USING UML CLASS DIAGRAM AS A KNOWLEDGE ENGINEERING TOOL

Thomas Raimbault, David Genest and Stéphane Loiseau
Leria laboratory – University of Angers, 2 boulevard Lavoisier, 49045 Angers Cedex 1, France

Keywords: Knowledge Engineering, Model-Based Reasoning, UML, Conceptual Graphs, Visual Quering and Checking.

Abstract: UML class diagram is the de facto standard, including in Knowledge Engineering, for modeling structural knowledge of systems. Attaching importance to visual representation and based on a previous work, where we have given a logical defined extension of UML class diagram to represent queries and constraints into the UML visual environment, we present here how using the model of conceptus graphs to answer queries and to check constraints in concrete terms.

1 INTRODUCTION

In Knowledge Engineering (KE) several tools are used, especially knowledge representation methods, for knowledge modeling and processing; e.g. (Akkerman et al., 2000). We consider both visual modeling for human readability and capable processing by a machine as fundamental in KE area. For these two reasons, we advocate on the one hand the use of the Unified Modeling Language (UML) as visual modeler, on the other hand the use of the model of conceptual graphs (CG) as engine reasoner.

Based on our previous work (Raimbault et al., 2009), where we have given both an extension of UML class diagram to represent queries and constraints into the UML visual environment and a first-order logical (FOL) semantics for extended UML class diagram (EUCD), we present here how using the CG model to answer queries and to check constraints.

Due to the great deal in common between KE and Software Engineering (SE) and the high visual level of UML class diagram as easy human readable meaning, the integration of UML in KE becomes obvious; e.g. (Rhem, 2006). UML – the so-called language from SE – provides a way to software design by using diagrams of various kinds (Booch et al., 1998). In this article, we focus our attention on class diagram that is the main UML diagram and is used to model structure of a system.

During the knowledge acquisition phase, the designer needs to query and to check several parts of already represented knowledge to complete this phase. However, UML is merely a language, then knowledge is only represented as a drawing (no reasoning way is available at the origin). The Object Constraint Language (OMG, a) (OCL), which is an integral part of UML 2, gives a solution to express constraints or queries in addition to UML diagrammatic notations. Nevertheless, as a textual language, OCL leaves the visual interest of UML out. That is the main reason why we use CG model to answer queries and to check constraints rather than OCL tools (like USE (Gogolla et al., 2007)).

In this paper, our contribution is to show how to translate EUCD from (Raimbault et al., 2009) into the CG model of (Chein and Mugnier, 1992; Chein et al., 1998; Chein and Mugnier, 2004; Baget et al., 1999; Baget and Mugnier, 2002). It is a knowledge representation and calculation model with FOL semantics that is implemented, for instance, in the Cogitant platform (Genest, 2008). This translation is interesting for tree reasons. First, the logical semantics – called logical form – of EUCD (including query and constraint) from (Raimbault et al., 2009) can easily mapped into the CG model using nested graphs (Chein et al., 1998). Second, in the situation where some element of a given class diagram can not be represented in the visual environment of EUCD, the CG model that provide a graphical environment (as graph theory meaning) seems more preferable than OCL that is a textual language. This second reason also concerns the use of the CG model instead of “classical” theorem prover because the visual representation of CGs is more intuitive to use than logical formulas. Third, Cogitant includes operators that can be used to validate our work: queries and constraints.
This paper is organized as follows. Section 2 gives a short introduction to UML class diagram. Section 3 recalls the logical form of EUCD and the definition of queries and constraints as kind of EUCD from our previous works. Section 4 indicates how to integrate EUCD into the model of conceptual graphs.

2 UML CLASS DIAGRAM

Class diagram shows statical structure of a system with graphical notations. It describes the system’s elements (especially classes), their features and their relationships to other system’s elements. In a class diagram, classes are modeled and are linked by two types of relations: generalization and association.

