Volume 1 No. 4, JULY 2011 ISSN 2222-9833
ARPN Journal of Systems and Software

http://www.scientific-journals.org

M-REFACTOR: A New Approach and Tool for Model Refactoring

[1] Maddeh Mohamed, [2] Mohamed Romdhani, [3] Khaled GHEDIRA
[1,3]SOIE, Ecole Nationale des Sciences de l'Informatique
Campus Universitaire Manouba - Manouba 2010
Tunis, Tunisia
[2]INSAT Centre Urbain Nord
[1]maddeh_mohamed@yahoo.com , [2] Mohamed.Romdhani@insat.rnu.tn , [3] khaled.ghedira@isg.rnu.tn

ABSTRACT

We present a new approach and tool (MRefactor) for model refactoring; we propose an extension of the UML metamodel for the assisted Model Driven Refactoring (MDR). Based on model qualities metrics and design flaws, we propose a new demarche allowing the automated detection of model refactoring opportunities and the assisted model restructuration. Precisely we focus on class and sequence diagrams.

Keywords: *Metrics, metamodeling, refactoring, UML profile.*

I. RESEARCH CONTEXT AND MOTIVATION

Software quality is assessed and improved mainly during formal technical reviews, which primary objective is to detect errors and defects early in the software process, before they are passed on to another software engineering activity or released to the customer. The detection and correction of design defects early in the process substantially reduce the cost of subsequent steps in the development and support phases. The solution advocated for correcting design defects is to apply refactorings.

The majority of previous researches on refactoring are focused at the code level and less concerned with the earlier stages of design [16][2][5][4]. A promising approach is to deal with refactoring in a language independent way [7][10][11][12][15][17].

In this paper, we present our framework for model driven refactoring, specifying the model refactoring by in-place model transformation [3][9][18]. We follow the model driven architecture (MDA) [8] approach as guideline for model refactoring. We extend the UML metamodel to support assisted refactoring. This extension has tow advantages, the first one, is that it allows the modeling of any model refactoring opportunities, which are Antipatterns and BadSmells. The second one concerns the assisted model restructuration.

The paper is structured as follows. Section 2, presents the general process of model refactoring. Section 3, presents the case study and the MRefactor tool. Section 4, gives the conclusion.

II. GENERAL PROCESS

Figure 1 presents our approach: first, we follow the principle of domain analysis to propose an extended UML meta-model to specify and formalize refactoring opportunities at the design level. After analyzing the design flaws we instantiate our metamodel to define the refactoring opportunities. The detection process creates the RefactoringTags on the source model. They indicate the refactoring primitives that will be applied for the model restructuring. The user validates the RefactoringTags he judge necessary. After what, the refactoring factory uses the information stored in the RefactoringTags for the assisted model refactoring.

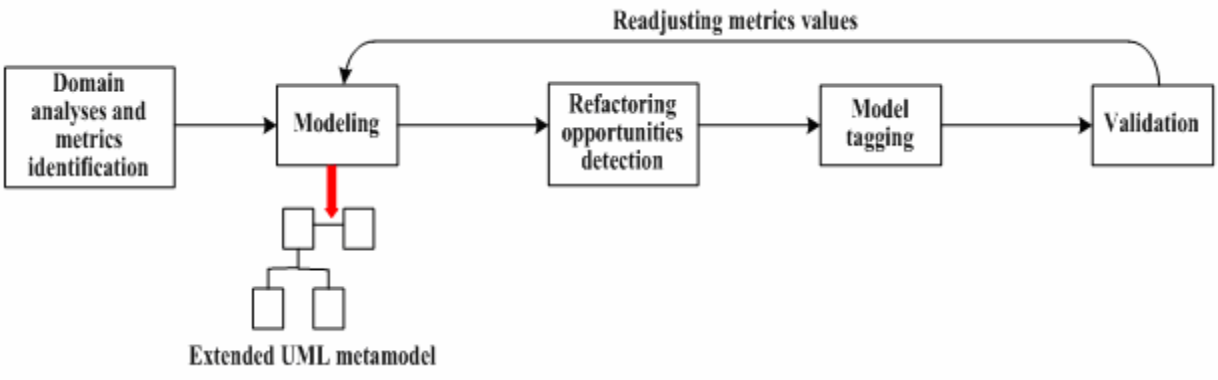


Fig. 1: The model refactoring process

A. Domain analyses

Domain analysis is a process by which information used in developing software systems is identified, captured, and organized with the purpose of making it reusable when creating new systems. In the context of refactoring, information about Antipatterns and BadSmells must be well structured and reusable for the automated detection of refactoring opportunities. Thus, we have studied the textual descriptions of design defects in the literature to identify, define, and organize the key concepts of the domain.

