Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MAPPING UML TO OWL2 ONTOLOGY

[1]OUSSAMA EL HAJJAMY, [2]KHADIJA ALAOUI, [3]LARBI ALAOUI , [4]MOHAMED BAHAJ
[1,2,4] University Hassan I, FSTS Settat, Morocco
[3]International University of Rabat 11100 Sala Al Jadida, Morocco
E-mail: [1]elhajjamyoussama@gmail.com, [2]khadija.alaoui.ma@gmail.com, [3]larbi.alaoui@hotmail.de, [4]mohamedbahaj@gmail.com

ABSTRACT

UML (Unified Modeling Language) is a standardized modeling language widely used by domain experts to model real-world objects in developing object oriented applications. On the other hand, the conceptualization, which is represented in OWL, is designed for use by applications that need to process the content of information instead of just presenting information. Therefore, the problem of migrating UML to OWL is becoming an active research domain. In this paper we present a detailed and comprehensive comparison of the differences between the two languages, analyze the existing mapping methods between them and propose a novel process of direct and automatic mapping solution UML2OWL2 that generalizes these methods. Our process preserve the semantic of some features of UML class diagrams such as inheritance, data types, types of associations (compositions, class associations, N-ary associations, reflexive associations, dependency and simple associations …). It defines precise transformation rules and provides a model of ontology while covering the semantic of the source UML class diagrams. A tool based on our approach has also been developed and tested to demonstrate the practical applicability of our strategy.

Keywords: *UML, OWL2, direct mapping, automatic mapping, semantic, ontology, UML class diagram*

1. INTRODUCTION

UML, the Unified Modeling Language, is the most used language in the requirements specification [15] and design of object oriented software in the middle tier of enterprise applications and it is developed for the purpose of business and general domain modeling. In the meantime OWL [18] defines a common set of concepts and terms that are used to describe and represent a domain of knowledge and it is developed for representing semantic information for the World Wide Web by providing a vocabulary to represent classes, class's hierarchies, associations between classes and properties.

Clearly, the formal structure of UML is quite different from that of OWL. The major difference between these languages is that the modeling of OWL is less constrained than that of UML, which means that many OWL models have no equivalent in UML, and OWL provides more primitives than UML such as the disjointness, union, intersection and equivalence of classes. Consequently, the limitation of UML for being used as a visual syntax for knowledge representation, the increase of semantic web technologies and the fast development of web applications based on ontology have all made the problem of migrating UML to OWL an active research domain.

However the existing studies do not provide a complete solution to this problem and so far there still be no effective proposals that could be considered as a standard method that preserves the original structure of the source UML class diagrams.

Our aim in this work is to take a further step in the existing research works by identifying the weaknesses and limitations of the different existing techniques and proposals, and address other very important aspects that have not been touched yet in the world of conversion from UML to OWL. These aspects are mainly related to inheritance (generalization between classes, associations), data types (enumeration, primitive and complex type) and the different types of associations (composition, class associations, N-ary associations, dependency, reflexive and simple associations).

We perform our work at two levels, one providing a comparison of the existing mapping methods from UML to OWL and the other proposing a novel migration solution UML2OWL2 that generalizes these methods, optimizes the constraints extraction

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

and refines the mapping rules to be more expressive and less complicated.

The remainder of this paper is organized as follow. In the following section we present an overview of the different UML to OWL schema transformation proposals. Needful terminology and several rules to convert UML into OWL2 are presented in section 3. To illustrate how to combine the rules together for a concise mapping, sections 4 and 5 respectively outline the automatic mapping algorithm and its implementation. Finally, section 6 includes some conclusions and future work.

2. COMPARISON OF EXISTING MAPPING METHODS

Due to the widespread use of UML and OWL languages, it is no wonder that there are many works in the literature whose goal is to study the different relationships between UML and OWL and propose a transformation from UML to OWL [1, 3, 5, 10, 11, 14, 17 and 18].

In this section we aim at giving a summarized review of RDB-to-OWL existing methods. The investigation of these methods is done with the focus on their shortcomings with regards to the relevant elements that are not considered in their mapping process. In the following section all such elements will be treated by our mapping strategy and a comparison table of the different approaches will is given with the UML class diagram elements that are considered by each proposal and also the strategy followed in order to translate them into OWL ontology.

Cranefield [17] provide a UML-based visual environment for modeling web ontology. He creates an ontology in a UML tool and then save it as an XMI-coded file. Then an XSLT stylesheet translates the XMI-coded file into the corresponding RDF Schema (RDFS). However this method has inherent drawbacks because RDFS does not have enough expressive power to capture the knowledge and constraints of UML. Although the work in [17] deals with the mapping of UML into RDF we choose to also consider its mapping approach in our comparison of existing works since it could be considered as one of the starting works that motivated the other approaches [1, 3, 5, 10, 11, 14, 18] that came after for the conversion from UML into OWL.

In [14] and for mapping from UML into OWL, Gherabi defined a correspondence between the class diagrams of UML and OWL by using a mathematical representation of the class diagram. However this method neglects many properties of the source UML class diagram such as the various types of association and generalization, data types and composition.

In [3] Zedlitz considered the mapping between UML elements and OWL2 constructs such as disjoint and complete generalization, generalization between associations, composition and enumeration. However, the constraints specific model elements (e.g. composition, multiple inheritance, datatypes and class association) imposed on the model are not mapped. In an extension of this work as presented in [5] Zedlitz focused on the data types of static data models often neglected in the previous approach and showed some differences and similarities in the representation of datatypes in UML and OWL2. Furthermore, a formal algorithm and a tool have not been introduced in their works.

Tschirner et al. described in [18] some conversion rules from UML-data models to OWL. They specify four main rules to map UML classes and attributes to OWL classes and properties. However, some important constraints (e.g. disjointness, composition, n-ary association …) were ignored in the transformation.

All aforementioned contributions and limitations of the mapping approaches are summarized in Table III which clearly shows the completeness of our mapping strategy in comparison with these works. Our strategy starts indeed from the limitations of existing mapping approaches and aims at giving concise mapping rules for all relevant elements in UML class diagrams. Such rules will be the basis for deriving a simple conversion algorithm that we implement using the programming language Java. We do it with the aim to come up with a solution to all aforementioned limitations of existing approaches in order to provide the semantic world as complete as possible conversion technique that allow to easily and fully deduce all conceptual details of the considered UML specifications relative to the analysis, conception and design of the associated modeled systems. This is fully justified by the need to handle all relevant concepts of the domain being modeled by considering there associated UML constructs in the mapping process. Each of the previous works does not handle many of such elements. In particular all these do not treat the case of multiple inheritance, dependency, complete generalization, reflexive association and each of them treats the other constructs only partially. In the following

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

section we propose to give clear and concise conversion rules by taking into account all such constructs of a UML class diagrams. The rules allow us to derive an algorithm that is as simple as possible and which does not use any intermediate language.

3. UML TO OWL2 MAPPING RULES

In order to create better OWL2 ontologies from a UML class model, we detail in this section our migration solution and we comprehensively give out the transformation rules. The proposed conversion rules are based on considering all possible cases in a UML class diagram.

Our approach begins with the extraction of the structure of the source UML class diagram. Then, by applying the rules of transformation from UML to OWL2 we create the classes and the properties of the objects and types of data that make up the model of the ontology.

3.1. Mapping Classes

Rule 1. Both UML and OWL2 use classes to represent concepts of a domain. Because both concepts are similar a basic transformation can be done.

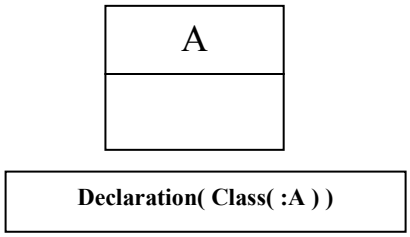


Figure 1. *Transformation of a class*

3.2. Mapping Relationships

Before introducing our mapping rules for relationships, we briefly give a new categorization for all types of relations. The relations are divided into the six following distinct categories:

Rel = {SAS, NAS, ASC, RAS}

- ✓ SAS (Simple association): specifies a semantic relationship that can occur between two different classes.
- ✓ NAS (N-ary association): is an association among three or more classes. The NAS can be defined as NAS = (AssocName, C1, C2… Cn) , n>2, where C1, C2, …, Cn are the names of the classes related by the association "AssocName".
- ✓ CAS (Class Association): is a modeling element that has both association and class properties. It allows us to add attributes, operations, and other features to associations
- ✓ RAS (Reflexive association): is an association between instances of the same class.

