Model and Criteria for the Automated Refactoring of the UML Class Diagrams

Evgeny Nikulchev
Moscow Technological Institute
Moscow, Russia

Olga Deryugina
Moscow Technological University MIREA
Moscow, Russia

Abstract—Many papers have been written on the challenges of the software refactoring. The question is which refactorings can be applied on the modelling level. Based on the UML model, for example. With the aim of evaluating this possibility the algorithm and the software tool of automated UML class diagram refactoring were introduced. The software tool proposed reduces the level of the UML class diagram complexity metric.

Keywords—UML; refactoring; class diagrams; software architecture; software design; UML transformation

I. INTRODUCTION

MDA (Model Driven Architecture) [1] approach is supported by OMG (Object Management Group). In the MDA approach design process starts with the creation of the PIM (Platform Independent Model). Then the PIM is automatically transformed into the PSM (Platform Specific Model).

MDA proposes standards of model creation, transformation and exchange. For example, UML (Unified Modelling Language) [2] is used to describe models, XMI (XML Metadata Interchange) [3] is used to exchange models between tools. One tool can be used to create a model, and another one – to analyze or transform it. Thus, model transformations play an important role in the MDA conception. One of the possible model transformation objectives can be model refactoring.

Refactoring means restructuring of the system without changing its behavior. Originally, refactoring was connected with the code-level transformations. A number of studies have investigated different means of the software refactoring [4, 5, 6].

Some refactorings apply design patterns to the existing code. First design patterns were proposed by Erich Gamma et al. in [7]. Software design patterns describe solutions of commonly occurring problems. Design patterns are aimed at the improvement of such software characteristics as modifiability, reusability, maintainability, etc.

Many papers have been written on the challenges of the software refactoring. The question is which refactorings can be applied on the PIM level described with the UML.

There are two main approaches to the problem of UML refactoring. The first one is connected with the search based software engineering [8] (SBSE), which is a topic of growing interest nowadays. In SBSE software engineering problems are formulated as optimization problems, which then are solved by search algorithms (Genetic algorithm, Simulated annealing, Swarm intelligence algorithms, etc.) SBSE is used in order to solve UML class diagram refactoring problem in [9–14]. However, the result of applying search algorithms to class diagrams sometimes can be meaningless.

The second approach is connected with the developing frameworks of automated model refactoring [15–18], where the main role is given to the software designer. And the framework applies transformations, using some transformation rules in interaction with the designer.

This paper explores the problem of the automated UML class diagram refactoring. This problem is significant as long as the model refactoring is less time-consuming than the code refactoring. Furthermore, model refactoring is connected with the creation of PIM rather than PSM.

The rest of this paper is organized as follows: First, the problem of the automated UML class diagram refactoring is formulated in Section 2. Then an algorithm of the automated UML class diagram refactoring is proposed in Section 3. In Section 4, the software tool UML Refactoring is introduced before concluding in Section 5.

II. FORMULATION OF THE PROBLEM

The scheme of the UML class diagram analysis is shown in Fig. 1. Algorithm takes as input UML class diagram $d$, fitness function $f(d)$ and a set of semantically equivalent transformations $T$. The output is a list of transformations $T^*$, which reduce fitness function value and are recommended to apply.

![Fig. 1. Scheme of the UML class diagram analysis](image)

Let $d$ be a UML class diagram $d = \{C, I, R\}$, where $C$ is a set of classes $C = \{c_0, ..., c_k\}$, $I$ is a set of interfaces $I = \{i_0, ..., i_l\}$, $R$ is a set of relations $R = \{r_0, ..., r_g\}$. 

www.ijacsa.thesai.org
Then $c_i$ is a class $c_i = \{A_i, M_i, F_i\}$, where $A_i$ is a set of attributes $A_i = \{a_{i_0}, a_{i_1}, ..., a_{i_n}\}$, $M_i$ is a set of methods $M_i = \{m_{i_0}, m_{i_1}, ..., m_{i_m}\}$, and $F_i$ is a set of features $F_i = \{s_{i_0}, s_{i_1}, ..., s_{i_p}\}$.

In addition, $i_0$ is an interface which defines a set of methods $M_i = \{m_{i_0}, m_{i_1}, ..., m_{i_m}\}$.

Let us assume that the UML class diagram transformation $t$ can be described as the following mapping function:

$$t(d, E) : d \rightarrow d'$$

where $E$ is a set of diagram elements $e_1 \in d$, $e_1 \in \{C, R, I\}$.

Assume that the semantically equivalent transformation $t$ is such a transformation $t(d, E) : d \rightarrow d'$, that:

$$S(d) - S(d')$$

where $S(d)$ is a structural semantic value of the diagram $d$, which can be described as follows:

$$S_i = \{c_{i_0} = \{\}, c_{i_1} = \{\}, ..., c_{i_k} = \{\}\, | \, i_0 = \{\}, i_1 = \{\}, ..., i_k = \{\}\}$$

$$i_0, ..., i_k \in d$$

where $c_{i_0}, ..., c_{i_k} \in d$, are the classes of the diagram $d$; $i_0, ..., i_k \in d$, are the interfaces of the diagram $d$; $r_{i_0}, ..., r_{i_k} \in d$, are the relations of the diagram $d$.

