

From Software Parameterization to Software Profiling

Philippe Bouaziz^{1,2} and Lionel Seinturier¹

¹ Univ. Paris 6, Lab. LIP6, 4 place Jussieu, F-75252 Paris cedex 05, France
{Philippe.Bouaziz, Lionel.Seinturier}@lip6.fr

² Prodware Group, 45 quai de la Seine, F-75019 Paris, France

Abstract. Over the last years, software engineering research applied to separation of concerns has focused on new software paradigms such as Aspect Oriented Programming. Aspects are abstractions which capture and localize crosscutting concerns. Many works have been conducted with regard to non functional concerns such as performance or semantics of component. This paper wants to demonstrate their interest for functional concerns. It introduces a new software engineering process that we call Software profiling which represents a step further in software parameterization using functional/non-functional aspects to provide highly adaptable and evolving software.

1 Introduction

Over the last years, software engineering research applied to separation of concerns has focused on new software paradigms such as Aspect Oriented Programming [8][7]. It aims at optimizing software development by providing tools to improve modularization, so that it corresponds to the natural view of concerns such as defined by developers [10]. Aspects are abstractions which capture and localize crosscutting concerns e.g. code which cannot be encapsulated within one class but that is tangled over many classes. Synchronization, failure handling, load balancing, real time constraints, memory management, optimization are classical examples of aspects. Many works have been conducted with regard to non functional concerns such as performance or semantics of component whereas crosscutting concerns are widely presents in codes dealing with functional aspect.

Research in AOP is in its early stage with programming tools available, but its contribution to software development process is still not clearly defined, especially in term of penetration degree in the functional part. We think that capabilities of aspect are broader than system and environmental concerns, and that they can be widely used in the software development process, and especially concerning software parameterization that has been the last ten years goal in software industry.

We present here the foundation of a software engineering process that we call Software profiling which represent a step further in software parameterization by

using functional/non-functional aspects to provide highly adaptable and evolving software. It permits to obtain clear, modular and strongly evolving software that can be profiled statically or dynamically depending on the problematic of the user with no alteration of the standard source code which is only concerned by standard evolutions, keeping this way ascendant compatibility in software versions.

First of all, we describe quickly Aspect Oriented Programming. In a second part we present the goals and fundamentals of Software profiling and its contribution to software engineering industry. Finally we consider the benefits of a Case tools for software profiling as a base for further works.

2 AOP

Aspect Oriented Programming aims to achieved separation of concerns by improving code modularization using Aspects. Most of the time, aspects represent non-functional requirements. In AOP, components and aspects are separate codes and the weaving process is done by a static (compile time) or dynamic (run time) compiler that is called an aspect weaver. The aspect weaver, weaves aspects and components at specific points named join points which can be implicit such as language keywords or explicit. Specific code is then added at this points. [6] classify join points among open, class-directional, aspect-directional, and closed depending on the fact that the aspect or the class knows about each other or not:

- Open: Both classes and aspects know about each over,
- Class-directional: the aspect knows about the class,
- Aspect-directional: the class knows about the aspect,
- Closed: neither the aspect nor the class knows about the other.

AspectJ [1] and the Composition Filter Object Model (CFOM) [2] are some of the leader tools in AOP. AspectJ is an aspect-oriented extension to Java that is being developed at the Xerox Palo Alto Research center. It offers a language to define a new kind of module, called an aspect. Aspects are defined separately from the standard code. AspectJ provides a static aspect weaver in Java, and other development tools. It is widely used in AOP research community, and version 1.0 is due next fall. A version of AspectJ for C [4] is under development. CFOM is a project developed by the Trese group. The composition filter approach [2] extends objects with filters that deal with inter-objects messages. Input and output filters are used to localize aspect code. Other AOP approach exists, among them Subject Oriented Programming [5], Adaptive Programming [9], and other language extension like AOP/St for Smalltalk [3].

3 Software profiling

3.1 Software parameterization

Customer requirements and needs tend to evolve as technology, business and company processes advance. Software developed in the last decade with initial

customer needs in mind tend to be unsuitable to follow these changes as no proper design technique is available for this. Development teams that try nevertheless to achieve this goal, end up with source code more and more difficult to maintain, upgrade, and reuse. Along a long embryo period, software is install with many difficulties and is in permanent beta stages to meet new requirements or requirements that emerge from the analysis.

The increasing speed of technological evolution over the last twenty years and the fact that computer software became more and more common, created a clear need for rationalizing software processes to deliver cheaper products, with short integration times, quality-oriented maintainability and evolution capabilities to guarantee a longer software life.