We analyze the key concepts, we extract metric-based heuristics as well as structural and semantic information. Designer experts are in charge of this step, if well and carefully done it guarantees the efficiency of the detection process.

We present an antipattern example: the Blob. The Blob (called also God class [14]) corresponds to a large controller class that depends on data stored in surrounded data classes. A large class declares many fields and methods with a low cohesion. A controller class monopolizes most of the processing done by a system, takes most of the decisions, and closely directs the processing of other classes.

Despite the clarity of the definition it gives no information on how user could identify this antipattern. The objectives of the domain analysis and the modelling stages we propose, is to standardize the detection of the antipatterns and BadSmells.

After the domain analysis for the Blob we remarque that the Blob antipattern and God Class are closed. The detection of the God Class indicates the presence of the Blob antipattern.

The automated detection of this design flaws could be based on the WMC, ATFD and APM metrics, following the heuristic H1.

Access To Foreign Data (ATFD) [13] represents the number of external classes from which a given class accesses attributes, directly or via accessor-methods. The higher the ATFD value for a class, the higher the probability that the class is or is about to become a god-class.
Weighted Method Count (WMC) [1] is the sum of the statical complexity of all methods in a class. In model context, complexity is considered unitary, WMC measures in fact the number of methods (NOM).
Attribute Per method (APM) is defined as the ratio of the metrics Number of attributes NOA and NOM. We assume that for each class the getter and setter methods are defined so if for a class C the APM =0.5 it means that the class C contains only getter and setter methods.
Tight Class Cohesion (TCC) [6] is defined as the relative number of directly connected methods. Two methods are directly connected if they access a common instance variable of the class.

The heuristic is H1: ATFD<20% and ATFD >4 and WMC>20 and APM<0.4). The Blob can be modeled now using the UML extention.

We identified 31 metrics that composes the heuristics of detection of design defects.

B. Modeling

Based on MOF meta-metamodel and profile capabilities, we extend the UML metamodel to support all key concepts used in the specifications of design defects. We instantiate the metamodel for each concept, representing a catalogue of design flaws. The information stored in the design flaws model can be programmatically manipulated. This stage allows the automated treatment of the information. We formalize a set of textual and informal design flaws description, subject to a subjective interpretation, in a well-structured model enclosing all necessary information to deal with refactoring. In figure 2 we present the extended UML metamodel.

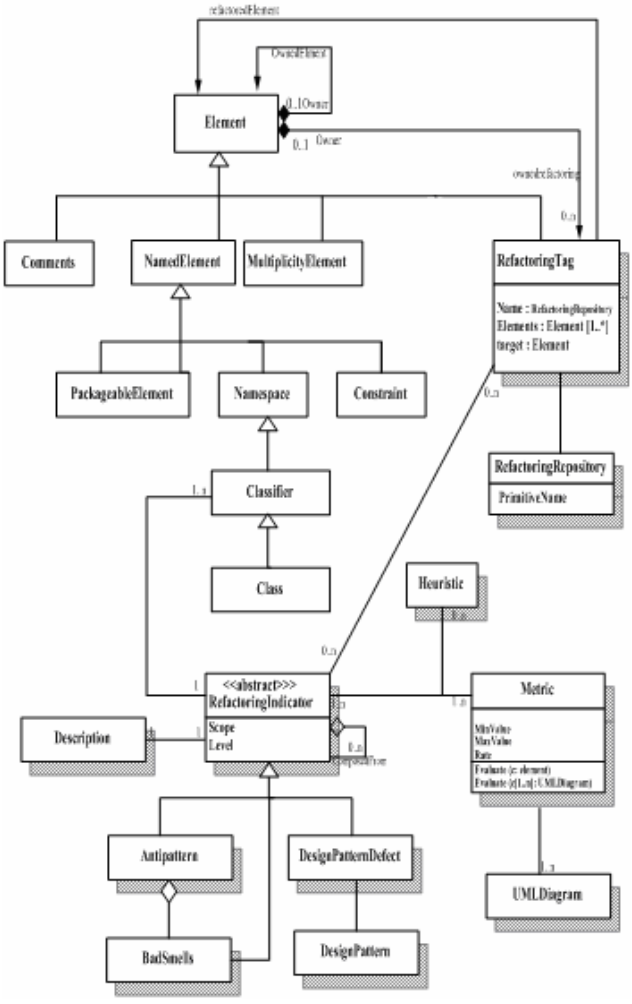


Fig. 2: Extended UML metamodel

We tend to formalize the design flaws using the UML profile. We related each one to the model construct in which it occurs, we define a set of metrics, and heuristics helping us to attend assisted detection.

