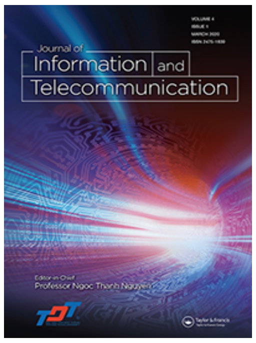


Journal of Information and Telecommunication

ISSN: 2475-1839 (Print) 2475-1847 (Online) Journal homepage: https://www.tandfonline.com/loi/tjit20

Transformation of UML class diagram into OWL Ontology

Minh Hoang Lien Vo & Quang Hoang

To cite this article: Minh Hoang Lien Vo & Quang Hoang (2020) Transformation of UML class diagram into OWL Ontology, Journal of Information and Telecommunication, 4:1, 1-16, DOI: 10.1080/24751839.2019.1686681

To link to this article: https://doi.org/10.1080/24751839.2019.1686681

JOURNAL OF INFORMATION AND TELECOMMUNICATION
2020, VOL. 4, NO. 1, 1–16
https://doi.org/10.1080/24751839.2019.1686681

Transformation of UML class diagram into OWL Ontology

Minh Hoang Lien Vo and Quang Hoang

University of Sciences, Hue University, Hue City, Vietnam

ABSTRACT
UML class diagrams are used in order to demonstrate the model information systems. Most of the former systems were previously framed by applying UML class diagrams. Besides, the current trend has a growing demand of reusing data from older information systems, which is very beneficial for the implementation of semantic knowledge. The upgradation and conversion of UML class diagrams into ontology aim to reuse older systems; therefore, cost reduction is essential. There have been many studies for transformation from UML class diagrams into ontology; however, they have not fully proposed the transformation rules. This paper will suggest the addition of rules for transforming UML class diagrams to ontologies, such as attribute transformations which have data types as classes, the structured attributes, the association with association class, the recursive association, the aggregation with a qualification.

ARTICLE HISTORY
Received 15 December 2017
Accepted 27 October 2019

KEYWORDS
Semantic Web; OWL; ontology UML; UML

1. Introduction

UML class diagram is a conceptual model which is often used for designing the logic model of information systems. The crucial advantage of designing system by utilizing UML lies in its ability to describe, and reflect the real world of information systems better. Furthermore, it has also got the further support of developers (Rational Rose, Enterprise Architect, etc.) so the UML design has gradually gained its popularity.

In recent years, the ontology has become a common term in many various aspects of computer science, promising to support a shared and common understanding of a specific knowledge domain that can be communicated between people and computerbased systems, as well as help computer understanding and processing information more efficiently. Basically, the ontology provides vocabulary to describe data with semantics that computers can understand. There is a range of different languages designed to perform ontology for semantic web. Outstandingly, RDFS and OWL are the two most basic languages which are used commonly. OWL is considered as an extension of RDFS to overcome the disadvantages of RDFS. Currently, OWL is being considered as the standard language for representing ontology for semantic web. OWL is a language describing the classes, attributes and relationships between objects in a way that machines can

CONTACT Minh Hoang Lien Vo minhvhl@gmail.com

understand. OWL1 is classified into three different types with the ability to perform incremental semantics: OWL Lite provides classes, hierarchy attributes and simple constraints. OWL DL increases expressiveness and yet retains decidability of the classification problem, which is expected to improve reasoning efficiency. OWL DL includes all OWL language constructs, but under certain restrictions. OWL Full is an entire language with no restriction; however, the disadvantage of this language is undecidable.

The OWL2 is an ontology language for the Semantic Web. It is considered as an extension of OWL DL by adding the syntax to the representation without losing its decidability. In particular, OWL2 is capable of representing the key of a class in an ontology similar to the key of an entity type in the ER model. For ontology editing, Stanford University has introduced the open source Protégé tool, which defines ontological concepts such as classes, properties, semantic categories, and different restrictions.

Many studies have transformed from the conceptual database model into the ontology. In relation to the transformation of the Entity-Relationship model to the OWL ontology, studies such as Fahad (2008) have proposed a method for designing OWL Ontologies from the ER model based on a set of rules transforming the components of an ER model (entities, attributes, and relationships between entities) into corresponding OWL components. Myroshnichenko and Murphy (2009) presented an automated conversion solution from the ER model to equivalently on the OWL Lite Ontology. The author presented a transformation algorithm based on five rules that map the components of the ER model to the corresponding components on OWL. Chujai, Kerdprasop, and Kerdprasop (2008) proposed an approach for constructing Ontology OWLs using Protégé from a relational database designed on a given ER model. Van Nguyen, Vo, Hoang, and Hoang (2016) introduced the fully EER model rules to the OWL.

There have been several studies on the transformation from UML into ontology. Gaevi, Djuri, Devedi, and Damja (2004) and Djuri, Gaevi, and Devedi (2005) presented the transformation method into OWL from a UML. Their solution was based on an MDAdefined architecture for ontology development and the Ontology UML Profile (OUP). The author presents a method to transform an ontology from its OUP definition into OWL description.

Brockmans et al. (2006) introduced a UML profile that allows visualization of OWL ontology by notation symbols which is similar to UML. This allows the development of ontology using UML tools. de Almeida Ferreira and da Silva (2007) analysed the feasibility and ease of an ontology model using common UML CASE tools.

Gherabi and Bahaj (2012) described how UML can be transformed into OWL Ontology, which enables reasoning on them by semantic web applications. However, the author used a mathematical model to represent, so many instances of the class diagram were ignored, such as structured attributes, attributes in which the datatype is class, recursive association, and association with association class, etc. because such cases are difficult to perform on a mathematical model.

Zarembo and Kodors (2013) proposed the automatic transformation into OWL2 ontology from the data model of the UML ISO 19.103 class diagram. However, Zarembo and Kodors (2013) have proposed only common cases, such as classes, attributes, and associations. There are many missing instances of the UML class diagram, such as recursive association, association with association class, aggregation, etc.

Zedlitz and Luttenberger (2012, 2012, 2014) analysed similarities and differences between UML class models and OWL ontologies, and identified incompatible language features, as well as presented the transformation between the UML class diagram and the OWL 2 ontology. The authors have also proposed transformation rules as well as analysed some differences and similarities in the representation of data types in UML and OWL2. Like the above-mentioned authors, they have put forward the addition of cases such as generalizations/specializations, but there are still cases where such combinations have not yet been displayed, such as association with association class, recursive association, etc.