Based on the aforementioned categorization our mapping rules for relationships are as follows.

Rule 2. Every simple association in UML is converted into an ObjectProperty axiom in OWL2. If the navigability is given in both directions, then "InverseObjectProperties" axiom is added to the ontology.

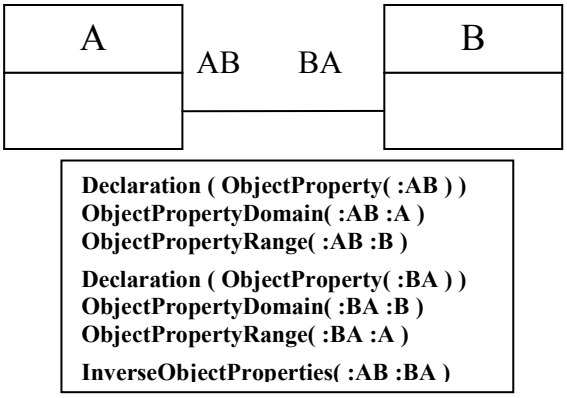
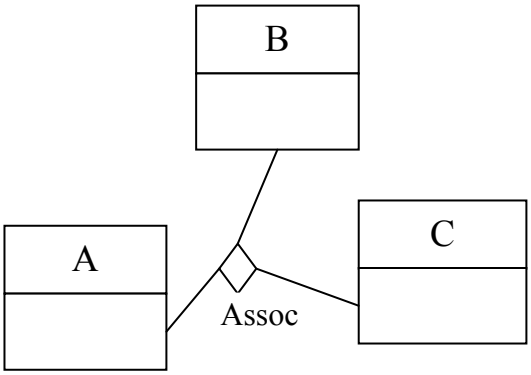


Figure 2. *Transformation of simple binary associations*

Rule 3. For each N-ary association with n classes (n>2) we create a new class, having a name equal to the name of the association, whose instances are instances of links in the association and n ObjectProperty whose domains are the new class and whose ranges are the classes attached to the member ends of the association.



Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

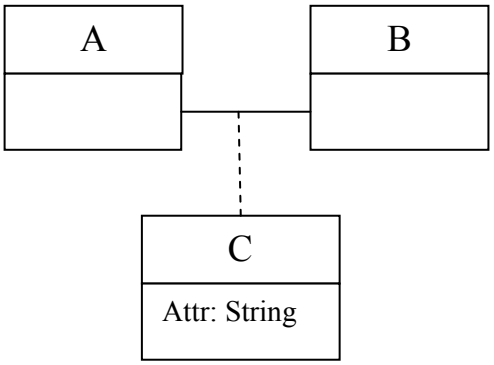
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
Declaration(  Class( :A ) )
Declaration(  Class( :B ) )
Declaration(  Class( :C ) )
Declaration(  Class( : Assoc ) )

Declaration( ObjectProperty( :A_Assoc ) )
ObjectPropertyDomain( :A_Assoc :A )
ObjectPropertyRange( :A_Assoc :Assoc )

Declaration (ObjectProperty( :B_Assoc ) )
ObjectPropertyDomain( :B_Assoc :B )
ObjectPropertyRange( :B_Assoc : Assoc )

Declaration (ObjectProperty( :C_Assoc ) )
ObjectPropertyDomain( :C_Assoc :C )
ObjectPropertyRange( :C_Assoc : Assoc )

SubClassOf( :Assoc :A_Assoc )
SubClassOf( : Assoc :B_Assoc )
SubClassOf( : Assoc :C_Assoc )
```

Figure 3. *Transformation of an N-ary association*

Rule 4. A class association with attributes is formally transformed to:

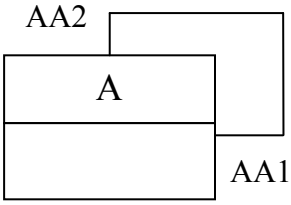
- ✓ an OWL class (named className), with data property for every additional attribute and two pairs of inverse object properties for every class connected to the association class
- ✓ and object property chains between the different classes connected to the association class




```
Declaration(  Class( :A ) )
Declaration(  Class( :B ) )
Declaration(  Class( :C ) )

Declaration ( ObjectProperty( :C_A ) )
ObjectPropertyDomain( :C_A :C )
ObjectPropertyRange( :C_A :A )
Declaration ( ObjectProperty( : A_C ) )
ObjectPropertyDomain( : A_C : A )
ObjectPropertyRange( : A_C : C)
InverseObjectProperty( : C_A : A_C )
Declaration ( ObjectProperty( :C_B ) )
ObjectPropertyDomain( :C_B :C )
ObjectPropertyRange( :C_B :B )
Declaration ( ObjectProperty( :B_C ) )
ObjectPropertyDomain( :B_C :B )
ObjectPropertyRange( :B_C :C )
InverseObjectProperty( : C_B : B_C )

Declaration( Data Property( :Attr ) )
DataPropertyDomain( :Attr :C )
DataPropertyRange( :Attr  xsd:String )

SubObjectPropertyOf(
   ObjectPropertyChain( :A_C :C_B )
    :A_B )
 SubObjectPropertyOf(
    ObjectPropertyChain( :B_C :C_A )
    :B_A)
```

Figure 4. *Transformation of a class association*

Rule 5. Every reflexive association in UML is converted into OWL2 by using the "ReflexiveObjectProperty" axiom.




