Class model extraction from procedural code: Confronting a few ideas from the 2000's against the reality of an industrial system

Marianne Huchard, Ines Ammar, Ahmad Bedja Boana, Jessie Carbonnel, Theo Chartier, Franz Fallavier, Julie Ly, Daniel Alias Nguyen Vu-Hao, Florian Pinier, Ralf Saenen, et al.

Class model extraction from procedural code: Confronting a few ideas from the 2000's against the reality of an industrial system

Marianne Huchard*, Ines Ammar, Ahmad Bedja Boana, Jessie Carbonnel, Theo Chartier, Franz Fallavier, Julie Ly, Vu-Hao Alias Daniel Nguyen, Florian Pinier, Ralf Saenen, and Sebastien Villon

LIRMM, Université Montpellier 2 et CNRS
{marianne.huchard}@lirmm.fr
http://www.lirmm.fr/~huchard

Abstract. In this extended abstract, we report an ongoing experience conducted during a Master project on the migration of two industrial software systems. The project was proposed by a major IT service company (not cited here for confidentiality reasons) which would like to investigate a migration processing chain, in order to renovate legacy software composed of man-machine interfaces, databases and procedural source code. The aim of the renovation is to migrate to the object-oriented paradigm and generate new source code. Due to the limited time that the Master students had in their curriculum for this project, we restricted the study to the extraction of a class model. A processing chain was proposed and a few heuristics, taken in the literature, have been tested, with no really conclusive results. We report here the current status of the project in order to get feedback and new ideas to build for the future.

Keywords: Software reengineering, software migration, class model extraction, object identification

1 Context and problematics

Software renovation still remains a costly and time-consuming process for IT service companies, that can be viewed as a waste of resource, compared to the development of new software. Nevertheless, if timely and effective measures are not taken to regularly update the design, the source code, the documentation and all related artifacts, it may arrive the day where the software can no longer be understood by human, or compiled by the new compilers, or even ran on the new servers. The challenge is to maintain and migrate with a low cost the legacy software, before real problems arise. In this extended abstract, we report an

* The authors would like to thank the IT service companies that brought the renovation project and followed the work in progress and the master students (Luc Debène, Chaymae Regragui and Cedric Cambon) that made a tutorial for the use of Famix in the context of this project.

experience, in which we designed a processing chain for renovating a particular legacy software.

The two studied legacy software systems are part of a larger software suite, but can be analyzed independently. They are composed of code describing man/machine interface (HTML, VBScript/ASP, Javascript), Visual Basic Code (VB6), SQL procedures (SQL Server 2000) and two databases. The source code (VB and SQL) is composed of 909 functions, 30437 LOC for the largest software and 346 functions, 26042 LOC for the smallest software. One database contains 45 tables, while the other contains 103 tables.

Due to the limited time that the Master students had in their curriculum for this project, we restricted the study to the extraction of a class model. The students were divided in three groups, but they collaborated throughout the project. In the next sections, we develop the proposed approach and its current results (Section 2), then we conclude in Section 3.

2 The renovation approach

The global process The current study is concerned with specific programming languages, but we aim at proposing a generic processing chain which could be applied to other procedural and database programming languages (input) and other object-oriented language (output) of the same company. We decided to follow a classical processing chain such as that which is presented in Fig. 1, organized around an intermediate model.

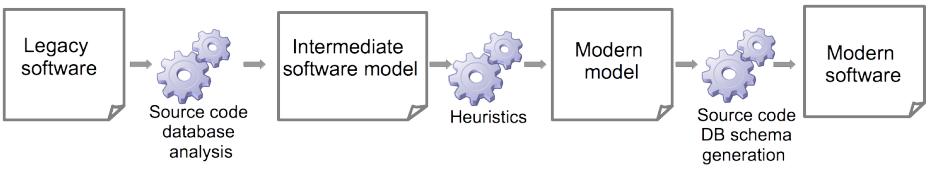

Fig. 1. The generic process

Fig. 2 shows the current instantiated process that we follow. We describe in next paragraphs the tested tools and methods of each step.

