A USEFUL LOGICAL SEMANTICS OF UML FOR QUERYING AND CHECKING UML CLASS DIAGRAM

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

Keywords: UML, First Order Logic, Knowledge Representation and Reasoning, Model-Based Reasoning, Knowledge Engineering, Visual Quering, Visual Checking, Positive Constraints, Negative Constrains.

Abstract: In Knowledge Engineering, UML class diagram is the defacto standard for modeling object oriented systems. We propose a way for logical reasoning on UML class diagram, concerning *querying* and *checking* class diagram. First, we define an original *logical semantics* to UML class diagram. Our approach differs from other existing works, because we use a same set of predicates to translate any class diagram instead of other "ad hoc" approaches. Second, we extend UML, especially with *variable* and *bicoloration*, to express query and constraint into the visual environment of (extended-)UML.

1 INTRODUCTION

In Knowledge Engineering, several models and tools are used to represent and manipulate knowledge, e.g. (Akkerman et al., 1999). For modeling object oriented systems, the Unified Modeling Language (Booch et al., 1998) (UML) has been widely accepted as standard. Currently, UML provides an object oriented language for modelling knowledge by using diagrams of various kinds. However, UML is merely a language, then knowledge is only represented as a drawing and no reasoning way is available. In this article, we focus our attention on *class diagram* that is the main UML diagram.

During the knowledge acquisition phase, the designer need to query and to check knowledge. The Object Constraint Language (OMG, a) (OCL) gives a start of solution to express constraints to be checked and queries. OCL is an integral part of UML2 (OMG, c; OMG, b), it is a textual language that provides a way to express specific constraints on object oriented models in addition to diagrammatic notations. However, OCL is only a language with few reasoning tools and without visual representation.

The aim of this work is to provide a way for logical reasoning on UML class diagram, concerning querying and checking class diagram. We extend UML both to associate an original logical semantics of UML class diagram and to express queries and constraints into the visual environment specific to (extended-)UML.

For this purpose, we propose two contributions. First, we define for UML class diagram an associated *Logical Form* (LF), which is based on first order predicate logic (FOL). Translating UML class diagram into FOL (Beckert et al., 2002) or more generally into languages with inference power (Soon-Kyeong and Carrington, 2000; Berardi et al., 2005) [1] is not a new idea. *However, all these approaches are ad hoc approaches*: for each UML class diagram, a specific set of predicates is defined. For instance, in (Beckert et al., 2002) a class Person is translated into a predicate called Person. This predicate is useful for logical reasoning on only this given class diagram that contains the class Person. Unfortunately, it is impossible in FOL to refer to a given predicate using a variable. For example, it is impossible to query the class diagram like "Is there a class that has an attribute of string type?". The originality of our approach is to consider the different notations that compose the UML class diagram language as ordered set of FOL predicates. Then, we translate specific information of a specific class diagram as constants included in FOL formulas. In other words, each UML class diagram can be translated into FOL by using the same set of predicates, and the specifics of a given class diagram are given by using constants. Thus, our approach makes pos-

[1] About using Description Logics, like in (Berardi et al., 2005), (Rosati, 2007), the difficulty of querying knowledge within this logic is discussed.

Raimbault T., Genest D. and Loiseau S. (2009).
A USEFUL LOGICAL SEMANTICS OF UML FOR QUERYING AND CHECKING UML CLASS DIAGRAM.
In *Proceedings of the International Conference on Agents and Artificial Intelligence*, pages 179-184
DOI: 10.5220/0001556101790184

sible general inferences on the LF of any UML class diagram.

Second, we *extend UML class diagram* with generic elements, i.e. *variables*. Indeed, to construct a reliable model the designer needs both to query it and to check it. On the one hand, variables are required to express a *query* in UML class diagram: a query is an incomplete representation of knowledge whose some parts must be identified. On the other hand, variables are required to formulate *constraints* for *checking* if a class diagram can be considered as valid. We integrate queries and checks in the visual representation of UML, and we associate a logical semantics to these notions.

This paper is organized as follows. Section 2 defines the logical semantics of UML class diagram. Section 3 presents our extension to query and check UML class diagram by using variables.

