UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
DISCo - Dipartimento di Informatica Sistemistica e Comunicazione

MARPLE

Metrics and Architecture Recognition Plug-in for Eclipse

Project team:

Francesca Arcelli	arcelli@disco.unimib.it
Stefano Maggioni	stefano.maggioni@disco.unimib.it
Christian Tosi	christian.tosi@essere.disco.unimib.it
Marco Zanoni	marco.zanoni@essere.disco.unimib.it

A software engineering area that is getting prominent in the vast field of software maintenance and evolution is reverse engineering. One of the aims of this discipline aims at the obtainment of views of already existing complex software systems, in order to try to understand which are its constituent components and have a general "easy to manage" view of its architecture. This activity should heavily simplify the system's management and re-documentation phases, as engineers won't focus onto programming details and won't have to manage directly the source code, but will on the other side work exclusively on the system architecture, focusing on those parts that look critical eventually going into their details.
In the ambit of this discipline we propose a tool called MARPLE (Metrics and Architecture Recognition Plug-in for Eclipse) which focuses on the detection of design patterns as possible components of systems architectures, trying to infer their presence starting from a static analysis of the source code aimed at the extraction of little code structures which, resembled, could be significant indicators of the presence of certain design patterns inside the code. Moreover, it provides means for software architecture reconstruction activities, allowing the users to understand the relationships among the various components of the analyzed systems, dealing with their overall architectures views, which help in managing their complexity.

Reverse engineering and design pattern detection

A software engineering research area that is getting more and more importance for the maintenance and evolution of software systems is reverse engineering. An important objective of this discipline is to let the engineer identify the fundamental components of an analyzed system

and obtain its constituent structures. Getting this information should heavily simplify the restructuring and maintenance activity, as the system would be seen as a set of little coordinated components, rather than as a unique monolithic block.
Considering these structures, particular relevance is given to *design patterns* [GHJV94]. They are extremely useful during the project design phases, as they can be considered as a sort of directives to follow in order to solve a problem in a defined context. In fact, a design pattern describes a problem that can be faced a lot of times and the core of the solution to that problem, so that the solution can be used many times.
Finding these design patterns in a software system can therefore give hints on what kind of problems have been found during the development of the system itself; the presence of design patterns (assuming that their implementation is done correctly) can be considered as an indication of good software design, which demonstrates to have been done with the help of structures that are, for their self definition, reusable.
In this perspective, design pattern detection is very useful for the comprehension of a software system: design patterns can give some helpful information about the organization of the system, indicating the logical fundamentals of its implementation. Moreover, they are very important during the re-documentation process, in particular when the documentation is very poor, incomplete or not up-to-date.
Different tools for design pattern detection have been proposed in the literature (e.g. [AG01, AR06, FBTG02, NS+02, SO05a, SO05b, Smi05]). They usually have problems in finding all the design patterns of the GoF catalogue [GHJV94], some tools recognize only a small subset of these patterns, many false positive results are found and moreover they usually don’t work well when trying to analyze medium/large systems: scalability is surely a critical problem.
Our approach to design pattern detection is based on design pattern decomposition [AR05, AMRT05, AMR06, Mag06], obtaining from the source code those structures that can be considered indicators of the presence of patterns in the source code. We use static analysis techniques in order to parse the ASTs of the analyzed projects and to obtain the structures we need for our inspection.
But we should remember that one of the main objectives of reverse engineering is the obtainment of information about the general architectures of the analyzed systems in their final state. This activity is commonly known as *Software Architecture Reconstruction* (SAR). SAR activities are fundamental while dealing with large-sized or obsolete systems, where a general overview on the architecture, on the modules and on the functionalities exposed by the system can enhance its comprehension and its maintenance. Many tools for SAR have been proposed, both coming from the industry and from the academia. IBM developed Structural Analysis for Java [SA4J], a tool that analyzes the jar files of any Java application and provides views for its architectural analysis, for the underlining of critical or complex classes and components, for the analysis of the components involved in an eventual system modification, and moreover for the calculation of metrics about systems complexity and stability. IBM’s Rational Architect [RA] provides SAR functionalities for the obtainment of static diagrams (like UML class diagrams) that should represent the architecture of a target system. Other tools, like CodeLogic [CL], also provide the generation of dynamic diagrams (like sequence diagrams and flow charts) that help in the comprehension of the execution process of a well defined functionality. Another category of tools derive their results basing on the system documentation. Doxygen [Doxy] generates enriched documentation (also completed with different kinds of diagrams generated with the GraphViz tool [GV]) starting from the javadoc tags that can be found inside source code.
Some other tools work on a separately generated model of the target system. This is the case of CodeCrawler [CC], which accepts a FAMIX model as input, to be obtained with ad-hoc tools. The views obtained with this tool are exclusively based on the calculation of metrics. The results are presented through polymetric views, where each rectangle (representing a class, a method, or an attribute, depending on the view) is associated with at most five metrics identified by color, dimensions and position of the rectangle itself. Basing on these views, the user can have an idea about the system complexity, the critical components and the interactions among the various components.