Using the example in Figure 1, we briefly present the main notations of a UML class diagram. A class (e.g. Customer) is drawn as a solid-outline rectangle. It contains the name of the class in the top compartment, the attributes (e.g. name) in the middle compartment, and the operations (e.g. bill) in the bottom compartment. The generalization relation is represented by an arrowed line drawn from the specialized class to the general class. In Figure 1, the class FrequentFlyer has for generalization the class Customer, and the class Customer has for subclasse the classe FrequentFlyer. An association between the classes Flight and Customer is defined by the association class travel and by the properties present at the ends of the association, like the multiplicity 1..N or the role person. The association class, which is a kind of class, is shown as a class symbol (the top compartment is only represented here) linked by a dashed line to a line that represent the association.

Note that we consider a UML class diagram as a finite set of UML notations. We call a UML notation a pair (concept, element), where concept is one of the symbols that constitute the UML class diagram language, and element is an instance of this concept. For example, in Figure 1 Flight is an element of the concept ‘class’. We divide concepts in three sets: the set of entities, the set of relations and the set of properties. An entity is used to refer to general concepts such as: class, attribute, operation, etc. A relation is either a generalization or an association between classes. A property provides more precisely a meaning, like visibility or multiplicity.

3 PREVIOUS WORKS

The presented work in this paper is based on our previous works (Raimbault et al., ) and (Raimbault et al., 2009). In (Raimbault et al., ) is proposed a first way for querying and checking UML class diagram by using the model of conceptual graphs. (Raimbault et al., 2009) gives a simplification and an explicit formalization in FOL of the implicit logical semantics using in (Raimbault et al., ) to transforme class diagrams into conceptual graphs.

We now recall the main contributions from (Raimbault et al., 2009): logical form of a class diagram, an extension of class diagram with variables and bicoloration, and queries and constraints.

3.1 Logical Form of UML Class Diagram

In our previous work (Raimbault et al., 2009), we associate a FOL semantics, called Logical Form (LF), of a given UML class diagram. The predicate set that are used expresses the UML meta-model (OMG, c).

In a class diagram, an element may be described by some others: these other elements are enclosed into it. For instance, a class is described by its attributes and its operations, which are nested into the class representation. We call the entity that encloses the others the context of them. In light of those observation and rather than other approaches to transforme UML class diagram into FOL formula – e.g. (Beckert et al., 2002) –, in our approach a predicate that represents an entity (respectively a relation or a property) is a binary predicate (respectively a ternary predicate): the first one argument is used to represent the context of an element.

Logical Form. The Logical Form $\Phi_D$ of a UML class diagram $D$ is a FOL formula of conjunction of
FrequentFlyer
to different kind of UML notations that can be used in
different LFs is the LF of class diagram in Figure 1, based on par-
tomer (i.e. an instance of a concept). According to
different kind of UML notations that can be used in D, ΦD is obtained as follows:
• an instance e of an entity E is represented by the binary predicate E(cᵦ, idₑ) where cₑ is the context of e and idₑ is the identifier of e (see Φ₁ as example). The entity predicate set is partially ordered in Figure 2.
• a generalization link between the more general class C₁ and the more specific class C₂ is represented by the ternary predicate generalization(cₑ, id₁, id₂) where cₑ is the context of the generalization link, id₁ and id₂ are respectively the identifiers of C₁ and C₂ (see Φ₂ as example).
• a association link among classes C₁, . . . , Cₙ is in UML - a group of several association ends. The association link, strictly speaking, is represented as an association class, i.e. an instance of the entity assoClass (see first item); The j-th association end is represented by the ternary predicate assoEnd(cᵦ, idₑ, idₗ) where cᵦ is the context of the association link, idₑ is the identifier of class Cₑ and idₗ is the identifier of role that is associated to Cₗ according to this association link; The group is composed by the use of the ternary predicate association(cᵦ, idᵦ, idᵣ) where idᵦ is the identifier of the association link (i.e. the association class); (see Φ₃ as example).
• a value v for the property P that is applied to an entity or a relation a is represented by the ternary predicate P(cₐ, idₐ, v) where cₐ is the context of a and idₐ is the identifier of a (see Φ as example). The property predicate set is partially ordered in Figure 3.