```
ReflexiveObjectProperty( :AA1 )
ReflexiveObjectProperty( :AA2 )
```

Figure 5. *Transformation of a reflexive association*

3.3. Mapping Attributes

In a UML class diagram an attribute x in class C can be one of the following:

Attr = {SimpleAttr, idAttr}

- ✓ SimpleAttr (Simple attribute): is an attribute whose type is a Primitive Type.
- ✓ idAttr (id attribute): UML offers the possibility to define a single key "idAttr" per class. This latter can be used to enforce that

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

there are no two different instances of a class for which all relations specified in the key have an identical value.

Rule 6. For each SimpleAttr we create a data type property by respectively associating with its domain and range the URI of the class corresponding to the attribute and the XSD type corresponding to the type of the attribute in the UML class diagram.

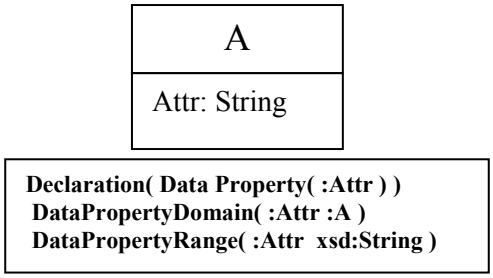


Figure 6. *Transformation of an attribute*

Rule 7. idAttr implies that the values of the data type property that represent this attribute must be unique. Therefore, these properties must be declared with HasKey properties.

Declaring a predicate as a HasKey property is similar to saying that it is an InverseFunctionalObjectProperty. The difference between both is that:

- ✓ HasKey is applicable only to individuals that are explicitly named by an IRI in ontology.
- ✓ InverseFunctionalObjectProperty is applicable to any kind of individual (named individual, anonymous individual, and any individual whose existence is implied by existential quantification).

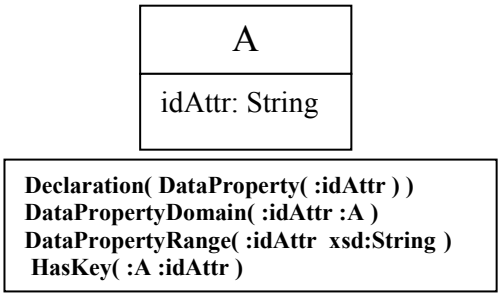


Figure 7. *Transformation of an id-attribute*

3.4. Mapping data Types

A UML datatype is a classifier, similar to a class, whose instances are identified only by their value. It is shown using a rectangle symbol with keyword «dataType». On the other hand an OWL2 datatype is defined by assigning an Internationalized Resource Identifier (IRI) to a DataRange using a DatatypeDefinition axiom.

In the UML2OWL2 transformation process, this aspect should not be ignored, especially as OWL2 comes with an elaborate support for datatype properties.

DataType = {PrimType, ComplexType, Enum}

- ✓ PrimType (Primitive type): is a data type which represents atomic data values, i.e. values having no parts or internal structure. UML supports the predefined primitive data types defined in the PrimitiveTypes package of the Auxiliary Constructs package.
- ✓ ComplexType (Complex type): In contrast to primitive data types, complex data types have an internal structure. For example attribute whose type is a class, it belongs to a class and connect it with another class.
- ✓ Enum (Enumeration): An enumeration in UML is a designated collection of literals, is used to create a datatype with a predefined list of allowed values.

Rule 8. Primitive data types are transformed into their corresponding datatype from XML Schema because owl uses the majority of the datatypes integrated into XML schema.

Rule 9. The UML to OWL2 transformation of complex datatypes is similar to the transformation of associations. Therefore the most similar concept in OWL2 for a complex datatype is "ObjectProperty" axiom. Therefore it is converted to a unidirectional ObjectProperty.

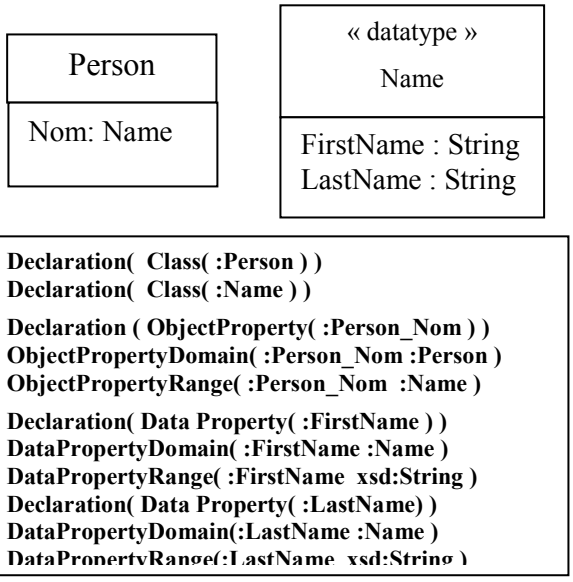


Figure 8. *Transformation of complex datatypes*

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Rule 10. In OWL 2 the data range "DataOneOf" axiom is suitable for defining an equivalent datatype to enumeration. DataOneOf defines a datatype with a fixed predefined value space.

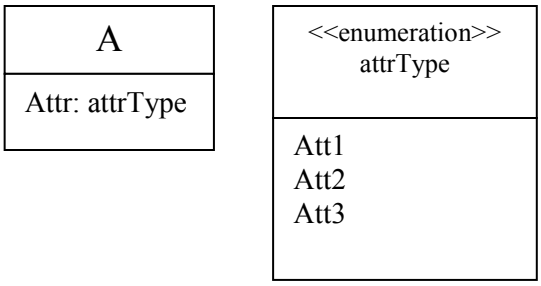


Figure 9. *Transformation of an enumeration*

3.5. Mapping Multiplicity Constraints

UML allows the user to specify the multiplicity for the association's source and target. OWL2 achieves this by specifying minimum and maximum cardinalities.

If a binary UML association has a multiplicity on its both ends, then the corresponding OWL property will be an inverse ObjectProperty, each having one of the multiplicity declarations.

The following table gives the restrictions to apply to object properties based on the UML cardinalities:

TABLE I. *Transformation of multiplicity constraints*

	Multiplicity in UML	Equivalent into OWL 2
Rule11	0..1	ObjectMaxCardinality(1 :A_B)
	1..1	ObjectExactCardinality(1 :A_B)
	0..n	No restriction
	1..n	ObjectMinCardinality(1 :A_B)

3.6. Mapping Generalization-Specialization

The Generalization-Specialization relationship is a relationship between the top class of a hierarchy, called the super-class, and the lower level classes in the hierarchy, called the sub-classes. The sub-classes have the properties of the parent but also have additional properties peculiar to the child. In UML class models it is not only possible to use generalization for classes but also for associations and data types.

In this sense, the definition for a Generalization-Specialization relationship can be specified as:

GS = {GSC, GSA, GSD}

- ✓ GSC: Generalization-Specialization between classes and it is divided into five categories:

GSC = {GSCS, GSCD, GSCC, GSCDandC, MI}

- GSCS: Generalization-Specialization relationship between classes without constraints;
- GSCD: Disjoint (but not complete) generalization-Specialization between classes where an instance of one sub-class must not be instance of another sub-class of the generalization;
- GSCC: Complete generalization-Specialization between classes where all subclasses have been specified and no additional subclasses can be added;
- GSCDandC: Disjoint and complete generalization-Specialization between classes;
- MI: Multiple inheritance indicates that a class is an immediate subclass of several other classes at the same time;

- ✓ GSA: In UML class diagrams it is not only possible to use generalization-specialization for classes but also for associations;
- ✓ GSD: Generalization-Specialization between data types.

Due to the very similar structure and semantics of Generalization elements in UML on the one hand and SubClassOf or SubProperty axioms in OWL 2 on the other hand, a transformation from UML to OWL2 is easily possible as follow:

Rule 12. For every two elements that are connected via an instance of the UML meta-class "Generalization" we create an instance of the OWL meta-class "SubClassOf".

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

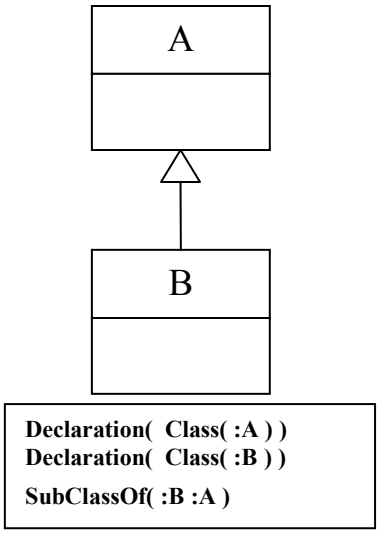


Figure 10. *Transformation of a generalization between classes*

Rule 13. We transform a disjoint (but not complete) generalization-Specialization between classes by adding a DisjointClasses axiom with all sub-classes.

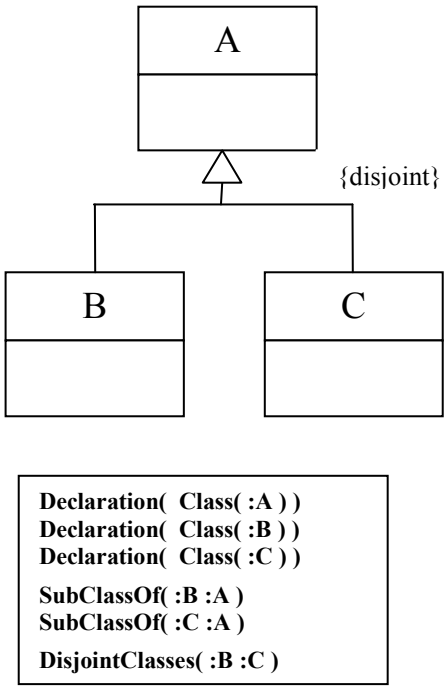


Figure 11. *Transformation of a disjoint generalization*

Rule 14. Complete Generalization-Specialization in UML defines a class as a set of subclasses. We transform this kind of constraint to OWL 2 by using the "EquivalentClasses" and "ObjectUnionOf" axioms.

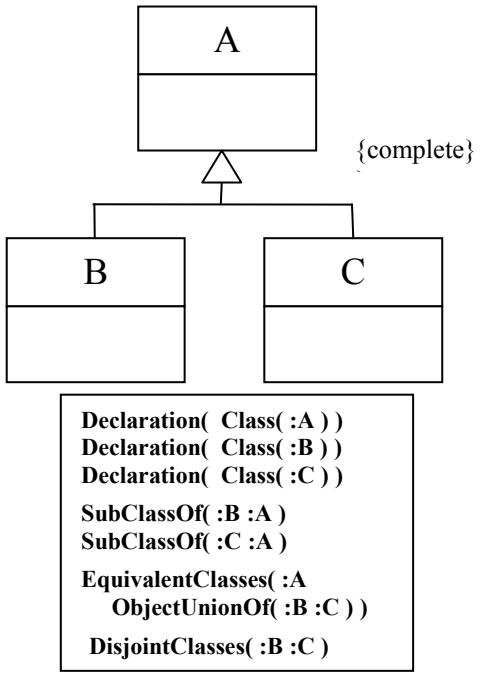


Figure 12. *Transformation of a complete generalization*

Rule 15. In UML a disjoint and Complete Generalization-Specialization states that all the subclasses are pairwise disjoint and semantically equivalent to the super-class. In this case, we use the "DisjointUnion" axiom to generate its equivalent in OWL 2.

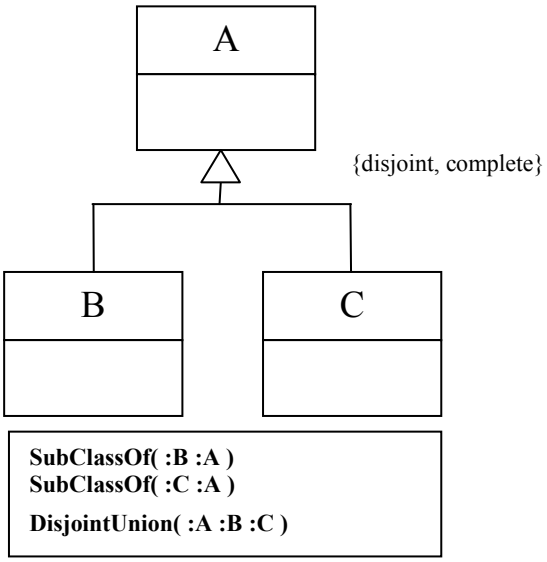


Figure 13. *Transformation of a disjoint and complete generalization*

Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Rule 16. Multiple inheritance involves that an instance of the sub-class is an indirect instance of the super-classes. The element has to obey the necessary conditions of all super classes and is therefore an element of the intersection of all super classes. This can be transferred to the OWL2 world using "IntersectionOf" axiom.

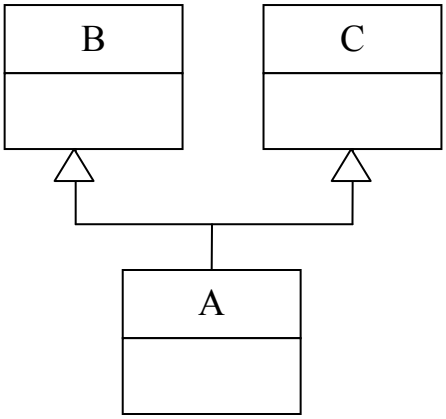
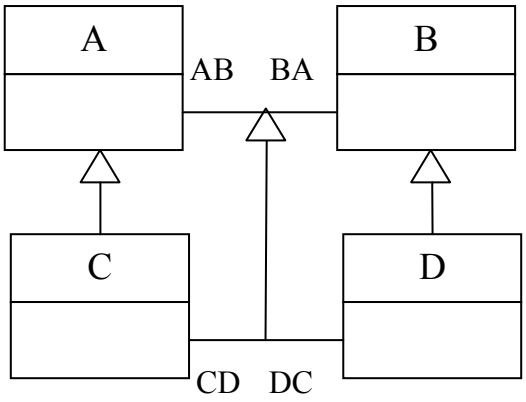


Figure 14. *Transformation of a multiple inheritance*

Rule 17. Generalization between associations can be transformed into OWL 2 by using "SubProperty" axiom. Since the generalization between two bidirectional associations must be transformed into two "SubPropertyOf" axioms.



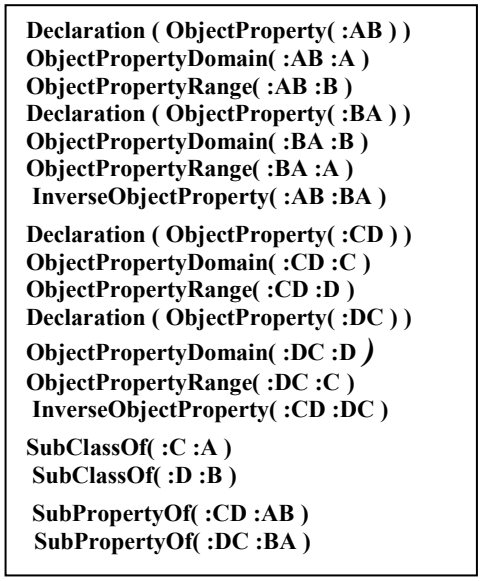
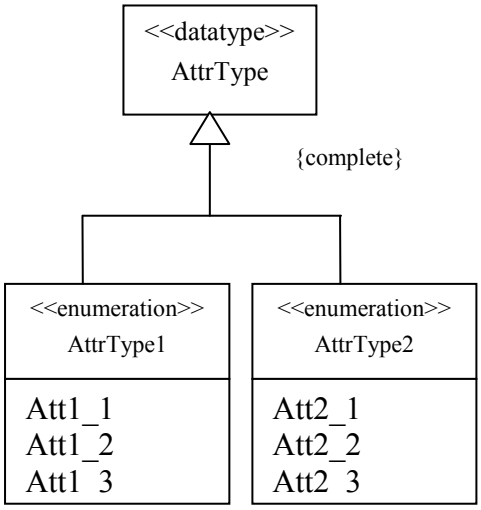


Figure 15. *Transformation of a generalization between associations*

Rule 18. For the transformation of generalization between datatypes, we define a new data range in OWL2 by adding a "DatatypeDefintion" axiom. This data range contains the sub-datatypes combined in the "DataUnionOf" axiom.



Journal of Theoretical and Applied Information Technology
15th August 2016. Vol.90. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
Declaration( Datatype( :AttrType) )
Declaration( Datatype( :AttrType1 ) )
Declaration( Datatype( :AttrType2 ) )
DatatypeDefinition(
    :AttrType1
    DataOneOf(Att1_1  Att1_2   Att1_3 ) )
DatatypeDefinition(
   : AttrType2
   DataOneOf(Att21_1  Att2_2   Att2_3) )
DatatypeDefinition(
  :AttrType
  DataUnionOf( :AttrType1 : AttrType2 ) )
```

