SAPIENZA
UNIVERSITÀ DI ROMA

UML schema to logic translator

Formal Methods

A.A.2012/2013

Antonio Gallo

Fernanda Maria Grella

Igor Atzeni

Contents

1.Introduction

Unified Modeling Language(UML) is a standard formalism to software design and analysis. A big issue in this subject is to check some UML Diagram's properties for the formal reasoning on the concepts about an application domain.

Firstly we know that UML is an uncomprehensible machine and has no precise semantics, so automated reasoning is completely out of discussion.
Respect this issue, it would be highly desirable to have case tools that provide automated reasoning services to detect relevant formal properties of UML diagrams, such as inconsistencies and redundancies.
In our case, we have designed an application that provide the first order logic and description logic representation of UML Class Diagram.

In order to achieve this task, we have developed a Java application that takes as input a graphml file (generated by a graph editor named yEd) and provide the corresponding FOL/DL translation.

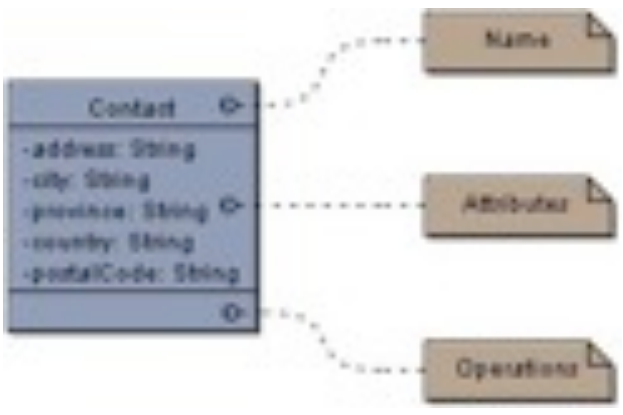
2.Translation: UML-->FOL

In the following chapter we briefly recall some theoretical sketches about the translation of UML diagram in FOL. In particular how to formalize the translation of diagram's principal component.

The UML Class Diagrams models the domain of interest in term of:
a) object grouped into classes;
b) proprierties of classes (as attributes and operations);
c) associations between classes;
d) sub-classing (ISA/Generalization associations)

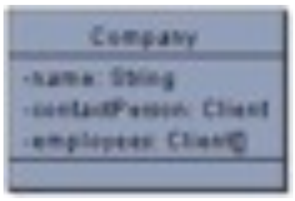
A..Classes

A class in UML models is a set of objects that share certain common properties as attributes and operations.Each class is characterized by a name and a set of properties.



B1..Attributes

An attribute models a local property of a class,it's characterized by a name,a type and possibly a multiplicity.This because we can find attributes without multiplicity and attributes with minimal and maximal number of values.



Since attributes may have a multiplicity different from 1 . . . 1 they are better formalized as binary predicates, with suitable assertions representing types and multiplicity:
Given an attribute a of a class C with type T and multiplicity i . . . j we capture it in FOL as a binary predicate aC(x,y) with the following assertions:

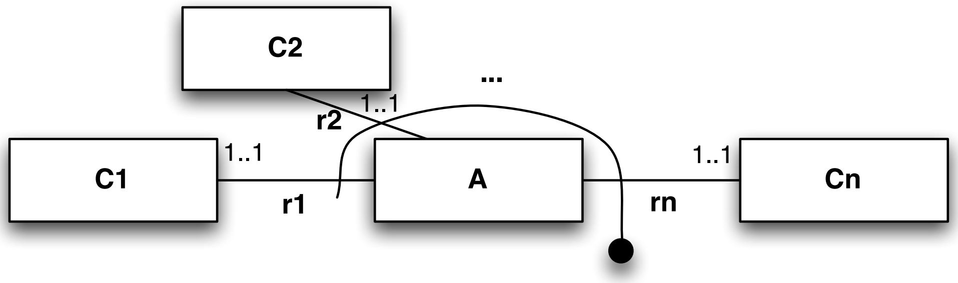
-Assertion for the attribute type

$$\forall x, y.\ a(x, y) \supset C(x) \wedge T(y)$$

-Assertion for the multiplicity

$$\forall x.\ C(x) \supset (i \leq \sharp\{y \mid a(x, y)\} \leq j)$$

Sometimes we may want to assert properties of associations. In UML to do so we resort to Association Classes,we associate to an association a class whose instances are in bijection with the tuples of the association,then we use the association class exactly as a UML class.