Example: Φ₁ is the LF of the part of class diagram in Figure 1 amount to the class Customer, Φ₂ is the LF of the generalization link between FrequentFlyer and Customer, Φ₃ is the LF of the association travel. Finally, Φ is the LF of class diagram in Figure 1, based on partial LFs Φ₁ to Φ₃ that are competed w.r.t. appropriate properties.

\[ \Phi_1 = \text{package}(\top, \text{pkg}) \land \text{class}(\text{pkg, customer}) \land \text{attribute}(\text{customer, name}) \land \text{operation}(\text{customer, bill}) \land \text{parameter}(\text{bill, f}) \]
\[ \Phi_2 = \text{generalization}(\text{pkg, customer, frequentflyer}) \]
\[ \Phi_3 = \text{assoClass}(\text{pkg, travel}) \land \text{role}(\text{pkg, assigflight}) \land \text{role}(\text{pkg, person}) \land \text{assoEnd}(\text{pkg, flight, assigflight}) \land \text{assoEnd}(\text{pkg, customer, person}) \land \text{association}(\text{pkg, travel, assigflight}) \land \text{association}(\text{pkg, travel, person}) \]
\[ \Phi = \Phi_1 \land \text{named}(\text{pkg, customer, "Customer"}) \land \text{propVal}(\text{pkg, "Customer"}) \land \text{ofType}(\text{bill, f, flight}) \land \ldots \land \Phi_2 \land \Phi_3 \land \text{multiplicity}(\text{pkg, person, 1-N}) \land \ldots \]

First, note that the more general context is represented by the constant \( \top \). Second, note that each element is identified by a constant: for visibility reasons, in our example an identifier is the element’s name prefixed by its full context (but identifiers may be numbers as \( id₁, id₂, etc. \)). Third, note that named is a property that makes the link between an identifier of an entity and its real name in a class diagram. Fourth, note that predicate propVal is used to define a property’s value as an entity itself.

### 3.2 Querying and Checking EUCD

In (Raimbault et al., 2009), we have proposed an extension of UML class diagram, called extended UML class diagram (EUCD), using variables and/or bicoloration. Then, we have defined two family of extended UML class diagrams: queries and constraints. Queries are used to find some elements, and constraints to check some other. We recall in this section how to query and check UML class diagram.

**Variable and Mapping.** Any element in a EUCD can be a variable. It seems that the element exists whereas it is not identified (a temporary identifier is used: the variable’s name). The new visual notation associated of a variable \( x \) is denoted in EUCD \( D \) by the symbol \( S_x \). In \( \Phi_D \), the existential quantified variable \( x \) is used.

A mapping from a EUCD \( D_1 \) to a EUCD \( D_2 \) is a minimal subset \( D_2' \) of \( D_2 \) such as knowledge modeled by \( D_1 \) can be inferred by knowledge modeled by \( D_2' \).
Querying UML Class Diagram. A query is a EUCD (see left part of Figure 4 for instance). The result(s) of a query \( Q \) to interrogate a EUCD \( D \) is the set of mappings from \( Q \) to \( D \).

Example: The query in Figure 4 expresses the fact that “In a given package, does a class named Customer has a subclass that has an attribute of type int?” The LF of this query is:

\[
\Phi_Q = \exists x,y,z \text{ package}(\top, x) \land \text{class}(x, \text{id}_1) \land \text{named}(x, \text{id}_1, “Person”) \land \text{propVal}(x,”Person”) \land \text{class}(x,y) \land \text{generalization}(x, \text{id}_1, y) \land \text{attribute}(y,z) \land \text{visibility}(y,z, “private”) \land \text{propVal}(\top,”private”) \land \text{ofType}(y,z,\text{int}) \land \text{primType}(\top,\text{int}).
\]

The answer of this query to interrogate class diagram on Figure 1 is presented on the right side in Figure 4. One can see that the intended class is Frequentflyer: into package Pkg with the attribute bonusPoints.

Checking UML Class Diagram. Two kind of constraints are defined: negative and positive constraints. A negative constraint \( C^- \) expresses a specification like “no A exists” in a class diagram D: no mapping from \( C^- \) to \( D \) must be found. A negative constraint is represented by a EUCD. A positive constraint \( C^+ \) expresses a specification like “if premise A, then obligation B”. In other words, if a mapping from the premise of \( C^+ \) to a class diagram \( D \) (which has to be check by \( C^+ \)) exists, then this mapping must be extend from whole \( C^+ \) to \( D \). A positive constraint is represented by a EUCD where premise is white background colored and obligation is black background colored.

