Translating UML Class Diagrams into First Order Logic

Love Ekenberg, Paul Johannesson, Marcelo Viriato Munguanaze, and Rika Manka Tesha

Department of Computer and Systems Sciences, SU/KTH

1 Introduction

The Unified Modelling Language, UML, has gained increased popularity in recent years. It is now regularly used not only for systems analysis and design, for which it was originally conceived, but also for other phases in the systems life cycle such as requirements engineering and business analysis. The success of UML can to a large extent be attributed to two factors. First, UML has received extensive industry support from IT suppliers as well as users and has been effectively standardised. Secondly, UML makes use of intuitive and visual modelling constructs as the main components of the language, which facilitates its adoption among large user groups. However, this reliance on graphical constructs poses problems when it comes to precise and unambiguous semantics. The constructs of UML are typically informally defined, which leaves room for ambiguities, loose interpretations and misunderstandings. An important task is, therefore, to formalise the basic notions and constructs of UML.

The main purpose of this work is to describe how specifications in UML language can be translated into a logic based formalism so as to prepare a unifying view of a general class of conflict detection. Translating specifications into first order logic has the added advantage that by introducing 2nd order analyses, it becomes possible to analyse a broader spectrum of conflicts, i.e. using first order logic extended with transaction mechanisms provides tools for systematically classifying conflicts, including: individual inconsistency; protocols for multi-agent behaviour; incongruence when events are incompatible; whether certain combinations of initial states are incongruent; whether event paths are incompatible, etc. cf. [Ekenberg 96, Ekenberg 00]. This kind of unified view is made possible through translations of the kind suggested in this paper, even when the processes are described in an event-driven representation. We argue that first order logic, when used in conjunction with conceptual modelling, provides a sound basis on which specifications written in a process based language can be transformed, merged, and verified for the purpose of detecting interference.

Different varieties of temporal logic [Dixon 98] and BDI logic [Rao 98] on the other hand are strong contenders for the target language. In fact, much previous work on formalising UML has been based on different versions of temporal logic, e.g. [Knapp 99], who defines the semantics of UML interactions in temporal logic. Another approach has been to map UML constructs to some formal specification language, e.g. [Kim 99], who maps class diagram to Object-Z. Dynamic logic has been used as a basis for UML semantics, e.g. in [Pons 99]. One of the most complete formalisations of UML is given in [Övergaard 00], who bases the semantics on π-calculus and labelled transition systems. However, first order logic remains the choice for this work. This is in agreement with [Johannesson 98] and [van Benthem 95], that argue for the advances of first order logic to model dynamics in contrast to approaches based on temporal logic. Furthermore, the representation in first-order logic has some convenient features from a theorem proving perspective. Given access to an efficient theorem prover, NP-complete problems can, in many cases, be solved within a reasonable time [Stålmarck 96]. Most propositional theorem provers have been based either on the resolution principle [Robinson 65] or on semantic tableaux [Beth 59], [Smullyan 68]. A proof system formulated in [Mondadori 88ab], that is similar to [Stålmarck 95], has been investigated by [D'Agostino 90]. His conclusion is, roughly, that natural deduction systems with a discharge rule based on the bivalence principle are generally better than those with other types of discharge rules, such as the discharge rule of semantic tableaux. [1,]

In the first section, show how the static constructs of class diagrams in UML can be translated into a first Order logic framework. In section 2, we describe the tool created using rational rose that simulates the translation process. In section 3, we introduce the theorem prover Leantap, that enables us to perform some theorem proving on the translation output of our tool.

3 Translating UML Class Diagrams into Logic

Static concepts and relations modelled in UML can straightforwardly be expressed in terms of first order formulae. In this section, we suggest such a translation for static properties of class diagrams.

3.1 UML Class Diagrams

A class diagram is the standard modelling concept for static properties in UML. The components of a class diagram are classes that represent concepts in the world. These can have one or more lexical attributes. Classes can be related to each other, which is represented in UML by associations. Associations can be further specified by cardinality constraints expressing properties such as, e.g., injectivity, surjectivity and totality. Compositions are particular kind of associations, and are expressed by aggregations in UML. Furthermore, various kinds of subset relations can be represented.

Using definition 5 below, a class diagram in UML can readily be translated into a set of first order formulae.