FOL Assertions needed for stating bijection between association class and association:

$$\forall x, y.\ r_i(x, y) \supset A(x) \land C_i(y), \qquad \text{for } i = 1, \dots, n$$

$$\forall x.\ A(x) \supset \exists y.\ r_i(x, y), \qquad \text{for } i = 1, \dots, n$$

$$\forall x, y, y'.\ r_i(x, y) \land r_i(x, y') \supset y = y', \qquad \text{for } i = 1, \dots, n$$

$$\forall y_1, \dots, y_n, x, x'.\ \bigwedge_{i=1}^{n} (r_i(x, y_i) \land r_i(x', y_i)) \ \supset\ x = x'$$

B2..Operations

The operations listed in a class represent the functions or tasks that can be performed on the data in the class. An operation definition for a class C has the form

f(P1,P2,.....,Pm) : R

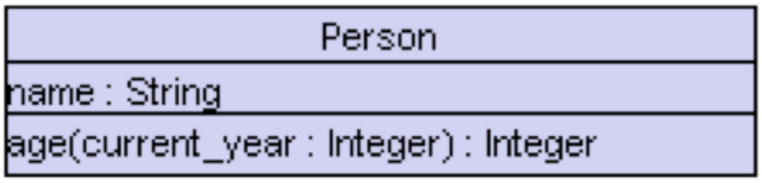
where "f" is the name of the operation, "P1,P2,...Pm" are the types of the m parameters,and "R" is the type of the result.

The predicate $f_{P1,...Pm}$ has to satisfy the following FOL assertions:

$$\forall x, p_1, \dots, p_m, r.\ f_{P_1,\dots,P_m}(x, p_1, \dots, p_m, r) \supset \bigwedge_{i=1}^{m} P_i(p_i)$$
$$\forall x, p_1, \dots, p_m, r, r'.\ f_{P_1,\dots,P_m}(x, p_1, \dots, p_m, r) \land f_{P_1,\dots,P_m}(x, p_1, \dots, p_m, r') \supset r = r'$$

$$\forall x, p_1, \dots, p_m, r.\ C(x) \land f_{P_1,\dots,P_m}(x, p_1, \dots, p_m, r) \supset R(r)$$

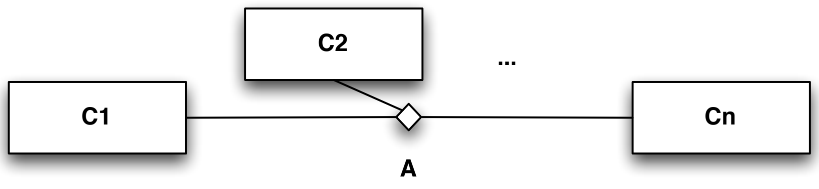
As following:



$$\forall x, p, r.\ age_{Integer}(x, p, r) \supset Integer(p)$$
$$\forall x, p, r, r'.\ age_{Integer}(x, p, r) \land age_{Integer}(x, p, r') \supset r = r'$$
$$\forall x, p, r.\ Person(x) \land age_{Integer}(x, p, r) \supset Integer(r)$$

C..Associations

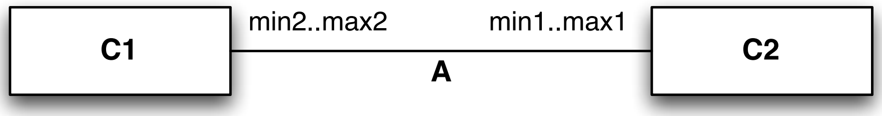
Relationships between classes are modeled in UML Class Diagrams as Associations. An association in UML is a relation between the instance of two or more classes moreover an association between two classes is a property of both classes.



We can represent an n-ary association A among classes C1,C2,....,Cn as n-ary predicate A in FOL.
We assert that the components of predicate must belong to correct classes:

$$\forall x_1, \ldots, x_n.\ A(x_1, \ldots, x_n) \supset C_1(x_1) \wedge \ldots \wedge C_n(x_n)$$

As following the multiplicities of binary associations are easily expressible in FOL:



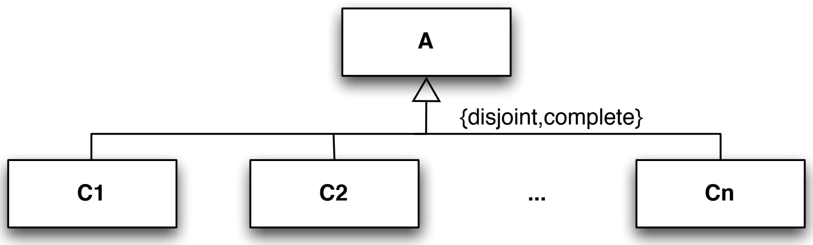
$$\forall x_1.\ C_1(x_1) \supset (min_1 \leq \sharp\{x_2 \mid A(x_1, x_2)\} \leq max_1)$$
$$\forall x_2.\ C_2(x_2) \supset (min_2 \leq \sharp\{x_1 \mid A(x_1, x_2)\} \leq max_2)$$

Multiplicities can be anything we specify,as following:

0	zero
1	one
1..*	one or many
1..2,10..*	one,two or ten and above but not three through nine

D..ISA/Generalization

The ISA relationship is of particular importance in conceptual modeling:a class C ISA a class C’ if every instance of C is also an instance of C’.
In UML the ISA relationship is modeled through the notion of generalization.
The generalization link is used between two classes to show that a class incorporates all of the attributes and operations of another, but adds to them in some way.