An overview of the MARPLE project

As introduced before we are developing a source code analysis tool, called *MARPLE* (*Metrics and Architecture Recognition PLug-in for Eclipse*), that will be able to detect with good approximation the presence of software architectures, at now focusing on design patterns, inside a Java program. The tool is being developed as a plug-in for the Eclipse framework. In addition to the design pattern recognition, our project will also provide other means to support reverse engineering activities, such as calculation of metrics for the visual inspection of the code itself and measurements on the code and on the architecture of a software system, all integrated with the design pattern detection module.
We can see a general architecture of our tool in the next figure.

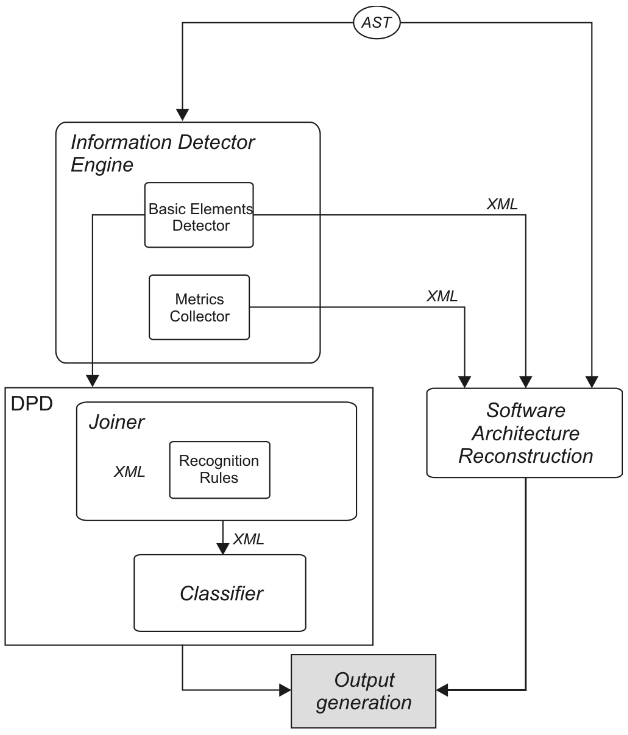

Figure 1 - The architecture of the MARPLE project

It is constituted by 4 main modules, that interact with one another through XML data transfers.
The 4 modules are:

- *Information Detector Engine*: starting from an AST representation of the source code of the analyzed project, it collects both basic elements (namely design pattern clues [Mag06], elemental design patterns [Smi02a, Smi02b] and micro patterns [GM05]), which are to be used in the actual process of design pattern detection, and metrics that could be useful indicators of the presence of design patterns inside the code;
- The *Joiner*, that extracts architectures from the project that could match those of design patterns, basing on the information extracted by the *Information Detector Engine*, considering the extracted classes as graph nodes and the basic elements as edges connecting those nodes;
- The *Classifier*, which tries to infer whether the structures detected by the *Joiner* could effectively be realizations of design patterns or not. This module helps to detect possible false positives identified by the *Joiner* and to evaluate the similarity with the theoretical design patterns by assigning different levels of probabilities;
- The *Software Architecture Reconstruction* module, which obtains abstractions from the target project basing on the elements and metrics extracted mainly by the *Information Detector Engine*, but also directly from the ASTs of the analyzed system;
- The activity of *Output generation* provides an organic view of the project analysis results. Through this activity, the user will see both the results produced by the detection of design patterns and the views provided by the SAR module.

Currently, the *Information Detector Engine* and the *Joiner* module have been completely developed, together with the views that let the user deal with their results. The *Basic Elements Detector* identifies 48 clues that are hints for the various GoF design patterns, while the *Metrics Collector* calculates various object-oriented metrics that are to be used during the SAR activities.
The *Joiner* has been completely developed as far as its algorithms are concerned, we have at now defined the rules for the extraction of the candidates for 8 design patterns, and we are proceeding in the definition of the remaining ones.
We have started the development of the SAR modules, which at now provides six views for the analysis of the architectures of complex Java systems. These views are both generated by metrics values (like what happens with CodeCrawler) and by the analysis of the basic elements identified by the *Information Detector Engine.* We are now working on the extension of this module for the implementation of new views in order to make it more complete.

