Instructor}, \mathsf{taughtCourse} : \mathsf{Course}, \mathsf{teachingSemester} : \mathsf{Semester}]$

$ass_{\mathsf{Dept}}(\mathsf{Taking}) = [\mathsf{takenCourse} : \mathsf{Course}, \mathsf{courseTaker} : \mathsf{Student}]$

$ass_{\mathsf{Dept}}(\mathsf{Supervision}) = [\mathsf{supervisor} : \mathsf{Professor}, \mathsf{supervisee} : \mathsf{Graduate}]$

$mult_{\mathsf{Dept}}(\mathsf{String}, \mathsf{Professor}, \mathsf{expertise}) = (1, 3)$

$mult_{\mathsf{Dept}}(\mathsf{Instructor}, \mathsf{Teaching}, \mathsf{courseTeacher}) = (0, \infty)$

$mult_{\mathsf{Dept}}(\mathsf{Course}, \mathsf{Teaching}, \mathsf{taughtCourse}) = (1, \infty)$

$mult_{\mathsf{Dept}}(\mathsf{Semester}, \mathsf{Teaching}, \mathsf{teachingSemester}) = (1, \infty)$

$mult_{\mathsf{Dept}}(\mathsf{Professor}, \mathsf{Supervision}, \mathsf{supervisor}) = (1, \infty)$

$mult_{\mathsf{Dept}}(\mathsf{Graduate}, \mathsf{Supervision}, \mathsf{supervisee}) = (0, 1)$

$mult_{\mathsf{Dept}}(\mathsf{Course}, \mathsf{Taking}, \mathsf{takenCourse}) = (5, 200)$

$mult_{\mathsf{Dept}}(\mathsf{Student}, \mathsf{Taking}, \mathsf{courseTaker}) = (1, 50)$

$\mathsf{Administrator}\ gene_{\mathsf{Dept}}\ \mathsf{Staff}$ and $\mathsf{Instructor}\ gene_{\mathsf{Dept}}\ \mathsf{Staff}$, which forms a generalization set with constraints $dsjt_{\mathsf{Dept}}(\mathsf{Administrator}, \mathsf{Instructor})$ and $covr_{\mathsf{Dept}}(\mathsf{Administrator}, \mathsf{Instructor}, \mathsf{Staff})$, and $\mathsf{Professor}\ gene_{\mathsf{Dept}}\ \mathsf{Instructor}$ and $\mathsf{Graduate}\ gene_{\mathsf{Dept}}\ \mathsf{Student}$, which impliedly form a generalization set, respectively.

Figure 2 The formal syntax of UML class diagram `Dept`.

  - a subset $DRID_O$ of *data range* identifiers; each data range identifier is a predefined XML Schema datatype reference such as `xsd:integer`,
  - a subset $OPID_O$ of *object property* identifiers; object properties link individuals to individuals, and
  - a subset $DPID_O$ of *datatype property* identifiers; datatype properties link individuals to data values.

- $Axiom_O$ is a finite OWL *axiom* set partitioned into a subset of *class axioms* and a subset of *property axioms*; each axiom is formed by applying OWL constructs (e.g., `Class` and `ObjectProperty`) to the identifiers or descriptions that are the basic building blocks of a class axiom and describe the class either by a class identifier or by specifying the extension of an unnamed anonymous class via the OWL "Property Restriction" construct restrictions and "Boolean combination" construct `unionOf`.

Note that the complete syntax format for the identifiers in Definition 3 is a *Uniform Resource Identifiers (URI) reference* that consists of an absolute URI (or a prefix name) and a *fragment identifier*. Figure 3 summarizes OWL DL constructs used here, including their abstract syntax, description logic (DL) syntax and model-theoretic semantics [16]. The formal semantics of the OWL DL ontology will be further specified in Section 4.2.

3.4 Translation algorithm

The rationale behind the automatic translation from a UML class diagram to an OWL DL ontology is the existence of semantic correspondences between the two models. These correspondences are briefly listed in Table 1. For instance, a UML package corresponds to an OWL ontology; the package name corresponds to the namespace for the ontology.

Note that in UML, the scope of an attribute is limited to the class on which it is defined, thus an attribute name is not globally unique within the diagram. Similarly, a UML association name can be duplicated in a given diagram, with each occurrence having a different semantics. However, in OWL all properties and classes are a first class object, which means that they are defined globally and uniquely identified by URI references. These incompatibilities should be tackled in the translation process by renaming and globalization of identifiers.

Based on the semantic correspondences, the formal approach for extracting an OWL DL ontology from a UML class diagram can be specified using the translation algorithm **U2OTrans** as depicted in Figure 4. The algorithm performs two kinds of translation operations: (i) the translation from UML symbols to OWL identifiers, (ii) the translation from UML elements to OWL axioms by invoking the procedure **AxiomCreation** shown in Figure 5. The algorithm virtually specifies a set of mapping rules, following which a UML class diagram (Definition 2) can be translated into an OWL DL ontology (Definition 3).

In order to enable readers to understand well the algorithm steps, let us use an example to briefly explain the translation process. Given UML class diagram `Dept` (Figure 1) and its formal syntax (Figure 2), the translation algorithm first performs the translation from UML symbols to OWL identifiers. For example, UML class symbol Staff $\in C_{\text{Dept}}$ is translated into OWL class identifier Staff $\in CID_{\text{Dept}}$ (For illustrative purposes,

OWL DL abstract syntax	DL syntax	Model-theoretic semantics
Descriptions (C)		
A (URI reference) owl:Thing owl:Nothing	A $\top$ $\bot$	$A^I \subseteq \Delta^I$ owl:Thing $^I = \Delta^I$ owl:Nothing $^I = \varnothing$
restriction(R allValuesFrom(C))	$\forall R.C$	$(\forall R.C)^I =$ $\{x \mid \forall y. <x,y> \in R^I \rightarrow y \in C^I\}$
restriction(R minCardinality(n))	$\geq n\ R$	$(\geq n\ R)^I =$ $\{x \mid \#\{y. <x,y> \in R^I\} \geq n\}$
restriction(R maxCardinality(n))	$\leq n\ R$	$(\leq n\ R)^I =$ $\{x \mid \#\{y. <x,y> \in R^I\} \leq n\}$
restriction(U allValuesFrom(D))	$\forall U.D$	$(\forall U.D)^I =$ $\{x \mid \forall y. <x,y> \in U^I \rightarrow y \in D^I\}$
restriction(U minCardinality(n))	$\geq n\ U$	$(\geq n\ U)^I =$ $\{x \mid \#\{y. <x,y> \in U^I\} \geq n\}$
restriction(U maxCardinality(n))	$\leq n\ U$	$(\leq n\ U)^I =$ $\{x \mid \#\{y. <x,y> \in U^I\} \leq n\}$
unionOf(C_1 C_2 ...)	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I$
Data Ranges (D)		
D (URI reference)	D	$D^I \subseteq \Delta_D^I$
Object Properties (R)		
R (URI reference)	R ; R^-	$R^I \subseteq \Delta^I \times \Delta^I$; $(R^-)^I = (R^I)^-$
Datatype Properties (U)		
U (URI reference)	U	$U^I \subseteq \Delta^I \times \Delta_D^I$
Class Axioms		
Class(A partial C_1 ... C_n)	$A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$A^I \subseteq C_1^I \cap \ldots \cap C_n^I$
EquivalentClasses(C_1 ... C_n)	$C_1 = \ldots = C_n$	$C_1^I = \ldots = C_n^I$
DisjointClasses(C_1 ... C_n)	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^I \cap C_j^I = \varnothing,\ i \neq j$
Property Axioms		
ObjectProperty(R domain(C_1)...domain(C_n) range(C_1)...range(C_m) [inverseOf(R_1)])	 $\geq 1\ R \sqsubseteq C_i$ $\top \sqsubseteq \forall R.C_j$ $R = (^- R_1)$	 $R^I \subseteq C_i^I \times \Delta^I,\ i = 1,\ldots,n$ $R^I \subseteq \Delta^I \times C_j^I,\ j = 1,\ldots,m$ $R^I = (R_1^I)^-$
DatatypeProperty(U domain(C_1)...domain(C_n) range(D_1)...range(D_m))	 $\geq 1\ U \sqsubseteq C_i$ $\top \sqsubseteq \forall U.D_j$	 $U^I \subseteq C_i^I \times \Delta_D^I,\ i = 1,\ldots,n$ $U^I \subseteq \Delta^I \times D_j^I,\ j = 1,\ldots,m$

Figure 3 The OWL DL abstract syntax, description logic (DL) syntax and model-theoretic semantics of OWL DL constructs.

there is no renaming operation here). Then the algorithm performs the UML elements to OWL axioms translation by invoking **AxiomCreation**. For instance, for UML class Staff

Table 1 Semantic correspondences between UML class diagram and OWL DL ontology.