ISA:

$$\forall x.\ C_i(x) \supset C(x), \qquad \text{for } i = 1, \ldots, n$$

The most used constraints are disjointness and completeness:

-**disjointness** asserts that different subclasses cannot have common instances

$$\forall x.\ C_i(x) \supset \neg C_j(x), \qquad \text{for } i \neq j$$

-**completeness** asserts that every instances of the superclass is also an instance of at least one of the subclasses.

$$\forall x.\ C(x) \supset \bigvee_{i=1}^{n} C_i(x)$$

3.Translation: FOL-->DL

Description logics(DL) based systems are currently in use in many applications:
-Configuration
-Conceptual Modeling
-Query Optimization and View Maintenance
-Natural Language Semantics
-Planning

Description Logics formalize many Object-Oriented representation approaches.
Any (basic) Description Logic is a fragment of FOL.
The representation is at the predicate level: no variables are present in the formalism.
A Description Logic theory is divided in two parts:
-the definition of predicates (TBox)
-the assertion over constants (ABox)

The basic step of the construction is the choise of disjoint sets of symbols for primitive concepts,and primitive roles.
Terms are then defined using the alphabets by means of constructors/operators.

Set constructors

•intersection C ⊓ D
• union C ⊔ D
•complement ¬C

$$\text{Person} \sqcap \neg\text{Female} \qquad \text{and} \qquad \text{Female} \sqcup \text{Male}.$$

Role restrictions

•universal role restriction, ∀R.C, requires that all those individual objects that are in the relation R belong to C.
•existential role restriction, ∃R.C, requires the existence of an infividual in the relation R that belongs to C.

$$\exists\text{hasChild.Person} \sqcap \forall\text{hasChild.Person}.$$

Number restrictions

Number restrictions denote sets of individuals with atleast/atmost the specified number of role fillers.

$$(\leqslant 3 \ \text{hasChild}) \sqcap (\geqslant 2 \ \text{hasChild})$$

Syntax and semantics of AL

$\mathcal{AL}$ is the basic language in the family of $\mathcal{AL}$ languages

Construct	Syntax	Example	Semantics
atomic concept	A	**Doctor**	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
atomic role	P	**child**	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
atomic negation	$\neg A$	**¬Doctor**	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
conjunction	$C \sqcap D$	**Hum ⊓ Male**	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
(unqual.) exist. res.	$\exists R$	**∃child**	$\{ a \mid \exists b. (a,b) \in R^{\mathcal{I}} \}$
value restriction	$\forall R.C$	**∀child.Male**	$\{a \mid \forall b. (a,b) \in R^{\mathcal{I}} \supset b \in C^{\mathcal{I}}\}$

(C, D denote arbitrary concepts and R an arbitrary role)

The $\mathcal{AL}$ family

Typically, additional constructs w.r.t. those of $\mathcal{AL}$ are needed:

Construct	$\mathcal{AL}\cdot$	Syntax	Semantics
disjunction	$\mathcal{U}$	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
qual. exist. res.	$\mathcal{E}$	$\exists R.C$	$\{ a \mid \exists b. (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}} \}$
(full) negation	$\mathcal{C}$	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
number	$\mathcal{N}$	$(\geq k\, R)$	$\{ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}}\} \geq k \}$
restrictions		$(\leq k\, R)$	$\{ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}}\} \leq k \}$
qual. number	$\mathcal{Q}$	$(\geq k\, R.C)$	$\{ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq k \}$
restrictions		$(\leq k\, R.C)$	$\{ a \mid \#\{b \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq k \}$
inverse role	$\mathcal{I}$	P^{-}	$\{ (a,b) \mid (b,a) \in P^{\mathcal{I}} \}$

In the following we show how apply that syntax.

Encoding of classes and attributes

An UML class C is represented by an atomin concept C. Each attribute a of type T for C is represented by an atomin role a_C .

