Description Logic Application for UML Class Diagrams Optimization

Maxim Sergievskiy
National Research Nuclear University MEPhI
Moscow, Russia

***Abstract*—Most of known technologies of object-oriented developments are UML-based; particularly widely used class diagrams that serve to describe the model of a software system, reflecting the regularities of the domains. CASE tools used for object-oriented developments, often lack verification and optimization functions of diagrams. This article will discuss one of the ways to present class diagram in the form of statements description logic, and then perform their verification, and optimization. Optimization process is based on design patterns and anti-patterns. We will show that some transformations could be done automatically, while in other cases suboptimal models need to be adjusted by a designer.**

***Keywords*—UML; domain models; description logic; concept; role; class diagram; design patterns; anti-patterns**

I. Introduction

UML has recently become the standard, widely applied method for software design and analysis [1]. Most of the technologies of object-oriented development (including DevOps and Agile) use wide toolset of this language. At a design stage, the most widely applied tools of UML are class diagrams (CD). The main advantages of UML are high expressiveness and declarative nature, the richness of the structures, which are negatively affecting the ability of automatic verification.

That is, checking whether the CD contains structural errors, in particular, incompatible components whether redundancy, how CD optimal from the standpoint of subsequent implementation difficult.

It is common, that possible mistakes at the design stage often migrate to the implementation phase leading to the need for additional debugging. In the worst case, it could even require an extra iteration and creation of additional prototype that could slow down the development process. Software redundancy is particularly relevant while performing an integration of several autonomous components, e.g. Web services.

There is an approach based on the use of design patterns [2], which allows applying certain structural solutions in the initial phase of software development. In this case, you can immediately focus on the use of a number of standard patterns, not to deviate from methodology associated with them. More often, the developer has to deal with CD, which require adjustments (these cases often occur due to the lack of experience). The process can be organized so that changes are performed manually in a visual format, which is simple and clear. To avoid errors, it is preferable to perform these actions automatically with the help of specialized CASE tools that have built-in validation function. It is important to have a formal description of the CD and the transformation rules in accordance with, for example, design patterns. Besides, this formal description should allow identifying structural errors in class diagrams.

The rest of this paper is organized as follows: first, Section II discusses the key targets and tools allowing formalization of class diagrams. Then the description logic as a basic tool of formalization CD is offered in Section III. Section IV shows, how the CD are described using description logic ALCQI. In Section V, concrete examples of optimization CD are presented before concluding in Section VI.

II. Formalization of Class Diagrams

There are several approaches to the formal description of class diagrams; the most commonly applied are those based on the use of OCL, Z+ language and description logics (DL). The real verification and transformation of texts feasible for Z language (partially) and description logics.

Let us focus on the use of DL, provided it is the most universal of these mechanisms. DL is widely used to describe ontologies [3] and has been initially developed primarily for this purpose. Since CD can be represented in the form of ontologies, the use of DL is appropriate. From the family of description logics the one most appropriate to describe class diagrams should be taken. Large number of studies [4], [5], [6] propose to use ACLQI logic, expanded capabilities represent n-ary association relationships.

Formally, a class diagrams do not operate with objects, but often, for example, to describe relations between classes, objects are necessary for understanding semantics.

Let us answer the question, what formal description of CD in the form of DL is required for?

Firstly, for a convenient formal representation of CD.

Secondly, to check consistency of CD. It is known, that the UML semantics do not allow using certain combinations of elements. For example, classes do not allow self-inheritance, directly or indirectly, the class-association should not have the same attributes as the classes associated with it, etc. Formal description of those constraints (there are a few of them), without complicating description logic excessively [7], does not appear possible. Instead, additional procedures of DL analysis could be introduced and should help to identify errors.

Finally, third, to optimize i.e., replacement of some structures to other, more optimal. For example, it is known that n-ary association relationships in some cases could be replaced by binary [8].

Let us elaborate a bit more on software systems described by CD. Identifying an optimal CD is not a straightforward question. For example, the use of many standard design patterns (these include Facade, Abstract factory, Adapter) increases complexity of CD, but improves its quality in terms of a further generation of the code and modifiability. The concept of complexity of a given CD could be helpful in principal [9], but there is no consensus view on this topic. For example, the use of interfaces often improves universality and reuse of future program code, however it reduces usability in the same time.