Definition 5

Given a set C of class diagrams in a UML specification, where the strings *agg* or *lex* do not occur. R_C is the least set of first order formulae defined by the following clauses.

---

[1] More specifically, it is shown that Mondadori's proof system restricted to subformula proofs p-simulates [Cook 79] semantic tableaux but the opposite does not hold. The system of [Stålmarck 95] (restricted to subformula proofs) is easily seen to p-simulate Mondadori's system.

1. Alphabet

a) If *r* is a name of a class definition in C, then *r* is a predicate symbol of arity one in $L(R_C)$.
b) If *t* is a name of an association in C, then *t* is a predicate symbol of arity two in $L(R_C)$.
c) If *t* is a name of an attribute in C, then *t* is a predicate symbol of arity two in $L(R_C)$.
d) *agg* is a predicate symbol of arity two in $L(R_C)$.
e) *lex* is a predicate symbol of arity one in $L(R_C)$.

2. Typing constraints for associations

If *r* and *s* are names of class definitions in C, and *t* is a name of an association from *r* to *s* in C, then $\forall x \forall y(t(x,y) \rightarrow (r(x) \wedge s(y)))$ is in R_C.

3. Typing constraints for attributes

If *r* is a name of a class definitions in C and *t* is a name of an attribute of *r* in C, then $\forall x \forall y(t(x,y) \rightarrow (r(x) \wedge lex(y)))$ is in R_C.

4. Aggregation constraints

If *r* and *s* are names of class definitions in C, and *t* is a name of an aggregation from *r* to *s* in C, then $\forall x \forall y(t(x,y) \rightarrow (r(x) \wedge s(y) \wedge agg(x,y)))$ is in R_C.

5. ISA constraints

If *r* and *s* are names of class definitions in C, and the statement *r ISA s* belongs to C, then $\forall x(r(x) \rightarrow s(y))$ is in R_C.

5. Subclass constraints

Assume that *p*, *r* and *s* are names of class definitions in C, and that *p ISA s* and *r ISA s* belong to C. If *p* and *r* are disjoint in C, then $\forall x \neg(p(x) \wedge r(x))$ is in R_C. If *p* and *r* are exhaustive wrt *s* in C, then $\forall x(s(x) \rightarrow (p(x) \vee r(y)))$ is in R_C.

6. Cardinality constraints

If r and s are names of class definitions in C, and t is a name of an association from r to s in C, with cardinality $((\mathrm{min}_r .. \mathrm{max}_r), (\mathrm{min}_s .. \mathrm{max}_s))$, then the formulae below are in R_C.

6.1. Minimum number of associations for the domain

$$\forall y \exists x_1 .. \exists x_{\mathrm{min}_r} ((s(y)) \rightarrow (t(x_1, y) \wedge ... \wedge t(x_{\mathrm{min}_r}, y))) \wedge$$
$$\neg(x_1 = x_2) \wedge ... \wedge \neg(x_1 = x_{\mathrm{min}_r}) \wedge$$
$$\neg(x_2 = x_3) \wedge ... \wedge \neg(x_2 = x_{\mathrm{min}_r}) \wedge ... \wedge$$
$$\neg(x_{\mathrm{min}_{r-1}} = x_{\mathrm{min}_r})$$

6.2. Maximum number of associations for the domain

$$\forall y \forall x_1 ... \forall x_{\mathrm{max}_r} \forall x_{\mathrm{max}_r + 1} [((t(x_1, y) \wedge ... \wedge t(x_{\mathrm{max}_r}, y) \wedge t(x_{\mathrm{max}_r + 1}, y))$$
$$\rightarrow$$
$$((x_1 = x_2) \vee ... \vee (x_1 = x_{\mathrm{max}_r}) \vee (x_1 = x_{\mathrm{max}_r + 1}) \vee$$
$$(x_2 = x_3) \vee ... \vee (x_2 = x_{\mathrm{max}_r}) \vee (x_2 = x_{\mathrm{max}_r + 1}) \vee ... \vee$$
$$(x_{\mathrm{max}_r} = x_{\mathrm{max}_r + 1}))]$$