UML class diagram elements	Corresponding OWL DL ontology elements
a UML package	an OWL ontology
a UML package name	the namespace for the OWL ontology corresponding to the UML package
a UML class	an OWL class (which can be called an *entity class*)
a UML association or association class between two or more classes	an OWL class (which can be called a *relationship class*)
a UML attribute of a class or association class	an OWL datatype property whose domain is the OWL class corresponding to the UML class or association class; whose range is the predifined XML Schema datatype (e.g., xsd:integer) corresponding to the data type of this UML attribute
a UML role associated to a UML association and a UML class	an OWL object property between the two OWL classes that correspond to the UML association and class, respectively
the multiplicity an instance of a UML class may participate in a UML association via a UML role	the (minimum and maximum) cardinality of the OWL object property corresponding to the UML role
a UML generalization set	a set of OWL partial (i.e., subClassOf) class axioms; each axiom corresponding to a generalization relationship
the disjointness and covering constraints enforced on a generalization set	the OWL DisjointClasses and EquivalentClasses axioms, respectively

$\in C_{\text{Dept}}$ such that $att_{\text{Dept}}(\text{Staff}) = [\text{staffNo} : \text{String}, \text{staffName} : \text{String}]$, staffNo, staffName $\in A_{\text{Dept}}$, and String $\in T_{\text{Dept}}$, the procedure **AxiomCreation** creates an OWL class axiom (Step 1.1/axiom (i)):

```
Class(Staff partial restriction(staffNo allValuesFrom(xsd:string) cardinality(1))
restriction(staffName allValuesFrom(xsd:string) cardinality(1)))
```

and creates two property axioms (Step 1.2/axiom (ii)):

```
DatatypeProperty(staffNo domain(Staff) range(xsd:string))
DatatypeProperty(staffName domain(Staff) range(xsd:string)).
```

The resulting OWL DL ontology in the abstract syntax derived from UML class diagram `Dept` is given in the Appendix.

It is worth emphasizing that, although the syntax form of OWL axioms in the algorithm steps is the abstract one, it is possible for a programmer to design an algorithmic implementation that generates an OWL ontology in the exchange syntax (i.e., the RDF/XML serialization) based on the mapping rules. This is because the

Algorithm U2OTrans(U)

Input: A UML class diagram $U = (L_U, att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U)$.

Output: The OWL DL ontology (in the abstract syntax) $O = (ID_O, Axiom_O) = \phi(U)$ that is defined by a translation ϕ such that $ID_O = \phi(L_U)$ and $Axiom_O = \phi(att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U)$.

1. The translation from UML symbols to OWL identifiers $ID_O = \phi(L_U)$:
   1.1. For each class symbol $C \in C_U$, create a class identifier $\phi(C) \in CID_O$;
   1.2. For each association symbol $S \in S_U, S \notin C_U$, create a class identifier $\phi(S) \in CID_O$;
   1.3. For each attribute symbol $A \in A_U$, create a datatype property identifier $\phi(A) \in DPID_O$;
   1.4. For each data type symbol $T \in T_U$, create a data range identifier $\phi(T) \in DRID_O$, i.e., map it to a predefined XML Schema datatype reference;
   1.5. For each role symbol $R \in R_U$, create two identifiers for a pair of mutually-inversed object properties: $\phi(R) \in OPID_O$ and $V = \mathsf{inv\_}\phi(R) \in OPID_O$, where "inv_" is a terminal symbol part of the identifier.
2. The translation from UML elements to OWL axioms $Axiom_O = \phi(att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U)$:
   Invoke **AxiomCreation**($att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U$).

Figure 4 The UML-to-OWL translation algorithm **U2OTrans**.

correspondence between the two syntax forms is straightforward and has been established by W3C's OWL specification [24]. Let us take an example. The following is an instance of axiom (ii):

DatatypeProperty(staffNo domain(Staff) range(xsd:string))

This OWL abstract syntax's axiom corresponds to the following RDF/XML serialization:

```
<owl:DatatypeProperty rdf:ID="staffNo">
    <rdfs:domain rdf:resource="#Staff"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
...
<owl:Class rdf:ID="Staff"> ... </owl:Class>
```

Note that this serialization uses several predefined namespace prefixes (such as `owl:`) or the default namespace of the ontology to define ontological terms. The

Procedure AxiomCreation($att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U$)	Axiom#
Input: UML elements $att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U$.	
Output: OWL axioms $Axiom_O = \phi(att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U)$.	
1. For each class $C \in C_U$ such that $att_U(C) = [A_1 : T_1, \ldots, A_h : T_h]$, $A_1, \ldots, A_h \in A_U$, $T_1, \ldots, T_h \in T_U$, $h \geq 1$,	
1.1. create a class axiom: Class($\phi(C)$ partial restriction($\phi(A_1)$ allValuesFrom($\phi(T_1)$)) cardinality(1) ... restriction($\phi(A_h)$ allValuesFrom($\phi(T_h)$) cardinality(1))).	(i)
1.2. **for** $i = 1..h$ **do** create a property axiom: DatatypeProperty($\phi(A_i)$ domain($\phi(C)$) range($\phi(T_i)$);	(ii)
if $(l \leftarrow minmult_U(T_i, C, A_i);\ l \neq 0)$ **then** replace the cardinality constraint in $\phi(A_i)$'s property restriction clause of axiom (i) with: minCardinality(l);	(iii)
if $(u \leftarrow maxmult_U(T_i, C, A_i);\ u \neq \infty)$ **then** replace the cardinality constraint in $\phi(A_i)$'s property restriction clause of axiom (i) with: maxCardinality(u).	(iv)
2. For each association $S \in S_U$ such that $ass_U(S) = [R_1 : C_1, \ldots, R_n : C_n]$, $R_1, \ldots, R_n \in R_U$, $C_1, \ldots, C_n \in C_U$, $n \geq 2$,	
2.1. create a class axiom: Class($\phi(S)$ partial restriction($\phi(R_1)$ allValuesFrom($\phi(C_1)$) cardinality(1)) ... restriction($\phi(R_n)$ allValuesFrom($\phi(C_n)$) cardinality(1))).	(v)
2.2. **for** $i = 1..n$ **do** create a property axiom and a class axiom: ObjectProperty(V_i domain($\phi(C_i)$) range($\phi(S)$) inverseOf($\phi(R_i)$));	(vi)
Class($\phi(C_i)$ partial restriction(V_i allValuesFrom($\phi(S)$))).	(vii)
3. For each role $R \in R_U$ such that $ass_U(S) = [\ldots, R : C, \ldots]$, $S \in S_U$, $C \in C_U$,	
3.1. create a property axiom: ObjectProperty($\phi(R)$ domain($\phi(S)$) range($\phi(C)$);	(viii)
3.2. **if** $(l \leftarrow minmult_U(C, S, R);\ l \neq 0)$ **then** add a cardinality constraint to V 's property restriction clause of axiom (vii): minCardinality(l);	(ix)
if $(u \leftarrow maxmult_U(C, S, R);\ u \neq \infty)$ **then** add a cardinality constraint to V 's property restriction clause of axiom (vii): maxCardinality(u).	(x)
4. For each generalization set $C_1\ gene_U\ C$, $C_2\ gene_U\ C$, ..., $C_m\ gene_U\ C$, where $C_1, C_2, \ldots, C_m, C \in C_U$, $m \geq 1$,	
4.1. **for** $i = 1..m$ **do** create a class axiom: Class($\phi(C_i)$ partial $\phi(C)$);	(xi)
4.2. **if** $dsjt_U(C_1, \ldots, C_m)$ **then** create a class axiom: DisjointClasses($\phi(C_1) \ldots \phi(C_m)$);	(xii)
if $covr_U(C_1, \ldots, C_m, C)$ **then** create a class axiom: EquivalentClasses($\phi(C)$ unionOf($\phi(C_1) \ldots \phi(C_m)$)).	(xiii)

Figure 5 The UML elements to OWL axioms translation procedure **AxiomCreation**.

prefixes and default namespace are specified in the namespace declaration component of the ontology, like:

```
<!-- The default namespace of the current ontology: -->
xmlns = "http://www.uml2owl.org/university.owl#"
<!-- The base URI for the ontology document: -->
xml:base = " http://www.uml2owl.org/university.owl"
<!-- The namespace declaration for the “rdf:” prefix: -->
xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
<!-- The namespace declaration for the “rdfs:” prefix: -->
xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
<!-- The namespace declaration for the “owl:” prefix: -->
xmlns:owl = "http://www.w3.org/2002/07/owl#"
```

Obviously, utilizing the specified correspondence between the abstract syntax and the exchange syntax [24] as shown in the example, our proposed algorithm is applicable to the generation of OWL ontologies in the RDF/XML syntax so that the ontology can then be directly published and shared over the Web.