III. Description Logic

Let us describe the basic description logic ALC (Attributive Language with Complement), which is often used as a base to build many other logic [3].

Assume that there are a non-empty finite sets of atomic concepts A and atomic roles R. Then the composite concepts of the logic are defined following inductive way:

- every atomic concept A is a concept;
- the expressions T and $\perp$ are concepts;
- if C is a concept, then its complement ^{-}C is a concept as well;
- if C and D are concepts, then its intersection $C \cap D$ and union $C \cup D$ are concepts as well;
- if C is a concept and R is a role, then expressions $\forall R.C$ and $\exists R.C$ are concepts.

The axiom of inclusion of a concepts is described by the following expression: $C \sqsubseteq D$. While the axiom of an equivalence of concepts is an expression $C \equiv D$, where C and D are arbitrary concepts.

Similarly, the axiom of inclusion of a roles is described by the following expression: $R \sqsubseteq S$. While the axiom of an equivalence of roles is an expression $R \equiv S$, where R and S are any given roles.

Terminology or a set of terminological axioms (TBox) is a finite set of axioms of the above types. Sometimes axioms for particular roles are allocated in separate sets called role hierarchy or RBox.

The semantics of a DL is defined by interpretation of its atomic concepts as sets of objects chosen from a fixed set (domain), and atomic roles as sets of pairs, i.e. binary relations on the domain.

Formally, an interpretation I consists of a nonempty set (domain) Δ^I and interpretation function, which assigns to each atomic concept A a subset $A^I \sqsubseteq \Delta^I$, and each atomic role - a subset $R^I \sqsubseteq \Delta^I \times \Delta^I$. If the pair of individuals belongs to the interpretation of a specific role R, that is

$(e, d) \in R^I$, we say that the individual d is an R-successor of the individual e.

An interpretation function extends to compound concepts of logic according to the rules described in the study [3]:

For descriptions of class diagrams it is preferable to use the logic ALCQI. Extension ALCQI relative to the ALC views are:

Q - constraints of cardinality of roles: concepts of the form $<n$ R. C meaning: there is no more than n R-successors in C.

I - inverse roles: if R is a role, then R^{-} is also a role, meaning the inverse of binary relation.

Note that ALC logic (and many of its extensions, including ALCQI) can be considered as fragments of predicate logic with two variables, which is solvable [10]. This allows to transfer results of solvability, computational complexity and decision algorithms from the field of logic predicates into the area of description logics.

For CD we will only deal with TBox, and will be addressing the following three problems:

1) are not axioms that describe CD in terms of DL, conflicting, i.e., if there is a possibility for at least one formula to be inference simultaneously with its denial.

2) is it possible to identify sets of statements (axioms), showing the ineffectiveness of a given CD;

3) is it possible to optimize a model by modifying original axioms (refactoring of a software model existing in the form of DL).

IV. Presentation of Class Diagrams in Description Logic

Let us describe a method of representing, or rather coding CD in the form of DL axioms [4]. It this case class will be matching concept, while association - role.

Each attribute A of type K of class C is represented as follows:

$C \sqsubseteq \forall A.K$

Every operation f () : P (a result belong to P) of class C is represented role P, for which the following is valid:

$C \sqsubseteq \forall P_f.P \cap (\leq 1\ P_f.\perp)$

The generalization relation between classes C1 and C2, obviously, is represented as follows:

$C2 \sqsubseteq C1$,

where C1 is the ancestor.

For coding parameters for relations binary association and aggregation (aggregation degree higher than two is pointless) we use the following [4]:

$\exists A.C_2 \sqsubseteq C1$
$\exists A^{-}.C_1 \sqsubseteq C2$
$C1 \sqsubseteq (\geq m_1\ A.C_2) \cap (\leq m_2\ A.C_2)$

$C2 \sqsubseteq (\geq n_1 A^-.C_1) \cap (\leq n_2 A^-.C_1)$,

where C1, C2 are concepts corresponding to different classes; A is role corresponding to a binary association; A is a inverse role (relative to A); n1, n2, m1, m2 are numerical values, corresponding to the multiplicities.