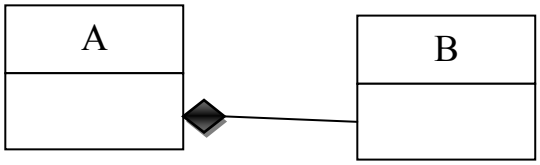
Figure 16. *Transformation of a generalization between datatypes*

3.7. Mapping a composition

CP (Composition): In UML a composition is a special kind of association between classes that specifies that every instance of a given class (of parts) can be a part of at most one whole.

Rule 19. A direct mapping of a composition is not feasible in OWL2 ontology. However, the part of the whole can be transformed like other simple associations into an "ObjectProperty" axiom by taking into account the following restrictions:

- ✓ The composition association is antisymmetric. We can transform this constraint by adding an "AsymmetricObjectProperty" axiom.
- ✓ The composition association is irreflexive (a class must not be in a composition relation to itself). For this restriction we can use the "IrreflexiveObjectProperty" axiom.
- ✓ An object of a class must not be part of more than one composition. We can achieve this restriction by adding "InverseFunctionalObjectProperty" axiom.



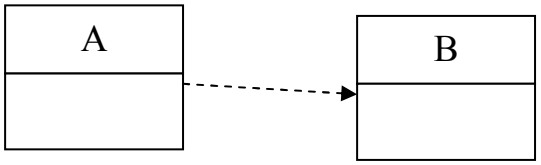

```
Declaration ( ObjectProperty( :B_isPartOf_A ) )
ObjectPropertyDomain( : B_isPartOf_A  :B )
ObjectPropertyRange( : B_isPartOf_A  :A )
InverseFunctionalObjectProperty( :B_isPartOf_A )
IrreflexiveObjectProperty ( : B_isPartOf_A )
AsymmetricObjectProperty ( : B_isPartOf_A )
```

Figure 17. *Transformation of a composition*

3.8. Mapping dependencies

A dependency in UML class diagram denotes that the existence of a target class is dependent of a source class.