This extension has three advantages fist it allows the modeling of refactoring opportunities second it uses refactoringTag which contains the refactoring information attached to a set of elements. The information consists on the refactoring name, the refactoring elements and the target element. Third, using refactoringTag we can show graphically to the users the model elements that are suspicious. Applying any refactoring consists on marking the selected class by associating an RefactoringTag containing the refactoring name, the refactoring elements and the target element (if necessary) then executing a refactoring transformation module.

As presented in figure 3 we consider this description as a way that could be investigated for the standardization of the design flaws description. The only information that could be readjusted for better refactoring opportunities detection is the Heuristic H1. Each metric is related to the model from which it is measured. In this example, WMC and APM are a structural metrics calculated from the class diagram, ATFD is a behavioral metric that could be measured from sequence diagrams. The two comments C1 and C2 give a textual description of the design flaws representing the semantic aspect.

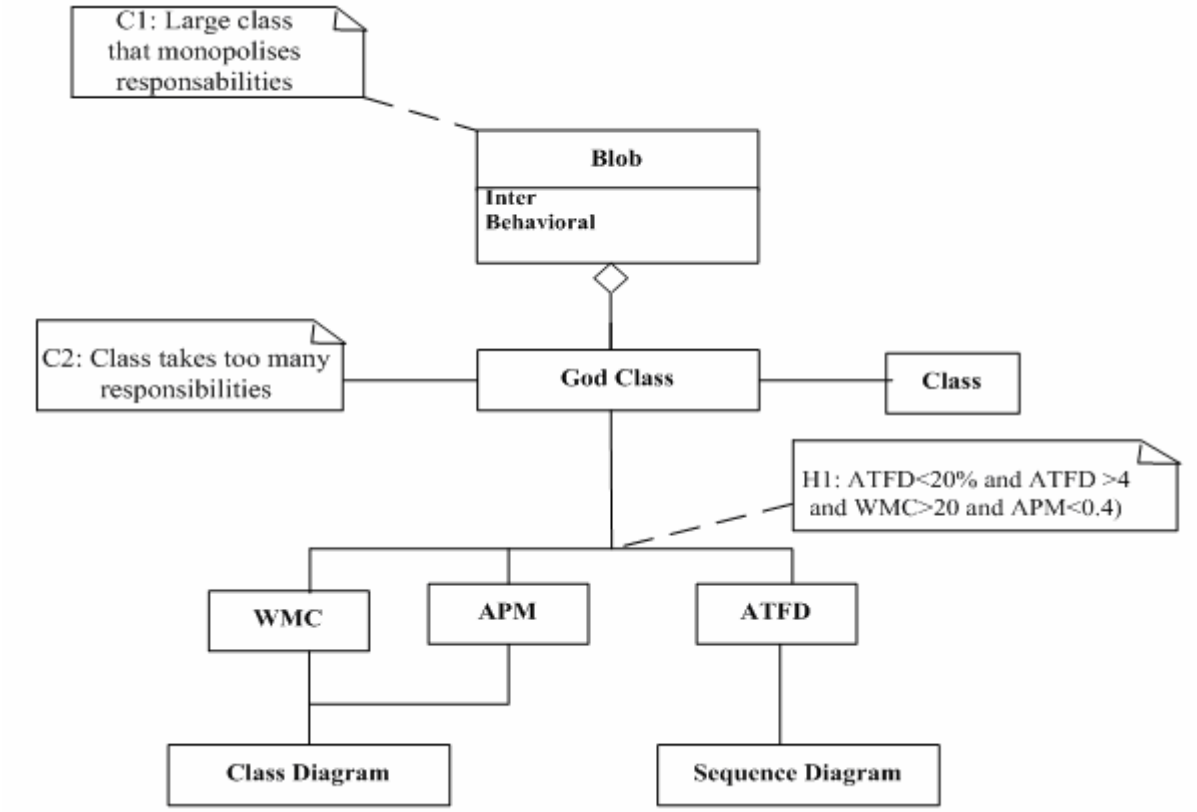


Fig. 3: The Blob antipattern

C. Refactoring opportunities detection

For each design flaws, we identify the metric-based heuristics, and the UML diagrams on which we base the model analyze. Then, the correction refactoring factory extracts the information from the RefcatoringTag, and restructures the model after the user validation. This validation represents the only intervention of users

