CHAPTER 6

RESOLVING INCONSISTENCIES IN UML MODELS

This chapter describes about proposed consistency checker and inconsistency resolver and discuss about the modified algorithm. The chapter also discusses about the implementation and the experimental results obtained for various domains.

6.1 INTRODUCTION

Ontology contains set of axioms with constraints on the each individual class. The class, attribute and relationships in the UML class diagram have strong defined meaning. These inconsistencies are detected using reasoning properties of Description logic and resolved using ranking approach which is based on arity and impact factors as defined by Aditya Kalyanpur (2006). The axioms will be modified based on the above factors. This resolving process is repeated until all the DL statements obtained are consistent for the entire Schema. When the consistent set is obtained, it is then mapped to OWL which results in Consistent OWL.

6.2 MODIFIED ALGORITHM TO RESOLVE INCONSISTENCY

The inconsistency is detected and resolved in the algorithm proposed in the previous chapter does not consider about the dependencies with other classes and deleting an axiom abruptly may lead to loss in vital data or property. Hence a modified inconsistency resolving algorithm is proposed which considers the dependency loss also. The steps involved in the
algorithm are the UML class diagram is converted to XMI format and is parsed to obtain the class elements. Then the transformation rules are applied to the class elements and corresponding axioms are obtained. The proposed algorithm is given below.

**Algorithm for Detecting and Resolving the inconsistency**

**Input** : set of axioms

**Output** : erroneous path and its explanation / consistent information with set of Axioms

Begin

For D \(\in\) Set of (Axioms)

If (D contains class C)

\[ C_1 = \text{set of super class or Derived class C} \]

\[ C_2 = \text{disjoint class of} \ C_1 \]

//class with same name in generalization

If \( C = C_2 \) then

Remove D with class C2

End if

Else if (D contains attribute 'a')

For D1 \(\in\) (Axiom list)

//collect axioms with similar attribute

If (D1.attribute_Type.equals (D.attribute_Type)) List=D1

End

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

End Else if

Else if (D contain multiplicity)

If (i>j) \{i, \geq j\} then (interchange i, j)

End else if

Else if (D contain association)

**ARC (erroneous path, unsatisfiable classes)**

End Else if

Else

Do nothing

End

End for

End
ARC (Axiom Rank Computing) Algorithm

**Input** : Erroneous axiom set for unsatisfiable classes
**Output** : Consistent axiom set

Function ARC (Erroneous axiom, unsatisfiable classes)

Begin
For $c \in$ set of (unsatisfiable classes)
For $D \in$ set of (Erroneous path)
   If (D contains Associations)
      Arity = No. of unsatisfiable class with ‘D’
      Impact = other dependent Axioms in both Domain set and
      Derived Set.
      Rank = Arity + Impact.
      //Delete the axiom with minimum rank
      End if
   End for
End for
End

From the obtained set of axioms dummy ontology is created and is reasoned using the DL properties. If consistent, the axioms are mapped to OWL statements. If the error is due to the redundant Axioms i.e. duplicate class in the generalizations, it is resolved by removing one of the duplicates. If attribute type error i.e., two attributes of two different classes have different type then type of the attribute is changed based on library or maximum occurrence. The multiplicity inconsistency is resolved by interchanging the orders. If cyclic inheritance problem occurs, it can be resolved by removing an axiom with minimum rank or random rank. The dependency loss is less in random rank axiom deletion compared to minimum rank.

The above mentioned ARC algorithm detects multiple inheritances, attribute type error and cardinality constraints. It calculates the rank for each axiom in the erroneous path and the loss of dependency is calculated. This is
done by finding number of occurrences of each axiom and the number of axioms that are affected if a particular axiom is removed. The axioms can be deleted using minimum rank or random ranking.

The cyclic inheritance problem is resolved by considering the dependency between classes and removing an axiom with minimum rank or random rank.

