CHAPTER 5

UML MODELS TO OWL CONVERSION AND DETECTING INCONSISTENCY USING DL RULES

This chapter deals with UML model conversion and describes formal methods and the need to transform these models. It also detects the inconsistencies in these models and discusses about the algorithm proposed and the implementation of it.

5.1 INTRODUCTION

In the Modeling level, Class diagram plays important role among all UML diagrams. Both UML and OWL are based on the classes, objects and relationship thus conversions are possible. OWL is the knowledge representation language and helps in creating ontological model. It assists the machine to automatically process and integrate the information. OWL can be generated during the modelling phase by using the UML class diagram because similarities exists between the two languages and Enterprise models have their schemas in UML. So by reusing the existing schema OWL can be generated.

The challenges in UML conversion are problems like inconsistency, finite satisfiability, redundancy and incomplete design gets reflected in the resultant OWL on conversion. Ragnhild (2011), Tom Mens et al (2009) also describe about detecting inconsistencies. So an approach is
proposed for finding and resolving the inconsistency during the conversion process and OWL is generated.

The Class diagram may be modeled incorrectly also stated by Jocelyn Paola (2003), Mira Balaban et al (2009) and these will be ported to OWL also. This leads to inconsistency problems. One of the ways to remove inconsistency while creating OWL from UML is by using Description logics. It helps to determine the inconsistent information in the UML models and also remove the redundancy.

5.2 UML MODEL CONVERSION SYSTEM

The proposed UML model conversion systemis shown in Figure 5.1. Initially UML models are exported to XMI file. This XMI file is converted into Description logic statements by applying description logic transformation rules.

![Figure 5.1 UML Model Conversion Architecture](image)
Reasoning properties like satisfiability, equivalentness, disjointness and subsumption are applied to the obtained Description logic statements to find the inconsistent statements. The proposed algorithm is applied for solving the inconsistency. When the statements become satisfiable for the specific domain, the DL statements are converted to OWL and stored in the OWL Repository at the server level. The system has section like class diagram to DL statements conversion, checking the reasoning properties for consistency of the statements, resolving the inconsistency and generating a satisfied OWL description. The steps are as follows:

Step 1 : Generate XMI from UML class diagram
Step 2 : Extract class elements from XMI
Step 3 : Apply transformation rules to the class elements and get the DL statements.
Step 4 : Check the consistency by applying reasoning properties.
Step 5 : If Consistent then the DL statements are mapped to OWL statements by using DL toOWL mapping rule.
Step 6 : If inconsistent then use the solving technique as given in the Inconsistency resolving algorithm and goto step 4.

5.3 RESOLVING INCONSISTENCY IN UML

One of the inconsistency triggers is same name in the sibling classes of generalization as in Figure 5.2. In this case remove one of the classes with same name. The next inconsistency trigger is attribute value type error, which occur when two association appear with same or two attributes in two classes have same attribute name with different types a:T1 of class1, a:T2 of class2 which is represented in Figure 5.3. Here collect the classes with
same attribute name and if attribute is available in the library use its type or change type by using maximum occurrences of that attribute. The other inconsistency trigger is multiplicity constraint representation [i..j] where j<i as in Figure 5.4. In this case, interchange i and j so that multiplicity becomes [j…i]. The next inconsistency trigger is cyclic inheritance as represented in Figure 5.5. To remove this Cycle, find the set of classes in the loop and its corresponding association. If the association type is generalization without Disjointness then switch the direction of Generalization or remove the association.

Figure 5.2 Same Class Names in Generalization

Figure 5.3 Attribute Type Error

Figure 5.4 Multiplicity Constraint
The algorithm for resolving inconsistency in the Class diagram is described below.

Algorithm : Resolving Inconsistency

Input : Set of DL statements

Output : Set of consistent DL statements

Begin

For D ε(DL statements)

if (D contain class C)

C1=set of super class or Derived class C
C2= disjoint class of C1

if C1==C2 then //class with same name in generalization

Remove D with class C2

End if

Else if (D contain attribute ‘a’)

For D1 ε (DL statement D)

List=attribute Type(D1) // Collect similar attributes in a list

End for

If(list with different Type) then

If(a in Library(L)) then a.Type = L.a.Type

Else a.Type=Maxoccur.Type(list)

End

Else if (D contain multiplicity)
If (i>j) { ≤ i,j} then (interchange i,j) End
Else
if (D contain association)
//adjusting the association in the cyclic inheritance
For each D1 in (DL statement D)
    D2=Dependent DL of D1
    Apply subsumption property between the D2
    If inconsistent then
        S1=association of the classes in the cyclic inheritance
        For each S in (S1)
            If (association. Type==generalization&&
                association.Type!=disjoint)
                //Switch the direction of generalization
                Break;
            End if
        End for
        Remove the D and continue
    End if
End for
Else
do nothing
End for
end for

5.4 IMPLEMENTATION OF UML CONVERSION SYSTEM

The proposed system is implemented in Eclipse integrated development environment (IDE) with Galileo and Protégé. Ontology is designed using Protégé and the pellet reasoner in it is used to detect the inconsistency.

Cyclic Inheritance in a class diagram is identified and resolved. The system is tested using both consistent and inconsistent class diagrams and checked whether the inconsistency is captured. The sample test data considered for this system is tabulated in Appendix 1. Initially an inconsistent
class diagram is modeled by setting OWL properties using Profiles and stereotypes like OWL, RDF, owlOntology as in Figure 5.6 and is then converted to OWL RDF/XML format using Twouse toolkit. Figure 5.7 shows the OWL code for Inconsistent Class diagram.

Figure 5.6 Inconsistent Class Diagram

The corresponding DL statements for the Class Diagram shown in Figure 5.6 are given below:

a) instructor ⊆ department
b) Student ⊆ department
c) Department ⊆ college
d) college ⊆ course
e) course ⊆ student
f) student ⊆ ¬ instructor
The obtained OWL is of W3C standard. The inconsistency in the resultant OWL is detected by importing the OWL to protégé and checking the inconsistency using pellet reasoner which is in Figure 5.8. The obtained inconsistency is resolved by using the techniques given in the algorithm. The Cyclic Inheritance inconsistency is resolved by finding the set of classes which causes the loop and changing the direction of any one of the generalization in the loop. If loop contains disjoint property then remove the association between the two classes.

For example, the inconsistent class is “instructor” and since “department” is subset of “instructor”, the inconsistent class may be “instructor or department”. But the cyclic inheritance occurred with respect to class “department”. Therefore, set of classes in the cycle is found using “department” class.
Then generalization without disjointness is found.

\[
\{\text{department, college, course, student}\}
\]

Hence, change in the direction of generalization will solve the inconsistency. So, the link “college to department” is changed. Then the OWL code is generated and again reasoned to found that it is consistent which is shown in Figure 5.9

![Inconsistency Detection](image)

Figure 5.8 Inconsistency Detection
SUMMARY

UML models can be used to generate OWL because they have related properties between them. The generated OWL can be annotated and linked with web services. During conversion of class diagram to OWL, problems like inconsistency and redundancy occurs which is reflected in OWL. So the proposed algorithm detects and resolves inconsistency that happens in converting UML to OWL.

The proposed system detects and eliminates the cycle in the model without considering the dependency of the class. This may lead to instability in the model. So, instead of removing the link directly the dependencies between classes are to be considered and the algorithm needs to be modified accordingly.
This chapter discussed the various inconsistencies in the class diagram and implementation of inconsistencies detection. The following chapter describes about the other constraints, identifies dependencies between classes and illustrates the modified algorithm.