Rule 20. For A direct mapping of a dependency is not feasible in OWL2 ontology. Therefore the dependency is transformed into a unidirectional ObjectProperty with the name "ClassSource_depend_ClassTarget". The domain and the range are given according to the direction of association.




```
Declaration ( ObjectProperty( :A_depend_B ) )
ObjectPropertyDomain( : A_depend_B :A )
ObjectPropertyRange( : A_depend_B :B )
```

Figure 18. *Transformation of a dependency*

4. UML TO OWL2 MAPPING ALGORITHM

Our approach aims at defining a correspondence between the UML class diagram and OWL2 ontology. It consists of three separate phases as shown in figure 19.

The first step stores the model information in XMI document by using a Power Designer tool; the XMI format (XML Metadata Interchange) makes it possible to represent an UML model in an XML format. In the second step the different elements of the source UML class diagram (such as classes, attributes, generalizations, compositions, dependencies, and several relationships) as mentioned in section 3 are extracted and used as the input of our mapping algorithms. Finally the transformation tool applies our algorithm based on the list of rules to create the equivalent ontology in owl2. Figure 19 below shows the architecture of UML2OWL2 implementation.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

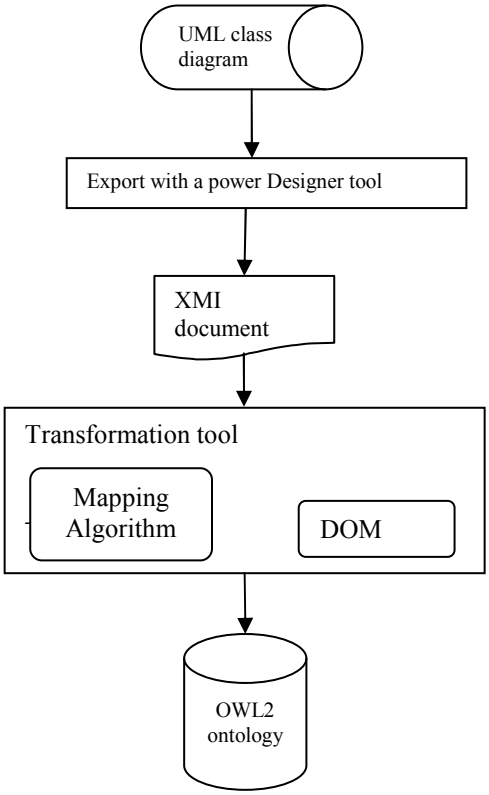


Figure 19. *UML2OWL2 framework architecture*

In this section we will introduce our algorithm for the automatic construction of OWL2 ontology from a UML class diagram. The algorithm captures important semantic properties of the UML class diagram such as inheritance, enumeration, composition, dependency … . Following the set of rules that have been given in the previous section, the problem that arises is how to apply the rules in an efficient manner in the transforming process. Different sequences of mapping the different relations will lead to results with different performances. For example a class should be mapped before enumeration and data type. Otherwise the mapping of a complete or disjoint generalization may be blocked and postponed until all subclasses are mapped into the owl2 ontology.

MapUMLClassDiagram()

```
Input: XMI Document
Output: OWL2 Ontology
Begin
    MappingClasses()
    MappingAssociations()
End
```

The algorithm has been divided into two parts. The first part **MappingClasses()** is a function that generates for each UML:class an OWL2:class and calls the sub-functions **MappingSubClassesOf()** and **MappingAttributesOf()** to respectively convert the inheritance constraint and the attributes.

MappingClasses ()

```
Input: Class C
Begin
 For each Class Ci in XMI Document loop
  If (isDataType(Ci) = false and isEnumeration(Ci) = false) then
    Apply rule 1: Create OWL2 Class Ci
    If (HasChild(Ci) = true) then
      MappingSubClassesOf(Ci)
    End If
    MappingAttributes(Ci)
  End If
 End loop
End
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MappingSubClassesOf(C)

```
Input: Class C
Begin
 ListP =ListParentOf(C)
 ListCh = ListChildrenOf(C)
 If (ListP.Length > 1)
   Apply rule 16: MappMultipleInheritance(C)
 Else If (ListP.Length ≤ 1) then
   For each Child Chi in ListCh loop
      Apply rule 12: Create Chi SubClassOf C
   End loop
   If (ListCh.Length > 1) then
     If (GetStereotype = "disjoint, complete") then
       Apply rule 15: Create DisjointUnionAxiom(C, ListCh)
      Else If (GetStereotype = "disjoint") then
       Apply rule 13: Create DsjointClassesAxiom(ListCh)
      Else If (GetStereotype = "complete") then
       Apply rule 14: Create EquivalentClassesAxiom(A, ObjectUnionOfAxiom(ListCh) )
      End If
    End If
  End If
End
```

MappingAttributes(C)

```
Input: Class C
Begin
 For each Attribute Ai in C loop
   If (isIdAttribute(Ai) =true) then
    Apply rule 7: Create  DataProperty Ai with HasKey axiom
   Else
     If (isEnumeration(Ai) = true) then
       Apply rule 10: Create DataProperty Ai with a range of elements using DataOneOf axiom
     Else If (isComplexeType(Ai) = true) then
       Apply rule 9 to map a complex type
     Else
      Apply rule 6 : DataProperty Ai
     End If
   End If
  End loop
End
```

The other part of the algorithm **Mapping-Relationships()**, converts the different types of relations that can exist between classes in the UML diagram. The relationships can be modeled in a number of different ways, depending on the association type (simple association, reflexive association, N-ary association, composition, dependency, class association).

MappingRelationships()

```
Input: Relation Rel
Begin
 For each Relation Rel_i in XMI Document
  If (ListOfRelatedRelations(Rel_i).Length > 2) then
    Apply rule 3 to map N-ary associations
  Else
    SouceRel = GetSouceClassOf(Rel_i)
    DestRel = GetDestClassOf(Rel_i)
    If (TypeOf(Rel_i) = Class) then
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
    Apply rule 4 to map Class Association
  Else If (TypeOf(Rel_i) = Composition) then
    Apply rule 19 to map composition
  Else If (TypeOf(Rel_i) = Dependency) then
    Apply rule 20 to map dependency
  End If
  Else
    If (SouceRel = DestRel) then
      Apply rule 5: Create ObjectProperty Rel_i and add ReflexiveObjectProperty axiom
    Else
      Apply rule 2 :Create ObjectProperty Rel_i
    End If
  End If
 End If
 If (HasChild(Rel_i) = true) then
   Apply rule 17: add SubPropertyOf axiom to map Generalization between associations
 End If
   Apply rule 11 to mapCardinality
 End loop
End
```

5. EXPERIMENTAL RESULTS

To evaluate the UML2OWL2 model a tool has been developed. This tool takes as input an XMI document that contains the overall elements of the source UML class diagram. Then it extracts these elements using DOM technology (Document Object Model) and applies our mapping algorithm to create the resulting OWL2 document. The tool is implemented using Java solutions mainly due to its platform-independent capabilities. DOM is employed for an easy and efficient reading, manipulation, and writing of an XMI document. The DOM parser allows a convenient method for accessing any piece of data in the XMI document and also preserves the order of elements.

As an example, Figure 20 shows a section of a UML class diagram developed using Rational Rose.

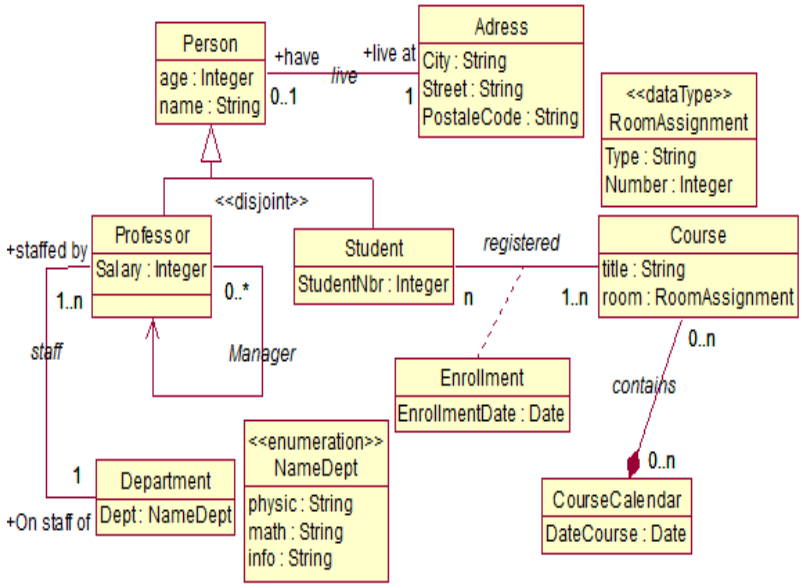


Figure 20. *Example of a Class Diagram*

Our implementation is based on the XMI version 1.1. The structure of the diagram is stored in XMI documents using Unisys Rose XML Tools (version 1.3). Part of the XMI document is shown in Figure 21. In this example the XMI element <UML:Class name="Adress"> matches with the class "Adress"

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

in the UML Class Diagram (Figure20), and the attribute "City" is represented in the <UML:Attrribute> element. Table III shows the UML elements extracted from Figure 20 and the corresponding XMI elements