The ARC algorithm calculates rank for each axiom in the erroneous path and the loss of dependency is evaluated. This is done by finding number of occurrences of each axiom and the number of axioms that are affected if a particular axiom is removed. The axioms can be deleted either using minimum rank or random ranking. The comparison between these two methods is done and the results are compared.

6.3 IMPLEMENTATION OF THE SYSTEM

The XMI is parsed by XMI_Parser and the class elements are extracted. The extracted class elements are checked for consistency by Pellet reasoner. The resolving algorithm gets the inconsistency error and resolves accordingly. This is done for each class element. If consistent, the class elements are mapped to OWL. The input XMI file is parsed by XMI_Parser and generalization relationships and its corresponding elements from the XMI file and DL axioms for domain, range, data type property and cardinality of the attribute from the class elements are collected.

The parsed elements in “Parser1” object are sent to consistency checker. The pellet consistency checker is imported and explanation for inconsistency statements is obtained by initializing the PelletReasoner, calling the consistency checker by using inconsistency() function and find the explanation for the unsatisfiable classes by getUnsatisfiableClasses() and getExplanation(). By using the obtained explanation the corresponding
resolving technique is called. Figures 6.1 and 6.2 show the code snippet for collecting axioms for the class elements and for generalization relationships and attributes respectively.

The cyclic generalization is resolved by applying the erroneous path to the ARC algorithm. The ranking for each axiom is obtained by the total of arity and impact. Arity is number of occurrence of erroneous axioms in the inconsistent class.

```java
void insertGeneralizations()
{
    OWLClass superClass = owlDataFactory_.getOWLClass(childEntry.getKey().getName());
    OWLClass subClass = owlDataFactory_.getOWLClass((childClass.getName());
    OWLAxiom myHierarchy =
        owlDataFactory_.getOWLSubClassOfAxiom(subClass, superClass);
    axiomList_.add(myHierarchy); //Generating a list of child complement's classes
    OWLObjectComplementOf negatedClass
        =owlDataFactory_.getOWLObjectComplementOf(subClass);
    negatedClassSet.add(negatedClass);   //Generating a list of child classes
    subClasses.add(subClass);
}

if (negatedClassSet.size() > 1)
{
    //Disjointness constraint
    OWLDisjointClassesAxiom dijointClassExpression =
        owlDataFactory_.getOWLDisjointClassesAxiom(subClasses);
    axiomList_.add(dijointClassExpression);
}

if (subClasses.size() > 1)
{
    //Covering constraint
    OWLObjectUnionOf unionOfSubClasses =
        owlDataFactory_.getOWLObjectUnionOf(subClasses);
    OWLAxiom unionClassAxiom =
        owlDataFactory_.getOWLSubClassOfAxiom(superClass, unionOfSubClasses);
    axiomList_.add(unionClassAxiom);
}
```
Figure 6.2 Code for Creating Axiom of Attributes and Cardinality

The unsatisfiable classes were found by counting the size of the OWLNothing. By using the collected unsatisfiable classes and its erroneous path the arity if found. Impact is number of axioms affected on removing this particular axiom and Figure 6.3 shows the code for finding impact of the axiom. The rank which is the total of arity and impact is found. The minimum rank axiom in the erroneous path is deleted and sent again to the consistency checker. When the axioms are consistent, Mapping to OWL is done.

