

Reverse Engineering of Object Oriented Code

Research Leader: Bruno Caprile

Participants: Alessandra Potrich and Paolo Tonella

Keywords: Object Oriented Programming, Code Analysis, Unified Modeling Language (UML).

The Problem: During maintenance, the most reliable and accurate description of the actual behaviour of a software system is its source code. However, not all questions about the system can be answered directly by resorting to this repository of information. What the reverse engineering methodology aims at is the extraction of abstract, goal-oriented “views” of the system, able to summarize relevant properties of the computation performed by the program.

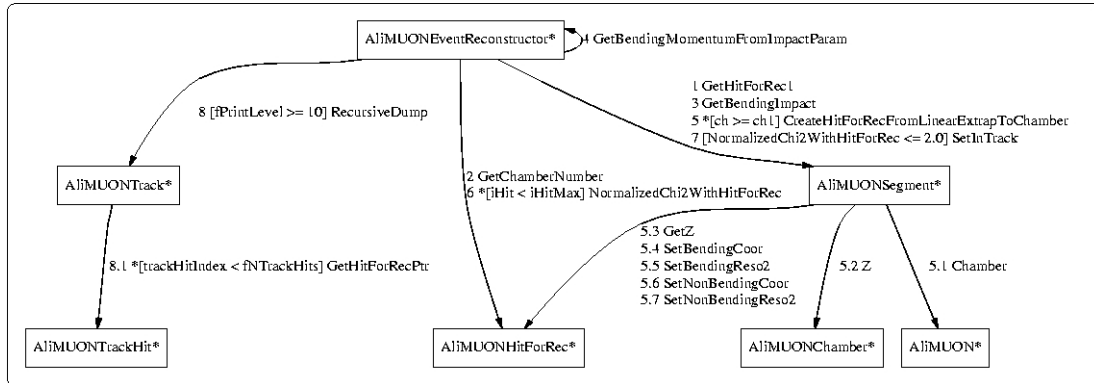


Figure 1: An example of UML Collaboration Diagram containing a set of communicating objects, as automatically extracted from the C++ code of the Alice experiment at CERN.

Motivation: Software is among the most complex human artifacts. The way it is produced and maintained is thus an interesting object of study in itself. Among the various phases of software development, maintenance is by far the most demanding and expensive along the whole software’s life cycle. Difficulties are mainly related to the understanding of extant systems and the estimate of the impact of changes. Reverse engineered diagrams provide useful information about the system being maintained. In this respect, they can support the program understanding activities, drive refactoring and restructuring interventions. Finally, they can be employed to trace back the compliance of the actual code with the intended design.

Previous Work: The difficulties in reverse engineering and maintenance of object oriented applications are discussed in [R1] and [R2]. Although available commercial tools (e.g., Rose, Together) can extract the UML Class Diagram from the code, it is only in a few research works that the problem of filtering relevant information and providing accurate and meaningful high level views has been addressed [R3], [R4]. For example, in presence of weakly typed containers a high risk exists of missing relations based on containers in the recovered class diagrams. A dynamic analysis of the interactions among the objects in an object oriented system is a relatively easy task [R5],[R6],[R4],[R7]. However, the diagrams that can be reverse engineered in this way are very incomplete – holding for a single, given execution of the program and given input values.

In our first works [P1],[P2] we concentrated on the problem of design-code compliance. We defined a process and developed tools able to recover an “as is” design from the code, and compare the recovered design with the actual design. The proposed approach has been experimented on industrial design and code provided by Sodalia SpA.

Approach: With respect to the traditional software development process, what we are dealing with is an *inverse problem*, in which higher level views are extracted moving backward from the implementation. Since several properties of the programs are in general undecidable, the views being extracted are necessarily approximate, and have to resort to heuristics and special purpose

algorithms. Nonetheless, they are conceived in such a way as to account for properties and limitations of humans' cognitive abilities: in particular, they have to be usable, of manageable size, and conservative in the approximations made.