6.3. Minimum number of associations for the range

$$\forall y \exists x_1 .. \exists x_{\mathrm{min}_s} ((r(y)) \rightarrow (t(y, x_1) \wedge ... \wedge t(y, x_{\mathrm{min}_s}))) \wedge$$
$$\neg(x_1 = x_2) \wedge ... \wedge \neg(x_1 = x_{\mathrm{min}_s}) \wedge$$
$$\neg(x_2 = x_3) \wedge ... \wedge \neg(x_2 = x_{\mathrm{min}_s}) \wedge ... \wedge$$
$$\neg(x_{\mathrm{min}_{s-1}} = x_{\mathrm{min}_s})$$

6.4. Maximum number of associations for the range

$$\forall y \forall x_1 ... \forall x_{\mathrm{max}_s} \forall x_{\mathrm{max}_s + 1} [(((t(y, x_1) \wedge ... \wedge t(y, x_{\mathrm{max}_s}) \wedge t(y, x_{\mathrm{max}_s + 1}))$$
$$\rightarrow$$
$$((x_1 = x_2) \vee ... \vee (x_1 = x_{\mathrm{max}_s}) \vee (x_1 = x_{\mathrm{max}_s + 1}) \vee$$
$$(x_2 = x_3) \vee ... \vee (x_2 = x_{\mathrm{max}_s}) \vee (x_2 = x_{\mathrm{max}_s + 1}) \vee ... \vee$$
$$(x_{\mathrm{max}_s} = x_{\mathrm{max}_s + 1}))]$$

Example

Consider the UML class diagram below.

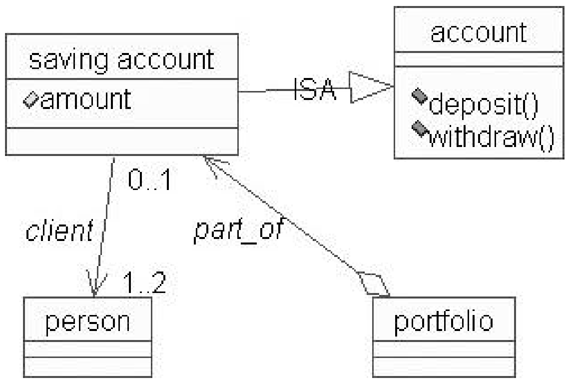

Fig. 2 A UML class diagram

Thus, *saving_account*, *account*, *person* and *portfolio* are classes in UML, representing concepts. The relation *client* is represented by an association from *saving_account* to *person*, meaning that a client is a person with an account. The attribute *amount* represents the current balance. An account must be owned by a person, and a person can have at most one saving account. Furthermore, at maximum two clients can share an account. The class *saving_account* is a subclass to *account*. Furthermore, *saving_account* is a component of the portfolio of the bank. This is represented by the aggregation form *saving_account* to *portfolio*. The methods *deposit* and *withdraw* represent possible transactions for an account and will be treated in section 4 below.

Using the rules in definition 5, the class diagram is translated to the following set of formulae:

Translation of static properties	**UML components**
$\{\forall x(\text{saving\_account}(x) \rightarrow \text{account}(x))$,	ISA relation
$\forall x \forall y(\text{client}(x,y) \rightarrow (\text{saving\_account}\ (x) \wedge \text{person}(y)))$,	association *client*
$\forall x \forall y(\text{amount}(x,y) \rightarrow (\text{saving\_account}(x) \wedge \text{lex}(y)))$,	attribute *amount*
$\forall x \forall y(\text{part\_of}(x,y) \rightarrow (\text{saving\_account}(x) \wedge \text{portfolio}(y) \wedge \text{agg}(x,y)))$,	aggregation *part_of*
$\forall x \exists y(\text{saving\_account}(x) \rightarrow \text{client}(x,y))$,	cardinality 1
$\forall x \forall y \forall y \forall w(\text{client}(x,y) \wedge \text{client}(x,z) \wedge \text{client}(x,w)) \rightarrow$ $(y=z \vee y=w \vee z=w))$,	cardinality ..2
$\forall x \forall y \forall y(\text{client}(y,x) \wedge \text{client}(z,x)) \rightarrow (y=z)\}$	cardinality ..1

```
' ------------------------------------------------------------
' File: TranslationToFOL_Leantap.ebs
'
' Description:    1. Translates a rose Class diagram model into FOL syntax
'                 Each class, attribute, association, etc is written in a text files
(FolQuoted.txt)
'              2. Remove the quota marks and writes a new txt file      (Fol.txt)
'         by:  Manka Tesha & Marcelo Munguanaze
'              KTH/Stockholm University (DSV) @2001/2002
'  created on:  October 2001
' last update:  March 2002
'----------------------------------------------------------------------------------------------------
-

Const Banner$ = "%-----------------------------------------------------------------------------
---------"
Dim number As Integer

Sub PrintClass (aClass As Class, Indent As Integer)
'write each class Name as predicate symbol of arity one
                  Write #1, Space$(Indent) + aClass.Name
End Sub

Sub PrintAttribute (anAttribute As Attribute, Indent As Integer)
'write each attribuite Name as predicate symbol of arity two
                  Write #1, anAttribute.Name
End Sub


Sub PrintClassAttributes (aClass As Class, Indent As Integer)
    Dim AllAttributes As AttributeCollection
                  For i% = 1 To aClass.Attributes.Count
                                  Set AllAttributes = aClass.Attributes
                                  Call PrintAttribute (AllAttributes.GetAt (i%), 0)
                  Next i%
End Sub


Sub PrintAttributeAssociation (anAttribute As Attribute, theClassName As String,
Indent As Integer)
                  If Len (anAttribute.Type) > 0 Then
                                  Write #1, "For All x For All y(" + anAttribute.Name
+ "(x,y)->(" + theClassName + "(x) And " + anAttribute.Type + "(y)))."
                  Else
                                  Write #1, "For All x For All y(" + anAttribute.Name
+ "(x,y)->(" + theClassName + "(x) And lex(y)))."
                  End If
```

```
End Sub

Sub PrintClassAttributesAssociation (aClass As Class, Indent As Integer)
    Dim AllAttributes As AttributeCollection
                   'for each Class, print their attibuites
                   For i% = 1 To aClass.Attributes.Count
                                 Set AllAttributes = aClass.Attributes
                                 Call PrintAttributeAssociation (AllAttributes.GetAt
(i%), aClass.Name, 0)
                   Next i%
End Sub

Sub PrintClassGeneralization (aClass As Class, Indent As Integer)
                   Dim theSuperClasses As ClassCollection
                   Set theSuperClasses = aClass.GetSuperClasses ()

                   If theSuperClasses.Count > 0 Then
                                 For i% = 1 To theSuperClasses.Count
                                               Write #1, "For All x(" +
aClass.Name + " (x) ->" + theSuperClasses.GetAt (i%).Name + "(x))."
                                 Next i%
                   End If
End Sub


Sub PrintClassAssociations (aClass As Class, Indent As Integer)

                   Dim theAssociations As AssociationCollection
                   Dim Association As Association
   Dim myRole As Role
                   Dim directionRole As Role

                   Set theAssociations = aClass.GetAssociations ()
    For i% = 1 To theAssociations.Count
                                 Set Association = theAssociations.GetAt (i%)
                                 Set directionRole =
Association.GetRoleForNameDirection()
                                               If (directionRole.Name =
aClass.Name) Then
                                                             If Len
(Association.Name) > 0 Then

                   Write #1, Association.Name
                                                             Else

                   Write #1, "<Not Named>"