D. Model tagging

This stage involves the usage of the Refactoring Tag. The detection tool generates automatically the refactoringTags indicating to user visually what are the problems detected in the model and how the tool will remediate to them. User has only to validate the RefactoringTag and run the correction module.

E. Validation

We validate the detection algorithm and readjust the metric-based heuristics if necessary. After experimentations, this stage creates efficient and reusable detection algorithms. Expert can modify the metrics values to adapt them to the context of detection he can increase or decrease the metrics values depending on the largest of the model.

III. CASE STUDY

In this section we validate our approach as well as the tool we proposed. We choose The AC Airline Company, which proposes to his customers a set of flights for various national and international destinations. The information system of the company is intended to assure the management of the various fields of activity of the company:

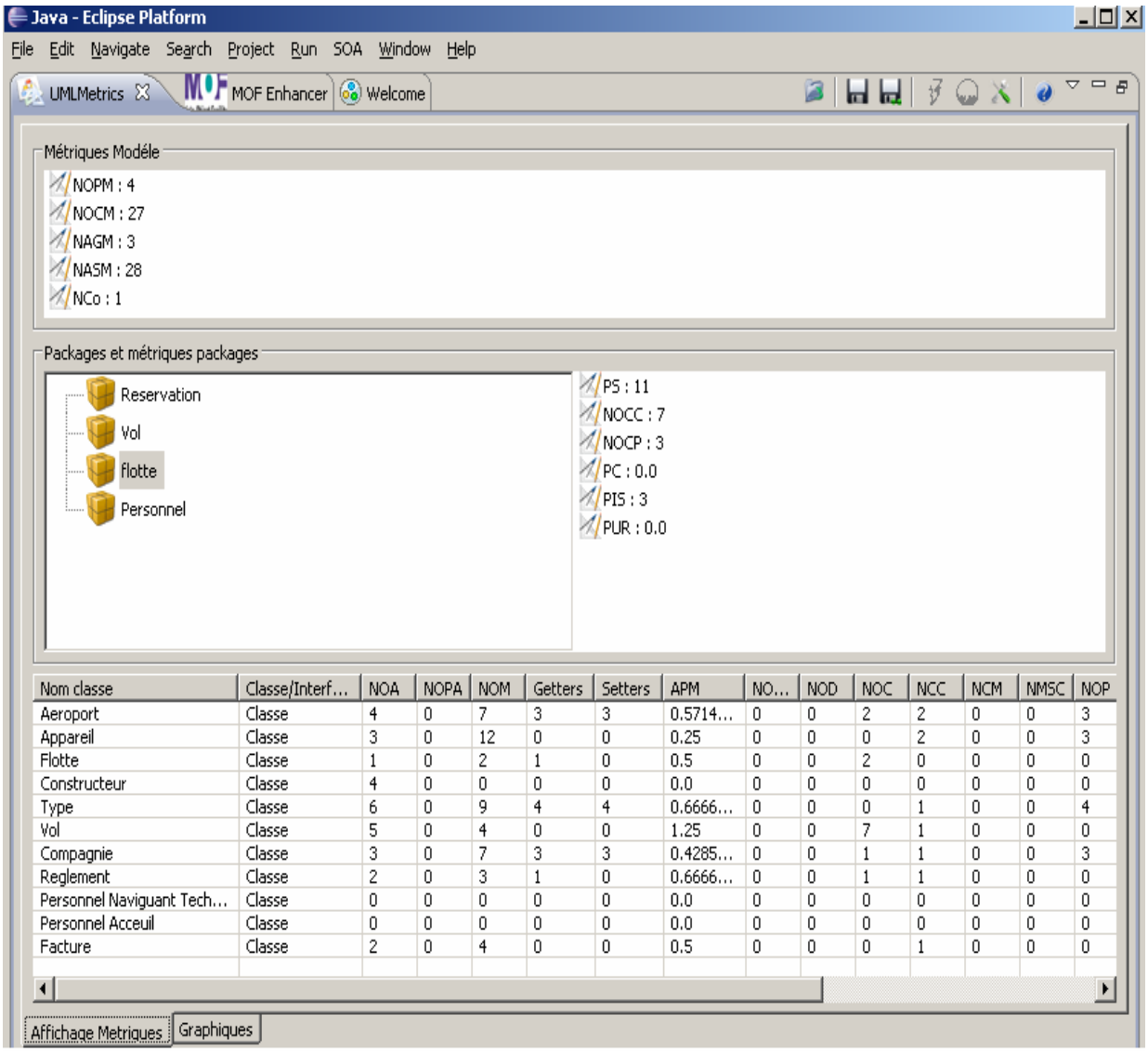
Management of Devices
Management of the Flights: planning
Management of the Reservations
Management of Staff