The system was tested using both consistent and inconsistent class diagrams and checked whether the inconsistency was captured and resolved. Initially an inconsistent class diagram with cyclic inheritance is modeled by
using ArgoUML. The exported XMI of the class diagram is given as input to
the code for detecting and resolving the inconsistency. The input XMI is
transformed to the Set of axioms by passing to the XMIParser.

```java
If (axiom instanceof OWLSubClassAxiom)
{
    OWLSubClassAxiom subAx = (OWLSubClassAxiom) axiom;
    if (subAx.getSubClass() instanceof OWLClass && subAx.getSuperClass() instanceof OWLClass) {
        OWLClass sub = (OWLClass) subAx.getSubClass();
        OWLClass sup = (OWLClass) subAx.getSuperClass();
        for (Iterator it = SetUtils.union(pellet.descendantClassesOf(sub)).iterator();
            it.hasNext();)
        {
            OWLDescription s1 = (OWLDescription) it.next();
            for (Iterator it2 = SetUtils.union(pellet.ancestorClassesOf(sup)).iterator();
                it2.hasNext();)
            {
                OWLDescription s2 = (OWLDescription) it2.next();
                if (!s2.equals(ontology.getOWLDataFactory().getOWLThing()) &&
                    !s1.equals(ontology.getOWLDataFactory().getOWLNothing()))
                    OWLSubClassAxiom ax =
                        pellet.getOntology().getOWLDataFactory().getOWLSubClassAxiom(s1, s2);
                    this.addAxiom(axiomSOSMap, axiom, ax);
                }
            }
        }
    } else if (axiom instanceof OWLDisjointClassesAxiom) {
        Set dis = ((OWLDisjointClassesAxiom) axiom).getDisjointClasses();
        for (Iterator d = dis.iterator(); d.hasNext();)
        {
            OWLDescription desc = (OWLDescription) d.next();
            OWLDisjointAxiom dx =
                pellet.getOntology().getOWLDataFactory().getOWLDisjointClassAxiom(d, desc);
            this.addAxiom(axiomSOSMap, dis, dx);
            impact=((HashSet) this.axiomSOSMap.get(axiom)).size();
        }
    }
}

Figure 6.3 Code Snippets for Finding the Impact of the Axiom
The Other inconsistency that occurs in class diagram is attribute type error. This error arise if two different class has same attribute name but different type which is shown in Figure 6.4. An object is created for the attribute in protégé to find the axioms that causes inference.

![Class Diagram Example](image)

**Figure 6.4 Attribute Type Error**

### 6.4 RESULTS AND DISCUSSION

The system is tested for eight different domain with about 5-10 classes in each domain with consistent and inconsistent class diagrams. For each class diagram, the number of classes and total no of axioms is calculated. Then the number of inconsistent classes, type of inconsistency occurred and erroneous path are found using the pellet reasoner. Based on the explanation for the inconsistency and the erroneous path, the appropriate resolving technique is used.

#### 6.4.1 Minimum axiom Vs Random Axiom

The inconsistency for multiple inheritance inconsistency can be resolved either by removing the random rank axiom or by removing the
minimum rank axiom. In the erroneous path, if the axiom with random rank (Example: College ⊆ course) is deleted then the dependency will be reduced in large scale which is found using the Impact. Figure 6.5 shows the comparison between minimum and random ranking.

**Table 6.1 Comparison of Impact before and after Removing Axiom**

<table>
<thead>
<tr>
<th>S.No</th>
<th>Axiom</th>
<th>Impact of axioms before removing inconsistency</th>
<th>Impact after removing inconsistency by choosing minimum rank Axiom</th>
<th>Impact after removing inconsistency by choosing Random rank Axiom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Department ⊆ college</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>College ⊆ course</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Course ⊆ student</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Student ⊆ department</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Instructor ⊆ department</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Student ⊆ ¬ instructor</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 6.5 Comparison between Minimum and Random Ranking**
But if the axiom with minimum rank is deleted (Example: Course \( \subseteq \) student) then the dependency will not be affected much when compared to the random rank axiom deletion. By analyzing the calculated Impacts of Minimum rank axiom deletion and random rank axiom deletion, it is shown that impact loss is reduced on choosing random Axiom for removing the inconsistency.

### 6.4.2 Loss of Dependency

The dependency in the system is affected when inconsistency is resolved by ranking approach. The percentage loss of dependency is found by

\[
\text{% loss of dependency} = \frac{\text{Impact before removing inconsistency} - \text{Impact after removing inconsistency}}{\text{Impact before removing inconsistency}}