3.5 Time-complexity of the algorithm

In this section we analyze the time complexity of algorithm **U2OTrans** by measuring the amount of work done by the algorithm. Here we consider the *translation* operations and ignore the *preprocessing* operations (i.e., diagram parsing and element extraction). That is to say, we exclude the amount of work done by an XML parser (e.g., the DOM API for Java in our implementation) that parses the UML class diagram (i.e., an XMI-coded file) and prepares the element data in computer memory for the usage in the translation procedure of the algorithm. Moreover, since the translation from UML symbols to OWL identifiers (Step 1 of algorithm **U2OTrans**) can be simultaneously made as sub-operations in creating the OWL axioms (Step 2 of algorithm **U2OTrans**), we can ignore the amount of work done in the symbol-to-identifier translation and consider only the creation of axioms (plus judgment of the corresponding conditions) in Step 2 of algorithm **U2OTrans** (i.e., the procedure **AxiomCreation**) to be the basic operation of the algorithm. With these in mind, we now could measure the amount of work done by the algorithm by roughly counting the total number of operations for axiom creation in the procedure **AxiomCreation** (Figure 5).

Let U be the UML class diagram to be translated by the algorithm. We use the total number of main elements (including classes, attributes, associations, roles, and generalizations) in U to measure the diagram size. Thus we have $N = N_C + N_A + N_S + N_R + N_g$, where $N_C = |C_U|$ ($|C_U|$ denotes the *cardinality* of set C_U, i.e. the number of elements in C_U), $N_A = |A_U|$,$N_S = |S_U|$, $N_R = |R_U|$, and N_g is the number of generalizations specified in U. The basic operations are counted as follows:

- the executing times of creating axiom (i) in Step 1.1 is exactly N_C;
- the total executing times of creating axiom (ii) in Step 1.2 is exactly N_A;
- the total executing times of creating property restrictions (iii) and (iv) in Step 1.2 is at most $2N_A$;

- the executing times of creating axiom (v) in Step 2.1 is exactly N_S;
- the executing times of creating axiom (vi) and that of creating axiom (vii) in Step 2.2 are all exactly N_R, so do the creation of axiom (viii) in Step 3.1;
- the total executing times of creating property restrictions (ix) and (x) in Step 3.2 is at most $2N_R$;
- the executing times of creating axiom (xi) in Step 4.1 is exactly N_g;
- the executing times of creating axiom (xii) and that of creating axiom (xiii) in Step 4.2 are all at most $N_g/2$ because in an extreme case, every generalization set contains exactly one superclass plus two subclasses and has covering and disjointness constraints enforced on it; obviously the number of such generalization sets is at most $N_g/2$.

Summing up the above basic operations, the amount of work done by the algorithm is: $W(N) = N_C + 3N_A + N_S + 5N_R + 2N_g < 5(N_C + N_A + N_S + N_R + N_g) = 5N$. That is, the worst-case time complexity of the translation algorithm is $O(N)$.

4 Semantics-preservation of translation

This section gives the formal proof for the semantics preservation of our UML-to-OWL translation. Before giving the proof, we first introduce the semantics of the UML class diagram (Definition 2) and the semantics of the OWL DL ontology (Definition 3).

4.1 Semantics of UML class diagram

The semantics of a database conceptual schema (such as an ER schema and a UML class diagram) can be given by specifying which *database states* [5] are consistent with the information structure represented by the schema, see Definition 4. A database state is considered acceptable if it satisfies all integrity constraints that are part of the schema. This is captured by the notion of *legal database state* [5], see Definition 5.

Definition 4 A *database state* $B=(\Lambda^B,\Theta^B)$ corresponding to a UML class diagram is constituted by a nonempty finite set Λ^B that is assumed to be disjoint from all basic domains, and a mapping function Θ^B that maps:

- every data type symbol $T \in T_U$ to the corresponding basic domain $D(T)$,
- every class $C \in C_U$ to a subset C^B of Λ^B,
- every attribute $A \in A_U$ to a set $A^B \subseteq \Lambda^B \times \cup_{T \in T_U} B(T)$, and
- every association $S \in S_U$ to a set S^B of R_U-labeled tuples over Λ^B.

The elements of C^B, A^B, S^B are called *instances* of C, A, S, respectively.

Definition 5 Let B be a database state for a UML class diagram $U = (L_U, att_U, ass_U, mult_U, gene_U, dsjt_U, covr_U)$. B is said to be a *legal database state* for U, if it satisfies the following conditions:

(1) For each class $C \in C_U$ such that $att_U(C) = [A_1 : T_1, \ldots, A_h : T_h], h \geq 1$, the following holds: for $\forall c \in C^B$ and for each $i=1,\ldots,h$, if multiplicity $mult_U(T_i, C, A_i)$ is not explicitly stated, then there is exactly one element $a_i = < c, t_i > \in A_i^B$ whose first component is c, and the second component is $t_i \in D(T_i)$.

(2) For each class $C \in C_U$, attribute $A \in A_U$ and its data type $T \in D(T)$ such that $att_U(C) = [\ldots, A : T, \ldots]$, if multiplicity $mult_U(T, C, A)$ is explicitly stated, then it holds that $\forall c \in C^B$, $minmult_U(T, C, A) \leq \#\{t \in D(T) | < c, t > \in A\} \leq maxmult_U(T, C, A)$.

(3) For each association $S \in S_U$ such that $ass_U(S) = [R_1 : C_1, \ldots, R_n : C_n], n \geq 2$, all instances of S are of the form $[R_1{:}c_1, \ldots, R_n{:}c_n]$, where $c_i \in C_i^B$, $i{=}1,\ldots,n$.

(4) For each role $R \in R_U$ associating a class $C \in C_U$ to an association $S \in S_U$ such that $ass_U(S) = [\ldots, R{:}C, \ldots]$, it holds that $\forall c \in C^B$, $minmult_U(C, S, R) \leq \#\{s \in S^B | s[R] = c\} \leq maxmult_U(C, S, R)$.

(5) For each pair of classes $C_1, C \in C_U$ such that $C_1\ gene_U\ C$, it holds that ${C_1}^B \subseteq C^B$.

(6) For some classes $C_1,\ldots,C_m \in C_U$ such that $dsjt_U(C_1, \ldots, C_m), m \geq 2$, it holds that $C_1^B \cap \ldots \cap C_m^B = \emptyset$.

(7) For some classes $C_1,\ldots,C_m$, $C \in C_U$ such that $covr_U(C_1, \ldots, C_m, C), m \geq 2$, it holds that $C^B = C_1^B \cup \ldots \cup C_m^B$.

Note that the definition of database state reflects the usual assumption in databases that database states are finite structures [5]. Although the basic domains are not required to be finite, for each legal database state for a UML class diagram only a finite set of values from the basic domains are actually of interest in real-life applications. Thus we can define the *active domain* ${\Delta_{act}}^B$ of a database state B as the finite set of values. More formally, $\Delta_{act}^B = \left\{t \in D(T) | T \in T_U \wedge \exists A \in A_U, c \in \Lambda^B. < c, t > \in A^B\right\}$. Accordingly, it could be assumed without loss of generality that each basic domain is finite, and $D(T) \subseteq \Delta_{act}^B, T \in T_U$.

Now let us explain some conditions of the legal database state for a UML class diagram using several elements of class diagram `Dept` shown in Figures 1 and 2. With regard to condition (3), for association Supervision $\in S_{\text{Dept}}$ such that $ass_{\text{Dept}}(\text{Supervision}) = [\text{supervisor} : \text{Professor}, \text{supervisee} : \text{Graduate}]$, all instances of Supervison are of the form $[\text{supervisor} : c_1, \text{supervisee} : c_2]$, where $c_1 \in \text{Professor}^B$, $c_2 \in \text{Graduate}^B$. With regard to condition (5), for pair of classes Administrator, Staff $\in C_{\text{Dept}}$ such that Administrator $gene_{\text{Dept}}$ Staff, it holds $\text{Administrator}^B \subseteq \text{Staff}^B$; for pair of classes Instructor, Staff $\in C_{\text{Dept}}$ such that Instructor $gene_{\text{Dept}}$ Staff, it holds $\text{Instructor}^B \subseteq \text{Staff}^B$. As for condition (6), for classes Administrator, Instructor $\in C_{\text{Dept}}$ such that $dsjt_{\text{Dept}}$ (Administrator, Instructor), it holds $\text{Administrator}^B \cap \text{Instructor}^B = \emptyset$.

4.2 Semantics of OWL DL ontology

OWL DL uses a description logic style model theory to formalize the meaning of the language [12, 16, 19, 24]. The semantics for the OWL DL language is based on *interpretations*, see Definition 6 as follows.