```
<UML:Class xmi.id = 'S.026.1606.44.8'
  name = 'Adress' visibility = 'public' isSpecification = 'false'
  isRoot = 'true' isLeaf = 'true' isAbstract = 'false'
  isActive = 'false' >
  <UML:ModelElement.namespace>
    <UML:Namespace xml:link = 'simple' href = '..\Model.UMLDiagram.xml|G.0'/>
  </UML:ModelElement.namespace>
  <UML:Classifier.feature>
    <!-- ==================== ouss1::Adress.City    [Attribute] ================
    <UML:Attribute xmi.id = 'S.026.1606.44.9'
      name = 'City' visibility = 'private' isSpecification = 'false'
      ownerScope = 'instance'
      changeability = 'changeable' targetScope = 'instance' >
      <UML:StructuralFeature.multiplicity>
        <UML:Multiplicity >
          <UML:Multiplicity.range>
            <UML:MultiplicityRange xmi.id = 'id.0271606.13'
              lower = '1' upper = '1' />
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:StructuralFeature.multiplicity>
      <UML:Attribute.initialValue>
        <UML:Expression
          language = '' body = '' />
      </UML:Attribute.initialValue>
      <UML:StructuralFeature.type>
        <UML:Classifier xml:link = 'simple' href = '..\Model.UMLDiagram.xml|G.17'/>
      </UML:StructuralFeature.type>
    </UML:Attribute>
```

Figure 21. *Part of XMI document corresponding to the example of the class diagram of Figure 20*

TABLE II. *UML elements extracted from Figure 20 and the corresponding XMI elements*

UML elements	XMI elements
Class	<UML:Class>
Attribute	<UML:Attribute>
Association	<UML:Association>
Class Association	<UML:AssociationClass>
Multiplicity	<UML:Multiplicity>
Generalization	<UML:Generalization>
Enumeration	<UML:Stereotype name="enumeration"
DataType	<UML:DataType>
Composition	<UML:AssociationEnd aggregation = 'composite'>

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In the following, we provide an example of our platform conversion. Figures 22 and 23 respectively show the screenshot of UML2OWL2 tool and the OWL2 structure corresponding to the class diagram in Figure 20.

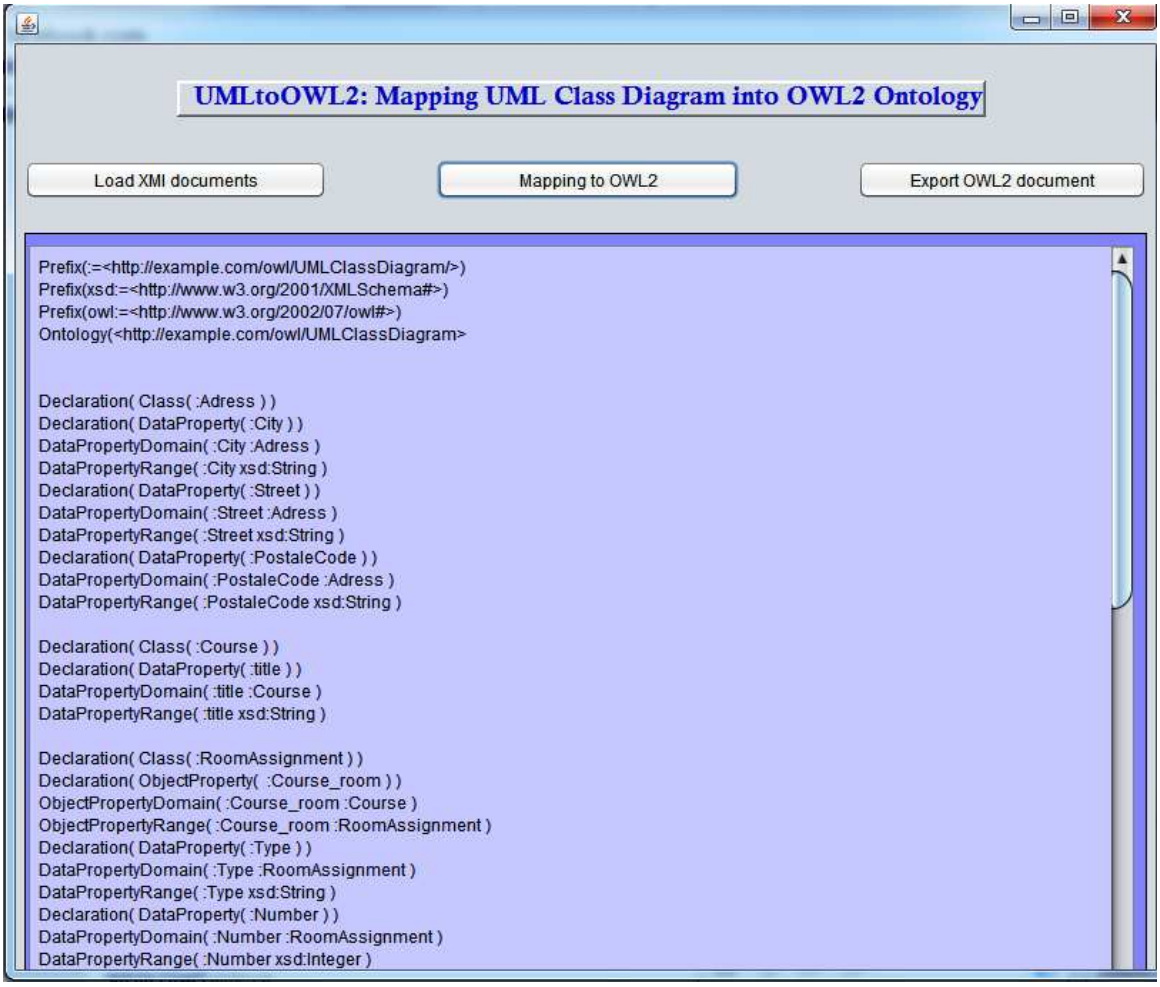


Figure 22. *Screenshot of UML2OWL2 tool*

The basic ontology graph structure of our example (Figure 20) is as follow:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
Prefix(:=<http://example.com/owl/UMLClassDiagram/>
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Ontology(<http://example.com/owl/UMLClassDiagram>

Declaration( Class( :Adress ) )
Declaration( DataProperty( :City ) )
DataPropertyDomain( :City :Adress )
DataPropertyRange( :City xsd:String )
Declaration( DataProperty( :Street ) )
DataPropertyDomain( :Street :Adress )
DataPropertyRange( :Street xsd:String )
Declaration( DataProperty( :PostaleCode ) )
DataPropertyDomain( :PostaleCode :Adress )
DataPropertyRange( :PostaleCode xsd:String )

Declaration( Class( :Course ) )
Declaration( DataProperty( :title ) )
DataPropertyDomain( :title :Course )
DataPropertyRange( :title xsd:String )

Declaration( Class( :RoomAssignment ) )
Declaration( ObjectProperty( :Course_room ) )
ObjectPropertyDomain( :Course_room :Course )
ObjectPropertyRange( :Course_room :RoomAssignmen
Declaration( DataProperty( :Type ) )
DataPropertyDomain( :Type :RoomAssignment )
DataPropertyRange( :Type xsd:String )
Declaration( DataProperty( :Number ) )
DataPropertyDomain( :Number :RoomAssignment )
DataPropertyRange( :Number xsd:Integer )

Declaration( Class( :CourseCalendar ) )
Declaration( DataProperty( :DateCourse ) )
DataPropertyDomain( :DateCourse :CourseCalendar )
DataPropertyRange( :DateCourse xsd:Date )

Declaration( Class( :Department ) )
Declaration( DataProperty( :Dept ) )
DataPropertyDomain( :Dept :Department )
DataPropertyRange( :Dept :NameDept )
DatatypeDefinition(
 :NameDept
 DataOneOf( "physic"^^xsd:String "math"^^xsd:String
            "info"^^xsd:String ) )

Declaration( Class( :Person ) )
Declaration( DataProperty( :age ) )
DataPropertyDomain( :age :Person )
DataPropertyRange( :age xsd:Integer )
Declaration( DataProperty( :name ) )
DataPropertyDomain( :name :Person )
DataPropertyRange( :name xsd:String )

Declaration( Class( :Professor ) )
Declaration( DataProperty( :Salary ) )
DataPropertyDomain( :Salary :Professor )
DataPropertyRange( :Salary xsd:Integer )

Declaration( Class( :Student ) )
Declaration( DataProperty( :StudentNbr ) )
DataPropertyDomain( :StudentNbr :Student )
DataPropertyRange( :StudentNbr xsd:Integer )
HasKey( :Student () ( :StudentNbr ) )

SubClassOf( :Professor :Person )
SubClassOf( :Student :Person )
DisjointClasses( :Professor :Student )

Declaration( ObjectProperty( :Enrollment_Course ) )
ObjectPropertyDomain( :Enrollment_Course :Enrollment )
ObjectPropertyRange( :Enrollment_Course :Course )
Declaration( ObjectProperty( :Course_Enrollment ) )
ObjectPropertyDomain( :Course_Enrollment :Course )
ObjectPropertyRange( :Course_Enrollment :Enrollment )
InverseObjectProperties( :Enrollment_Course
                         :Course_Enrollment )

Declaration( ObjectProperty( :Enrollment_Student ) )
ObjectPropertyDomain( :Enrollment_Student :Enrollment )
ObjectPropertyRange( :Enrollment_Student :Student )
Declaration( ObjectProperty( :Student_Enrollment ) )
ObjectPropertyDomain( :Student_Enrollment :Student )
ObjectPropertyRange( :Student_Enrollment :Enrollment )
InverseObjectProperties( :Enrollment_Student
                         :Student_Enrollment )

SubObjectPropertyOf(
   ObjectPropertyChain( :Course_Enrollment
                        :Enrollment_Student )
   :Course_Student
)
SubObjectPropertyOf(
   ObjectPropertyChain( :Student_Enrollment
                        :Enrollment_Course )
   :Student_Course
)

Declaration( DataProperty( :EnrollmentDate ) )
DataPropertyDomain( :EnrollmentDate :Enrollment )
DataPropertyRange( :EnrollmentDate xsd:Date )

Declaration( ObjectProperty( :live_at ) )
ObjectPropertyDomain( :live_at :Person )
ObjectPropertyRange( :live_at :Adress )

Declaration( ObjectProperty( :have ) )
ObjectPropertyDomain( :have :Adress )
ObjectPropertyRange( :have :Person )
InverseObjectProperties( :live_at :have )
SubClassOf(
   :Person
   ObjectExactCardinality( 1 :live_at :Adress )
   )
SubClassOf(
   :Person
   ObjectMaxCardinality( 1 :have :Adress ) )
Declaration( ObjectProperty( :Manager ) )
ObjectPropertyDomain( :Manager :Professor )
ObjectPropertyRange( :Manager :Professor )
Declaration( ObjectProperty( :ManagerBy ) )
ObjectPropertyDomain( :ManagerBy :Professor )
ObjectPropertyRange( :ManagerBy :Professor )
InverseObjectProperties( :Manager :ManagerBy )
ReflexiveObjectProperty( :Manager )
ReflexiveObjectProperty( :ManagerBy )

Declaration( ObjectProperty( :contains ) )
ObjectPropertyDomain( :contains :Course )
ObjectPropertyRange( :contains :CourseCalendar )
Declaration( ObjectProperty( :containsBy ) )
ObjectPropertyDomain( :containsBy :CourseCalendar )
ObjectPropertyRange( :containsBy :Course )
InverseObjectProperties( :contains :containsBy )

Declaration( ObjectProperty( :staffed_by ) )
ObjectPropertyDomain( :staffed_by :Department )
ObjectPropertyRange( :staffed_by :Professor )
Declaration( ObjectProperty( :On_staff_of ) )
ObjectPropertyDomain( :On_staff_of :Professor )
ObjectPropertyRange( :On_staff_of :Department )
InverseObjectProperties( :staffed_by :On_staff_of )
SubClassOf(
   :Department
   ObjectMinCardinality( 1 :staffed_by :Professor )
   )
SubClassOf(
   :Department
   ObjectExactCardinality( 1 :On_staff_of :Professor )
   )
)
```