2 UML CLASS DIAGRAM INTO FIRST-ORDER LOGIC

2.1 A Short UML Class Diagram Description

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

We present with an example, in Figure 1, the main notations of a UML class diagram. A class (e.g. Pilot) is drawn as a solid-outline rectangle. It contains the name of the class in the top compartment, the attributes (e.g. licenseDate) in the middle compartment, and the operations (e.g. fly) in the bottom compartment. This operation has one parameter, which is an instance of the class Plane. The operation bonusPoints of the class FrequentFlyer returns a data type int. The generalization relation is represented by an arrowed line drawn from the specialized class to the general class. Then, the class Pilot has for generalization the Person class, and the Person class has for subclasses the classes Pilot, Customer and indirectly FrequentFlyer. An association between the classes Pilot and Plane is defined by the central Flight association class and by the properties present at the ends of the association, like the multiplicity 0..N. The association class, which is a kind of class, is shown as a class symbol linked by a dashed line to a line symbol of association. The association between the classes Flight and Customer is

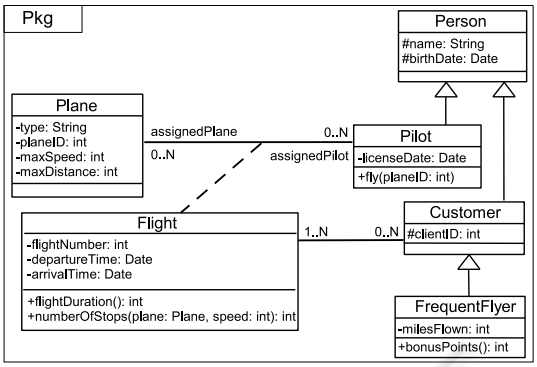

Figure 1: Example of a UML class diagram.

a more simple link that not explicitly needs an association class.

2.2 Logical Form

We specify some terms that are used in this paper. 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 Plane 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.

We recall that in a class diagram an element may be described by some others. Then, these elements are enclosed into it. For instance, a class is described by its attributes and its operations. The entity that encloses the others is the *context* of them.

Definition 1 (Logical Form). *The* Logical Form Φ_D *of a UML class diagram D is a FOL formula of conjunction of predicates. A* predicate *in* Φ_D *represents a concept of the UML class diagram language. A* constant *in* Φ_D *refers to an element of D.* Φ *is obtained according to* Definitions 2 to 6.

In a UML class diagram some properties of elements are implicit. For instance, if nothing is specified, a class is by default not abstract. Please notice that we explicitly express all implicit information from a class diagram into its LF.

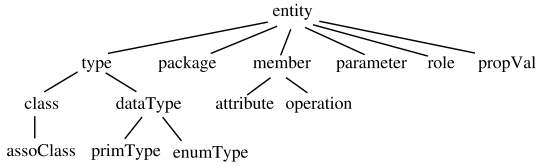

Figure 2: Set of entities.

2.2.1 Entities in Class Diagram

Definition 2 (**LF of Entities).** *Let D be a UML class diagram. A UML notation (*E, e*) of D, where* e *is an instance of an entity* E*, is represented in* Φ_D *by a binary predicate* E*. The first term of* E *is a constant, called the* context *of* e*, and the second term is a constant, called the* identifier *of* e*.*

The entity predicates are partially ordered by a kind-of *relation, denoted by* $\prec$ *(Figure 2).*

Example: The LF of the class Pilot is Φ_1 = *package(*$\top$*, Pkg)* $\wedge$ *class(Pkg, Pkg.Pilot)* $\wedge$ *attribute(Pkg.Pilot, Pkg.Pilot.licenseDate)* $\wedge$ *operation(Pkg.Pilot, Pkg.Pilot.fly)* $\wedge$ *parameter(Pkg.Pilot.fly, Pkg.Pilot.fly.planeID)*. The predicates *class*, *attribute*, *operation* and *parameter* represents some entities. Φ_1 indicates that Pilot, with the single identifier Pkg.Pilot, is a *class*, licenseDate is an *attribute*, fly is an *operation*, and planeID is a *parameter* of fly. All of these elements are defined in package Pkg. Remember that each element is enclosed in a context, then the context of Pilot is Pkg, the one of both licenseDate and fly is Pilot and the one of planeID is fly.