Definition 6 An *interpretation* $I = \left(\Delta^I, \Delta_D^I, \bullet^I\right)$ of an OWL DL ontology O consists of a domain of discourse which is divided into two disjoint sets, individual domain Δ^I and data-value domain ${\Delta_D}^I$, and an interpretation function $\bullet^I$ which maps:

- every class $C \in CID_O$ to a set $C^I \subseteq \Delta^I$,
- every data range $D \in DRID_O$ to a set $D^I \subseteq \Delta_D^I$,
- every object property $R \in OPID_O$ to a set $R^I \subseteq \Delta^I \times \Delta^I$, and
- every datatype property $U \in DPID_O$ to a set $U^I \subseteq \Delta^I \times \Delta_D^I$.

Besides, the interpretation function $\bullet^I$ can be extended to descriptions and axioms, the corresponding semantics conditions are described in Figure 3.

An OWL DL ontology O is satisfied by an interpretation I just when all of the semantic conditions resulting from the axioms in O are satisfied by I. If such interpretation I exists, it is called as a *model* of O.

Furthermore, it is implicit in the semantics of a UML class diagram that there cannot be two labeled tuples connected through all roles of the association to exactly the same elements of the domain.

In addition, according to condition (3) of the legal database state for a UML class diagram (see Definition 5), the semantics of a UML class diagram U implies such constraints that for every $s_1, s_2 \in S^B$, where $S \in S_U$, $ass_U(S) = [R_1 : C_1, \ldots, R_k : C_k]$, it always holds $(\underset{1\le i\le k}{\wedge} \forall c \in C^B.(s_1[R_i] = c \leftrightarrow s_2[R_i] = c)) \rightarrow s_1 = s_2$. The model of OWL DL ontology $\phi(U)$, on the other hand, do not have the corresponding constraints. Therefore we need to further impose such constraints on the model of the ontology. This is captured by the notion of *relation-descriptive model*, see Definition 7 as follows.

Definition 7 Let U be a UML class diagram and $\phi(U)$ be an OWL DL ontology translated from U (with algorithm **U2OTrans**). A model I of $\phi(U)$ is called *relation-descriptive*, if for every association $S \in S_U$ with $ass_U(S) = [R_1 : C_1, \ldots, R_n : C_n], n \ge 2$, for every $s_1, s_2 \in (\phi(S))^I$, we have $(\underset{1\le i\le n}{\wedge} \forall f \in \Delta^I.(< s_1, f > \in (\phi(R_i))^I \leftrightarrow < s_2, f > \in (\phi(R_i))^I)) \rightarrow s_1 = s_2$.

4.3 Semantics-preservation proof

We give the formal proof for the semantics preservation of the translation algorithm **U2OTrans** (in Section 3.4) as follows.

Theorem 1. The translation from UML class diagram U to OWL DL ontology $\phi(U)$ in the algorithm **U2OTrans** is semantics-preserving, i.e.,

(1). For each legal database state B for U, there exists a mapping $\alpha : B \rightarrow I$ so that $I = \alpha(B)$ is a model of $\phi(U)$.
(2). For each relation-descriptive model I of $\phi(U)$, there exists a mapping $\beta : I \rightarrow B$ so that $B = \beta(I)$ is a legal database state for U.

Proof (1). Given a database state B for U, we define an interpretation $I = \alpha(B)$ of $\phi(U)$ as follows:

- individual domain $\Delta^I = \Lambda^B \cup \bigcup_{S \in S_U} S^B$.
- data-value domain $\Delta^I_D = \bigcup_{T \in T_U} \Delta^B_{act}$.
- for each symbol $X = C_U \cup A_U \cup S_U$,

$$(\phi(X))^I = X^B. \tag{1}$$

- for each data type symbol $T \in T_U$,

$$(\phi(T))^I = \Delta^B_{act} = \{t \in D(T) | T \in T_U \wedge \exists A \in A_U, c \in \Lambda^B. < c, t > \in A^B\}. \tag{2}$$

- for each association $S \in S_U$ such that $ass_U(S) = [R_1 : C_1, \ldots, R_n : C_n], n \geq 2$,

$$(\phi(R_i))^I = \left\{ < s, c_i > \in \Delta^I \times \Delta^I | s \in S^B \wedge c_i \in C_i^B \wedge s[R_i] = c_i \right\}, i = 1, \ldots, n. \quad (3)$$

Let B be a legal database state. We now prove that I is a model of $\phi(U)$, i.e., I satisfies every axiom in $\phi(U)$ (i.e., axiom (i)~(xiii) in the procedure **AxiomCreation** of the algorithm **U2OTrans**. Also note that B is a finite structure, so I is also finite.)

(a). Let $C \in C_U$ be a UML class with $att_U(C) = [A_1 : T_1, \ldots, A_h : T_h], h \geq 1$. By condition (1) in the definition of legal database state (Definition 5), and by formula (1) and (2), for $\forall c \in C^B = (\phi(C))^I$ and for each i=1, …, h, if multiplicity $mult_U(T_i, C, A_i)$ is not explicitly stated, then there is exactly one element $< c, t_i > \in A_i^I$ whose second component is $t_i \in D(T_i) = (\phi(T_i))^I$. It follows that $(\phi(C))^I \subseteq \bigcap_{i=1}^{h} \{c | \forall t_i. < c, t_i > \in (\phi(A_i))^I \rightarrow t_i \in (\phi(T_i))^I \wedge \#\{t_i | < c, t_i > \in (\phi(A_i))^I\} = 1\}$. So I satisfies axiom (i). By the definition of database state (Definition 4), $A_i^B \subseteq \Delta^B \times \cup_{T_i \in T_U} D(T_i)$, i=1, …,h. Furthermore, by condition (1) in Definition 5, for $\forall c \in C^B$, there exists $< c, t_i > \in A_i^B$ with $t_i \in D(T_i), i = 1, \ldots, h$. Thus $A_i^B \subseteq C^B \times D(T_i), i = 1, \ldots, h$. By formula (1) and (2), $(\phi(A_i))^I \subseteq (\phi(C))^I \times (\phi(T_i))^I, i = 1, \ldots, h$. So I satisfies axiom (ii). Let $l = minmult_U(T_i, C, A_i) \neq 0, i = 1, \ldots, h$. Consider $\forall c \in C^B = (\phi(C))^I$. By condition (2) in Definition 5, we have $\#\{t_i \in D(T_i) | < c, t_i > \in A_i\} \geq minmult_U(T_i, C, A_i) = l$, thus $(\phi(C))^I \subseteq \{c | \#\{t_i \in D(T_i) | < c, t_i > \in (\phi(A_i))^I\} \geq l\}, i = 1, \ldots, h$ by formula (2). So I satisfies axiom (iii). So does axiom (iv).

(b). Let $R \in R_U$ be a role such that $ass_U(S) = [\ldots, R : C, \ldots]$. By condition (3) in Definition 5, for $\forall s \in S^B, s = [\ldots, R : c, \ldots]$ where $c \in C^B$. By formula (3), $(\phi(R))^I \subseteq \left\{ < s, c > \in \Delta^I \times \Delta^I | s \in S^B \wedge c \in C^B \wedge s[R] = c \right\}$, i.e., $(\phi(R))^I \subseteq S^B \times C^B$. Then by formula (1), $(\phi(R))^I \subseteq (\phi(S))^I \times (\phi(C))^I$. So I satisfies axiom (viii).

(c). Let $S \in S_U$ be an association such that $ass_U(S) = [R_1 : C_1, \ldots, R_n : C_n], n \geq 2$. By condition (3) in Definition 5 and formula (1) and (2), for $\forall s \in S^B = (\phi(S))^I$, we have $s = [R_1 : c_1, \ldots, R_n : c_n], c_i \in C_i^B = (\phi(C_i))^I, i = 1, \ldots, n$. So s is a function defined on [R_1, …, R_n]. By formula (3), there is exactly one $c_i = s[R_i] \in C_i^B = (\phi(C_i))^I$ which holds $< s, c_i > \in (\phi(R_i))^I$. It follows that

$$\begin{aligned} (\phi(S))^I \subseteq \bigcap_{i=1}^{n} \{s | \forall c_i. < s, c_i > \in (\phi(R_i))^I \rightarrow c_i \in (\phi(C_i))^I \\ \wedge \#\{c_i | < s, c_i > \in (\phi(R_i))^I\} = 1\} \end{aligned} \quad (4)$$

Thus I satisfies axiom (v). For i=1,…,n, since V_i is the inverse of $\phi(R_i)$, we have $V_i^I = \left((\phi(R_i))^I\right)^-$. According to the proof in (b), we have $(\phi(R_i))^I \subseteq (\phi(S))^I \times (\phi(C_i))^I$, thus have $V_i^I = \left((\phi(R_i))^I\right)^- \subseteq (\phi(C_i))^I \times (\phi(S))^I$. So I satisfies axiom (vi). Consider $\forall s \in (\phi(S))^I$. By formula (4), $\exists c_i \in (\phi(C_i))^I$ such that $< s, c_i > \in (\phi(R_i))^I$, i.e., $< c_i, s > \in V_i^I, i = 1, \ldots, n$. So $(\phi(C_i))^I \subseteq \{c_i | \forall s. < c_i, s > \in V_i^I \rightarrow s \in (\phi(S))^I\}$. It follows that I satisfies axiom (vii).