Besides, El Hajjamy, Alaoui, Alaoui, and Bahaj (2016) presented a detailed and comprehensive comparison of the differences between the two languages, as well as analysed the existing mapping methods between them and proposed a novel process of direct and automatic mapping solution. Their process preserves the semantic of some features of UML class diagrams such as inheritance, data types, types of associations (compositions, class associations, n-ary associations, dependency and simple associations, etc.). In comparison with the above authors (El Hajjamy et al., 2016) it suggests that any cases which are often neglected in the previous approaches such as aggregation, recursive association. But still some cases have not yet been proposed as attributes in which the datatype is class, association with association class, instances of recursive association, etc.

Grunwald (2014) has researched and evaluated of existing UML to OWL approaches, however, the author does not specify transformation algorithms for cases, as well as many missing instances such as multiplicity of attribute value, enumeration, data constraints, association classes, n-ary associations, overlapping/disjoint class annotation, etc.

In general, these studies have delivered their concentration on developing methods for transforming the UML class schema into OWL ontology, such as transformation of the class, attributes, and relationships between classes. However, these studies have not fully proposed the transformation rules of the UML class diagram.

Therefore, on the basis of the proposal of the previous studies, the paper adds transformation rules for the following cases such as structured attribute, recursive association, association with association class, the aggregation with a qualify, etc. From these rules, it is possible to transform directly from the UML class diagram into the OWL2.

Accordingly, the paper will seek to address the following questions. In the next section, it will illustrate clearly the rule of transforming a UML class diagram into OWL ontology. In the next part, we will attempt to put forward the conclusion and discussion of the next research direction.

2. Transformation of UML into OWL2

UML and OWL2 are designed for different purposes, but they have shared many similarities. Studies (Kiko, 2008; Grunwald, 2014; Zarembo & Kodors, 2013; Zedlitz, Jörke, & Luttenberger, 2012; Zedlitz & Luttenberger, 2012) provide an overview of the fundamental differences between UML and OWL2.

In this section, we propose the rules for transforming UML into OWL. In these transformation rules, the components on the UML will be transformed into corresponding OWL components so that these components must represent the data and constraints on the UML.

Before introducing our transforming rules, the paper defines the notation as follows: In the UML class diagram, the class is E (or F), the attribute is $attE$, the relationship is R. In OWL, the class is $C(E)$, datatype property is $attE$ and object property is E_1RE_2 (R is the relationship between two classes E_1 and E_2). The OWL result is described by VisioOWL, which is a Microsoft Visio application used to create graphical representations of OWL ontologies (Flynn, 2005).

To illustrate the transformation rules, we use the UML class diagram as follows: The relationship between the Employee, Department, and Project that the employee joins is shown in Figure 1 (Elmasri, 2015).

2.1. Transformation of class

UML and OWL2 all have a concept of class. The class description for a set of objects has the same structure, behaviour, and relationships. The class can have attributes that determine the structure and the operation determines the behaviour of the instance of the class. Class in OWL2 is an empty set or has multiple instances and do not define operations on those instances. A class in UML is a more general structure, containing a set of instances (Zarembo & Kodors, 2013).

The previous studies have proposed the rules of transformation the class in a UML class diagram to a class in OWL ontology. So we will construct the transformation rules as follows (Zedlitz et al., 2012):

Rule UML1: For each UML class *E* that is transformed into class *C*(*E*) respectively in OWL 2, set *DisjointClasses* characteristics for each class if not they are not generalization.

In Figure 2, the class *Employee* was transformed into class *Employee* in OWL.

2.2. Transformation of an attribute

The attribute in UML must be unique in the context of the class. Attributes may have different levels of usage, and they are described as being accessible from other classes. In OWL, the datatype properties do not support the retrieval access. So, the transformation of attributes in UML into OWL ontology depends on the type of the attribute. If the type of the attribute is a primitive data type, attributes are transformed into a datatype property in

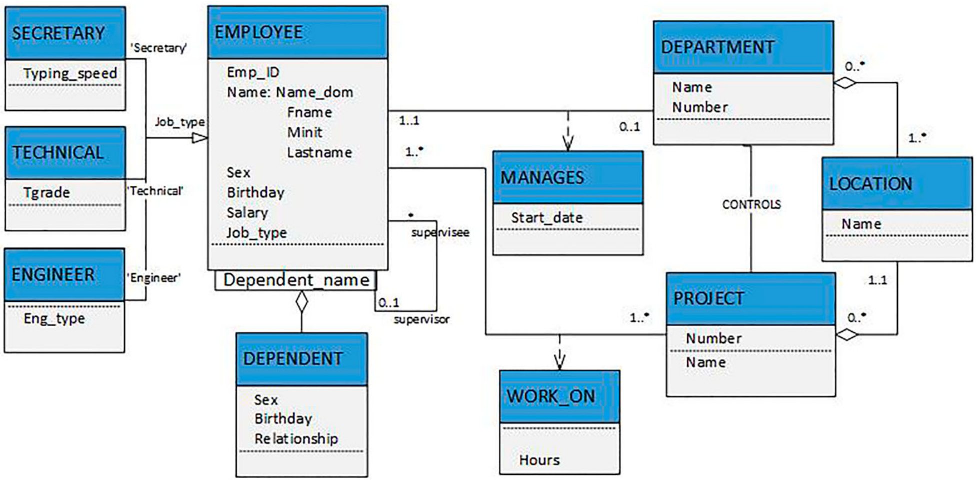


Figure 1. An example of UML class diagram.