We modeled the problem by using "Poseidon for UML". Le model present 4 classes diagrams and 61 sequences diagrams. The sequences diagrams represent all possible scenarios in our use case. This constraint is fundamental for the analyze efficiency when measuring metrics values.

A. MRefactor Tool

We follow the refactoring process we proposed. The domain analysis and modeling steps are done only once. We have already studied design defects and modeled all refactoring opportunities using the extended UML model. These two steps are the basis for the implementation of the refactoring tool we proposed. The Refactoring opportunities detection step is based on the analysis of the class diagram and all sequences diagrams representing all scenarios.

As shown in figure 4, the first operation MRefactor will perform is the measure of metrics values. The UML diagrams are transformed in an XMI file which constitutes the input data.



Nom classe	Classe/Interf...	NOA	NOPA	NOM	Getters	Setters	APM	NO...	NOD	NOC	NCC	NCM	NMSC	NOP
Aeroport	Classe	4	0	7	3	3	0.5714...	0	0	2	2	0	0	3
Appareil	Classe	3	0	12	0	0	0.25	0	0	0	2	0	0	3
Flotte	Classe	1	0	2	1	0	0.5	0	0	2	0	0	0	0
Constructeur	Classe	4	0	0	0	0	0.0	0	0	0	0	0	0	0
Type	Classe	6	0	9	4	4	0.6666...	0	0	0	1	0	0	4
Vol	Classe	5	0	4	0	0	1.25	0	0	7	1	0	0	0
Compagnie	Classe	3	0	7	3	3	0.4285...	0	0	1	1	0	0	3
Reglement	Classe	2	0	3	1	0	0.6666...	0	0	1	1	0	0	0
Personnel Naviguant Tech...	Classe	0	0	0	0	0	0.0	0	0	0	0	0	0	0
Personnel Acceuil	Classe	0	0	0	0	0	0.0	0	0	0	0	0	0	0
Facture	Classe	2	0	4	0	0	0.5	0	0	0	1	0	0	0

Fig. 4: the metrics measure

At the two first steps we have identified for each design defect a metrics based heuristic and the diagrams form which it could be measured. Based on the value of this heuristic our tool infer that the input model suffer from such design defect. Once the detection done, the first output is graphically visualized for users. In fact the plug-in we propose allows the graphical representation of the UML class diagram. As presented in figure 5, the originality of this visualization reside in the fact that, as described in model tagging step, the refactoring proposed solution are represented using the new element we introduced the RefactoringTag.