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 id_1, id_2, etc.). Third, Φ_1 will be completed in Section 2.2.3 with some properties of these entities.

2.2.2 Relations in Class Diagram

There are two kinds of relations between classes: generalization and association.

Definition 3 (**LF of Generalization).** *Let D be a UML class diagram with notations (*class,C_1*), (*class,C_2*) and (*generalization link,g*), such as* g *links the more* general *class* C_1 *and the more* specific C_2*. The notation (*generalization link,g*) is represented in* Φ_D *by a ternary predicate* generalization*, where the first term is the context of* g*, the second term is the identifier of* C_1 *and the third term is the identifier of* C_2*.*

Example: Descendants of Person are defined by Φ_2 = *generalization(Pkg, Pkg.Person, Pkg.Pilot)* $\wedge$ *generalization(Pkg, Pkg.Person, Pkg.Customer)* $\wedge$ *generalization(Pkg, Pkg.Customer, Pkg.FrequentFlyer)*.

We have always chosen to represent a UML association relation as its most complete form, i.e. by using an association class. Then, each association from a class diagram can be expended to an association that is centralized by an association class.

Definition 4 (**LF of Association).** *Let D be a UML class diagram with notations (*class,C_1*), (*class,C_2*), (*association class,A*), (*role name,r_1*) and (*role name,r_2*) such as:* C_1 *and* C_2 *are linked together through* A*;* r_1 *and* r_2 *are the two ends of* A *and are respectively combined with* C_1 *and* C_2*. An element* r_i *is an entity represented in* Φ_D *by a binary predicate* role*. A role* r_i *and a class* C_i *are combined in* Φ_D *by using a ternary predicate* assoEnd*, where the first term is the context of* A*, the second is the identifier of* C_i *and the third is the identifier of* r_i*. Then, a role* r_i *and* A *are linked in* Φ_D *by a ternary predicate* association*, where the first term is the context of* A*, the second is the identifier of* A *and the third is the identifier of* r_i*.*

Example: Association between Pilot and Plane: Φ_3 = *assoClass(Pkg, Pkg.Flight)* $\wedge$ *role(Pkg, Pkg.assignedPilot)* $\wedge$ *role(Pkg, Pkg.assignedPlane)* $\wedge$ *assoEnd(Pkg, Pkg.Pilot, Pkg.assignedPilot)* $\wedge$ *assoEnd(Pkg, Pkg.Plane, Pkg.assignedPlane)* $\wedge$ *association(Pkg, Pkg.Flight, Pkg.assignedPilot)* $\wedge$ *association(Pkg, Pkg.Flight, Pkg.assignedPlane)*. The constants *Pkg.assignedPilot* and *Pkg.assignedPlane* identify the two ends of the association, which is centralized by the association class Flight.

Please first remark that the Definition 4 can be easily extended for n-ary associations (n$\geq$ 2). Second, note that Φ_3 will be completed in Section 2.2.3 with properties.

2.2.3 Properties of Elements

Definition 5 (**LF of Properties' Elements).** *Let D be a UML class diagram with a notation (*P, (a,v)*), where the value for property* P *that is applied to the entity*[2] *or relation* a *is* v*. Notation (*P, (a,v)*) is represented in* Φ_D *by a ternary predicate* P*, where the first term is the context of* a*, the second term is the identifier of* a *and the third term is the identifier of the notation (*propVal,v*) in the same context.*

The property predicates are partially ordered by a kind-of *relation, denoted by* $\prec$*, whose greatest predicate is* property *(Figure 3).*

[2]except *propVal* of course

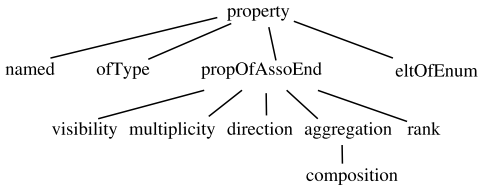

Figure 3: Set of properties.

Please observe that *named* is a property that makes the link between an identifier of an entity and its real name in the class diagram.