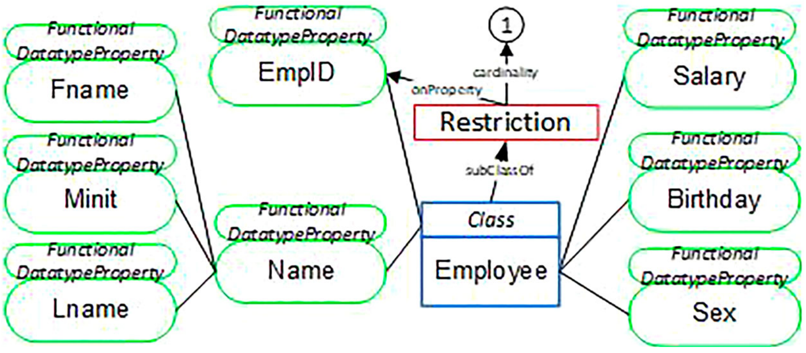


Figure 2. An example of UML class diagram.

the OWL, if the type of the attribute is a class, that attribute is transformed into an object property. The author El Hajjamy et al. (2016) has proposed to transform the key attribute of the class, but in practice, there are still instances where the class does not have a key attribute. So we have the transformation rules as follows:

Rule UML2: Transform the attribute *attE* of the class *E* which has a primitive data type in UML into the datatype property *attE* with range is corresponding with data type in OWL, and domain is *C*(*E*), set function characteristics for datatype properties *attE*.

Apply Rule UML2, the attributes *Sex*, *Birthday* and *Salary* as Figure 1 will be transformed into the datatype properties as Figure 2.

Similar, the transformation rules for attribute of class that the datatype is class in UML as follows:

Rule UML3. Transformation of attribute *attE* of class *E* that the datatype is class *F* in UML into datatype property *attE* in class *C*(*E*) with domain is *C*(*E*), range is *C*(*F*).

For example, in Figure 3, the attribute *att1* is transformed into object property *att1* with domain is class *CLASS1* and range is class *CLASS2*.

An attribute can be declared with an enumerated, it means that we are able to receive any literal value which predefined by the user. To represent the enumeration in the OWL ontology, there are two different ways to do this. The first is to declare a range of property with a list of the values by the *DataOneOf* syntax. The second is to define the new datatype

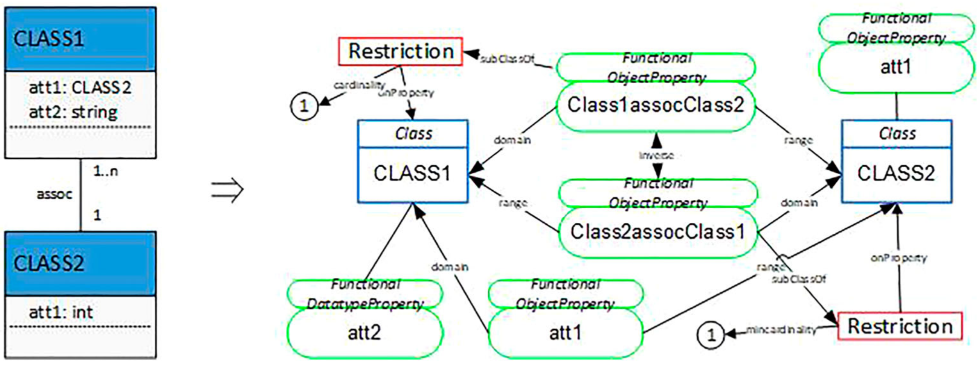
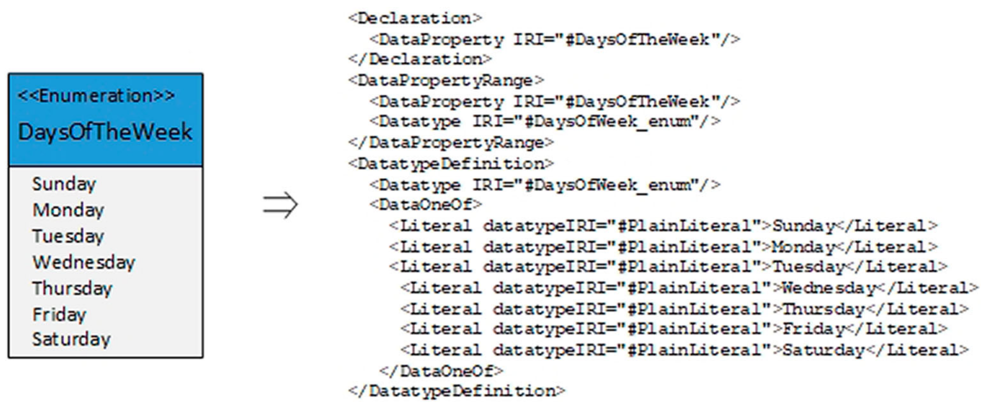


Figure 3. An example of transformation of the association and the attribute whose data type is class.



```
<Declaration>
  <DataProperty IRI="#DaysOfTheWeek"/>
</Declaration>
<DataPropertyRange>
  <DataProperty IRI="#DaysOfTheWeek"/>
  <Datatype IRI="#DaysOfWeek_enum"/>
</DataPropertyRange>
<DatatypeDefinition>
  <Datatype IRI="#DaysOfWeek_enum"/>
  <DataOneOf>
    <Literal datatypeIRI="#PlainLiteral">Sunday</Literal>
    <Literal datatypeIRI="#PlainLiteral">Monday</Literal>
    <Literal datatypeIRI="#PlainLiteral">Tuesday</Literal>
     <Literal datatypeIRI="#PlainLiteral">Wednesday</Literal>
     <Literal datatypeIRI="#PlainLiteral">Thursday</Literal>
     <Literal datatypeIRI="#PlainLiteral">Friday</Literal>
     <Literal datatypeIRI="#PlainLiteral">Saturday</Literal>
   </DataOneOf>
</DatatypeDefinition>
```

Figure 4. An example of transformation of the enumeration.

in OWL with the *DataOneOf* syntax, and the range of datatype property is the type of new datatype which is defined above.

In this case, we find that using the second method is more accurate. The reason is that the enumerated is a user-defined data type, therefore defining the new data type in the OWL is correct. Also, if you apply the first method, suppose that there is another attribute which employs the values of that enumerated, then declare it again. But if you use the second method, does not need to be declared, that enumeration can be used.

Rule UML4. Defines the new data type *attE_enum* with the *DataOneOf* syntax. Transform the attribute *attE* of the class *E* which has an enumeration data type into the datatype property *attE* with range is an enumerated data type att*E_enum*, and domain is *C*(*E*).

For example, in Figure 4, we defines the new data type *DayOfTheWeek* with the *DataOneOf* syntax.