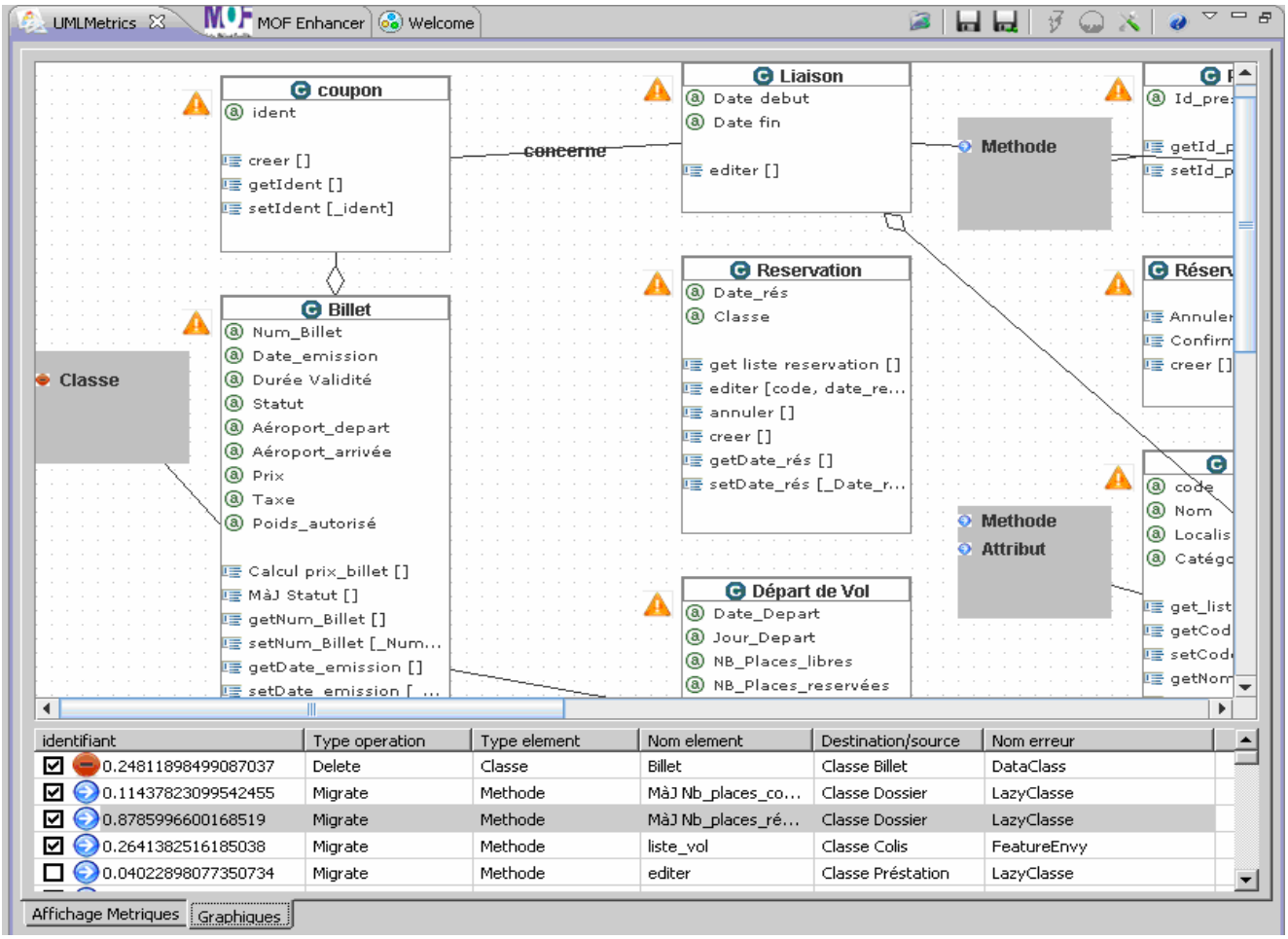


Fig. 5: The refactoringTag presentation

This element shown in gray color contains the information needed for the refactoring automation process, and when it is attached to one element in the model it gives graphical, easy and very comprehensive information on the design defect and the element in which it occurs. Users have only to validate the proposed solutions. Each solution is composed by:

The operation type: represents the refactoring primitive that will be applied to restructure the model (i.e. delete operation, migrate …)
The element type: is the model element concerned by the design defect.
The element name: represents the model element name.

The source/destination: represents the element in which the modification will occur. For example in the case of primitive “migrate operation”, it will indicate the class in which a given operation will be migrated.

The design flaws name: is the name of the design defect.

In figure 5 the second refactoring suggestion proposed indicate that MRefactor will migrate the method “Maj Nb_places_commander” to the class “Dossier” due the design defect “LazyClasse” the designer have to confirm this proposition.

B. Results

We define the user precision as the Number of users divide by the number of detection it gives an idea on the degree of efficiency of the tool for users, the value integrate the subjective aspect of design domain. Users can ignore some design defects.

Table I: Refactoring Results

Number of classes	Number of Design flaws	Number of detection	Number of user Validation	Precision for users	General Precision
100	24	21	15	0,714	0,875

The General Precision represents the number of detection divide by the Number of Design flaws it gives an idea on the degree of efficiency of MRefactor for the detection of all the design defects in a given design.