(d). Let $R \in R_U$ be a role with $ass_U(S) = [\ldots, R : C, \ldots]$ and $l = minmult_U(C, S, R) \neq 0$. Consider $\forall c \in C^B = (\phi(C))^I$. By condition (4) in Definition 5, we have $\#\{s \in S^B | c[R] = s\} \geq minmult_U(C, S, R) = l$, thus $(\phi(C))^I \subseteq \{c | \#$

$\{s \in S^B | s[R] = c\} \geq l\}$. By formula (3), $(\phi(C))^I \subseteq \{c | \#\{s \in S^B | < s, c > \in (\phi(R))^I\} \geq l\}$. It follows that $(\phi(C))^I \subseteq \{c | \#\{s \in S^B | < c, s > \in V^I\} \geq l\}$ because I satisfies axiom (vii). Therefore I satisfies axiom (ix). So does axiom (x).

(e). Let $C_1, C \in C_U$ be two classes such that $C_1\ gene_U\ C$. By condition (5) in Definition 5, $C_1^B \subseteq C^B$, and by formula (1), $(\phi(C_1))^I \subseteq (\phi(C))^I$. So I satisfies axiom (xi). Let $C_1, \ldots, C_m \in C_U, m \geq 2$ be classes such that $dsjt_U(C_1, \ldots, C_m)$. By condition (6) in Definition 5, $C_1^B \cap \ldots \cap C_m^B = \emptyset$, thus $(\phi(C_1))^I \cap \ldots \cap (\phi(C_m))^I = \emptyset$ by formula (1). So I satisfies axiom (xii). Let $C_1, \ldots, C_m, C \in C_U, m \geq 2$ be classes such that $covr_U(C_1, \ldots, C_m, C)$. By condition (7) in Definition 5, $C^B = C_1^B \cup \ldots \cup C_m^B$, thus $(\phi(C))^I = (\phi(C_1))^I \cup \ldots \cup (\phi(C_m))^I$ by formula (1). So I satisfies axiom (xiii).

(2). Given an interpretation $I = \left(\Delta^I, \Delta_D^I, \cdot^I\right)$ of $\phi(U)$, we define a database state $B = \beta(I)$ for U as follows:

- $\Lambda^B = \Delta^I - \cup_{S \in S_U} (\phi(S))^I$;
- For every symbol $X \in C_U \cup A_U$,

$$X^B = (\phi(X))^I; \tag{5}$$

- For every data type symbol $T \in T_U$, let

$$D(T) = (\phi(T))^I; \tag{6}$$

- For every association $S \in S_U$ such that $ass_U(S) = [R_1 : C_1, \ldots, R_n : C_n], n \geq 2$,

$$\begin{aligned} S^B = \{[R_1 : c_1,\ \ldots,\ R_n : c_n] | c_1, \ldots, c_n \in \Delta^I \\ \wedge \left(\bigwedge_{i=1}^{n} (\exists s \in (\phi(S))^I . < s, c_i > \in (\phi(R_i))^I)\right)\} \end{aligned} \tag{7}$$

Let I be a relation-descriptive model. We now prove that B is a legal database state for U, i.e., B satisfies all conditions of legal database state for U (i.e., conditions (1)~(7) in Definition 5).

(a). Since I satisfies axiom (i) in the algorithm, for every class $C \in C_U$ such that $att_U(C) = [A_1 : T_1, \ldots, A_h : T_h]$, $h \geq 1$, we have that $(\phi(C))^I \subseteq \bigcap_{i=1,\ldots,h}^{n} \{c | \forall t_i . < c, t_i > \in (\phi(A_i))^I \rightarrow t_i \in (\phi(T_i))^I \wedge \#\{t_i | < c, t_i > \in (\phi(A_i))^I\} = 1\}$. By formula (5) and (6), for $\forall c \in (\phi(C))^I = C^B$, there exists exactly one $< c, t_i > \in (\phi(A_i))^I = A_i^B, t_i \in (\phi(T_i))^I = D(T_i), i = 1, \ldots, h$. So B satisfies condition (1).

(b). Since I satisfies axiom (iii), for every attribute $A \in A_U$ with $att_U(C) = [\ldots, A : T, \ldots]$, we have that $\forall c \in (\phi(C))^I$, $(\phi(C))^I \subseteq \{c | \#\{t \in D(T) | < c, t > \in (\phi(A))^I\} \geq l\}$. By formula (5), we have that $\forall c \in C^B$, $\#\{t \in D(T) | < c, t > \in A\} \geq minmult_U(T, C, A)$. So does $\#\{t \in D(T) | < c, t > \in A\} \leq maxmult_U(T, C, A)$. Thus B satisfies condition (2).

(c). Since I satisfies axiom (v), for every association $S \in S_U$ such that $ass_U(S) = [R_1 : C_1, \ldots, R_n : C_n]$, $n \geq 2$, we have that $(\phi(S))^I \subseteq \bigcap_{i=1,\ldots,n}^{n} \{s | \forall c_i . < s, c_i > \in (\phi(R_i))^I \rightarrow c_i \in (\phi(C_i))^I \wedge \#\{c_i | < s, c_i > \in (\phi(R_i))^I\} = 1\}$, $i = 1, \ldots, n$. By formula (7), it always holds that for $\forall s \in S^B$ we have $s = [R_1 : c_1, \ldots, R_n : c_n]$, $c_i \in C_i^B, i = 1, \ldots, n$. So B satisfies condition (3).

(d). Since I satisfies axiom (ix), for role $R \in R_U$ such that $ass_U(S) = [\ldots, R : C, \ldots]$, it holds that $\forall c \in (\phi(C))^I$, $(\phi(C))^I \subseteq \{c \mid \#\{s \in S^B \mid < c, s > \in V^I\} \geq l\}$, $l = minmult_U(C, S, R)$. Moreover, since I satisfies axiom (vi), we have $V^I = ((\phi(R))^I)^-$, thus have $(\phi(C))^I \subseteq \{c \mid \#\{s \in S^B \mid < s, c > \in (\phi(R))^I\} \geq l\}$. By

formula (5), it holds that $\forall c \in C^B$, $\#\{s \in S^B | s[R] = c\} \geq minmult_U\ (C, S, R)$. Similarly, it holds $\#\{s \in S^B | s[R] = c\} \leq maxmult_U\ (C, S, R)$. So B satisfies condition (4).

(e). Since I satisfies axiom (xi), for every pair of classes C_1, $C \in C_U$ with $C_1\ gene_U\ C$, we have $(\phi(C_1))^I \subseteq (\phi(C))^I$. By formula (5), $C_1^B \subseteq C^B$. So B satisfies condition (5).

(f). Since I satisfies axiom (xii), for some classes $C_1, \ldots, C_m \in C_U$, $m \geq 2$ such that $dsjt_U(C_1, \ldots, C_m)$, we have $(\phi(C_1))^I \cap \ldots \cap (\phi(C_m))^I = \emptyset$. By formula (5), $C_1^B \cap \ldots \cap C_m^B = \emptyset$. So B satisfies condition (6).

(g). Since I satisfies axiom (xiii), for some classes $C_1, \ldots, C_m, C \in C_U$, $m \geq 2$ such that $covr_U(C_1, \ldots, C_m, C)$, we have $(\phi(C))^I = (\phi(C_1))^I \cup \ldots \cup (\phi(C_m))^I$. By formula (5), $C^B = C_1^B \cup \ldots \cup C_m^B$. So B satisfies condition (7).

For the previous example, according to the proven **Theorem 1**, the translation from UML class diagram `Dept` (Figures 1 and 2) to OWL DL ontology ϕ(Dept) (see the Appendix) in the algorithm **U2OTrans** is semantics-preserving because: (i) for each legal database state B for `Dept`, there exists a mapping $\alpha : B \rightarrow I$ so that I = $\alpha(B)$ is a model of ϕ(Dept); (ii) for each relation-descriptive model I of ϕ(Dept), there exists a mapping $\beta : I \rightarrow B$ so that $B = \beta(I)$ is a legal database state for `Dept`.

5 Tool implementation and experiments

As a proof-of-concept for our proposed approach (i.e., the algorithm **U2OTrans**), we have developed a prototype tool called UML2OWL based on J2SE 1.5.0 platform and conducted algorithmic efficiency tests and case studies with the tool.