Example: The example of the LF Φ_1 should now be completed: Φ'_1 = Φ_1 $\wedge$ ... $\wedge$ *named(Pkg, Pkg.Pilot, Pilot)* $\wedge$ *propVal(Pkg, Pilot)* $\wedge$ ... $\wedge$ *visibility(Pkg.Pilot, Pkg.Pilot.licenseDate, private)* $\wedge$ *propVal(Pkg.Pilot, private)* $\wedge$ *multiplicity(Pkg.Pilot, Pkg.Pilot.licenseDate, 1-1)* $\wedge$ *propVal(Pkg.Pilot, 1-1)* $\wedge$ ... $\wedge$ *rank(Pkg.Pilot.fly, Pkg.Pilot.fly.planeID, #1)* $\wedge$ *propVal(Pkg.Pilot.fly, #1)*. This completed LF expresses also the fact the private attribute licenseDate has a multiplicity of 1-1 and it expresses that the parameter planeID is the first parameter of the operation fly. In the same way, Φ'_3 = Φ_3 $\wedge$ *named(Pkg,Pkg.assignedPlane,assignedPlane)* $\wedge$ *propVal(Pkg, assignedPlane)* $\wedge$ ... $\wedge$ *multiplicity(Pkg, Pkg.assignedPilot, multiplicity.0-N)* $\wedge$ *propVal(Pkg, 0-N)*.

Definition 6 (**Partial Ordering in Set of Concepts).** *A FOL formula* Φ_{UML} *to the set of UML concepts is assigned. It corresponds to the interpretation of the partial orderings of set of entities (Figure 2) and set of properties (Figure 3). For all predicates* p_1 *and* p_2 *such as* $p_2 \prec p_1$*, the formula* $\forall x_1 \ldots x_n (p_2(x_1 \ldots x_n) \rightarrow p_1(x_1 \ldots x_n))$ *is assigned, where* $n = 2$ *for entity predicates and* $n = 3$ *for property predicates.*

In Figure 2 or 3 the order between two concepts is represented by a link between themselves such as the more general concept is above the more specific concept. For instance, the ordering *assoClass* $\prec$ *class* in Figure 2 where *class* is the more general entity means that, in a given context an association class is also a class: $\forall c,x$ *assoClass(c,x)* $\rightarrow$ *class(c,x)*.

3 EXTENDING UML TO QUERY AND TO CHECK CLASS DIAGRAM

In addition to UML as a model for representing knowledge, we propose a way for querying and checking knowledge. Queries are used to find some elements, and constraints to check some other. We indicate in this section how to query and check UML class diagram. Two new visual notations are introduced into the UML model: variables and coloration.

3.1 Variables and Mappings

Definition 7 (**Extended UML Class Diagram).** *Let D be a UML class diagram. A variable* x *of D is denoted by the symbol $x. An entity notation (E,$x) of D is represented in* Φ_D *as it is defined in Proposition 1; but the second term of predicate* E *is* x*. All variables are existentially quantified in* Φ_D*.*

We call an extended UML class diagram *a UML class diagram that may have variables.*

Asking for the existence of a *mapping* from an extended class diagram *Q* to a class diagram *D* can be seen as a search such as all information contained in *Q* is also contained in *D*.

Definition 8 (**Mapping).** *Let Q be an extended UML class diagram and D a UML class diagram.*

A subset D' *of D is called a* mapping from *Q* to *D if* $\Phi_{UML}, \Phi_{D'} \vDash \Phi_Q$ *and* $\forall\, D'' \subset D', \Phi_{\text{UML}}, \Phi_{D''} \nvDash \Phi_Q$.

3.2 Querying UML Class Diagram

Definition 9 (**Result of a UML Query).** *A* UML query *is an extended UML class diagram. Let D be a UML class diagram and Q a UML query. The result of the query Q to D is the set of mappings from Q to D.*

In a general way, a designer is interested in the correspondence between each variable from *Q* and the identifiers of instances of entities in a subset of *D* according to a given mapping from *Q* to *D*.