                                                             End If
```

```
                                        End If
                Next i%
End Sub

Sub PrintAssociationsStatements (aClass As Class, Indent As Integer)

                Dim theAssociations As AssociationCollection
                Dim Association As Association
  Dim allAssociations As New AssociationCollection

  Dim myRole As Role
                Dim myOtherRole As Role
                Dim directionRole As Role

                Set theAssociations = aClass.GetAssociations ()

   For i% = 1 To theAssociations.Count
                                Set Association = theAssociations.GetAt (i%)
                                Set directionRole =
Association.GetRoleForNameDirection()
                                Set myRole = Association.GetCorrespondingRole
(aClass)
                                Set myOtherRole = Association.GetOtherRole
(aClass)
                                If (directionRole.Name = aClass.Name) Then
                                        Write #1, "For all x For all y(" &
association.name &"(x,y) => (" & myOtherRole.Name & " (x) And " &
myRole.Name &"(y))."
                                End If
                Next i%

End Sub

Sub PrintMultiplicityConstraintStatements (aClass As Class, Indent As Integer)

 ' multiplicity code here  for 0..1;0..n; min..max, ...

                Dim theAssociations As AssociationCollection
                Dim Association As Association
  Dim allAssociations As New AssociationCollection

  Dim myRole As Role
                Dim myOtherRole As Role
                Dim directionRole As Role

                Dim multiplicity As String
                Dim min_value As Integer
```

```
Dim max_value As Integer

Dim tempString As String
Dim tempStringLean As String

Set theAssociations = aClass.GetAssociations ()

For i% = 1 To theAssociations.Count
        Set Association = theAssociations.GetAt (i%)
        Set directionRole =
Association.GetRoleForNameDirection()
        Set myRole = Association.GetCorrespondingRole
(aClass)
        Set myOtherRole = Association.GetOtherRole
(aClass)
        'If (directionRole.Name = aClass.Name) Then

        Select Case myOtherRole.Cardinality

                Case "n"
                        Write #1," Case n"
                Case "0..n"
                        tempString =  "For
all x ("
                        tempString =
tempString & directionRole.name & "(x) ->(For all z1..For all zn,For all zn+1("
                        tempString =
tempString & Association.Name & "(x,z1) ^ " & Association.Name & "(x,zn) ^ " &
Association.Name & "(x,zn+1)) ->(z1=z2 v..v z1=zn v z1=zn+1) v(zn=zn+1)))"
                        Write #1, tempstring
                Case "1..n"
                        tempString =  "For
all x (" & myOtherRole.Name & "(x) ->(y1(" & Association.Name & " (x,y1) ^(For
all z1..For all zn,For all zn+1(" & Association.Name & "(x,z1) ^ "
                        tempString =
tempString & Association.Name & "(x,zn) ^ " & Association.name & "(x,zn+1))
=>(z1=z2 v..z1=zn v z1=zn+1) v(zn=zn+1)))."
                        Write #1,tempString
                Case "0..1"
                        tempString = "For
all x For all y For all z((" & Association.Name & "(x,y) ^ " & Association.Name &
"(x,z)) -> y=z) "
                        Write #1,tempString
                Case Else
                        multiplicity  =
myOtherRole.Cardinality
```

```
min_value = CInt(Mid$(multiplicity,1,1))
max_value = CInt(Mid$(multiplicity,4,1))

tempstring =  "For all x (" & directionRole.Name & "(x) ->("

For i% = 1 To min_value
    If i% = min_value Then
        tempstring = tempstring &  "Exist y" &i & "("
    Else
        tempstring = tempstring &  "Exist y" &i & ","
    End If
Next i%

For i% = 1 To min_value
    If i% = min_value Then
        tempstring = tempstring & Association.Name & "(x,y" &i & "))^"
    Else
        tempstring = tempstring &  Association.Name & "(x,y" &i & ") ^"
    End If
Next i%


For i% = 1 To min_value - 1
    For j% = (i% + 1) To min_value
        If i% = min_value - 1 Then
            tempstring = tempstring &  "(y"&i & "<>" & "y" &j & ") ^"
        Else
            tempstring = tempstring &  "(y" &i & "<>" & "y" &j & ") ^"
```

```
End If
Next j%
Next i%

'Dealing with the max value

For i% = 1 To max_value
If i% = max_value Then
tempstring = tempstring & "For all z" &i & "("
Else

tempstring = tempstring & "For all z" &i & ","
End If
Next i%

For i% = 1 To max_value + 1
If i% = max_value + 1 Then
tempstring = tempstring & Association.Name & "(x,z" &i & ")) ->"
Else
tempstring = tempstring & Association.Name & "(x,z" &i & ") ^"
End If
Next i%

For i% = 1 To max_value
For j% = (i% + 1) To max_value + 1
If i% = max_value  Then