5.1 The UML2OWL tool

The UML2OWL tool can take an XML Metadata Interchange (XMI 2.0) coded UML class diagram file exported from a UML 2 modeling tool as *input* and produce the corresponding

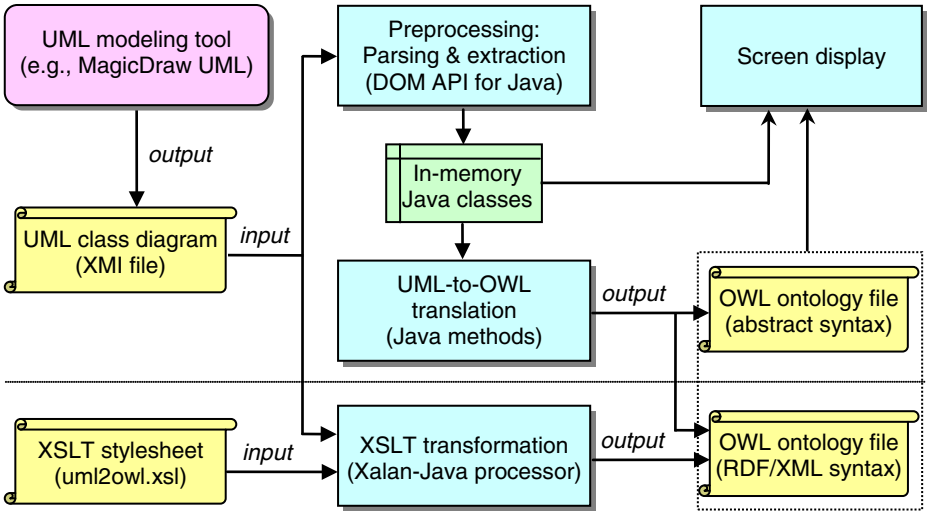

Figure 6 Architecture and process flow of UML2OWL tool.

OWL DL ontologies in both the abstract syntax and the RDF/XML syntax as *output*. The architecture and process flow of UML2OWL is illustrated in Figure 6.

In our implementation, the XMI file of the UML class diagram is parsed by the DOM API for Java to construct an in-memory document representation (tree structure). Java methods are used to extract UML model data from the tree structure; the extracted data are stored as in-memory data in Java classes listed in Figure 7 and simultaneously displayed on the tool screen. Then the in-memory data are used by Java methods to perform the translation from the UML class diagram to the OWL DL ontology. The resulting ontology is saved as text files and displayed simultaneously on the tool screen.

In order to draw a comparison of algorithmic efficiency between two kinds of implementation mode for UML-to-OWL translation, the *Java coding* mode as in our implementation and the *XSL transformation* mode as in most existing solutions, we implemented an add-in module in our tool to perform XSL transformation based UML-to-OWL translation. As depicted in

```
abstract class ElementModel{   // UML element abstract model
    String id = null;              // UML element identifiers
    String name = null;            // UML element names
}
class ClassModel extends ElementModel{   // UML classes
    Vector attributeIdVector = null;          // UML attributes of the class
    Vector roleIdVector = null;               // UML roles associated to the class
    Vector superclassIdVector = null;         // UML superclasses of the class
}
class AttributeModel extends ElementModel{   // UML attributes
    String classId = null;            // UML (association) class owning the attribute
    String dataTypeId = null;         // data type of the attribute
    String maxMultiplicity = null;    // maximum multiplicity of the attribute
    String minMultiplicity = null;    // minimum multiplicity of the attribute
}
class AssociationModel extends ElementModel{   // UML associations
    Vector roleIdVector = null;   // UML roles associated to the association
}
class AssociationClassModel extends AssociationModel{  // UML association classes
    Vector attributeIdVector = null;                  // attributes of the association class
}
class RoleModel extends ElementModel{   // UML roles
    String classId = null;                // UML class associating the role
    String associationId = null;          // UML association (class) associating the role
    String maxMultiplicity = null;        // maximum multiplicity of the role
    String minMultiplicity = null;        // minimum multiplicity of the role
}
class GeneralizationSetModel extends ElementModel{   // UML generalization sets
    String superclassId = null;             // UML (common) superclass
    Vector subclassIdVector = null;       // UML subclasses
    boolean disjointness = false;  // disjointness constraint enforced on the generalization set
    boolean covering = false;      // covering constraint enforced on the generalization set
}
```

Figure 7 Java classes for storing UML class diagram element data.

Table 2 The sizes of ten UML class diagrams.

UML class diagram	Diagram size	Numbers of main elements				
		Class	Association	Attribute	Role	Generalization
#1	50	9	5	26	7	3
#2	100	18	9	50	15	8
#3	150	23	15	79	24	9
#4	200	29	17	113	28	13
#5	250	34	24	140	39	13
#6	300	40	30	167	48	15
#7	350	46	35	197	55	17
#8	400	51	40	228	62	19
#9	450	57	43	261	68	21
#10	500	62	51	287	78	22

Figure 6 (the part under the dashed line), the translation process is described by a self-developed XSL Transformations (XSLT) stylesheet and executed by calling the Xalan-Java v2.7.0 (http://xml.apache.org/xalan-j/), an open-source XSLT processor for Java.

5.2 Algorithmic efficiency tests

We carried out UML-to-OWL translation experiments with our UML2OWL tool on a PC (CPU P4/3.0GHz, RAM 512MB). Ten UML class diagrams created with UML 2 tools were tested in the experiments, as listed in Table 2. All resulting OWL DL ontologies have passed the syntactic validation by the WonderWeb OWL Ontology Validator (http://phoebus.cs.man.ac.uk:9999/OWL/Validator).

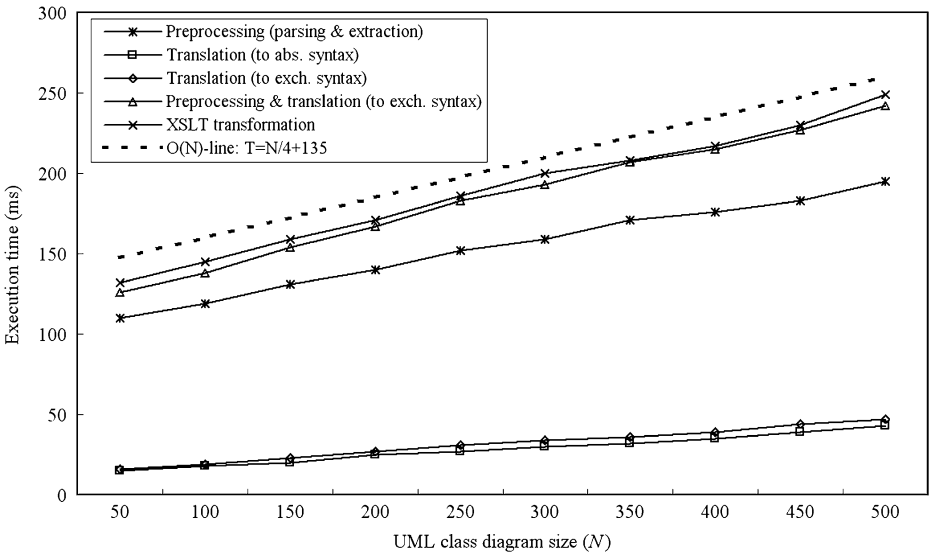

Figure 8 The actual execution time of algorithmic routines running the ten UML class diagrams.

Figure 8 is a line chart which shows the actual execution time of the algorithmic routines in the UML2OWL tool running the ten UML class diagrams. The experimental results indicate that: (i) the time complexity of our translation algorithm (**U2OTrans**) is $O(N)$ according to the contrastive $O(N)$-line $T = N/4 + 135$ (the dashed line) in the figure, which validates the analytical results in Section 3.5; (ii) the actual execution time was mainly spent on the preprocessing process (parsing and element extraction of the diagrams), that is, the translation process in our algorithm is efficient; (iii) Java coding implementation compares favourably with XSL transformation implementation in terms of algorithmic efficiency.

5.3 Case study

We have conducted several case studies using our UML2OWL tool and several popular UML modeling tools (such as No Magic, Inc.'s MagicDraw UML, Gentleware's Poseidon for UML, and Sybase's PowerDesigner). To save space, we just give a small-scale case here. Figure 1 given in Section 3.1 is an example UML class diagram, `Dept`, created with No Magic, Inc.'s MagicDraw UML Community Edition v11.5. As mentioned earlier, the formal syntax of the class diagram can be found in Figure 2.

Figures 9 and 10 are the screenshots of UML2OWL tool running the `Dept` case. In the two figures, the left tree is a visualization of the parsed UML class diagram, showing a UML class/association hierarchy with attributes/roles, where C denotes a class, S an association, (S) an association class, A an attribute with its type, and R a role with the association it participates in; the right text-areas display the resulting OWL DL ontology in the abstract syntax and the RDF/XML syntax, respectively. As mentioned earlier, the resulting OWL DL ontology in the abstract syntax is given in the Appendix.