2.3. Transformation of structure attributes

If attributes are complex, then it should be separated into a class that is independent of each other. However, there are still cases where structured attributes still retain the complex structure to represent the true meaning of the real world. In the transformation of an attribute, the previous papers only proposed transformation the single attribute, nobody mentioning a structured attribute instance. El Hajjamy et al. (2016) proposed to convert the structured attribute to a new class and an object property which show relationship between classes. However, the datatype property in OWL support hierarchical structure, so that the transformation of structured attributes into sub datatype properties will be more contextually expressed, and not the creation of new classes and object property.

Rule UML5. The structure attribute *attE* of class *E* which has the set of sub attributes *sub_attE* is transformed into datatype property *attE* in class *C*(*E*). The sub attribute *sub_attE* will transform into sub datatype properties *sub_attE* of the datatype property *attE*. The *sub_attE*'s domain is *attE* and range is corresponding with data type in OWL.

For example, in Figure 2, the sub attribute *Fname*, *Minit*, *Lname* of the structure attribute *Name* is transformed into sub datatype properties.

2.4. Transformation of the relationship between classes

Object-oriented systems are sets of objects that interact with each other to perform tasks as required. Relationships are semantic links between object classes, which express their relevance to the properties and operations of each other in the system. In UML, there are different types of relationships: association, aggregation, inheritance, and dependency.

2.4.1. Transformation of the association

In UML, the association can be one-directional or uni-directional. An association can be transformed into an object property in OWL. For two-dimensional association, the two attributes are converted into two object properties for each direction. To ensure that both of these object properties are part of the association, thus it setups inverse properties (Zedlitz et al., 2012).

Rule UML6. Let consider that the association R between classes E_1 and E_2, we will construct the transformation rules as follows:

- Add two inverse object properties E_1RE_2 and E_2RE_1 which show relationship between class $C(E_1)$ and $C(E_2)$.
- The cardinality constraint in UML will be change the position between two classes when transforming into OWL. For each value of cardinality constraint (*min*, *max*), if *min* differential 0 or *max* differential N on the association R, add *min*/*max* constraint to the corresponding object properties.

For example, in Figure 3, the association *assoc* between class *CLASS1* and *CLASS2* is transformed into two inverse object properties *Class1assocClass2* and *Class2assocClass1*. The cardinality constraint will be change the position between two classes when transforming into OWL, so we add minimum and maximum cardinality restriction by 1 to the object property *Class1assocClass2*. In Figure 3, with cardinality constraint $1 \ldots n$, add minimum cardinality restriction by 1 to the object property *Class2assocClass1*.

2.4.2. Transformation of the association with association class

The association with association class is basically a class attached to an association, which is used to model association as a UML class. This association class is similar to any normal class, with names, properties, and operations. However, the association class is characterized by complementary attributes that do not belong to any object in the association. In OWL, it is not possible to represent an association class, so in this case it must create new classes and two object properties to represent relationships between classes. We have a transformation rule as follow:

Rule UML7. Let consider the association between two classes E_1 and E_2 with association class A, we have the following transformation rules:

- Add class $C(A)$, attributes *attA* of association class A are transformed into datatype properties *attA* of class $C(A)$.
- Add two inverse object properties E_1HasA and $AOfE_1$ which show the association between class $C(A)$ and $C(E_1)$; Two inverse object properties E_2HasA and $AOfE_2$ which

show the association between class $C(A)$ and class $C(E_2)$. Set function characteristics and minimum constraint is 1 in two attributes: $AOfE_1$, $AOfE_2$.

- The cardinality constraint in UML will be change the position between two classes when transforming into OWL. For each value of cardinality constraint (*min*, *max*), if *min* is differential 0 and *max* is differential N on the association, add the corresponding *min*/*max* constraints to the object properties E_1HasA and E_2HasA. If the association is an n-n relationship, add two object properties $AOfE_1$, $AOfE_2$ to the set of key attributes of class $C(A)$.

In Figure 5, the association between two classes *Employee* and *Department* with association class *Manages* was transformed into new class *Manages*, and the attributes *Start_-date* of association class *Manages* was transformed into datatype properties of class *Manages*.

2.4.3. Transformation of the recursive association

It is possible to associate a class with itself in an association relationship. The association still represents a semantic relevance, but the linked objects fall into the same class. At this time, a set of instances can take a single role or two different roles in the same association. Checking the role allows us to classify all of the inheritance: symmetric or asymmetric. These are issues that previous studies have not addressed.

A recursive relationship is symmetric when all of the instance participating in the relationship has a unique role and has the same semantics. Accordingly, if *R* is a recursive association which is symmetric then $role_1$ and $role_2$, in which $role_1$ and $role_2$ are the name of two roles of the recursive association *R*. Conversely, if *R* is not symmetric, we call this association asymmetric.

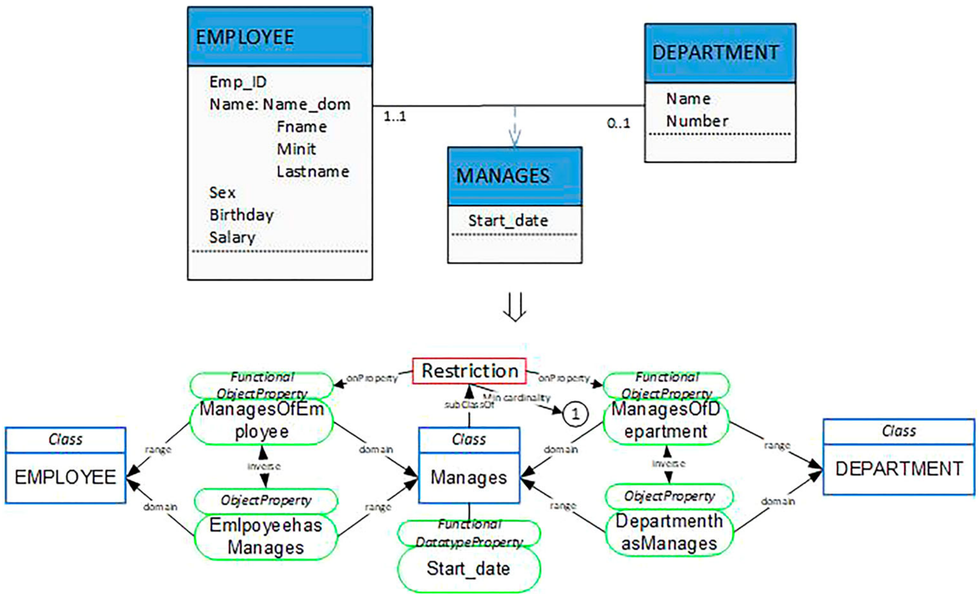


Figure 5. An example of transforming the association relationship with the association class.