-To encode the typing of a for C:

$$\exists a_C \sqsubseteq C \qquad \exists a_C^{-} \sqsubseteq T$$

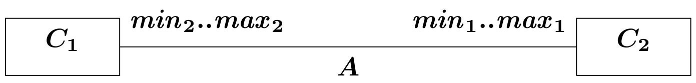
-To encode the multiplicity [i..j] of a:

$$C \sqsubseteq (\geq i\, a_C) \sqcap (\leq j\, a_C)$$

in particular in that case when j is * so we omit the second conjunct,while when the multiplicity is [0..*] we omit the whole assertion,while when the multiplicity is missing the assertion becomes:

$$C \sqsubseteq \exists a_C \sqcap (\leq 1\, a_C)$$

Encoding of binary associations without association class



- A is represented by an $\mathcal{ALCQI}$ role A, with:

$$\exists A \sqsubseteq C_1 \qquad \exists A^- \sqsubseteq C_2$$

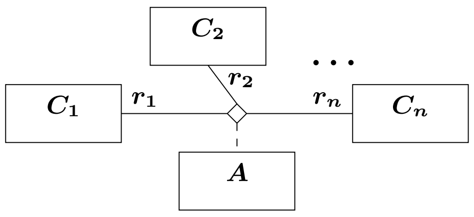
- To encode the multiplicities of A:
  - each instance of C_1 is connected through A to at least min_1 and at most max_1 instances of C_2:

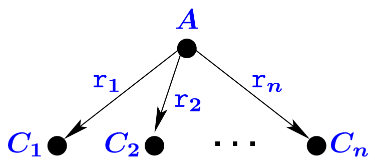
$$C_1 \sqsubseteq (\geq min_1\, A) \sqcap (\leq max_1\, A)$$

  - each instance of C_2 is connected through A^- to at least min_2 and at most max_2 instances of C_1:

$$C_2 \sqsubseteq (\geq min_2\, A^-) \sqcap (\leq max_2\, A^-)$$

Encoding of associations via reification

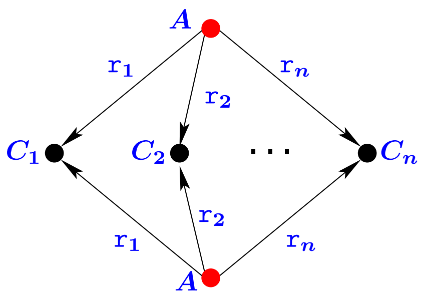




- Association A is represented by a concept A
- Each instance of the concept represents a tuple of the relation
- n (binary) roles $\mathtt{r}_{A,1}, \dots, \mathtt{r}_{A,n}$ are used to connect the object representin a tuple to the objects representing the components of the tuple
- To ensure that the instances of A correctly represent tuples:

$$\exists \mathtt{r}_{A,i} \sqsubseteq A \qquad \exists \mathtt{r}_{A,i}^- \sqsubseteq C_i \qquad i = 1, \dots, n$$
$$A \sqsubseteq \exists \mathtt{r}_{A,1} \sqcap \cdots \sqcap \exists \mathtt{r}_{A,n} \sqcap (\leq 1\, \mathtt{r}_{A,1}) \sqcap \cdots \sqcap (\leq 1\, \mathtt{r}_{A,n})$$

We have not ruled out the existence of two instances of A representing the same tuple of association A:

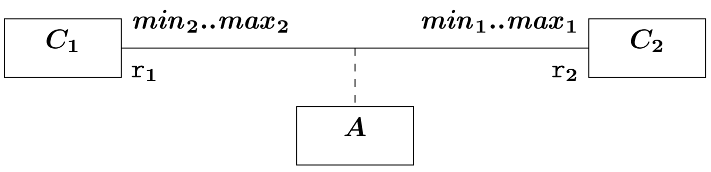


To rule out such a situation we could add a key assertion:

$$(\text{id } A \; \mathtt{r}_1, \ldots, \mathtt{r}_n)$$

Note: in a tree-model the above situation cannot occur
$\leadsto$ Since in reasoning on an $\mathcal{ALCQI}$ KB we can restrict the attention to tree-models, we can ignore the key assertions

Multiplicities of binary associations with association class



To encode the multiplicities of A we need qualified number restrictions:

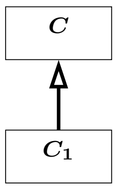
- each instance of C_1 is connected through A to at least min_1 and at most max_1 instances of C_2:

$$C_1 \quad \sqsubseteq \quad (\geq min_1 \, \mathtt{r}^-_{A,1}) \sqcap (\leq max_1 \, \mathtt{r}^-_{A,1})$$