Intermediate model The choice of the intermediate model was partly guided by simplicity reasons (avoiding complex UML meta-model as proposed in UML2tools for example) and by the needs of the envisaged heuristics (study data access and function calls to determine connected sets of data and functions). In order to represent the source artifacts and the class model, we chose the FAMIX meta-model and its MSE serialization format [5]. For the representation of the output class model, this was quite natural, because of its language independency and its ability to describe the static structure of object-oriented software: main used

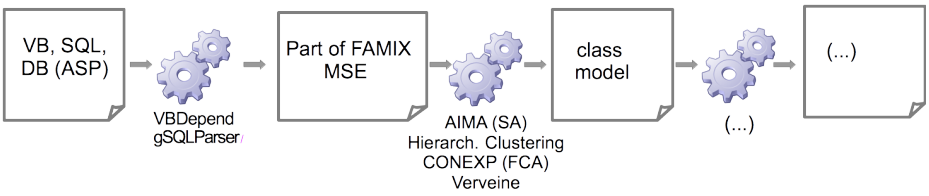

Fig. 2. The current process

concepts are meta-classse Class, Method, Attribute, Access and Invocation. For the representation of our input model, we partly used FAMIX in an unusual manner. Procedural code was easily represented using Function and Invocation. We met a problem for the representation of database elements. FAMIX being extensible, we thought that it would be the right approach to add meta-classes for representing data table and columns. Due to the delay, we provisionally abandon this track, but it remains a future work. Then, we used meta-classes Class and Attribute to respectively represent tables and columns, even if we don't consider the solution satisfactory. In this step, we found a difficulty in establishing a common strategy for the three student groups for the use of FAMIX, because initially, the groups were using different attributes of the meta-classes for representing the same information (*e.g.* using signature in Invocation meta-class, versus candidates).

Source code analysis Man-machine interface source code was partially manually analyzed, but in a first approach, we decided to abandon the track because of the little domain knowledge that this code seemed (at first sight) expose. This will be study into more details in future work. At the beginning, we would like to analyze VB code using Microsoft Visual studio, but it reveals to be impossible because of the old version of VB code. This again shows the importance to regularly update software source code. An evaluation version of VBdepend [1] was tested and allowed us to retrieve functions, invocations and parameters. As we were interested in finding the VB functions and the SQL functions that manipulated the database they called, some specific code has been developed because in the source code, this is done via a same function that takes as parameters the called SQL function and its parameters. SQL code has been analyzed with GSP [2] to extract which SQL procedure has which kind of access to which tables (and which columns of the tables). We met a few problems in using the two tools, but more than 99% of the VB functions and more than 91% of the SQL procedures were correctly analyzed. MSE entities coming from SQL analysis and from VB analysis are merged to give a unique MSE file for the remainder of the project.

Class model heuristics extraction Each student group had three or four papers about "object identification" (the term used in 2000's for talking about class

[1] http://www.vbdepend.com
[2] http://www.sqlparser.com

model extraction) and they had to choose one for implementation [9, 2–4, 7, 1, 6, 8, 10]. The three methods of [4, 6, 9] have been tried. The approach of [4] is an ad hoc method based on a hierarchical clustering technique that we had to entirely implement. The clusters are composed of columns "similarly accessed" by functions. In [6], they compare several meta-heuristics and we chose among them the simulated annealing method. The framework AIMA [3] has been used for the tests. It requires to implement a few Java interfaces by accessing the MSE files and computing a neighbor solution as well as cohesion and coupling metrics (on which an objective function is based) on the current solution (a set of candidate classes composed of data and functions). The approach of [9] uses Formal Concept Analysis to form concepts composed of functions and their accessed columns. After concept lattice building, an ad hoc algorithm merges some concepts and assigns the functions to the classes.

VerveineJ [4] is used in all implementations for finding the entities that are taken as input of heuristics and MSE files are generated that contain the output class models.