Example: The LF of the query (Figure 4) is Φ_Q = $\exists p,c,x,y,z$ *class(p,c)* $\wedge$ *named(p,c,Person)* $\wedge$ *propVal(p,Person)* $\wedge$ *class(p,x)* $\wedge$ *generalization(p,c,x)* $\wedge$ *operation(x,y)* $\wedge$ *visibility(x,y,public)* $\wedge$ *propVal(x,public)* $\wedge$ *parameter(y,z)* $\wedge$ *ofType(y,z,int)* $\wedge$ *primType(y,int)*. This query expresses the fact that "In a package, does a class named by the value Person has a subclass that has an operation with public visibility and with one parameter of type called int?". The result of this query to the class diagram of Figure 1 is presented on the right side in Figure 4. The designer can deduce that the intended class is Pilot with the operation fly that has the parameter planeID.

Theorem 1. *Let D be a UML class diagram and Q an extended UML class diagram.* $\mathcal{M}$ *is the result of query Q to D iff (1)* $\forall M \in \mathcal{M}\ \Phi_{UML}, \Phi_M \vDash \Phi_Q$*; and (2)* $\forall M \in \mathcal{M}, \nexists N \subset M$ *such as* $\Phi_{UML}, \Phi_N \vDash \Phi_Q$.

Proof. Let $\mathcal{S}$ be the set of subset of *D*. ($\Rightarrow$) Let $\mathcal{M}$ the result of *Q* to *D*. Let D' *a* mapping from *Q* to *D*. (1)

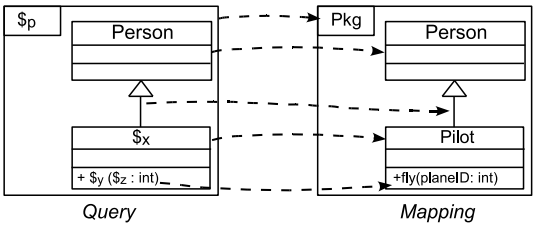

Figure 4: A UML query and a mapping.

Then, $D' \in \mathcal{M}$ and $\Phi_{\text{UML}}, \Phi_{D'} \vDash \Phi_Q$ (by hypothesis). (2) Let $D'' \in \mathcal{S}$ where $D'' \subset D'$, then $\Phi_{\text{UML}}, \Phi_{D''} \nvDash \Phi_Q$ (by hypothesis).

($\Leftarrow$) Let $M \in \mathcal{S}$ where $\Phi_{\text{UML}}, \Phi_M \vDash \Phi_Q$ and $\forall N \in \mathcal{S}$ where $N \subset M$, $\Phi_{\text{UML}}, \Phi_N \nvDash \Phi_Q$, then $\nexists N' \in \mathcal{S}$ where $N' \subset M$, $\Phi_{\text{UML}}, \Phi_{N'} \vDash \Phi_Q$, so $M \in \mathcal{M}$. □

3.3 Checking UML Class Diagram

A class diagram is valid with respect to a set of constraints if it verifies each constraint. A constraint is either *negative* or *positive*.

A negative constraint expresses an extended class diagram that must not be found in the class diagram to validate. A positive constraint expresses a specification like "if A, then obligation B". *Coloration* is used to distinguish premise from obligation.

Definition 10 (**UML Negative Constraint).** *A* UML negative constraint *is an extended UML class diagram.*

Definition 11 (**UML Positive Constraint).** Coloration *is an application that associates a color* 0 *or* 1 *with each notation of an extended UML class diagram. A* UML positive constraint C^+ *is a colored extended UML class diagram, which is divided into two parts: a* premise *and an* obligation. *The premise is the subset with* 0*-colored notations (white background) and obligation the subset with* 1*-colored notations (black background).*

Without considering coloration: C_p^+ is called the premise of C^+, $\Phi_{C_p^+}$ its LF, and Φ_{C^+} the LF of C^+. Notice that Φ_{C^+} is not the logical semantics (due to coloration) of C^+ [3].

Example: Figure 5 shows two negative constraints C_1^- and C_3^-, and two positive constraints C_2^+ and C_4^+. C_1^- expresses the following prohibition: "a class X can be a subclass of a class Y, which is a subclass of

[3]Let Ψ_p be defined as $\Phi_{C_p^+}$ without existential quantifiers, and Ψ_o as the LF of the obligation without existential quantifiers. Let $x_1,..,x_n$ be the variables of Ψ_p and $y_1,..,y_m$ the variables of Ψ_o that are not in Ψ_p. The *logical semantics* of C^+ is as $\forall x_1 \ldots \forall x_n \Psi_p \rightarrow \exists y_1 \ldots \exists y_m \Psi_o$.