And finally, n-ary associations (see Fig. 1) both with a class association, and without it, can be expressed by using the procedure of reification [3], i.e. a transformation of n-ary association into binary.

$A \sqsubseteq \exists R_1.C_1 \cap \ldots \cap \exists R_n.C_n \cap (\leq 1\ R_1) \cap \ldots \cap (\leq 1\ R_n)$

$C_1 \sqsubseteq (\geq m_1\ R_1^-.A) \cap (\leq l_1\ R_1^-.A)$

. . .

$C_n \sqsubseteq (\geq m_n\ R_n^-.A) \cap (\leq l_n\ R_n^-.A)$

Another important relation in class diagrams is a dependency relation. For completeness and consistency of the model, described using UML class diagrams, this relation is not affected. To encode the dependency relation let us introduce the following designation:

C1 —> C2

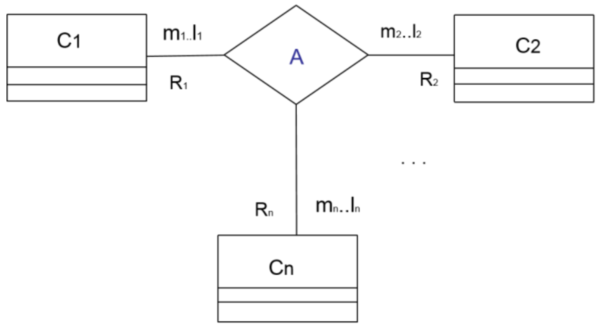

Fig. 1. N-ary association

This relation means that the class C2 depends on the class C1. Will consider it as an informal extension of the description logic.

Despite a somewhat arbitrary interpretation of the definition of concept of inclusion concepts, study [4] proves consistency of this coding method. Thus the first problem (1) can be considered solved.

Problem (2) associated with the search for suboptimal from the point of view of CD fragments of DL assertions. In other words, a formal description CD, presented in the form of DL, is analyzed for search notoriously inefficient parts. For example, searched for fragments of the diagram, for which is a more efficient model descriptions in accordance with design templates. To solve this problem can be applied an interesting approach based on the notion of anti-patterns design [11]. If an anti-pattern - a suboptimal fragment of CD - is found, than the designer is invited to change the CD.

The problem (3) – automatic conversion of assertions, describing a given class diagram, with the aim to optimize the model - could be solved only in certain cases, for example using the approach, proposed in the study [8], [11].

V. Examples of Optimization

As examples of the applicability of the proposed technology, we use a number of standard patterns and patterns introduced in the study [8].

A. The pattern "the chain of responsibilities"

Investigate one of the simplest cases of this pattern (see Fig. 2), when the request HandlerM() can be processed by the object of one of the two classes. In this case, an abstract class or interface could be introduced, that redirects the request to a particular class. Then the class diagram will look as follows (see Fig. 3).

Having a description in the form of description logic assertions:

$C_1 \sqsubseteq \forall P_f.P \cap (\leq 1\ P_f.\bot)$

$C_2 \sqsubseteq \forall P_f.P \cap (\leq 1\ P_f.\bot)$

and next informal extensions:

C —> C_1

C —> C_2,

where C, C_1 и C_2 are the classes Client, Handler1 and Handler2, respectively, f is the operation HandlerM, we can make a conclusion of applicability "the chain of responsibility" pattern.