\]

The Table 6.2 shows testing of Impacts and its Dependency loss for various domains. It describes the percentage loss of dependency calculated using minimum rank axiom deletion approach for various domain class diagrams, with number of inconsistent classes in the system, type of inconsistencies detected and not detected in the system, resolved inconsistencies, the impact level before and after removing inconsistency.

The dependency loss is high in maximum rank axiom compared to random rank axiom. A value is created to the attribute using the resultant OWL to detect attribute type error because no prerequisite in the class diagrams to create common values. The gradual or no loss of data or property is shown in Figure 6.6.
### Table 6.2 Testing of Impacts and its Dependency loss for Various Domains

<table>
<thead>
<tr>
<th>Domain Class Diagram</th>
<th>No. of classes</th>
<th>No. of Generated axioms</th>
<th>No. of inconsistent classes</th>
<th>Detected Inconsistency</th>
<th>Not Detected Inconsistency</th>
<th>Inconsistency Resolved</th>
<th>Impact Before Removing Inconsistency</th>
<th>Impact After Removing Inconsistency</th>
<th>% Loss of Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIC</td>
<td>8</td>
<td>45</td>
<td>2</td>
<td>Cyclic Inheritance, Data type error</td>
<td>Data type error</td>
<td>Cyclic inheritance</td>
<td>17</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>Catalyst reaction</td>
<td>6</td>
<td>12</td>
<td>1</td>
<td>Multiple Inheritance</td>
<td>-</td>
<td>Multiple inheritance</td>
<td>19</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>College Management</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>Cyclic inheritance</td>
<td>Cyclic Inheritance</td>
<td>15</td>
<td>11</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Hospital Management</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>Multipliedy Constrain</td>
<td>Multiplicity Constrain</td>
<td>19</td>
<td>19</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Online polling system</td>
<td>10</td>
<td>29</td>
<td>4</td>
<td>Multiple Inheritance</td>
<td>Multiple Inheritance</td>
<td>40</td>
<td>37</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>ATM</td>
<td>8</td>
<td>44</td>
<td>2</td>
<td>Cardinality constrain</td>
<td>-</td>
<td>Cardinality Constrain</td>
<td>14</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Parent-child</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>Data type error</td>
<td>-</td>
<td>-</td>
<td>j*</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
The system is tested for various domains and the dependency loss if particular axiom is deleted is charted. Moreover it is analysed that all multiplicity inconsistency which are detected are not solved because the axiom which has more dependency or impact if deleted may result in loss of captured data in the UML model.

![Figure 6.6 Comparison of Impacts and its Analysis](image)

**Figure 6.6 Comparison of Impacts and its Analysis**

From the testing results, the proposed system detects multiple inheritance, attribute type error and cardinality constraints. It resolves multiple inheritances using minimum rank axiom method. The attribute type error is detected only when the value for the attribute is created and class diagram does not provide any facility for creating common values to the attribute. Therefore a value is created to the attribute using the resultant OWL to detect attribute type error. Moreover it is analyzed that all multiplicity inconsistency which are detected are not solved.
6.5 SUMMARY

At the modeling level, to design and develop semantic web applications, there is a need to convert class diagrams to OWL. Also, consistency check is required during this conversion. A formal method like DL is used to detect and resolve it. A set of conversion rules is proposed for representing the class diagram in Intermediate DL statements with description about reasoning properties and various inconsistencies like duplicate name errors, Attribute value type error, generalization and multiple inheritances in UML which affect OWL. A modified algorithm is proposed in this chapter to detect and resolve each of the above inconsistency trigger by taking set of DL as input and resolve the inconsistent axioms like attribute type error, cardinality constraint, cyclic inheritance and duplicate name error. The inconsistency due to cyclic inheritance is resolved by computing the rank using ARC algorithm. After obtaining consistency, a description on how the consistent DL axioms are mapped to OWL is discussed.

In our approach, the resultant OWL obtained for the particular application is stored in the Repository. When the web request occurs the web service will invoke the OWL Repository based on the application and access the data automatically. The next chapter deals with how the web services can refer ontology when it is annotated and discuss about the discovery of web services.