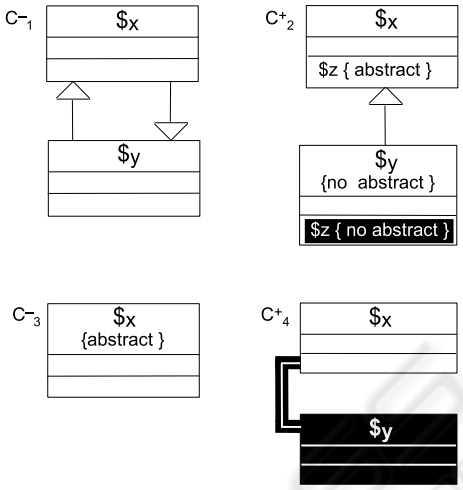

Figure 5: UML positive and negative constraints.

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". C_3^- expresses that "abstract classes" are prohibited. C_4^+ expresses that "each class has to be associated with another class".

Please observe that we consider constraints to be divided into two groups: *object oriented constraints* and *specific constraints*. The first group refers to constraints for verifying if class diagram respects the object oriented specifications. The second group may express any requirement in addition to the object oriented specifications that the designer needs according to corporate project specifications. C_1^- and C_2^+ are object oriented constraints, and C_3^- and C_4^+ specific constraints.

Definition 12 (**Verification of UML Constraint).** *Let D be a UML class diagram, C^- a UML negative constraint, C^+ a UML positive constraint.*

D verifies C^- *if there is no mapping from C^- to D.*

D verifies C^+ *if for each mapping M from the premise of C^+ to D, there is a mapping M' from C^+ to D such as $M \subseteq M'$.*

Example: The class diagram in Figure 1 is valid according to constraints C_1^-, C_2^+ and C_3^-, but it is not valid according to C_4^+. Indeed, there is a mapping from the premise of C_4^+ to the class Person, but there is no mapping from (whole) C_4^+ to the class diagram concerning Person (no association).

Theorem 2. *Let D be a UML class diagram and C^- a UML negative constraint. D verifies C^- iff $\Phi_{UML}, \Phi_D \nvDash \Phi_{C^-}$.*

Proof. ($\Rightarrow$) Let $\mathcal{S}$ be the set of subset of D. There is no mapping from C^- to D, then $\nexists D' \in \mathcal{S}$ where $D' \subseteq D$ such as $\Phi_{\text{UML}}, \Phi_{D'} \vDash \Phi_{C^-}$ (by hypothesis). So, $\Phi_{\text{UML}}, \Phi_D \nvDash \Phi_{C^-}$, because $D \subseteq D$.

($\Leftarrow$) Immediate from hypothesis. □

Theorem 3. *Let D be a UML class diagram, C^+ a UML positive constraint. D verifies C^+ iff for all substitution σ of all variables of C_p^+, if $\Phi_{UML}, \Phi_D \vDash \sigma(\Phi_{C_p^+})$, then there is a substitution σ' of all variables of C^+ where $\sigma \subseteq \sigma'$ and $\Phi_{UML}, \Phi_D \vDash \sigma'(\Phi_{C^+})$.*

Proof. ($\Rightarrow$) Let $\mathcal{M}_p$ be the set of mapping from C_p^+ to D; and $\mathcal{M}_c$ the set of mapping from C^+ to D. Let $M \in \mathcal{M}_p$, then there is a substitution σ_1 of all variables of C_p^+ such as $\Phi_{\text{UML}}, \Phi_D \vDash \sigma_1(\Phi_{C_p^+})$ (by hypothesis) and because there is no variable into Φ_D. $\exists M' \in \mathcal{M}_c$, then there is a substitution σ_1' of all variables of C^+ such as $\Phi_{\text{UML}}, \Phi_D \vDash \sigma_1'(\Phi_{C^+})$ and $\sigma_1 \subseteq \sigma_1'$ (because $M \subseteq M'$).