References

[AR05] F. Arcelli, C. Raibulet, "The Role of Design Pattern Decomposition in Reverse Engineering Tools", *Pre-Proceedings of the IEEE International Workshop on Software Technology and Engineering Practice* (*STEP 2005*), Budapest, Hungary, September, 24th -25th, 2005, pp. 230-233.

[AMRT05] F. Arcelli, S. Masiero, C. Raibulet, F. Tisato, "A Comparison of Reverse Engineering Tools based on Design Pattern Decomposition", *Proceedings of the IEEE Australian Software Engineering Conference* (*ASWEC 2005*), Brisbane, Australia, March, 27th -31st, 2005, pp. 262-269.

[AMR06] F. Arcelli, S. Masiero, C. Raibulet, "Elemental Design Patterns Recognition in Java", *Post-Proceedings of the 13th Annual International Workshop on Software Technology and Engineering Practice* (*STEP 2006*), IEEE Computer Society, Budapest, HU, September 24th - 25th, 2006.

[AR06] F. Arcelli, C. Raibulet, "Program Comprehension and Design Patterns Recognition: an experience report", submitted to the *Workshop on object-oriented reverse engineering*, *ECOOP 2006 Workshop*, Nantes France, April 2006.

[AG01] H. Albin-Amiot, Y.-G. Guéhéneuc, "Meta-modeling Design Patterns: application to pattern detection and code synthesis", In *ECOOP 2001 Workshop on Adaptative Object- Models and Metamodeling Techniques*, 2001.

[CC] CodeCrawler, www.inf.unisi.ch/faculty/lanza/codecrawler.html

[CL] CodeLogic, www.logicexplorers.com/

[Doxy] Doxygen, www.doxygen.org/

[FBTG02] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy, "Columbus – Reverse Engineering Tool and Schema for C++", in *Proceedings of The 6th International Conference on Software Maintanance* (*ICSM 2002*), pages 172-181. IEEE Computer Society, Oct. 2002.

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented* Software, Addison Wesley, Reading MA, USA 1994.

[GM05] J. Gil, I. Maman, "Micro Patterns in Java Code", *Proceedings of the 20th Conference on Object-Oriented Programming Systems Languages and Applications* (*OOPSLA'05*), October 2005.

[GV] GraphViz, www.graphviz.org/

[Mag06] S. Maggioni, "A new approach to design pattern recognition through design patterns clues", M.Sc. Thesis, University of Milano-Bicocca, Italy, February 2006.

[NS+02] J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, J. Welsh, "Towards Pattern-Based Design Recovery". In *Proceedings of the 24th International Conference on Software Engineering* (*ICSE*), Orlando, Florida, USA. ACM Press. May, 2002.

[RA] IBM Rational Architect, www-01.ibm.com/software/awdtools/architect/swarchitect/index.html

[SA4J] IBM Structural Analysis for Java, www.alphaworks.ibm.com/tech/sa4j

[SO05a] Nija Shi and Ronald A. Olsson, "Reverse Engineering of Design Patterns from Java Source Code", Department of Computer Science, University of California,2005. www.cs.ucdavis.edu/~shini/research/pinot

[SO05b] N. Shi, R. A. Olsson, “Reverse Engineering of Design Patterns for High Performance Computing”, Department of Computer Science, University of California, Davis. In *Proceedings of the 2005 Workshop on Patterns in High Performance Computing*, 2005.

[Smi02a] J. McC. Smith, “An Elemental Design Pattern Catalog”, Technical Report TR02-040, Department of Computer Science, University of North Carolina at Chapel Hill. December 10, 2002.

[Smi02b] J. McC. Smith and D. Stotts, “Elemental Design Patterns: A Formal Semantics for Composition of OO Software Architecure”, Department of Computer Science, University of North Carolina at Chapel Hill. In *Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop* (*SEW-27'02*), p.183, December 05-06, 2002.

[Smi05] J. McC. Smith and D. Stotts, “SPQR: Formalized Design Pattern Detection and Software Architecture Analysis”, Technical Report TR05-012, Department of Computer Science, University of North Carolina at Chapel Hill, May 30, 2005.