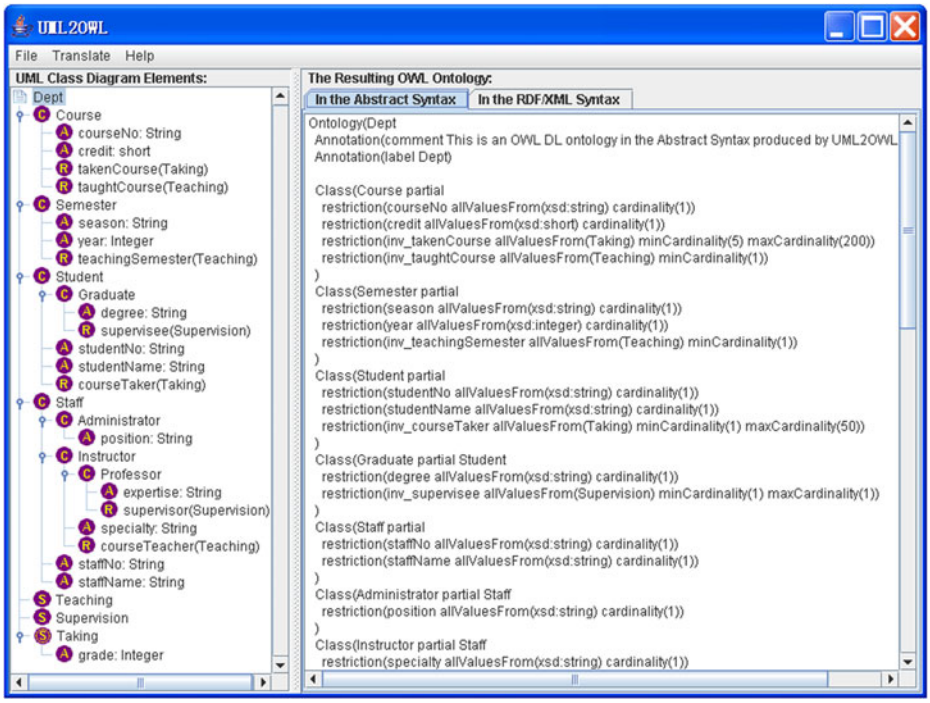

Figure 9 Screenshot of UML2OWL—the resulting OWL ontology in the abstract syntax.

Case studies show that our proposed approach is feasible and a fully automatic ontology extraction tool is achievable. The experiments also indicate that our UML2OWL tool can work with popular UML modeling tools on the market which support OMG's UML 2 model and XMI 2.0-coded UML class diagrams.

6 Conclusions

Aiming at providing a common framework that allows data to be shared and reused across application boundaries, the Semantic Web has been proposed as the next generation of the current Web. Ontologies play an essential role in such an effort, providing a concise and systematic means for defining Web data semantics in a formal, machine-processable way. However, manual ontology development using current OWL editors remains a tedious and cumbersome task that can easily result in a knowledge acquisition bottleneck. Conceptual schemata such as UML class diagrams, on the other hand, capture abundant domain knowledge. Thus developing an automatic tool for extracting OWL ontologies from UML class diagrams is meaningful to Web ontology development. In this paper we have presented an automatic, semantics-preserving approach for extracting OWL DL ontologies from existing UML class diagrams. The experiments with our implemented prototype tool, UML2OWL, show that the proposed approach is effective and a fully automatic ontology extraction tool is achievable.

The main contributions of our work are: (i) we propose and describe an efficient UML-to-OWL translation algorithm which uses normative Web Ontology Language as the target ontology language and captures richer knowledge of common UML constructs and constraints compared with existing approaches and/or tools; (ii) we give the formal proof for the semantics

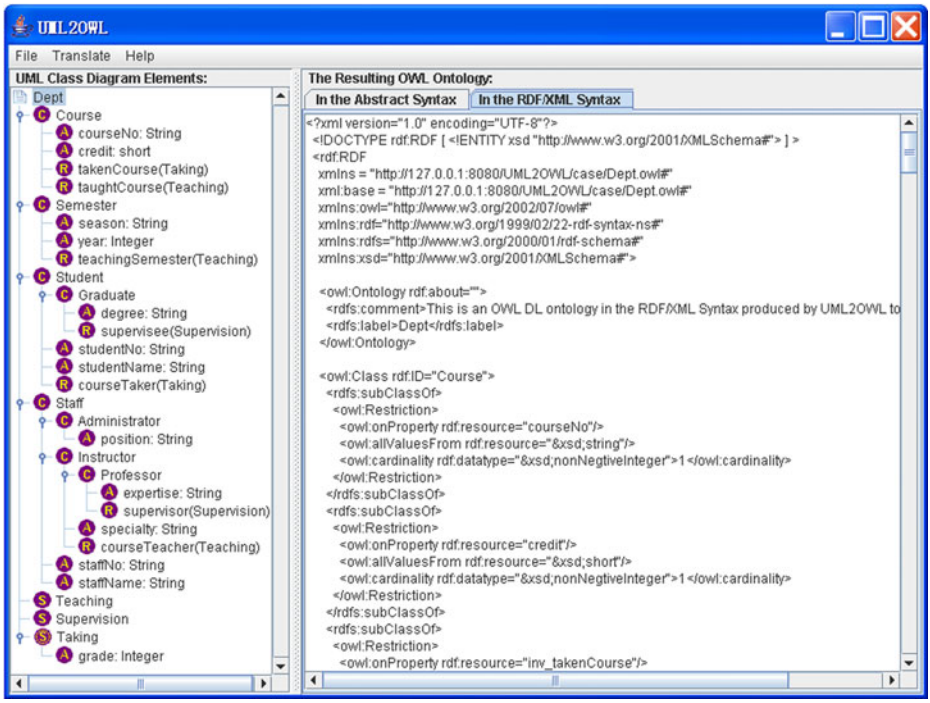

Figure 10 Screenshot of UML2OWL—the resulting OWL ontology in the RDF/XML syntax.

preservation of the UML-to-OWL translation algorithm, and analyze the time complexity of the algorithm; (iii) we present an automatic UML-to-OWL translation tool for extracting OWL ontologies from UML class diagrams, and indicate through experiments that the time complexity of the translation algorithm is $O(N)$. Since existing (legacy) UML class diagrams have been produced by popular UML tools which mostly support OMG's UML 2 model and export XMI 2.0-coded UML class diagram files, our approach can be widely applied in ontology development of Semantic Web applications whose underlying data sources are modeled in the UML model. In the sense of semantic interoperability, the proposed approach and tool can act as a gap-bridge between existing Web data sources and the Semantic Web.

The main limitation of the current version of our UML2OWL tool is that it cannot support several new syntaxes (including the Functional-Style syntax and the Manchester syntax) and richer datatypes of OWL 2 language. Our future work will be focused on improving our approach and tool to better support the new features of OWL 2.

Acknowledgements This work was supported by the following grants: (i) grant No. 20090094110015 from Research Fund for the Doctoral Program of Higher Education of China, (ii) grant No. BK2008354 and grant No. BK2010520 from the Natural Science Foundation of Jiangsu Province of China, and (iii) grant No. 2008135 from the "Six Talent Peaks Program" of Jiangsu Province of China.

Appendix: The resulting OWL DL ontology in the abstract syntax derived from UML class diagram `Dept`

Class(Staff partial restriction(staffNo allValuesFrom(xsd:string) cardinality(1)) restriction(staffName allValuesFrom(xsd:string) cardinality(1)));

Class(Administrator partial Staff restriction(position allValuesFrom(xsd:string) cardinality(1));

Class(Instructor partial Staff restriction(specialty allValuesFrom(xsd:string) cardinality(1)) restriction(inv_courseTeacher allValuesFrom(Teaching)));

Class(Course partial restriction(courseNo allValuesFrom(xsd:string) cardinality(1)) restriction(credit allValuesFrom(xsd:integer) cardinality(1)) restriction(inv_taughtCourse allValuesFrom(Teaching) minCardinality(1)) restriction(inv_takenCourse allValuesFrom(Taking) minCardinality(5) maxCardinality(200)));

Class(Student partial restriction(studentNo allValuesFrom(xsd:string) cardinality(1)) restriction(studentName allValuesFrom(xsd:string) cardinality(1)) restriction(inv_courseTaker allValuesFrom(Taking) minCardinality(1) maxCardinality(50)));

Class(Professor partial Instructor restriction(expertise allValuesFrom(xsd:string) minCardinality(1) maxCardinality(3)) restriction(inv_supervisor allValuesFrom(Supervision) minCardinality(1)));

Class(Graduate partial Student restriction(degree allValuesFrom(xsd:string) cardinality(1)) restriction(inv_supervisee allValuesFrom(Supervision) minCardinality(1) maxCardinality(1)));