In previous studies, no studies have proposed the classification of recursive association, therefore we propose a transformation rule as follows:

Rule UML8. Let consider the symmetric recursive association of class *E* with role *role*, we have transformation rules as follow:

- Add the object property with its name is the name of role, domain and range is class $C(E)$, set the symmetry characteristics for this object property.
- If the cardinality constraint is 1:1, add the maximum cardinality restriction by 1.
- With the minimum cardinality constraint is 1–1, add minimum cardinality restriction by 1 to this object property.

Apply these transformation rules of the symmetric recursive association *TaxedJointly-With* of *Person* class as in Figure 6.

In the case of an asymmetric recursive association, the role of relationship and the cardinality constraints is different, so when transforming into OWL, it adds two object properties. We have a transformation rule as follow:

Rule UML9. Considering the asymmetric recursive association of the class *E* with two roles: $role_1$ and $role_2$, we have the transformation rules as follow:

- Add a pair of inverse object properties with their name are $role_1$ and $role_2$, domain and range is class $C(E)$.
- The cardinality constraint on the asymmetric recursive association will converse the position between two classes when transforming into OWL. For each role with *min*/*max* cardinality constraint different 0 and *N*, add the *min*/*max* cardinality restriction corresponding to the object properties.

The asymmetric recursive association of the class *Employee* with two roles: *supervisee* and *supervisor* was transformed as in Figure 7.

2.4.4. Transformation of the aggregation

The aggregation relationship is a special case of association that focuses on the relationship between the aggregation (the whole) and a component part. In UML, there are three kinds of aggregation depending on the constraint of the part in its whole: the first one is normal aggregation, the second one is by reference (shared

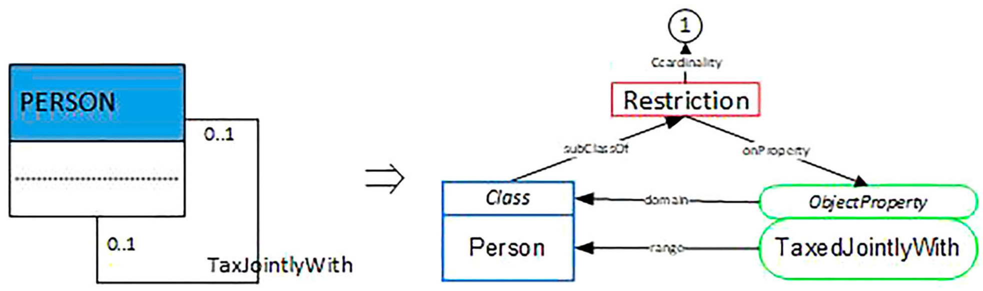


Figure 6. An example of transformation of symmetric recursive association.

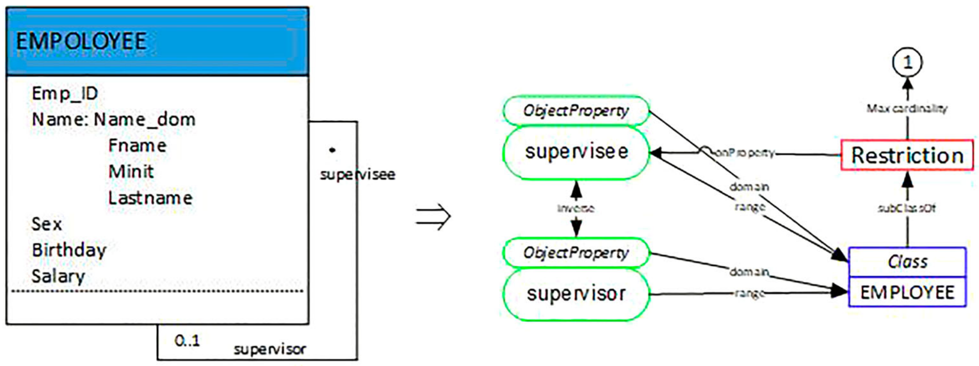


Figure 7. An example of transformation of non-symmetric recursive association.

aggregation), the third one is by value (composite aggregation). The composite aggregation is equivalent to the attribute composition, so the transformation rule is similar to the attribute transformation.

Aggregation indicates that the existence of the part is dependent on the whole. The existence dependency leads to the transforming of an aggregation into a pair of inverse object properties in OWL with corresponding constraint.

The normal aggregation represent the relationship between the two classes in which the object of the class consists of some objects of the other, but does not exist in its interior. We see that the normal aggregation is a combination of asymmetric classes, and an aggregation is not associated with itself. Therefore, we have the normal aggregation transformation rules as follow:

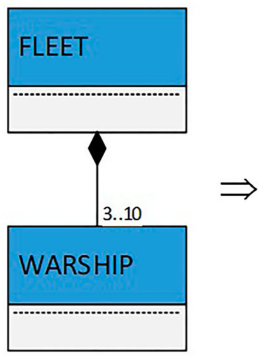
Rule UML 10. Let consider the normal aggregation class *E* associated with component class *F*, we have a transformation rule as follow:

- Add class *C*(*E*) and *C*(*F*).
- Add the object property *F_E* for representing the relationship between class *C*(*E*) and class *C*(*F*), with domain is *C*(*F*), range is *C*(*E*).
- Set *InverseFunctionalObjectProperty* for object properties *F_E*.
- Set *IrreflexiveObjectProperty* for the object attribute *F_E*.
- Set *AsymmetricObjectProperty* for the object attribute *F_E*.
- The cardinality constraint on the normal aggregation will change the position between two classes when transforming into OWL.

For example, a class *FLEET* consists of a number (3…10) warships of the class *WARSHIP*, but warships are not included in the class *FLEET*. When transforming into OWL ontology, the object property *Was_Ship* will have the following code as in Figure 8.

Shared aggregation is an aggregation in which the component can be more involved in the whole. We see that: The shared aggregation is a combination of irreflexive, so when transforming into OWL, it must use the *IrreflexiveObjectProperty* syntax. The shared aggregation is not recursive, a class isn't aggregate with itself, so must set the *AsymmetricObjectProperty* syntax.

