P-MARt: Pattern-like Micro Architecture Repository

Yann-Gaël Guéhéneuc
Département d'informatique et de recherche opérationnelle
Université de Montréal – CP 6128 succ. Centre Ville
Montréal, Québec, Canada, H3C 3J7
guehene@iro.umontreal.ca

Abstract

We introduce P-MARt*, a repository of pattern-like micro-architetcures. The purpose of* P-MARt *is to serve as baseline to assess the precision and recall of pattern identification tools. Indeed, several approaches have been proposed to identify occurrences of design patterns, yet few have been independently validated for precision and recall for lack of known occurrences. We hope that* P-MARt *can be shared and enriched by researchers interested in design pattern identification.*

1 Context

Maintainers must be aware of design choices in order to modify an object-oriented software system. Design choices include decisions made by developers when designing and implementing the system: the structures of classes and the relationships among them. However, design choices are often scattered in the source code of systems after implementation because, with available object-oriented programming languages, they do not transcribe directly into source code; developers must write several lines of code to implement their choices. Moreover, documentation is often obsolete, if it even exists, and these choices are thus lost.

However, design choices are often implemented with recurring *patterns*, "a form or model proposed for imitation" [9], to facilitate writing and understanding the source code. Idioms and design patterns are two types of patterns; architectural patterns and micro-patterns are others. *Idioms* are low-level patterns specific to some programming languages and to the implementation of particular characteristics of classes or their relationships. They are intra-class patterns describing typical implementation of, for example, relationships, object containment, and collection traversal. *Design patterns* [5] are recurring inter-class patterns that define solutions to common design problems in the organisation of classes. They are "tactics" that generate the structure and behaviour of classes and their relationships [2]. They influence the design of modules and classes but not the overall architecture. They are defined in terms of classes and relationships; thus their implementation uses idioms.

Developers search for some kinds of patterns in order to understand a system [10]; by recognising concrete manifestations of these patterns, they deduce from their experience choices underlying the presence of patterns in the source code. During maintenance and evolution, maintainers would greatly benefit by knowing the design choices made during implementation, see for example [11]. Therefore, several researchers have proposed tools to identify occurrences of patterns in object-oriented systems. Design patterns have been of much interest since Wuyts' precursor work [11].

2 Vocabulary

We use the term *motif* to express the solution of a pattern as "a reliable sample of traits, acts, tendencies, or other observable characteristics" [9] of a pattern. We distinguish between patterns and motifs because patterns encompass information that is not readily available for their identification. For example, the Composite design pattern [5, p.163] also includes information about its intent, motivation, applicability, and consequences, which are not observable characteristics. Only its structure, its participants, and their collaborations are observable in the source code. Thus, strictly speaking, we cannot use the terms design pattern "identification", "detection", or "instantiation" but rather the instantiation and identification of micro-architectures similar to some motifs; thus, we use the term "design motif identification" for the process traditionally called design pattern identification.

We define the term *micro-architectures* (μA) as concrete manifestations of some motifs in the implementation of a system. A micro-architecture is an occurrence of a design motif, composed of classes, methods, fields, and relationships having structure and organisation similar to one or more motifs. A micro-architecture can be similar to more than one motif because only developers may decide intent, motivation, and so on.

3 Problem

To support design pattern identification and program comprehension, we have been developing the PTIDEJ tool suite to model and identify design motifs [6, 7, 8] using a new multilayered framework. This framework makes it possible to recover two kinds of design choices from source code: idioms pertaining to the relationships among classes and design motifs characterising the organisation of the classes. The framework is extensible and scalable and ensures the traceability between motifs and source code by first identifying binary class relationships to obtain an *idiomatic model* of the code and using this model to identify design motifs and generate a *design model* of the system.

However, providing an identification framework is not enough without an independent assessment of its precision and recall to prove the soundness of the tool suite. Precision and recall are measures defined in information retrieval and computed as:

$$\text{precision} = \frac{|\{\text{true } \mu A\} \cap \{\text{identified } \mu A\}|}{|\{\text{identified } \mu A\}|}$$

$$\text{recall} = \frac{|\{\text{true } \mu A\} \cap \{\text{identified } \mu A\}|}{|\{\text{true } \mu A\}|}$$

Therefore, we looked for known micro-architetcures similar to some design motifs and found only a couple of systems with well documented use of design patterns (in particular JHOTDRAW [4]. This lack of known micro-architectures prevents the independent assessment of design pattern identification approaches and their comparison with one another.