MRefactor gives an encouraging results, it has allows non experts in design to made model refactoring faster and easier as well as for expert.

IV. CONCLUSION

This paper described our approach to model driven refactoring, specifying the refactoring of MOF models by in-place model transformation. We proposed an UML metamodel extension for assisted refactoring assisting the users to create their own refactoring.

Our approach focus on model level which is more abstract and allows the creation of generic refactoring primitive. As presented in the extended UML metamodel, the analyses of design flaws and metrics measure are the basis of the assisted detection of the refactoring opportunities. We consider an important point which is the traceability between original model and the refactored one. RefactoringTag illustrate all modifications applied on the source model. They could be transformed in textual description associated to the transformation module.

MRefactor is model refactoring tool, but it help also non expert to understand the notion of good design, since it make explicit many notion (Anitpatterns, BadSmells) that are subject to confused interpretations.

REFERENCES

[1] Chidamber, S., Kemerer, C.: A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6) (1994) 476–493.

[2] Corradini, A., H. Ehrig, H.-J. Kreowski and G. Rozenberg, editors,"Graph Transformation" Lecture Notes in Computer Science 2505, Springer-Verlag, 2002.

[3] E. J. Chikofsky and J. H. Cross. "Reverse engineering and design recovery - a taxonomy". IEEE Software, pages 13–17, January 1990.

[4] Ethan Hadar, Irit Hadar, "The Composition Refactoring Triangle (CRT) Practical Toolkit: From Spaghetti to Lasagna", OOPSLA 2006, Portland, Oregon, USA. ACM 1-59593-491-X/06/0010.

[5] E. Biermann, K. Ehrig, G. Kuhns, C. Kohler, G. Taentzer, and E.Weiss. "EMF Model Refactoring based on Graph Transformation Concepts", Electronic Communications of the Easst, Volume 3, 2006.

[6] J.M. Bieman and B.K. Kang. Cohesion and Reuse in an Object- Oriented System. Proc. ACM Symposium on Software Reusability, apr 995.

[7] Marc Van Kempen, Michel Chaudron, Derrick Kourie, Andrew Boake, "Towards Proving Preservation of Behaviour of Refactoring of UML Models", in proceedings of SAICSIT 2005, Pages 252.

[8] MDA Guide Version 1.0.1, omg/2003-06-01, 12th June 2003. Accessed on Jan 2006.

[9] Request for Proposal: MOF 2.0 Query /Views /Transformations RFP, OMG Document, 2002.

[10] Rui, K. and Butler, G. (2003). "Refactoring Use Case Models : The Metamodel". In Proc. Twenty-Sixth Australasian Computer Science Conference (ACSC2003), Adelaide, Australia. CRPIT, 16. Oudshoorn, M.J., Ed. ACS. 301-308.

[11] Raul Marticorena, "Analysis and Definition of a Language Independent Refactoring Catalog", 17th Conference on Advanced Information Systems Engineering (CAiSE 05). Portugal., page 8, jun 2005.

[12] Raul Marticorena and Yania Crespo. Refactorizaciones de especializacion sobre el lenguaje modelo MOON. Technical Report DI-2003-02, Departamento de Informatica. Universidad de Valladolid, septiembre 2003.

[13] R. Marinescu. Detecting Design Flaws via Metrics in Object-Oriented Systems. In Proceedings of TOOLS USA 2001, pages 103–116. IEEE Computer Society, 2001.

[14] Riel, A.J.: Object-Oriented Design Heuristics. Addison-Wesley (1996)

[15] Sander Tichelaar, Stéphane Ducasse, Serge Demeyer, Oscar Nierstrasz. "A Meta-model for Language-Independent Refactoring", published in the proceedings of ISPSE 2000.

[16] William F. Opdyke. Refactoring Object-Oriented Frameworks. PhD dissertation, University of Illinois at Urbana-Champaign, Department of Computer Science, 1992.

[17] Zhang, J., Lin, Y. and Gray, J. (2004) "Generic and Domain-Specific Model Refactoring using a Model Transformation Engine", Model-driven Software Development – Research and Practice in Software Engineering, accepted for publication in 2005.

[18] Maddeh Mohamed, Mohamed Romdhani, Khaled Ghédira: Classification of Model Refactoring Approaches. Journal of Object Technology 8(6): 143-158 (2009).