Therefore, we have the shared aggregation transformation rules as follow:



```
<owl:ObjectProperty rdf:about="#Warship_Fleet">
<rdf:type rdf:resource="#InverseFunctionalProp-
erty"/>
<rdf:type rdf:resource="#AsymmetricProperty"/>
<rdf:type rdf:resource="#IrreflexiveProperty"/>
<rdfs:domain rdf:resource="#Warship"/>
<rdfs:range rdf:resource="#Fleet"/>
</owl:ObjectProperty>
```

Figure 8. An example of normal aggregation transformation.

Rule UML11. Let consider the shared aggregation class *E* associated with component class *F*, we have a transformation rule as follow:

- Add class *C*(*E*) and *C*(*F*).
- Add two object properties for representing the relationship between class *C*(*E*) and class *C*(*F*): *E_F* with domain is *C*(*E*), range is *C*(*F*); *F_E* with domain is *C*(*F*), range is *C*(*E*).
- Set *IrreflexiveObjectProperty* for the object attribute *F_E*.
- Set *AsymmetricObjectProperty* for the object attribute *F_E*.
- The cardinality constraint on the share aggregation will change the position between two classes when transforming into OWL.

As shown in Figure 9, class *Project* aggregation with the class *Location* is transformed into two inverse object properties. In particular, the object property *Location_Project* is set to the irreflexive and antisymmetric axiom.

With the qualified aggregation between the whole class and the component class, we have the transformation rules as follow:

Rule UML12. Let consider the aggregation with a qualify *Q* between the whole class *E* and the component class *F*, we have the transformation rules as follow:

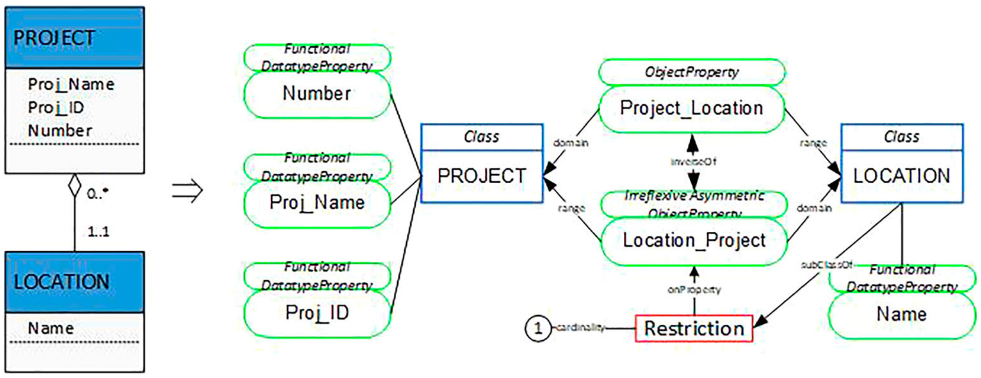


Figure 9. An example of share aggregation transformation.

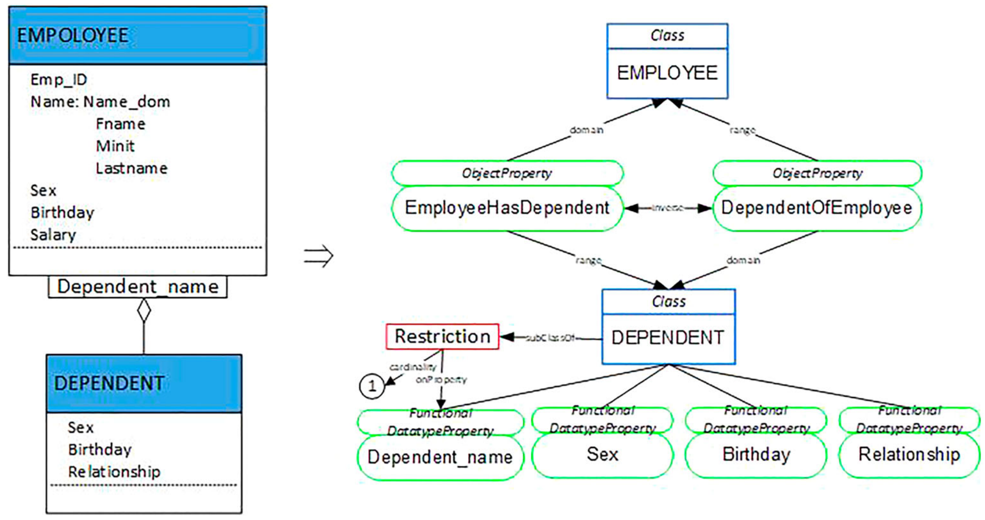


Figure 10. An example of transformation of the aggregation with a qualify.

- Add two classes $C(E)$ and $C(F)$.
- Add two inverse object properties for representing the relationship between class $C(E)$ and class $C(F)$: E_F has a domain is class $C(E)$, range is class $C(F)$; F_E has a domain is class $C(F)$, the range is class $C(E)$.
- The qualify Q is transformed into a datatype properties Q in class $C(F)$, domain is class $C(F)$ and range is the corresponding data type in OWL, add to the set of key properties of class $C(F)$.
- The cardinality constraint on UML will change position between two classes when transforming into OWL.

For example, the qualified aggregation with a qualify *Dependent_name* between the whole class *Employee* and the component class *Dependent* was transformed as shown in Figure 10.

2.4.5. Transformation of generalization/specialization

Generalization and specialization are two of the top and bottom views of class hierarchy, describing the complexity of managing the system by abstracting. The concepts of generalization and specialization in UML and OWL are similar. If C' is a specialized subclass of class **C** and *i* is an instance, then in both cases we have $C'(i)- > C(i)$ (Zedlitz et al., 2012).

In UML, there are two kinds of constraints for generalizations: completeness and disjointness. In the UML, the keyword *disjoint* or *complete* was added near the generalization arrow. Generalization without the keyword is not subjected to the two type constraints.

Rule UML13. Considering the generalization between the class E and the sub class E_i:

- Add the class $C(E)$ and class $C(E_i)$ which is subclass of class $C(E)$.
- If the generalization is *disjoint*, use syntax *owl:disjointWith*.
- If the generalization is *disjoint* and *complete*, use syntax *owl:disjointUnion*.