4 Our Solution: P-MARt

We have been developing since 2004 P-MARt, a repository of micro-architectures similar to design patterns. Every session, we have asked B.Sc. and M.Sc. students to analyse one system and identify micro-architectures using only the GoF's book [5]. We also collected from colleagues their collections of micro-architectures, we are in particular grateful to Arcelli et al. [1] and Bieman et al. [3].

```
<program type="LANGUAGE">
    <name>NAME</name>
    <designMotif name="NAME">
        <microArchitectures>
            <microArchitecture n="NUMBER">
                <roles>
                    <ROLES1>
                        <ROLE1>
                            <class>
                                NAME
                            </class>
                        </ROLE1>
                        ...
                    </ROLES1>
                    ...
                </roles>
            </microArchitecture>
            ...
        </microArchitectures>
    </designMotif>
    ...
</program>
...
```

Figure 1. Structure of the repository

We record micro-architectures in a XML file, which allows us to traverse the data to compute metrics and various statistics automatically. Figure 1 shows the general structure of the XML file: A program is written in a LANGUAGE and has a (unique) NAME; Each design motif has a (unique) NAME also; A micro-architecture possesses a unique NUMBER and associates each possible role in the design motif, ROLE1, ROLE2, ..., ROLEN, with the classes NAMEs (if any) playing it.

5 Conclusion and Future Work

With P-MARt, we hope to help fellow researchers assess their approaches to design pattern identification. We foresee three further direction of work revolving our P-MARt. First, we would like to discuss the interest in this repository and disseminate it further. In particular, we would like to know if similar endeavour exists elsewhere. Second, we would like to discuss the format of the repository and possibility to include more interesting information. Finally, we would like to discuss possible uses of P-MARt by researchers.

References

[1] F. Arcelli, S. Masiero, C. Raibulet, and F. Tisato. A comparison of reverse engineering tools based on design pattern decomposition. In *Proceedings of the* 16th *Australian Software Engineering Conference*, pages 262–269. IEEE Computer Society Press, March–April 2005.

[2] K. Beck and R. E. Johnson. Patterns generate architectures. In *Proceedings of* 8th *European Conference for Object-Oriented Programming*, pages 139–149. Springer-Verlag, July 1994.

[3] J. Bieman, G. Straw, H. Wang, P. W. Munger, and R. T. Alexander. Design patterns and change proneness: An examination of five evolving systems. In *Proceedings of the* 9th *international Software Metrics Symposium*, pages 40–49. IEEE Computer Society Press, September 2003.

[4] E. Gamma and T. Eggenschwiler. JHotDraw. Web site, 1998. members.pingnet.ch/gamma/JHD-5.1.zip.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1st edition, 1994.

[6] Y.-G. Guéhéneuc and H. Albin-Amiot. Recovering binary class relationships: Putting icing on the UML cake. In *Proceedings of the* 19th *Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 301–314. ACM Press, October 2004.

[7] Y.-G. Guéhéneuc, R. Douence, and N. Jussien. No Java without Caffeine – A tool for dynamic analysis of Java programs. In *Proceedings of the* 17th *Conference on Automated Software Engineering*, pages 117–126. IEEE Computer Society Press, September 2002.

[8] Y.-G. Guéhéneuc and N. Jussien. Using explanations for design-patterns identification. In *Proceedings of the* 1st *IJCAI Workshop on Modeling and Solving Problems with Constraints*, pages 57–64. AAAI Press, August 2001.

[9] Merriam-Webster. Merriam-Webster online dictionary, March 2003.

[10] C. Rich and R. C. Waters. *The Programmer's Apprentice*. ACM Press Frontier Series and Addison-Wesley, 1st edition, January 1990.

[11] R. Wuyts. Declarative reasoning about the structure of object-oriented systems. In *Proceedings of the* 26th *Conference on the Technology of Object-Oriented Languages and Systems*, pages 112–124. IEEE Computer Society Press, August 1998.