($\Leftarrow$) Let $\mathcal{S}$ be the set of subset of D. Let σ_1 a substitution of all variables of C_p^+ where $\Phi_{\text{UML}}, \Phi_D \vDash \sigma_1(\Phi_{C_p^+})$ (A), then there is a substitution σ_1' of all variables of C^+ where $\Phi_{\text{UML}}, \Phi_D \vDash \sigma_1'(\Phi_{C^+})$ (B). By (A), $\exists D' \in \mathcal{S}$, $\forall D'' \in \mathcal{S}$ where $D'' \subseteq D'$ such as $\Phi_{\text{UML}}, \Phi_{D'} \vDash \sigma_1(\Phi_{C_p^+})$ and $\Phi_{\text{UML}}, \Phi_{D''} \nvDash \sigma_1(\Phi_{C_p^+})$. By (B) and $\sigma_1 \subseteq \sigma_1'$ (by hypothesis), $\Phi_{\text{UML}}, \Phi_D \vDash \sigma_1'(\Phi_{C_p^+})$ and so $\Phi_{\text{UML}}, \Phi_D \vDash \sigma_1(\Phi_{C_p^+})$ (C). By (B) and (C), $\exists B' \in \mathcal{S}$ with $D' \subseteq B'$ and $\forall B'' \in \mathcal{S}$ where $B'' \subseteq B'$ such as $\Phi_{\text{UML}}, \Phi_{B'} \vDash \sigma_1'(\Phi_{C^+})$ and $\Phi_{\text{UML}}, \Phi_{B''} \nvDash \sigma_1'(\Phi_{C^+})$. □

4 CONCLUSION

We have proposed an original logical semantics to UML class diagram and an extention of UML to express queries and (positive and negative) constraints. Our approch is useful to query and to check UML class diagram for respectively answering queries and satisfying constraints that both can be modelled in extended UML as well. Notice that the definition in this article of constraints (using bicoloration) is inspired by constraints of (Chein and Mugnier, 1997) from the model of conceptual graphs (Chein and Mugnier, 1992; Wermelinger,). So, it could be interesting to translate logical form of UML class diagram into this model, and then use its reasoning operation (called *projection*) both to answer queries and to test constraints.

REFERENCES

Akkerman, H., Anjewierden, A., Hoog, R. D., Shadbolt, N., de Welde, W. V., and Wielenga, B. (1999). *Knowledge engineering and management: the CommonKads methodology*. MIT Press.

Beckert, B., Keller, U., and Schmitt, P. H. (2002). Translating the Object Constraint Language into First-order Predicate Logic. In *Proc.of VERIFY, Workshop at FLoC'02*.

Berardi, D., Calvanese, D., and De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1):70–118.

Booch, G., Jacobson, C., and Rumbaugh, J. (1998). *The Unified Modeling Language - a reference manual*. Addison Wesley.

Chein, M. and Mugnier, M.-L. (1992). Conceptual Graphs: Fundamental Notions. *Revue d'intelligence artificielle*, 6(4):365–406.

Chein, M. and Mugnier, M.-L. (1997). Positive nested conceptual graphs. In *Proc. of ICCS'97*, volume 1257 of *LNAI*, pages 95–109. Springer.

OMG. UML 2.0 Object Constraint Language Specification. http://www.omg.org/docs/ptc/03-10-14.

OMG. Unified Modeling Language Specification: Infrastructure, v2.0. http://www.omg.org/cgi-bin/doc?ptc/04-10-14.

OMG. Unified Modeling Language Specification: Superstructure, v2.0. http://www.omg.org/cgi-bin/doc?formal/05-07-04.

Rosati, R. (2007). The Limits of Querying Ontologies. In *Proceedings of 11th International Conference in Database Theory (ICDT'07)*, volume 4353 of *Lecture Notes in Computer Science (LNCS)*, pages 164–178. Springer.

Soon-Kyeong, K. and Carrington, D. (2000). A Formal Mapping between UML Models and Object-Z Specifications. In *Proc. of ZB '00*, volume 1878 of *LNCS*, pages 2–21. Springer.

Wermelinger, M. Conceptual Graphs and First-Order Logic. pages 323–337.