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.

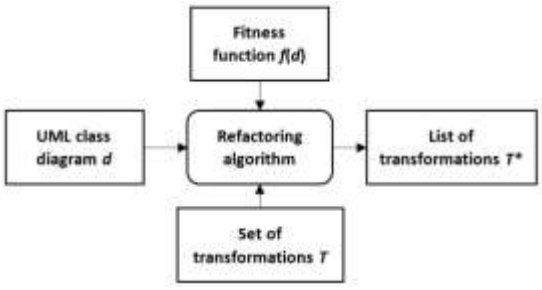

Fig. 1. Scheme of the UML class diagram analysis

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 $I = \{i_0, ..., i_1\}$, R is a set of relations $R = \{r_0, ..., r_g\}$.

Then c_i is a class $c_i = \{A_i, M_i, F_i\}$, where A_i is a set of attributes $A_i = \{a_0^i, a_1^i, ..., a_n^i\}$, M_i is a set of methods $M_i = \{m_0^i, m_1^i, ..., m_m^i\}$, F_i is a set of features $F_i = \{st_i, v_i, abs_i, ...\}, st \in \{0,1\}$ is isStatic feature, v_i is visibility parameter $v_i \in \{public, private, protected\}, abs_i$ is isAbstract feature $abs_i \in \{0,1\}$.

In addition, i_i is an interface which defines a set of methods $M_i = \{m_0^i, m_i^i, ...m_m^i\}$.

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

$$t(d, E): d \to d', \quad (1)$$

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

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

$$S(d) \sim S(d'), \quad (2)$$

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

$$S_i = \{\{c_1^i = \{...\}, c_2^i = \{...\}, ..., c_k^i = \{...\}\}, \{i_1^1 = \{...\}, i_2^i = \{...\}, ..., i_1^i = \{...\}\}, \{r_1^i, r_2^i, ..., r_m^i\}\}, \quad (3)$$

where $c_1, ..., c_k \in d_i$ are the classes of the diagram d; $i_1, ..., i_l \in d_i$ are the interfaces of the diagram d; $r_1, ...r_m \in d_i$ are the relations of the diagram d.

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

Assume that there is a UML class diagram d, a set of semantically equivalent transformations T, and a fitness function $f(d)$.

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 Refactoring Algorithm

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

```
for each t∈T
L = search(t,d,f) //search pairs {t,E} for which Δf(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:

```
L1 = analyze(t,d) //search element sets E ∈d for the
transformation
for each e∈L1
d' = refactor(e,t,d) //apply transformation t to the
diagram d
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
return L2
```

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 *l*1.

2) 2. For each class from *l*1 check whether its inheritors implement any interfaces. If yes – add them to the list *l*2.

3) 3. For each class from *l*1 calculate $\Delta K(d)$. If $\Delta K(d) < 0$ – add to the list *l*3: {parent_id, {child_classes_ids},{interfaces_ids}}.

4) 4. Return *l*3.

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

$$K(d) = k_1 \cdot |C| + k_2 \cdot |I| + k_3 \cdot |R| + k_4 \cdot \sum_{i=0}^{n} |A_i| + k_5 \cdot \sum_{i=0}^{n} |M_i| + k_6 \cdot \sum_{i=0}^{m} |M_j'|, \quad (4)$$

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 $i_i \in d$; $|R|$ is the number of relations $r_i \in d$; A_i is a set of attributes $a_j \in c_i, c_i \in d$; M_i is a set of class methods (except of the methods, declared in implemented interfaces) $m_i \in c_i, c_i \in d$; M_j' is a set of interface methods $m_j \in i_i, i_i \in d$; $k_1, k_2, k_3, k_4, k_5, k_6$ are weights for each group of elements.

IV. UML Class Diagram Refactoring 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.

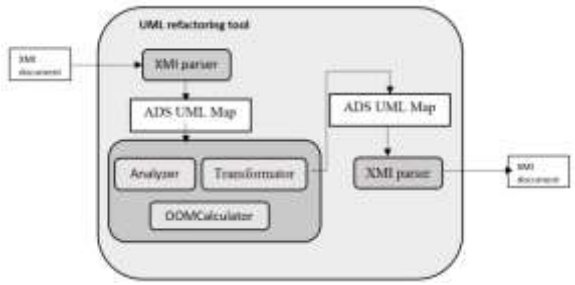

Fig. 2. Functional blocks of the UML class diagram refactoring tool

Let us introduce an example of applying the UML refactoring tool to the class diagram d 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:

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

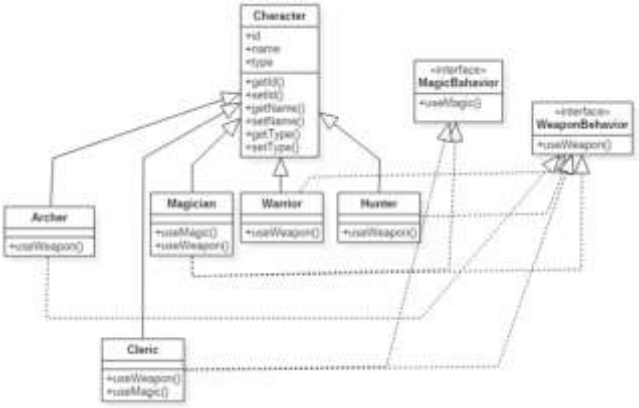

Fig. 3. Initial UML class diagram *d*

The UML Refactoring tool proposes the following transformations (see Fig. 4):

- Interface insertion for the classes Magician, Cleric and for methods useWeapon(), useMagic();
- Strategy insertion for the interfaces MagicBehaviour, WeaponBahaviour, parent class Character and child classes Archer, Magician, Warrior, Hunter, Cleric.

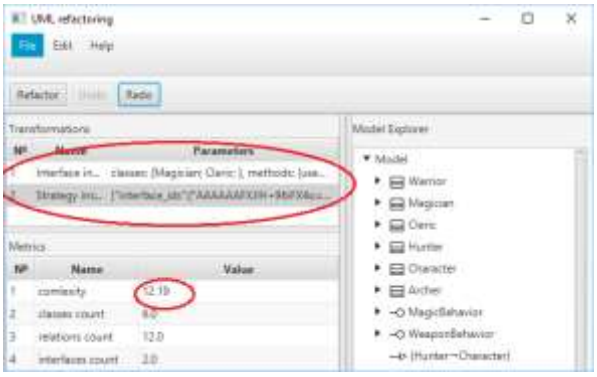

Fig. 4. Transformations proposed by the UML refactoring tool

The result of applying the Strategy transformation to the diagram d is shown in Fig. 5 and Fig. 6.

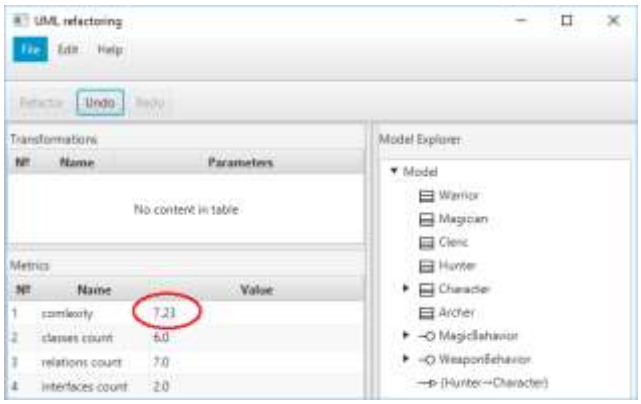

Fig. 5. The result of applying the Strategy transformation to the diagram *d*

After the transformation, the following is attained:

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

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

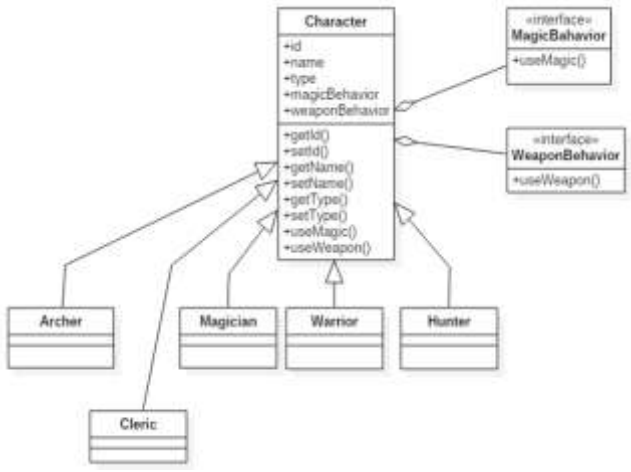

Fig. 6. Diagram *d'* attained as the result of the Strategy transformation

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

[1] OMG Model Driven Architecture (http://www.omg.org/mda)

[2] OMG Unified Modelling Language UML. Version 2.5 (http://www.omg.org/spec/UML/2.5)

[3] XML Metadata Interchange (XMI) Specification. Version 2.4.2 (http://www.omg.org/spec/XMI/2.4.2)