Class(Supervision partial restriction(supervisor allValuesFrom(Professor) cardinality(1)) restriction(supervisee allValuesFrom(Graduate) cardinality(1)));

Class(Teaching partial restriction(courseTeacher allValuesFrom(Instructor) cardinality(1)) restriction(taughtCourse allValuesFrom(Course) cardinality(1)) restriction(teachingSemester

allValuesFrom(Semester) cardinality(1)));
Class(Taking partial restriction(grade allValuesFrom(xsd:char)) restriction(takenCourse
allValuesFrom(Course) cardinality(1)) restriction(courseTaker allValuesFrom(Student) cardinality(1)));
DatatypeProperty(staffNo domain(Staff) range(xsd:string));
DatatypeProperty(staffName domain(Staff) range(xsd:string));
DatatypeProperty(position domain(Administrator) range(xsd:string));
DatatypeProperty(specialty domain(Instructor) range(xsd:string));
DatatypeProperty(courseNo domain(Course) range(xsd:string));
DatatypeProperty(credit domain(Course) range(xsd:integer));
DatatypeProperty(season domain(Semester) range(xsd:string));
DatatypeProperty(year domain(Semester) range(xsd:integer));
DatatypeProperty(studentNo domain(Student) range(xsd:string));
DatatypeProperty(studentName domain(Student) range(xsd:string));
DatatypeProperty(expertise domain(Professor) range(xsd:string));
DatatypeProperty(degree domain(Graduate) range(xsd:string));
DatatypeProperty(grade domain(Taking) range(xsd:integer));
ObjectProperty(courseTeacher domain(Teaching) range(Instructor));
ObjectProperty(inv_courseTeacher domain(Instructor) range(Teaching) inverseOf(courseTeacher));
ObjectProperty(taughtCourse domain(Teaching) range(Course));
ObjectProperty(inv_taughtCourse domain(Course) range(Teaching) inverseOf(taughtCourse));
ObjectProperty(teachingSemester domain(Teaching) range(Semester));
ObjectProperty(inv_teachingSemester domain(Semester) range(Teaching) inverseOf(teachingSemester));
ObjectProperty(takenCourse domain(Taking) range(Course));
ObjectProperty(inv_takenCourse domain(Course) range(Taking) inverseOf(takenCourse));
ObjectProperty(courseTaker domain(Taking) range(Student));
ObjectProperty(inv_courseTaker domain(Student) range(Taking) inverseOf(courseTaker));
ObjectProperty(supervisor domain(Supervision) range(Professor));
ObjectProperty(inv_supervisor domain(Professor) range(Supervision) inverseOf(supervisor));
ObjectProperty(supervisee domain(Supervision) range(Graduate));
ObjectProperty(inv_supervisee domain(Graduate) range(Supervision) inverseOf(supervisee));
DisjoitClasses(Administrator Instructor);
EquivalentClasses(Staff unionOf(Administrator Instructor)).

References

1. Baclawski, K., Kokar, M.K., Kogut, P.A., et al.: Extending UML to support ontology engineering for the Semantic Web. In: Proc. of the 4th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools (UML 2001). LNCS, vol. 2185, pp. 342–360. Toronto, Canada (2001)

2. Berardi, D., Calvanese, D., Giacomo, G.D.: Reasoning on UML class diagrams. Artificial Intelligence **168**(1–2), 70–118 (2005)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284**(5), 34–43 (2001)
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) **5**(3), 1–22 (2009)
5. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. Journal of Artificial Intelligence Research **11**, 199–240 (1999)
6. Christine Golbreich, Evan K. Wallace (eds.): OWL 2 Web Ontology Language New Features and Rationale. W3C Recommendation 27 October 2009. http://www.w3.org/TR/owl2-new-features/ (2009). Accessed 30 January 2011
7. Cranefield, S.: UML and the semantic web. In: Proc. of the first Semantic Web Working Symposium (SWWS'01), pp. 113–130. Stanford University, California, USA (2001)
8. Dean, M., Schreiber, G. (eds.): OWL Web Ontology Language Reference. W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-ref/ (2004). Accessed 30 January 2011
9. Djuric, D., Gasevic, D., Devedzic, V.: Ontology modeling and MDA. Journal of Object Technology **4**(1), 109–128 (2005)
10. Falkovych, K., Sabou, M., Stuckenschmidt, H.: UML for the semantic web: transformation-based approaches. In: Omelayenko, B., Klein, M.C.A. (eds.) Knowledge transformation for the semantic web, frontiers in artificial intelligence and applications, vol. 95, pp. 92–106. Ios Press, Amsterdam (2003)
11. Gasevic, D., Djuric, D., Devedzic, V.: MDA-based automatic OWL ontology development. International Journal on Software Tools for Technology Transfer **9**(2), 103–117 (2007)
12. Grau, B.C., Horrocks, I., Motik, B., et al.: OWL 2: The next step for OWL. Journal of Web Semantics **6** (4), 309–322 (2008)
13. He, B., Patel, M., Zhang, Z., Chang, K.C.-C.: Accessing the deep web. Communications of the ACM **50** (5), 94–101 (2007)
14. Hermida, J.M., Romá-Ferri, M.T., Montoyo, A., *et al.*: Reusing UML Class Models to Generate OWL Ontologies: a Use Case in the Pharmacotherapeutic Domain. In: Proc. of International Conference on Knowledge Engineering and Ontology Development (KEOD 2009), pp. 281–286, Funchal, Portugal (2009)
15. Hitzler, P., Krötzsch, M., Parsia, B., et al.: OWL 2 Web Ontology Language Primer. W3C Recommendation 27 October 2009. http://www.w3.org/TR/owl2-primer/ (2009). Accessed 30 January 2011
16. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: the making of a web ontology language. Journal of Web Semantics **1**(1), 7–26 (2003)
17. Jacob, E.K.: Ontologies and the semantic web. Bulletin of the American Society for Information Science and Technology **29**(4), 19–22 (2003)
18. Maedche, A., Staab, S.: Ontology learning for the semantic web. IEEE Intelligent Systems **16**(2), 72–79 (2001)
19. Motik, B., Patel-Schneider, P.F., Grau, B.C. (eds.): OWL 2 Web Ontology Language Direct Semantics. W3C Recommendation 27 October 2009. http://www.w3.org/TR/owl2-direct-semantics/ (2009). Accessed 30 January 2011
20. Na, H.-S., Choi, O-H., Lim, J.-E.: A method for building domain ontologies based on the transformation of UML models. In: Proc. of 4th International Conference on Software Engineering Research, Management and Applications (SERA 2006), pp. 332–338, Seattle, WA, USA (2006)
21. Object Management Group: Ontology Definition Metamodel (ODM) Version 1.0. OMG Specification, formal/2009-05-01. http://www.omg.org/spec/ODM/1.0/PDF/ (2009). Accessed 30 January 2011
22. Object Management Group: Unified Modeling Language: Infrastructure, version 2.1.2. OMG Specification, formal/2007-11-04. http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/ (2007). Accessed 30 January 2011
23. Object Management Group: Unified Modeling Language: Superstructure, version 2.1.2. OMG Specification, formal/2007–11–02. http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/ (2007). Accessed 30 January 2011
24. Patel-Schneider, P.F., Hayes, P., Horrocks, I. (eds.): OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation 10 February 2004. http://www.w3.org/TR/owl-absyn/ (2004). Accessed 30 January 2011
25. Sahoo, S.S., Halb, W., Hellmann, S., et al.: A Survey of Current Approaches for Mapping of Relational Databases to RDF. W3C RDB2RDF Incubator Group report. http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf (2009). Accessed 30 January 2011
26. Schneider, M. (eds.): OWL 2 Web Ontology Language RDF-Based Semantics. W3C Recommendation 27 October 2009. http://www.w3.org/TR/owl2-rdf-based-semantics/ (2009). Accessed 30 January 2011

27. Stollberg, M., Lausen, H., Lara, R., et al.: Towards semantic web portals. In: Proc. of the WWW2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web, New York, NY, USA. CEUR Workshop Proceedings vol. 105. http://CEUR-WS.org/Vol-105/TowardsSWPortals.pdf (2004). Accessed 30 January 2011
28. Tijerino, Y., Embley, D., Lonsdale, D., Ding, Y., Nagy, G.: Towards ontology generation from tables. World Wide Web **8**(3), 261–285 (2005)
29. Volz, R., Handschuh, S., Staab, S., et al.: Unveiling the hidden bride: deep annotation for mapping and migrating legacy data to the Semantic Web. Journal of Web Semantics **1**(2), 187–206 (2004)
30. Xu, Z., Ni, Y., Lin, L., Gu, H.: A semantics-preserving approach for extracting OWL ontologies from UML class diagrams. Database Theory and Application (DTA 2009), Communications in Computer and Information Science, vol. 64, pp. 122–136 (2009)