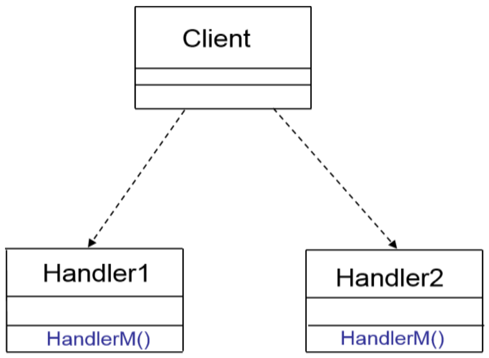

Fig. 2. Example of using of pattern "the chain of responsibilities"

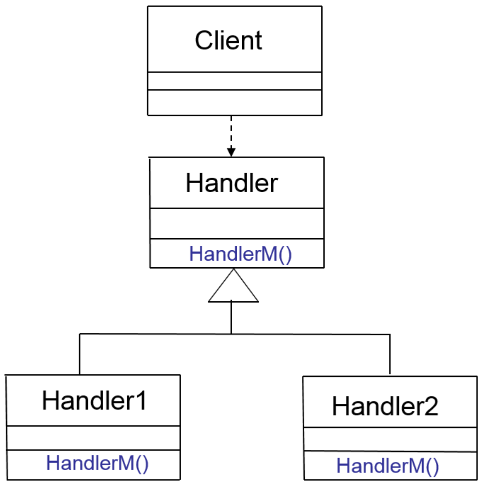

Fig. 3. Result of using of pattern "the chain of responsibilities"

B. The pattern that allows a transition from ternary association to binary

Assume that in the ternary association there is a class with multiplicity (1). Then ternary association could be replaced with a combination of binary association and class-association. Class diagrams, illustrating this situation, are shown in Fig. 4 and 5.

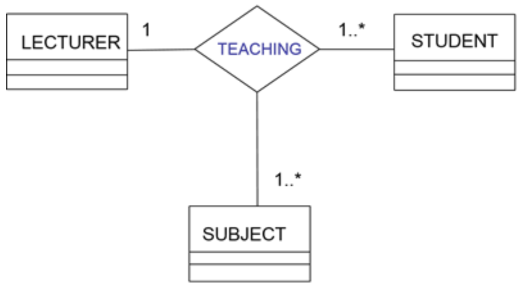

Fig. 4. The ternary association

Assume we have a description in the form of the following statements of description logic:

$A \sqsubseteq \exists R_1.C_1 \cap \exists R_2.C_2 \cap \exists R_3.C_3 \cap (\leq 1\ R_1) \cap$

$\cap (\leq 1\ R_2) \cap (\leq 1\ R_3)$

$C_1 \sqsubseteq (\geq 1\ R_1^-.A)$

$C_2 \sqsubseteq (\geq 1\ R_2^-.A)$

$C_3 \sqsubseteq (\geq 1\ R_3^-.A) \cap (\leq 1\ R_3^-.A)$,

where A is the ternary association Teaching; C_1, C_2, C_3 are the classes Student, Subject и Lecturer, respectively; R_1, R_2, R_3, R_1^-, R_2^-, R_3^- are direct and inverse roles of classes Student, Subject и Lecturer in association Teaching.

Then ternary association could be seamlessly replaced by a combination of binary association and class association.

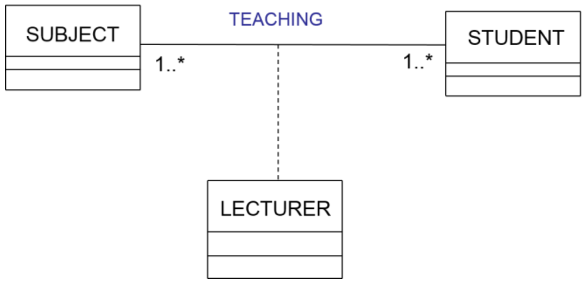

Fig. 5. Replacing ternary association on binary and class-association

C. Anti-pattern "the loop of the associations"

The idea of this anti-pattern (Fig. 6) is the following: if semantically related associations form a loop, it is possible that one of them is redundant and should be removed. The removal can be done only by the designer, hence the information about the detected anti-pattern "the loop of the associations" should be submitted to the designer.