Preliminary Results: Our recent work focused on the problems related to the extraction of UML diagrams from C++ code. In [P3] an algorithm is proposed for the inference of the container types. In [P4] two techniques for the automatic extraction of the object diagram are also investigated. The first technique is based on a static analysis of the source program; the second is based on the execution of the program on a set of test cases.

Main Publications:

[P1] G. Antoniol, B. Caprile, A. Potrich and P. Tonella: “*Design-Code Traceability for Object Oriented Systems.*” Ann. of Soft. Eng., Vol. 9, pp. 35-58, 2000.

[P2] G. Antoniol, B. Caprile, A. Potrich and P. Tonella: “*Design-Code Traceability Recovery: Selecting the Basic Linkage Properties.*” Sci. Comp. Prog., Vol. 40, N. 2-3, pp. 213-234, July 2001.

[P3] P. Tonella and A. Potrich: “*Reverse Engineering of the UML Class Diagram from C++ Code in Presence of Weakly Typed Containers.*” In Proc. of ICSM 2001, International Conference on Software Maintenance, pp. 376-385, Florence, Italy, November 7-9, 2001.

[P4] P. Tonella and A. Potrich: “*Static and Dynamic C++ Code Analysis for the Recovery of the Object Diagram.*” In Proc. of ICSM 2002, International Conference on Software Maintenance, Montreal, Canada, October 2002.

Impact: The impact we envisage is two-fold: first, we target the Software Engineering and Reverse Engineering communities through qualified contributions on journals and conferences. Secondly, we apply the outcomes of research to industrial systems – thereby empirically assessing the validity of our approach. In this respect, we currently provide reverse engineering and code analysis tools to 3 of the Large Hadron Collider (LHC) experiments of CERN – the European Center for Particle Physics in Geneva, (CH).

Future Work: UML diagrams related to the behavior of the objects and to the evolution of their internal state are not reverse engineered by any available tool or research prototype. We have started investigating the problem of recovering them. In addition to program understanding, possible usages of such kind of information are in the verification and validation of Object Oriented software, an area in which our analyses can be of high potential impact.

Research Support: Research contract with CERN, Geneva (CH).

Collaborations: Alice, ATLAS and LHCb Large Hadron Collider (LHC) experiments at CERN. Università del Sannio, Italy; Universities of Durham (UK) and Montreal (Canada).

References:

[R1] M. Lejter, S. Meyers, and S. P. Reiss: “*Support for Maintaining Object-Oriented Programs.*” IEEE Trans. on Soft. Eng., 18(12):1045-1052, Dec. 1992.

[R2] N. Wilde and R. Huitt: “*Maintenance support for object-oriented programs.*” IEEE Trans. Soft. Eng., 18(12):1038-1044, Dec. 1992.

[R3] L. Larsen and M. Harrold: “*Slicing Object-Oriented Software.*” Proc. of the Int. Conf. on Soft. Eng., pp. 495-505, 1996.

[R4] T. Richner and S. Ducasse: “*Recovering High-Level Views of Object-Oriented Applications from Static and Dynamic Information.*” In Proc. of the Int. Conf. on Software Maintenance, pp. 13-22, Oxford, England, 1999.

[R5] K. Koskimies and H. Mossenbock: “*Scene: Using Scenario Diagrams and Active Test for Illustrating Object-Oriented Programs.*” In Proc. of Int. Conf. on Soft. Eng., pp. 366-375, Berlin, Germany, March 25-29 1996.

[R6] W. D. Pauw, D. Kimelman, and J. Vlissides: “*Modeling Object-Oriented Program Execution.*” In Proc. of ECOOP'94 - Lecture Notes in Computer Science, pp. 163-182. Springer-Verlag, July 1994.

[R7] R. J. Walker, G. C. Murphy, B. Freeman-Benson, D. Wright, D. Swanson, and J. Isaak: “*Visualizing Dynamic Software System Information through High-Level Models.*” In Proc. of the Conf. on Object-Oriented Programming, Systems, Languages, and Applications, pp. 271-283, Vancouver, British Columbia, Canada, October 18-22, 1998.