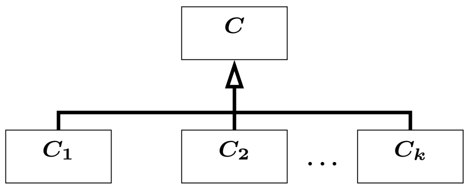
- each instance of C_2 is connected through A^- to at least min_2 and at mos max_2 instances of C_1:

$$C_2 \quad \sqsubseteq \quad (\geq min_2 \, \mathtt{r}^-_{A,2}) \sqcap (\leq max_2 \, \mathtt{r}^-_{A.2})$$

Encoding of ISA and generalization



$$C_1 \sqsubseteq C$$



$$\begin{array}{ccc} C_1 & \sqsubseteq & C \\ & \vdots & \\ C_k & \sqsubseteq & C \end{array}$$

- When the generalization is disjoint

$$C_i \sqsubseteq \neg C_j \qquad \text{for } 1 \leq i < j \leq k$$

- When the generalization is complete

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_k$$

Encoding of operations

Operation $f(P_1, \ldots, P_m) : R$ for class C corresponds to an $(m+2)$-ary relation that is functional on the last component

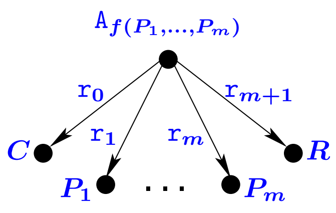
- Operation $f() : R$ without parameters directly represented by an atomic role $\mathtt{P}_{f()}$, with:

$$\exists \mathtt{P}_{f()} \sqsubseteq C \qquad \exists \mathtt{P}^-_{f()} \sqsubseteq R \qquad C \sqsubseteq (\leq 1\, \mathtt{P}_{f()})$$

- Operation $f(P_1, \ldots, P_m) : R$ with one or more parameters can be cannot expressed in $\mathcal{ALCQI}_{id}$ through reification:
  - relation is reified by using a concept $\mathtt{A}_{f(P_1,\ldots,P_m)}$
  - each instance of the concept represents a tuple of the relation
  - (binary) roles $\mathtt{r}_0, \ldots, \mathtt{r}_{m+1}$ connect the object representing a tuple to the objects representing the components of the tuple

Reification of operations

To represent operation $f(P_1, \ldots, P_m) : R$ for class C:



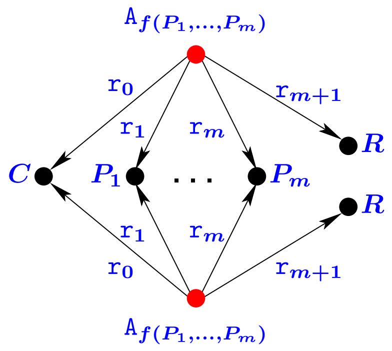
$$\begin{array}{llllr}
\mathtt{A}_{f(P_1,\ldots,P_m)} & \sqsubseteq & \exists\mathtt{r}_0 \sqcap \cdots \sqcap \exists\mathtt{r}_{m+1} \sqcap & & (1)\\
 & & (\leq 1\,\mathtt{r}_0) \sqcap \cdots \sqcap (\leq 1\,\mathtt{r}_{m+1}) & & \\
\exists\mathtt{r}_i \sqsubseteq \mathtt{A}_{f(P_1,\ldots,P_m)} & & \exists\mathtt{r}_i^- \sqsubseteq P_i & i = 1, \ldots, m & (2)\\
\exists\mathtt{r}_o \sqsubseteq \mathtt{A}_{f(P_1,\ldots,P_m)} & & \exists\mathtt{r}_i^- \sqsubseteq C & & (3)\\
\exists\mathtt{r}_{m+1} \sqsubseteq \mathtt{A}_{f(P_1,\ldots,P_m)} & & \exists\mathtt{r}_{m+1}^- \sqsubseteq R & &
\end{array}$$

(1) ensures that the instances of $\mathtt{A}_{f(P_1,\ldots,P_m)}$ represent tuples

(2) ensures that the parameters of the operation have the correct types

(3) ensures that, when the operation is applied to an instance of C, then the result is an instance of R

Again, we have not ruled out two instances of $\mathtt{A}_{f(P_1,\ldots,P_m)}$ representing two applications of the operation with identical parameters but different result:

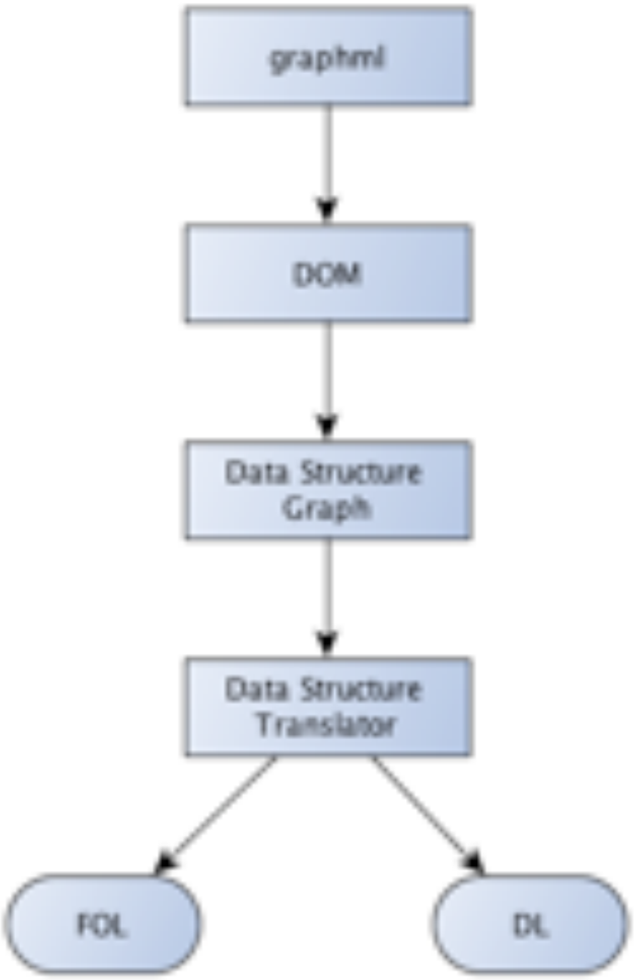


To rule out such a situation we could add a key assertion:
$(\mathbf{id}\ \mathtt{A}_{f(P_1,\ldots,P_m)}\ \mathtt{r}_0, \mathtt{r}_1, \ldots, \mathtt{r}_m)$

Again, by the tree-model property of $\mathcal{ALCQI}$, we can ignore the key assertion for reasoning

4.Stratified solution’s cost and explanation

The approach to the problem previews a stratified solution.
Firstly the user designs classes's UML diagram (called as "pseudo-UML" too) using the yEd editor and it is created the correspondent file,whose format is ".grapheme".
From that point we generated a DOM tree from the input xml code to populate,with a linear cost,a data structure created ad-hoc,that represent just also a graph. In this way we can explore our graph with a O(1) cost each step.It's necessary this exploration because each UML construct is a graph's portion (for example a binary association is composed from three nodes and two arcs) that need to be process (it using an other data structure whose each entity represent a construct) to generate the correspondent translation in FOL/DL.



We can assume that from XML to Building DOM the cost is polynomial .
The data structures graph requires one visit from the DOM tree to get the nodes, and another visit to get the edges, than put them into an hash map.

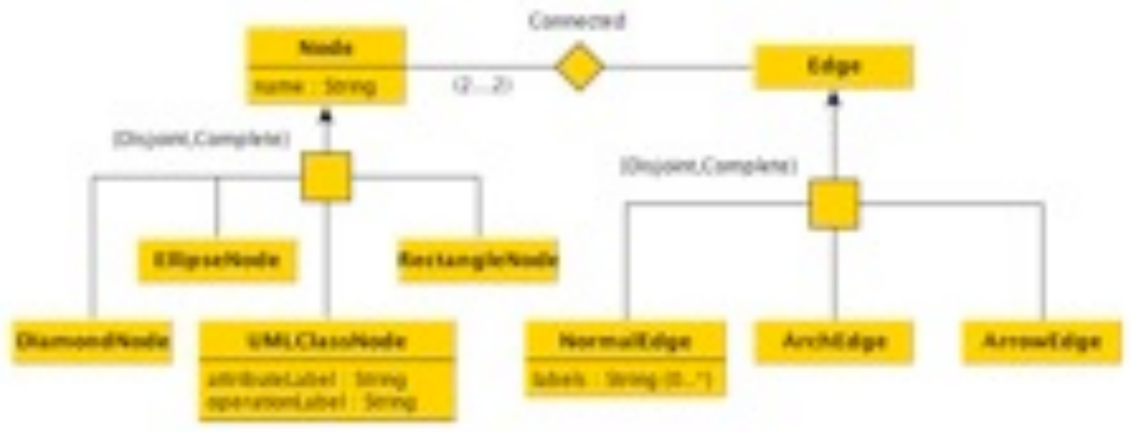
For every edge that connects two nodes , the algorithm get this nodes from the hash map, and link each other. Assuming that the cost of getting an element from the hash map is constant, we claim that the complexity of the entire algorithm is linear in the number of the element of the DOM.
The next step is to populate the data structure translator, iterating each node from the hash map and visiting the adjacent edges and nodes. Except the case of key constraint that, starting from a circular node, keep visiting the graph until there is an “arch shaped” edge. Considering this observations we can conclude that in the worst case the cost is quadratic, but in average case the degree of each node is very low, so we can consider it linear.
The cost of generating DL/FOL formulae is linear in the size of the translator data structure.
Summarizing, in the worst case total cost is quadratic.