The automated UML class diagram refactoring problem can be formulated as follows:

1. Assume that there is a UML class diagram $d$, a set of semantically equivalent transformations $T$, and a fitness function $f(d)$.
2. Then it is required to find such a set of pairs $\{t, E\}$, that:

$$d' = t(d, E), \Delta f(d') < 0.$$  

III. AUTOMATIC UML CLASS DIAGRAM REFACTORIZING ALGORITHM

Then the CDTA (Class Diagram Transformation Analysis) Algorithm can be described as follows:

```java
for each t \in T
L = search(t,d,f) //search pairs \{t, E\} for which \Delta f(d') < 0
Q.add(L) //add pairs \{t, E\} to the resulting list
return Q
```

The search(t,d,f) function can be defined as follows:

```java
L1 = analyze(t,d) //search element sets E \in d for the transformation
for each e \in L1
```

$\Delta f = \text{refactor}(e, t, d) // $apply transformation t to the diagram d

```java
if (f(d') < f(d)) //check the decrease of the f(d) value
L2.add(t,e) //add t and e to the resulting list
```

The analyze (t,d) method is specific for each transformation. For example, the algorithm of searching sets of diagram elements $E$ on which the Strategy transformation can be conducted can be described as follows:

1) Make a list of classes having inheritors $l1$.
2) For each class from $l1$ check whether its inheritors implement any interfaces. If yes – add them to the list $l2$.
3) 3. For each class from $l1$ calculate $\Delta K(d)$. If $\Delta K(d) < 0$ – add to the list $l3$: \{parent_id, \{child_classes_ids\}, \{interfaces_ids\}\}.
4) 4. Return $l3$.

Let us formulate an example of the fitness function – structural complexity metric $K(d)$:

$$K(d) = k_1 \cdot |C| + k_2 \cdot |I| + k_3 \cdot |R| + k_4 \cdot \sum_{m=0}^{n} |A_m| + k_5 \cdot \sum_{m=0}^{n} |M_m| + k_6 \cdot \sum_{m=0}^{n} |M_m'|.$$  

where $K(d)$ is the structural complexity of the diagram $d$; $|C|$ is the number of classes $c_i \in d$; $|I|$ is the number of interfaces

where $i_0 \in d$; $|R|$ is the number of relations $r_i \in d$; $A_i$ is a set of attributes $a_{i_0} \in c_i, c_i \in d$; $M_i$ is a set of class methods (except of the methods, declared in implemented interfaces) $m_{i_0} \in c_i, c_i \in d$; $M_m'$ is a set of interface methods $m_{i_0} \in c_i, c_i \in d$; $k_1, k_2, k_3, k_4, k_5, k_6$ are weights for each group of elements.

IV. UML CLASS DIAGRAM REFACTORIZING TOOL

Class diagram refactoring software should solve the following tasks:

1) UML class diagram analysis: searching the transformations which can be conducted to decrease the fitness function value;
2) UML class diagram transformation.

Main functional blocks of the UML refactoring tool are shown in Fig. 2. XMI parser translates XMI document to the abstract data structure UML Map [19], which stores UML class diagram elements in hash-maps. Then Analyzer searches pairs \{t,E\}, which reduce fitness function value and forms the Transformations table on the screen. OOM Calculator calculates various object-oriented metrics for the UML class diagram and forms the Metrics table on the screen. If a user has chosen some transformation from the table, the Transformer applies it to the diagram. User can export an attained diagram to XMI document.
Let us introduce an example of applying the UML refactoring tool to the class diagram shown in Fig. 3.

Assume that it is required to minimize relations count value. First, weight values for the fitness function should be set: $k_1 = 0.01; k_2 = 0.01; k_3 = 1; k_4 = 0.01; k_5 = 0.01; k_6 = 0.01$. Then it is attained, that:

$$K(d) = 0.06 + 0.02 + 12 + 0.03 + 0.06 + 0.02 = 12.19.$$  

Fig. 5. The result of applying the Strategy transformation to the diagram $d$

After the transformation, the following is attained:

$$\Delta K(d') = 0.06 + 0.02 + 7 + 0.05 + 0.08 + 0.02 = 7.23,$$

$$\Delta K(d) = -4.96.$$  

Fig. 6. Diagram $d'$ attained as the result of the Strategy transformation

V. CONCLUSIONS

The presented framework gives the possibility to analyze and automatically restructure UML class diagrams in accordance with aims of the refactoring, which can be formulated based on the fitness function (4).

The main role is given to the software designer, who can import/export UML class diagrams in XMI format, refactor them and save the results.

The framework calculates metrics and proposes a list of transformations, which minimize the fitness function value. Available transformations include Strategy, Façade, Interface insertion, etc.

Future research should examine the effectiveness of the proposed framework to the large software systems. Furthermore, lists of available transformations and calculated metrics should be expanded.
REFERENCES