Example: Figure 5 shows a negative constraints \( C^-_1 \) and a positive constraints \( C^+_2 \). \( C^-_1 \) expresses the following prohibition: “a class \( X \) can be a subclass of a class \( Y \), which is a subclass of \( X \)”. In other words, this constraint checks that “there is no simple inheritance cycle”. \( C^+_2 \) expresses the following obligation: “if a class \( X \) has an abstract operation \( z \), for each subclass \( Y \) of \( X \) that is non-abstract, \( z \) must necessarily be overwritten as a non-abstract operation”.

According to constraints \( C^-_1 \) and \( C^+_2 \), the class diagram in Figure 1 is valid.

4 INTREGRATION INTO CONCEPTUAL GRAPHS

In this article, we use the CG model from (Chein and Mugnier, 1992) with nested graphs (Chein and Mugnier, 1997; Chein et al., 1998), coreference links (Chein and Mugnier, 2004) and constraints (Baget et al., 1999; Baget and Mugnier, 2002). This CG model presents many interests. It is a formal and visual knowledge representation model. It provides a reasoning operator, called projection, which is sound and complete with respect to deduction in FOL (see (Sowa, 1984; Chein and Mugnier, 1992) for simple graphs, (Chein et al., 1998) for nested graphs. We use the CG model instead of “classical” theorem prover because in case of necessity to represent some knowledge that can not be represented in UML, the visual aspect of the CG model is more intuitive to use than logical formulas.

4.1 A Short Introduction to CGs

In the CG model, a support specifies the vocabulary representing ontological knowledge. It describes available concept and relation types ordered by a specialization relation. Facts are represented by conceptual graphs, which nodes represent individual instances of concepts and relations between them. The fundamental reasoning operation is a graph morphism called projection. It induces a subsumption relation between graphs.

Support. We consider here a support \( S = (T_C, T_R, I) \). \( T_C \) is a partially ordered set of concept types whose greatest element is \( \top \). \( T_R \) is a partially ordered set of relation types. For the paper, we only use binary relations whose greatest element is \( \top \) (but in general relations may be of any arity greater or equal to one). Each relation type has a
signature, which gives the greatest possible concept type for each argument. \(I\) is a countably infinite set of individual markers.

**Nested Conceptual Graphs.** A nested conceptual graph (NCG), related to a support \(S\), is a bipartite labelled multigraph \(G = (C, R, E, I)\). \(C\) is the set of concept nodes, \(R\) the set of relation nodes, and \(E\) the set of edges. Each label is given by the mapping \(l\). The label of a relation node is a relation type from \(T_R\), and the label of a concept node is composed of a concept type from \(T_C\), a marker from \([I ∪ \{\ast\}]\) (if \(\ast\), the node represents a generic individual, else a specific individual) and a description, which may be either sets of NCG or empty (refered by \(*\ast\) in (Chein and Mugnier, 1997; Chein et al., 1998)). For a graph \(G\), which is in a description of a concept node \(c\) of another graph, one says that \(G\) is nested into \(c\). Labels on concept nodes must respect the signature defined for their relation neighbors. Every edge is labeled by 1 or 2 (in the case of binary relations), for indicating which is the first and the second argument of each relation node.

### 4.2 EUCD into the CG Model

We first show how to translate an extended UML class diagram (EUCD) into a CG: the ordered sets of UML concepts are defined in a support of CGs; the translation of a given UML class diagram in a nested CG is presented (see (Raimbault et al., ) for details). Second, we describe how to translate queries and constraints into the CG model. Once we have it, we use the projection operator for making reasoning (querying and checking).