5.How design diagrams using yEd

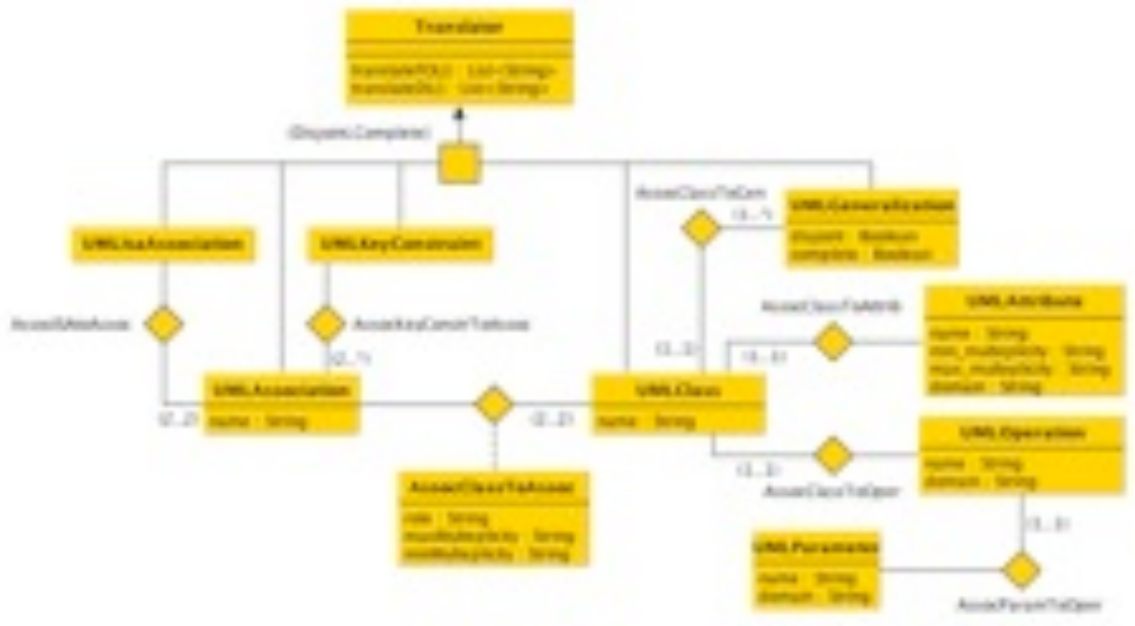
The graph editor (yEd) ,that we have chosen, is an editor to design a graph,in particular it natively support the creation of classes,attributes and operations.
The principal problem is that some UML construct natively can't be represent: for example the generalizations would have to be formed from one arc that it can connect more of two nods.
For that reason we have to modified the UML syntax:
-A R association between two classes A and B it represent by three nodes A,B,R which are connected between they using arcs;
-we are in the same situation with the generalizations whose it is introduced just one node between two nodes which represent respective the "father" class and the "son" classes.
-It's possible express a key constraint just only introducing a node with a circular form and connected it to the association's nodes using a bent arc.

Summarizing, the different application's layers are explained in the following work flow:

-Data structure graph's class diagram

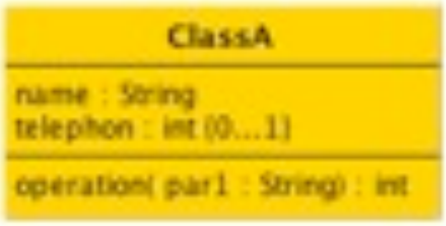


-Data structure translator's class diagram



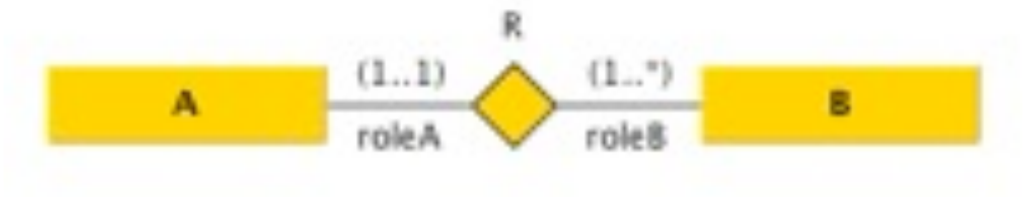
Following we observe each construct,so:

-Class:



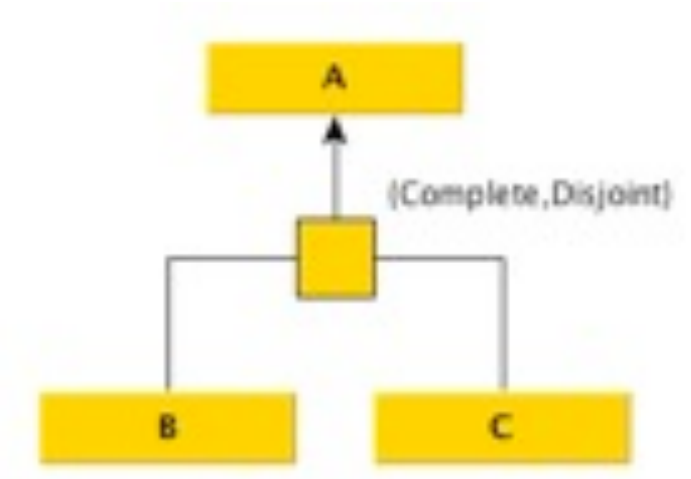
This is an example of a UML class named ClassA with two attributes and one operation. There are no differences between standard UML and this formalism. The first attribute with signature “name : String” has name “name” and domain “String”. The second attribute has multiplicity more. In the end we can see an example of an operation with one parameter.

-Association:



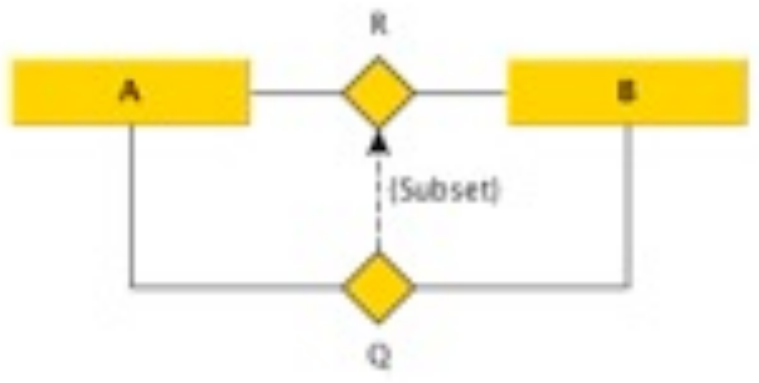
This is an example of an binary association with multiplicity, roles, and name. Notice, we have introduced a diamond node instead using a normal edge between the classes. The node is labeled with the name, and the two edges labeled with the roles and the multiplicity.

-Generalization:



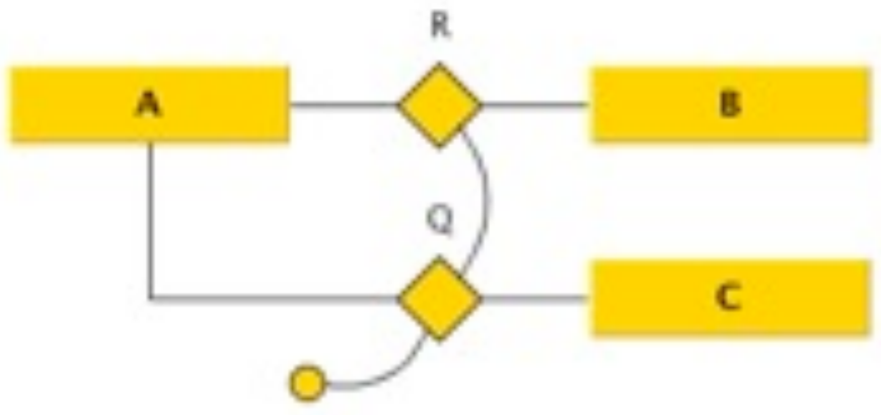
A generalization is like UML standard with a rectangle node among the classes. Notice that, the edges between the children classes and the rectangle node should be normal, and the edge the goes to the class A is arrowed.

-Association subset:



A subset between associations is an arrowed edge between the two diamond nodes. The label “{Subset}” is not mandatory, but belongs to UML.

-Key constraint:



A key constraint, in this case between R and Q, is composed by an ellipse node connected, using arch shaped edges, to the association.

6.How start the program

from command line:

```
> java -jar utl.jar diagram.graphml [output.txt]
```

where:
-diagram.graphml is the file exported from yEd and represents the UML class diagram;
-output.txt is the file in which the FOL/DL formulae will be saved.
If it is omitted, the output will be printed on screen.