Current results The hierarchical clustering based implementation allows us to find different size solutions, depending the chosen level for stopping the clustering. It produces classes which mainly correspond to connected groups of tables of the initial databases, thus with an explainable logics. Its current issue concerns the methods assignment (only functions that access to a single class are assigned to that class), thus a complementary approach has to be proposed. The metrics to be used in simulated annealing approach and their respective weights were not sufficiently described, and for the moment we are not able to obtain the same results as those obtained by the author on the example given in the paper. The FCA approach produces many classes (compared to the database table number). Some obtained classes have a coherence for people of the company, but some seem to be incidental groups of attributes and methods, technically explained by accesses and invocations, but with no clear meaning. Besides, even if the concept lattice guarantees maximal factorization with no redundancies, the applied post-treatments generate a huge attribute and method duplication.

3 Conclusion

In this extended abstract, we reported an ongoing work that aims at extracting a class model from procedural code, relying on work mostly done in the 2000s. Some difficulties were met to apply the methods: some are dedicated to specific contexts (as COBOL code, or C code) and they are not so generic; some parameters of the methods are not very precisely described, leaving space for interpretation. As we also think that structural aspects, including data access and function invocation are not sufficient to determine meaningful classes, we plan to investigate software identifier analysis. Besides, we expect that man-machine

[3] http://code.google.com/p/aima-java/
[4] http://www.moosetechnology.org/tools/verveinej

interface code or execution trace could be an help to extract connected functionalities. Last, as these techniques have to be understood as a technical assistance for expert engineers, we would like to study how the user should intervene in the process.

References

1. Bhatti, M.U., Ducasse, S., Huchard, M.: Reconsidering classes in procedural object-oriented code. In: International Conference on Reverse Engineering (WCRE) (2008), http://rmod.lille.inria.fr/archives/papers/Bhat08b-WCRE2008-ObjectIdentification.pdf
2. Canfora, G., Cimitile, A., Lucia, A.D., Lucca, G.A.D.: A case study of applying an eclectic approach to identify objects in code. In: IWPC. pp. 136–143. IEEE Computer Society (1999)
3. Cimitile, A., Lucia, A.D., Lucca, G.A.D., Fasolino, A.R.: Identifying objects in legacy systems using design metrics. Journal of Systems and Software 44(3), 199–211 (1999)
4. van Deursen, A., Kuipers, T.: Identifying objects using cluster and concept analysis. In: Boehm, B.W., Garlan, D., Kramer, J. (eds.) ICSE. pp. 246–255. ACM (1999)
5. Ducasse, S., Anquetil, N., Bhatti, M.U., Hora, A.C., Laval, J., Girba, T.: Mse and famix 3.0 : an interexchange format and source code model family. Tech. Rep. Cutter-Deliverable 22, ANR 2010 BLAN 0219 02, RMod INRIA Lille-Nord Europe (November 2011), http://rmod.lille.inria.fr/archives/reports/Duca11c-Cutter-deliverable22-MSE-FAMIX30.pdf
6. Glavas, G., Fertalj, K.: Solving the class responsibility assignment problem using metaheuristic approach. CIT 19(4), 275–283 (2011)
7. Lucca, G.A.D., Fasolino, A.R., Guerra, P., Petruzzelli, S.: Migrating legacy systems towards object-oriented platforms. In: ICSM. pp. 122–129. IEEE Computer Society (1997)
8. Maletic, J.I., Marcus, A.: Supporting program comprehension using semantic and structural information. In: Müller, H.A., Harrold, M.J., Schäfer, W. (eds.) ICSE. pp. 103–112. IEEE Computer Society (2001)
9. Sahraoui, H.A., Lounis, H., Melo, W.L., Mili, H.: A concept formation based approach to object identification in procedural code. Autom. Softw. Eng. 6(4), 387–410 (1999)
10. Zou, Y., Kontogiannis, K.: Incremental transformation of procedural systems to object oriented platforms. In: COMPSAC. pp. 290–295. IEEE Computer Society (2003)