tempstring = tempstring &  "(z" &i & "=" & "z" &j & "))."
Else
tempstring = tempstring &          "(z" &i & "=" & " z"&j &")v"
End If
```

```
                                                    Next j%
                                                    Next i%

                                                    'Print tempstring
                                                    Write #1, tempstring
                              End Select
                    Next i%    '{Until i=min_value}
End Sub

Sub PrintItems(aClass As Class, Indent As Integer)

                    Dim theAssociations As AssociationCollection
                    Dim Association As Association
   Dim allAssociations As New AssociationCollection

   Dim  myRole As Role
                    Dim myOtherRole As Role
                    Dim directionRole As Role
                    Dim multiplicity As String
                    Set theAssociations = aClass.GetAssociations ()

   For i% = 1 To theAssociations.Count
                              Set Association = theAssociations.GetAt (i%)
                              Set directionRole =
Association.GetRoleForNameDirection()
                              Set myRole = Association.GetCorrespondingRole
(aClass)
                              Set myOtherRole = Association.GetOtherRole
(aClass)
                              If (directionRole.Name = aClass.Name) Then

                              Write #1, "Class Name",aClass.Name
                              Write #1, "Role Name ",myRole.Name
                              Write #1, "Direction Name",DirectionRole.Name
                              Write #1, "my Other Role
Name",myOtherRole.Name
                              Write #1, "my Role Cardinality",myRole.Cardinality
                              Write #1, "my Other role
Cardinality",myOtherRole.Cardinality
                              End If
                    Next i%
End Sub

Sub RemoveQuotaMarkes
'Remove the quota markes and write to a new file
                    Dim f As String,s As String     ,m As String
                    f$ = OpenFilename$("FolQuoted","Text Files:*.TXT")
```

```
        m$ = OpenFilename$("Fol","Text Files:*.TXT")

        Open f$ For Input As #1
        Open m$ For Output As #2

        If f$ <> "" Then

                Do While Not Eof(1)
                Line Input #1,s$
                        a$ = Mid$(s$,2,(Len(s$)-2))
                        Print #2 , a$
                Loop
        End If

        Close #1
        Close #2

End Sub


Sub TranslationReport(myModel As Model, FileName As String) ', LeantapFile As
String)
        Dim theClasses As ClassCollection
        Dim theCategories As CategoryCollection
        Dim theModules As ModuleCollection
        Dim theSubsystems As SubsystemCollection


        Open FileName$ For Output Access Write As #1
        'Open LeantapFile$ For Output Access Write As #2

        Set theClasses = myModel.GetAllClasses ()

        Write #1, "%Translation of Rational Rose Class Diagrams Concepts
into First-Order Logic"
        Write #1, "%Alphabet"
        Write #1,  Banner
        Write #1, "%Predicates of Arity 1 "
        Write #1, Banner
        For i% = 1 To theClasses.Count
                PrintClass theClasses.GetAt (i%),0
        Next i%
        Write #1,"lex"

        Write #1, Banner
        Write #1, "%Predicates of Arity 2 "
        Write #1, Banner
```

```
		Write #1, "%Attributes"
		For i% = 1 To theClasses.Count
				PrintClassAttributes theClasses.GetAt (i%),0
		Next i%
		Write #1, "agg"

		Write #1, Banner
		Write #1, "%Attibute Associations Statements"
		Write #1, Banner
		For i% = 1 To theClasses.Count
				PrintClassAttributesAssociation
theClasses.GetAt(i%),0
		Next i%

		Write #1, Banner
		Write #1, "%Generalization Statements"
		Write #1, Banner
		For i% = 1 To theClasses.Count
				PrintClassGeneralization theClasses.GetAt(i%),0
		Next i%

		Write #1, Banner
		Write #1, "%associations"
		Write #1, Banner
		For i% = 1 To theClasses.Count
				PrintClassAssociations theClasses.GetAt (i%),0
		Next i%

		Write #1, Banner
		Write #1, "%Association Statements"
		Write #1, Banner
		For i% = 1 To theClasses.Count
				PrintAssociationsStatements theClasses.GetAt (i%),0
		Next i%
		Write #1, Banner
		Write #1, "%Multiplicity Statements"
		Write #1, Banner
		For i% = 1 To theClasses.Count
				PrintMultiplicityConstraintStatements
theClasses.GetAt (i%),0
		Next i%
		Close #1
		'Close #2
End Sub
```

```
Sub Main
		FileName$ = SaveFileName$ ("FolQuoted", "Text Files:*.txt")
		'LeantapFile$ = SaveFileName$ ("Saving the Leatanp File ", "Text 
Files:*.txt")
		'number=0

		If FileName$ <> "" Then TranslationReport RoseApp.CurrentModel, 
FileName$ ' , LeantapFile$
		RemoveQuotaMarkes
End Sub
```