[4] J. Kerievsky, "Refactoring to Patterns. Boston M. A.: Addison-Wesley, 2004.

[5] M. Fowler, Refactoring: Improving the Design of Existing Code", Boston M. A.: Addison-Wesley, 2000.

[6] T. Mens, T. Tourwe, "A survey of software refactoring", IEEE Transactions on Software Engineering, vol. 30, no. 2, 2004, pp. 126 – 139.

[7] E. Gamma, H. Richard, J. Ralph, and V. John, "Design patterns: Abstraction and reuse of object-oriented design", Springer Berlin Heidelberg, 1993.

[8] M. Harman, S. Mansouri, Y. Zhang, "Search-based software engineering: Trends, techniques and applications", ACM Computing Surveys, vol. 45, no. 1, 2012.

[9] M. Amoui, S. Mirarab, S. Ansari and C. Lucas, "A genetic algorithm approach to design evolution using design pattern transformation", International Journal of Information Technology and Intelligent Computing, vol. 1, 2006, pp. 235 – 245.

[10] M. Bowman, L. Briand, and Y. Labiche, "Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms", Technical report SCE-07-02, Carleton University.

[11] S. Hadaytullah, Vathsavayi, O. Räihä, and K. Koskimies, "Tool support for software architecture design with genetic algorithms", Proceedings of ICSEA'10. IEEE CS Press, August 2010, pp. 359–366.

[12] R. Lutz, "Evolving good hierarchical decompositions of complex systems, Journal of Systems Architecture", vol. 47, 2001, pp. 613 – 634

[13] C. Simons C., I. Parmee, "Single and multi-objective genetic operators in object-oriented conceptual software design", Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'07), 2007, pp. 1957 – 1958.

[14] O. Deryugina, "Improving the structural quality of UML class diagrams with the genetic algorithm", 6th Seminar on Industrial Control Systems: Analysis, Modeling and Computation, vol. 6., 2016, art. 03003.

[15] Y. Rhazali, Y. Hadi, A. Mouloudi, "A Model Transformation According Model Driven Architecture", Proceedings of 3rd International Conference on Computing Engineering & Enterprise Management (ICCEEM-2015), 2015, pp. 6 – 14.

[16] Y. Rahmounea, A. Chaoui, E. Kerkouche, "A Framework for Modeling and Analysis UML Activity Diagram using Graph Transformation", International Workshop on the Use of Formal Methods in Future Communication Networks (UFMFCN 2015), 2015, pp. 612 – 617.

[17] E. Kerkouche, A. Chaoui, E. Bourennane, O. Labbani, "A UML and Colored Petri Nets Integrated Modeling and Analysis Approach using Graph Transformation", Journal of Object Technology", vol. 9, no. 4, 2010. pp. 25–43.

[18] Sergievskiy Maxim, "N-ary relations of association in class diagrams: design patterns," International Journal of Advanced Computer Science and Applications, vol. 7, no. 2, 2016, pp. 265-268.

[19] O. A. Deryugina "Transformation of UML class diagrams using genetic algorithm", Russian Technological Journal, vol.1, no. 4 (9), 2015, pp. 36–42.