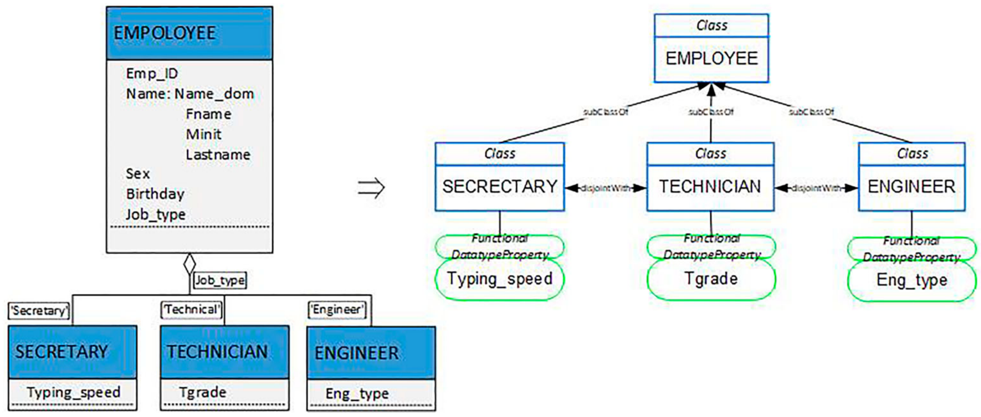


Figure 11. An example of transformation of the Generalization/Specialization.

The generalization between the class *Employee* and the sub class *Secretary*, *Technical*, *Engineer* was transformed as in Figure 11.

Executing our transforming rules with the UML class diagram of a human resource management company and projects as shown in Figure 1 as input produced a valid OWL2 presented in Figure 12 (notation described in Flynn, 2005). The ontology consists of classes corresponding to the class of the UML class diagram and object properties corresponding to UML relationships named in accordance with the conventions introduced by our transforming rules. We observe that all of the structural constraints in the input UML class diagram have been correctly mapped to equivalent OWL constraints. The OWL ontology results is verified on Protégé and reasoner, and found that the OWL result is plausible from the perspective of data and semantic requirements.

3. Conclusion

On the basis of the transformation of the conceptual database models into ontology, this paper has proposed the addition of transformation rules from the UML class diagram into OWL2.

Since previous studies have proposed transformation the classes in UML into classes in OWL, transformation attributes in UML into object properties in OWL, the paper adds the transformation rules such as the attribute that the datatype is class, the structured attributes. The paper also proposes to define the new datatype in OWL to apply to enumeration.

In the relationship transformation, the paper analyses case studies such as association, aggregation, dependency, and inheritance. In previous studies, we found that the proposals only dealt with association and dependence, but not enough. In particular, there have not yet been many researches which analyse and suggest transformations for association with associated classes, recursive association, aggregation with a qualify. In particular, the paper has analysed in detail the classification of recursive association, thus propose a method for transforming the corresponding cases. Comparing with previous studies, these are issues that previous studies have not addressed.

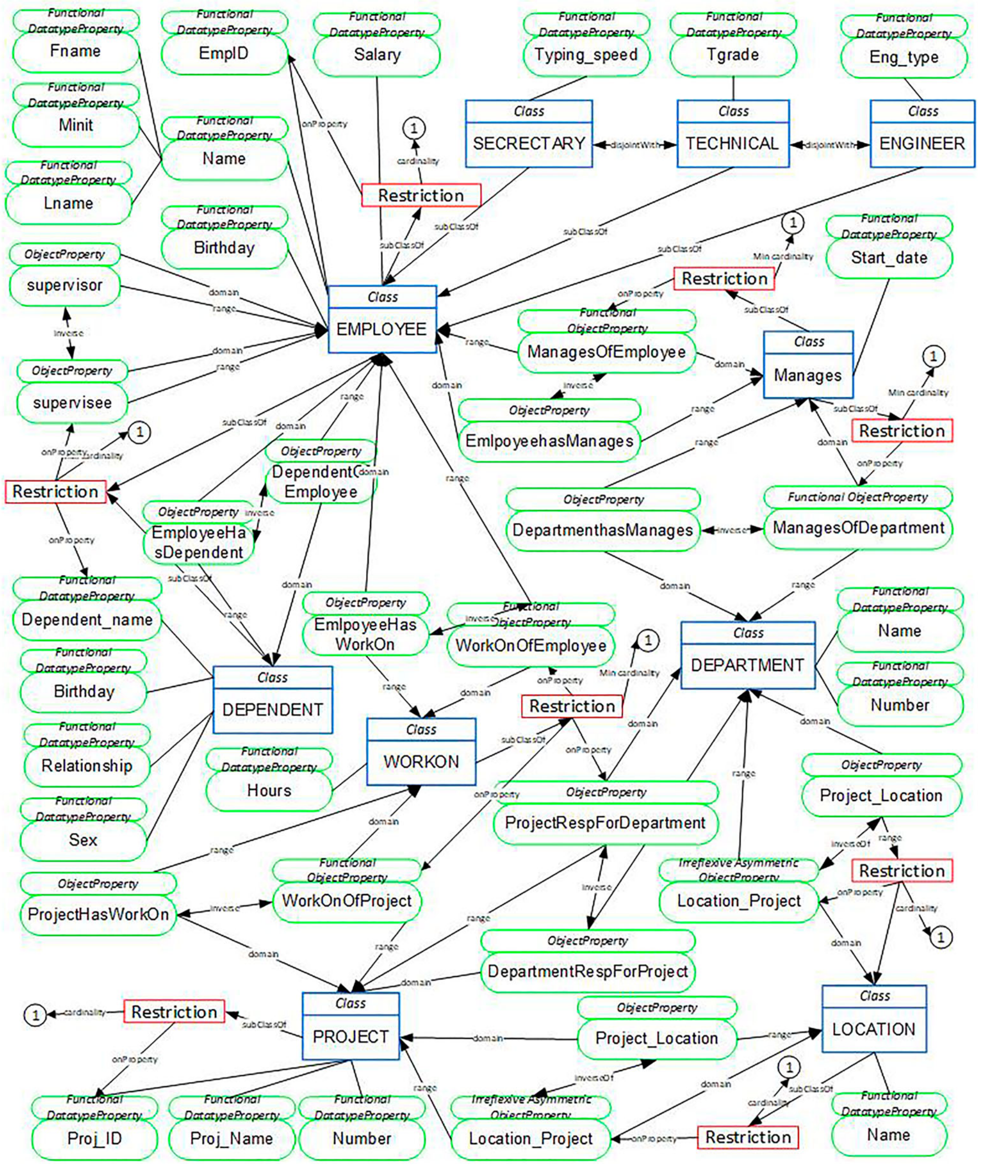


Figure 12. An example of transformation of the UML class diagram in Figure 1.