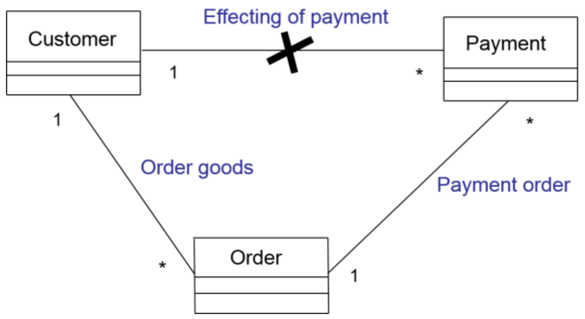

Fig. 6. Example of anti-pattern "the loop of the associations"

In the language of the DL it would look like this:

$\exists A_1.C_1 \sqsubseteq C_2$
$\exists A_2.C_2 \sqsubseteq C_3$
$\exists A_3.C_3 \sqsubseteq C_1$,

where A_1 – Effecting of payment, A_2 – Order goods, A_3 – Payment order, C_1 – Customer, C_2 - Order, C_3 – Payment.

Then the designer will be asked to remove one of the three axioms. In this case, it would be logical to remove from the class diagram the axiom

$\exists A_3.C_3 \sqsubseteq C1$

and the corresponding association relationship. This choice is determined by the semantics of the domain area.

VI. CONCLUSION

This study describes the new approach to optimizing software systems at the design stage. This approach consists of the transformation of class diagrams into description logic assertions and automated search for suboptimal fragments. For these purposes both design patterns and anti-patterns could be applied. Information about all detected suboptimal fragments is transmitted to the designer, who decides on potential modifications of the model. In addition, the system may suggest to apply certain transformations, and further to perform a series of transformations automatically.

The relevance of this approach is evidenced by the fact that verification and optimization of a model could be executed already at the design phase, which allows to minimize the processes of error correction and refactoring. Here are the key ideas proposed in this study:

1) A formal description of the model in the form of description logic assertions

2) Automatic model analysis to identify suboptimal fragments, using design patterns and anti-patterns

3) Automatic optimization of a model (for a number of design patterns) at the description logic level

REFERENCES

[1] J. Rumbaugh, I. Jacobson, G. Booch, "The Unified Modeling Language, Reference Manual", Addison-Wesley, Reading, MA, 1998.

[2] E.Gamma, R.Johnson, Helm R., J.Vlissides, "Design Patterns. Elements of Reusable Object-Oriented Software", Addison-Wesley, 2001

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, Cambridge, 2003.

[4] D. Berardi , D, Calvanese, and G. D. Giacomo, "Reasoning on UML Class Diagram," Artificial Intelligence, vol. 168, pp. 70-118, 2005.

[5] A. Cali, D. Clavanese, G. D. Giacomo, and M. Lcnzerini, "A Formal Framework for Reasoning on UML Class Diagram," in Proc. of the 13th Int. Sym. on Methodologies for Intelligent Systems (IS-MIS 2002), 2002.

[6] A. Queralt, A. Artale, D. Calvanese, E. Teniente, "OCL-Lite: Finite reasoning on UML/OCL conceptual schemas", Data & Knowledge Engineering 73, pp. 1–22, 2012

[7] A.Grigoriev, A.Kropotin, E. Ovsyannikova, "The Problem of Detecting Consistencies on UML Class Diagrams", in Proc. of International scientific-practical conference "Modern problems and ways of their solution in science, transport, production and education‘ 2012", http://www.sworld.com.ua/konfer29/721.pdf, 2013

[8] M. Sergievskiy, "N-ary Relations of Association in Class Diagrams: Design Patterns", International Journal of Advanced Computer Science and Applications, Vol. 7. № 2. pp. 265-268, 2016.

[9] E. Niculchev, O. Deryugina, "Model and Criteria for the Automated Refactoring of the UML Class Diagrams",. International Journal of Advanced Computer Science and Applications, Vol. 7. № 12. pp. 76-79, 2016.

[10] A. Cali, D. Clavanese, G. D. Giacomo, and M. Lcnzerini, "A Formal Framework for Reasoning on UML Class Diagram," in Proc. of the 13th Int. Sym. on Methodologies for Intelligent Systems (IS-MIS 2002), 2002.

[11] W. Brown, R. Malveau, H. McCormick, T. Mowbray, "AntiPatterns. Refactoring Software, Architectures, and Projects in Crisis", John Wiley & Sons, Inc., 1998