#### 4.2.1 LF and Nested Conceptual Graphs

**Proposition 1 (EUCD into CG).** A EUCD can be translated into a nested CG.

UML concepts are translated into a support \(S\). A EUCD is translated into a nested CG based on \(S\). We define \(S = (T_C, T_R, I)\) as follows:

- The set of concept types \(T_C\) is the set of entity predicates (Figure 2). Then the partial order between concept types is the partial order between entities, i.e. entity is the greatest concept type.
- The set of relation types \(T_R\) is composed of the set of property predicates (Figure 3) in accordance with partial order between them, plus the predicates generalization, assoEnd and association. Property, generalization, assoEnd and association have a greater common relation type \(T_2\).
- The set of individual markers \(I\) is the set of identifiers of elements.

Let \(D\) be a EUCD, we build a nested CG \(T(D)\) based on \(S\) such as by construction the FOL formula of \(T(D)\) (Chein and Mugnier, 1997; Chein et al., 1998; Chein and Mugnier, 2004) is equivalent to \(\Phi_D\). \(T(D)\) is the nested CG \(NCG(D, \top)\), which is recursively defined by \(NCG(D, c)\) as follows:

- For each \(P(c, t_1) \in \Phi_D\), a new concept node \(C_{t_1}\) is created. \(C_{t_1}\) is labelled by (type, marker, description); type is \(P\); marker is either \(t_1\) if \(t_1\) is a constant or \(\ast\) if \(t_1\) is a variable; description is either \(NCG(D, c)\) if \(NCG(D, t_1)\) is not empty or \(*\ast\).
- For each \(P(c, t_1, t_2) \in \Phi_D\), a new relation node is created between \(C_{t_1}\) and \(C_{t_2}\). This relation node is labelled by \(P\).

After this translation, coreference links are created between concept nodes \(C_x\) for each variable \(x\) of \(\Phi_D\).

#### 4.2.2 Queries and Constraints

**Proposition 2 (UML Query and Projection).** Let \(D\) be a UML class diagram and \(Q\) a EUCD. There is a mapping from \(Q\) to \(D\) iff there is a projection from \(\mathcal{T}(Q)\) to \(\mathcal{T}(D)\).

**Example:** Figure 6 presents in the CG model the example on Figure 4: on the left side is the translated CG \(\mathcal{T}(\text{UML query})\), on the right side a part of the translated CG \(\mathcal{T}(\text{class diagram Fig. 1})\). The projection from \(\mathcal{T}(\text{UML query})\) to \(\mathcal{T}(\text{class diagram Fig. 1})\) is the
application from the labelled nodes of $\mathcal{T}(\text{UML query})$ into labelled nodes of $\mathcal{T}(\text{class diagram Fig. 1})$ that may decrease node labels (Chein and Mugnier, 1997; Chein et al., 1998; Chein and Mugnier, 2004). A concept node is represented by a rectangle [$\text{typeConcept}$] or [$\text{typeConcept}$: $\text{D}$] that may have a nested CG into it; a relation node is represented by an oval ($\text{relationConcept}$).

**Proposition 3 (UML Constraint and CG Constraint).** A UML positive constraint (resp. negative constraint) is translated into a CG positive constraint (resp. CG negative constraint)\(^2\) by $\mathcal{T}$ that is extended such as the coloration is preserved (resp. each node is one colored.).

Let $D$ be a UML class diagram and $C$ a UML constraint. $D$ verifies $C$ iff $\mathcal{T}(D)$ verifies (Baget et al., 1999; Baget and Mugnier, 2002) $\mathcal{T}(C)$.

A prototype was developed to make an interface between a EUCD and a CG by using Cogitant (Genest, 2008). It translates UML class diagram, UML query and UML constraint into CGs. Cogitant operators (based on projection) compute results of queries and verify constraints on CGs. This interface makes the visual link between CGs and UML class diagram to display results computed by Cogitant (Raimbault et al.,) on UML class diagram.

### 5 Future Work

We have proposed a way to translate UML queries and UML constraints — both as EUCD — into the CG model, and so use projection to query and check class diagram. By using inverse translation, results can be displayed in the visual representation of UML.

Due to the importance of visual representation of knowledge, see for instance (OMG, b; Lukichev and Wagner, 2006), we will now focus our attention on how to represent rules into the UML visual environment by using bi-coloration (white background for head and black background for conclusion) and how to infer them by using the CG model.

### REFERENCES


OMG. Production rule representation. Technical report. br/2003-09-03.