Transformation from UML class diagram into OWL2 to describe the semantics of UML class diagrams, enabling systems to interact and integrate information. Intuitive and step-by-step visualization rules are easy to access. With the proposed transformation rules, the ontology construction is guaranteed according to the steps. Firstly, we define the target use of the ontology. Then, construct the ontology in the following steps:

(a) Identify concepts and relationships in the domain. Create definitions for those concepts, relationships and identify referring terms.
(b) Knowledge representation already in a) in OWL ontology.

Finally, we evaluate the resulting ontology, ensuring consistency of the transformation. The paper has been experimenting with a UML class diagram of a human resource management company and projects. Experimental results show that the conversion from UML layer diagram into OWL is guaranteed in terms of data.

If n is the number of class, l is the number of properties of each class, m is the number of relationships in the of UML class diagram, we see that if the apply the transformation rules for class, the complexity of algorithm will be $O(n)$, the complexity of the properties transformation will be $O(n \times l)$ and the complexity of the transformation of the relationship will be $O(m)$. Since the complexity of each rule is $O(1)$, the complexity of the transformation rules will be $O(\max(m, n \times l))$.

Due to dissimilarities of UML class diagram and OWL2, there have still been indicative challenges when transforming into OWL2 in order to ensure 'semantic preservation'. For example, OWL2 cannot present a similar structure to the '*profiles*' concept as well as the abstract classes in UML. In UML, the component data of a class can be encapsulated through access control properties. The visibility of attributes can be set by marking them as 'public', 'private'. OWL2 has no control mechanism to restrict access to model elements, etc.

The scope of UML application for modelizing the system is highly increasing. With the current development of the semantic web, it is essential to transform the previous systems in UML into OWL. In the future, we will continue to study the additional cases in order to complete the set of rules.

Disclosure statement

No potential conflict of interest was reported by the authors.

Notes on contributors

Minh Hoang Lien Vo is currently a doctoral student at the Hue University of Sciences, Vietnam. He received his master degree in Computer Science (2013) from Hue University of Sciences, Vietnam. His current research includes modelling and design of conceptual database models, temporal database, XML database systems and ontology. He currently works at the Thua Thien Hue Department of Tourism, Hue city, Vietnam.

Assoc. Prof. Quang Hoang received his PhD in Computer Science (2004) from the Institute of Information Technology, Vietnam Academy of Science and Technology. He is currently Dean of the Faculty of Information Technology, Hue University of Sciences, Vietnam. His research interests include modeling and design of conceptual database models, object-oriented database, temporal database, ontology and XML database systems.

ORCID

Minh Hoang Lien Vo http://orcid.org/0000-0002-1533-3647

References

Brockmans, S., Colomb, R., Haase, P., Kendall, E., Wallace, E. K., & Xie, G. (2006). A model-driven approach for building OWL DL and OWL full ontologies. *Proceedings of the international semantic web conference (ISWC) 200.*

Chujai, P., Kerdprasop, N., & Kerdprasop, K. (2008). On transforming the er model to ontology using Protégé OWL tool. *International Journal of Computer Theory and Engineering*, *6*, 484–489.

de Almeida Ferreira, D., & da Silva, A. M. R. (2007). UML to OWL mapping overview: An analysis of the translation process and supporting tools. *7th conference of Portuguese association of information systems (CAPSI06)*.

Djuri, D., Gaevi, D., & Devedi, V. (2005). Ontology modeling and MDA. *Journal of Object Technology*, *4*, 109–128.

El Hajjamy, O., Alaoui, K., Alaoui, L., & Bahaj, M. (2016). Mapping UML to OWL2 ontology. *Journal of Theoretical and Applied Information Technology*, *90*(1), 126–143.

Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of database systems* (7th ed.). United States of America: Addison-Wesley.

Fahad, M. (2008). ER2OWL: Generating OWL ontology from ER diagram. *IFIP – The international federation for information processing*. American Psychological Association.

Flynn, J. (2005). MS Visio VisioOWL Stencil. *VisioOWL*. Retrieved from http://projects.semwebcentral.org/projects/visioowl

Gaevi, D., Djuri, D., Devedi, V., & Damja, V., (2004). Converting UML to OWL ontologies. *ACM: Proceedings of the 13th international World Wide Web conference*.

Gherabi, N., & Bahaj, M. (2012). A new method for mapping UML Class into OWL ontology. *Special Issue of International Journal of Computer Applications on Software Engineering, Databases and Expert Systems*.

Grunwald, A. (2014). Evaluation of UML to OWL approaches and implementation of a transformation tool for visual paradigm and MS Visio. *Christian Doppler Laboratory CDL-Flex, Institute of Software Technology and Interactive Systems*, Vienna University of Technology, Favoritenstrasse 9-11/188, AT 1040 Vienna, Austria.

Kiko, A. (2008). A detailed comparison of UML and OWL. *Bericht 4*, Department for Mathematics and C.S., University of Mannheim.

Myroshnichenko, I., & Murphy, M. C. (2009). Mapping ER schemas to OWL ontologies. *Semantic computing, ICSC '09. IEEE international conference* (pp. 324–329).

Van Nguyen, T., Vo, H. L. M., Hoang, Q., & Hoang, H. H. (2016). A new method for transforming TimeER model based specification into OWL ontology. *8th Asian conference on intelligent information and database systems ACIIDS*. Da Nang, Viet Nam.

Zarembo, I., & Kodors, S. (2013). Automatic transformation of UML geospatial profile to OWL ontologies. *Virtual multidisciplinary conference QUAESTI*.

Zedlitz, J., Jörke, J., & Luttenberger, N. (2012). From UML to OWL 2. *Communications in Computer and Information Science*, *295*.

Zedlitz, J., & Luttenberger, N. (2012). Transforming between UML conceptual models. *Terra Cognita 2012 Workshop*.

Zedlitz, J., & Luttenberger, N. (2014). Conceptual modelling in UML and OWL2. *International Journal on Advances in Software*, *7*, 182–196.