Figure 23. *The generated OWL2 ontology from the UML class diagram in Figure 20*

To test the semantic consistency of our result ontology we loaded it in the Protégé OWL editor. The figure below (Figure 24) obtained using the plugin OntoGraf protégé shows the hierarchy of the classes.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

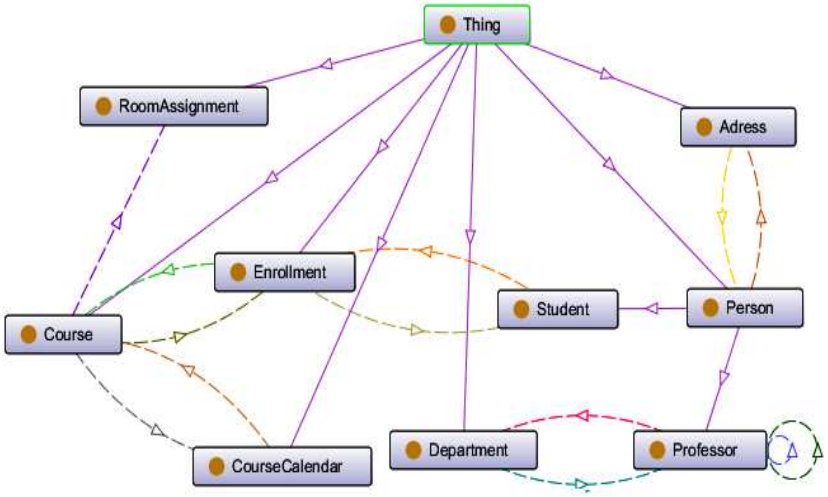


Figure 24. *Ontograf Diagram of the result Ontology*

We have compared our method to some of the existing approaches. The following table summarizes all mentioned rules and the approaches that have considered them.

TABLE III. UML TO OWL MAPPING COMPARISON METHODS

Constraints	**[1]**	**[3]**	**[5]**	**[10]**	**[11]**	**[14]**	**[17]**	**[18]**	**Our approach**
Ontology preparation	✓	×	×	×	✓	×	×	✓	✓
Classes	✓	✓	×	✓	✓	✓	✓	✓	✓
Simple association	✓	✓	×	✓	✓	✓	✓	✓	✓
N-ary association	×	×	×	×	×	✓	×	×	✓
Class association	✓	×	×	×	×	×	✓	×	✓
Reflexive association	×	×	×	×	×	×	×	×	✓
Simple attribute	✓	✓	×	✓	✓	✓	✓	✓	✓
id attribute	×	×	×	×	✓	×	×	×	✓
Primitive type	✓	✓	✓	✓	✓	×	✓	✓	✓
Complex type	×	×	✓	×	×	×	×	✓	✓
Enumeration	×	✓	✓	×	×	×	×	✓	✓
Multiplicity	✓	✓	×	✓	×	✓	✓	×	✓
Generalization without constraints	✓	✓	×	×	✓	×	×	×	✓
Generalization Disjoint	×	✓	×	×	×	×	×	×	✓
Generalization Complete	×	×	×	×	×	×	×	×	✓
Generalization Disjoint&Complete	×	✓	×	×	×	×	×	×	✓
Multiple inheritance	×	×	×	×	×	×	×	×	✓
Generalization between association	×	✓	×	×	×	×	×	×	✓
Generalization between data types	×	×	✓	×	×	×	×	×	✓
Composition	×	✓	×	✓	✓	×	×	×	✓
Dependency	×	×	×	×	×	×	×	×	✓

In this table we have identified commonalities and differences between existing mapping techniques and our mapping approach. With regards to our evaluation, none of the existing transformation tools satisfies requirements of transforming UML class diagrams into OWL. Table III shows that these transformation approaches do not provide a complete solution to the problematic. Contrary to these existing solutions our developed UML2OWL2 approach achieves a complete migration of UML class diagrams into OWL. Our approach does this conversion in an automatic way,

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

captures richer knowledge of common UML constructs and constraints and uses OWL2 as the target ontology language. Our approach utilizes the maximal intersection of UML features and OWL features.

6. CONCLUSION AND PERSPECTIVES

In this paper, a systematic approach UML2OWL2 for an automatic transformation of conceptual models between UML class diagrams and OWL2 is proposed. We especially gave a thorough analysis and comparison of existing mapping methods and identified their weaknesses and limitations. As a result we gave a complete list of elements that are crucial for the conversion and a complete list of associated mapping rules.

One of the reasons to act so is the importance of internationally standardized UML class diagrams as a powerful description tool. They are indeed widely used to tackle complexities of systems of the real world to be conceptualized and to abstract from any implementation platform. They offer a mean for simple conceptual models that reveal ideas of internal structure and behavior of the systems to be modeled by using a variety of constructs. Another reason is the importance of ontologies to the Semantic Web that has led to the development of the ontology languages RDf and OWL and associated supporting tools.

With this in mind we recommend that research goes a further step in encouraging the interaction between UML world and the semantic one. In this sense and with our UML2OWL2 algorithm our objective is principally to fill the gap in the existing mapping approaches from UML class diagrams into OWL that is related to the non consideration by these approaches of many of UML relevant constructs and give a formalization of the steps involved in the design of starting from and considering various elements in a UML class diagram.

Though some dissimilarity between structural elements of UML and OWL we were able to give concise rules for the mapping process and accordingly build an associated mapping algorithm Our implementation tool and case study show that the proposed approach is effective. The tool is fully automatic and allows obtaining schemas meaningful for developers of semantic applications.

Compared to the existing approach, our new solution optimizes constraints extraction, and supports all of the most common UML elements such as disjoint UML class annotations, attribute datatypes other than primitives and all type of associations. Thanks to OWL 2 the rules are also refined to be more expressive and less complicated using more expressive constructs (e.g., hasKey, ReflexiveObjectProperty, exactcardinality, DisjointClass, DisjointUinion, intersectionOf …). OWL2 also simplifies many programmatic tasks associated with ontologies, including ontology querying and processing. In addition OWL2 can be used to construct full applications that have dependencies on complex ontologies.

A limitation of our mapping approach is that it does not treat the mapping at the data-level yet. For our future research related to this topic the focus will be at this "data"-level in order to convert a UML object diagram into the instances part of ontology (ABOX) with all assertions of the different elements from the schema level.

ACKNOLWDGEMENT

This paper is not within the framework of any funded research project

REFERENCES

[1] A. BELGHIAT, M. BOURAHLA, "Transformation of UML Models towards OWL Ontologies", 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), 2012, pp. 840-846.

[2] D. Gasevic, D. Djuric, V. Devedzic, V. Damjanovi “Converting UML to OWL ontologies”. In Proceedings of the 13 th International World Wide Web Conference, NY, USA, pp. 488-489. 2004

[3] J. Zedlitz, J. Jorke, N. Luttenberger, "From UML to OWL2", Knowledge Technology: Third Knowledge Technology Week, KTW 2011, Kajang, Malaysia, July 18-22, 2011

[4] J. Zedlitz, N. Luttenberger, " Conceptual Modelling in UML and OWL-2", International Journal on Advances in Software, vol 7 no 1 & 2, year 2014

[5] J. Zedlitz, N. Luttenberger, " Data Types in UML and OWL-2", SEMAPRO 2013 : The Seventh International Conference on Advances in Semantic Processing

[6] K. Kiko, C. Atkinson, "A Detailed Comparison of UML and OWL", 2008 , technical report

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[7] L. Alaoui, O. EL Hajjamy, and M. Bahaj, "RDB2OWL2: Schema and Data Conversion from RDB into OWL2," International Journal of Engineering Research & Technology (IJERT), vol. 3, Issue. 11, November 2014

[8] L. Alaoui, O. EL Hajjamy, and M. Bahaj, "Automatic Mapping of Relational Databases to OWL Antology," Int. J. Engineering & Research Technology IJERT, Vol. 3, Issue 4 (April, 2014)

[9] L. Stojanovic, N. Stojanovic, R. Volz, "Migrating data-intensive web sites into the Semantic Web", In Proceedings of the 2002 ACM symposium on Applied computing (SAC '02), pp.1100-1107, ACM, 2002

[10] M. Ahlonsou, E. Blanchard, H. Briand, F. Guillet, "Transformation des concepts du diagramme de classe UML en OWL full", AEGC 2005, vol. RNTI-E-5, pp.13-18

[11] M. Bahaj, J. Bakkas, "Automatic Conversion Method of Class Diagrams to Ontologies Maintaining Their Semantic Features", International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6, January 2013

[12] M. K. Smith, C. Welty, D. L. McGuinness, OWL Web Ontology Language Guide (W3C Recommendation 10 February 2004) [EB/OL]. http://www.w3.org/TR/owl-features/, (last modified on 10 February 2004)

[13] M. Schneider, S. Rudolph2, G. Rudolph, "Modeling in OWL 2 without Restrictions arXiv: 1212.2902 v3 [cs.AI] 28 Apr 2013

[14] N. Gherabi, M. Bahaj, "A New Method for Mapping UML Class into OWL Ontology", Special Issue of International Journal of Computer Applications (0975 – 8887) on Software Engineering, Databases and Expert Systems – SEDEXS, September 2012

[15] OMG, "Unified Modeling Language, Superstructure Version 2.4," 2011, http://www.omg.org/spec/UML/2.4/Superstructure.

[16] S. Brockmans, R. M. Colomb, P. Haase, E. F. Kendall, E. K. Wallace, C. Welty, G. T. Xie, "A Model Driven Approach for Building OWL DL and OWL Full Ontologies", 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006. Proceedings, pp 187-200, 2006

[17] S. Cranefield, "UML and the semantic web", the first semantic web working Symposium, pp.113-130. Stanford University, California (2001).

[18] S.Tschirner, A.Scherp, S.Staab, "Semantic access to INSPIRE". Terra Cognita Workshop (2011)

[19] W.Xu, A. Dilo, "Modelling emergency response processes: Comparative study on OWL and UML", Proceedings of the Joint ISCRAM-CHINA and GI4DM Conference, Harbin, China, August 2008

[20] W3C, OWL Working Group,, "Web Ontology Language (OWL)" http://www.w3.org/2004/OWL, 2004

[21] W3C, OWL Working Group, "OWL 2 Web ontology language document overview. W3C Recommendation 27 October 2009,"http://www.w3.org/TR/owl2-overview/

[22] W3C, OWL Working Group, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. W3C Recommendation 11 December 2012," http://www.w3.org/TR/owl2-syntax/