To achieve this goal, software industry evolved, just as the textile industry did with ready-made clothes, and committed itself in developing standard business software, based on common needs of a significant community of customers. Base on this, customizations are being made possible with parameters that reflect the various management practices of customer organizations. These software are developed by editors that provide maintenance, that are permanently auditing their market, and that are arbitrating the functional and technological changes of software.

These days, the existing level of parameterization existing in major products, such as ERPs¹ (e.g. SAP), provides many important features to meet needs of various industry fields. Nevertheless, the level of adaptability of these products to non-mutualizable features is weak due to the complexity introduced by the huge number of available parameters. Most of the time, this adaptation is done in an *ad-hoc* manner. Faced with this problem, partial solutions have been proposed based on object technologies, n-tiers architectures and in the industry, relational databases. They allow delocalizing treatments such as reporting, that can (re) become specific, and to slightly amend software based on entry points, triggers or stored procedures, to better integrate it with the information system.

Nevertheless, for simple needs such as ascending compatibility, the behavior of the application or of the data model can never be altered. This prevents a straight and optimized answer to above mentioned issues. Furthermore, programs end up being tangled due to the many possibilities introduced by parameterization and cannot be reuse.

3.2 The benefits of functional aspects

Software design aims at: (1) factorizing functionalities, and (2) allowing that these functionalities be parameterized in order to meet customer specific needs. From an industrial point of view, the modification of these parameters can induce too deep modifications of the software internal structure preventing the addition of new or customer specific features. Many technical solutions exist, but they bring either an overload of scattered code, or a parameterization of existing parameters, and in all cases are too difficult to maintain (each case needs to be

¹ Enterprise Resource Planning

individually treated). Aspect oriented development should allow to standardize and centralize these specificities in order to avoid overloading and tangling.

3.3 Software profiling

The notion of aspect allows to profile software depending on the needs and notably:

- to encapsulate in a clear and centralized way, parameters leading to massive cross-cutting all along the code,
- to perform without altering the main code, modifications or extensions in functional part, for example altering the behavior of business objects to obtain a different action on the information flows, and even its replacement by another object,
- to encapsulate in a clear and centralized way system features such as synchronization, load balancing, real time, ...,
- to ease code reuse,
- to bring capabilities of dynamic reactivity to the software depending on the evolution of profiles such as dynamic design of GUI and contents (for instance to profile Internet applications furnished by ASPs - application service providers).

Usage of aspects in parameterized software therefore allows to profile software depending on data and strategy defined in static or dynamic ways. The pending difficulty is to be able to define what in a profile is relevant to aspects.

4 Further works

As mentioned before, parameterized software design, is a global process. To be a part of it, the notion of aspect should impact all levels, analysis, design and implementation. To do so, CASE tools should integrate this notion, furnishing an environment taking all this considerations in account. We can imagine to find there :

- a set of rules to help decide whether to use aspects or not,
- an extension to existing methods to take aspects into account during the analysis phase,
- a graphic design tool to describe aspects and their relationships with other aspects and components,
- an environment for managing and testing projects integrating aspects characteristics.

Some works have been realised in this sense such as UML/UXF [11] for the design phase or AJDE [1] as a development environment. Nevertheless, as far as we know, no break throws have been made in terms of designing aspects and their relationships or about methods, or decisional purpose. Our goal is to provide a CASE tools for software profiling design taking all this needs in account. One of the first steps will be to define, method specifics, components specifications and rules.

5 Conclusion

We show in this paper the interest of introducing aspects in software parameterization, and by the way demonstrate the capabilities of aspects in the functional field of software development. Aspects in the development process will bring clear, modular, strongly evolving and adaptable parameterized software. In this article we enlarged the scope of aspect to functional domains. We are now working, on defining rules to decide where and when aspects should apply. The next step will be for us to define fundamentals for Software profiling, especially in terms of method specifics, components specifications and rules.

References

1. AspectJ home page. <http://www.aspectj.org>.
2. Aksit, M., Wakita, K., Bosch, J., and Bergmans, L. Abstracting object interactions using composition filters. vol. 791 of LNCS, pp. 152–184.
3. Bollert, K. On weaving aspects. In Workshop Aspect-Oriented Programming at ECOOP'99 (June 1999). <http://trese.cs.utwente.nl/aop-ecoop99/>.
4. Coady, Y., Kiczales, G., Feeley, M., Hutchinson, N., and Ong, J. Structuring system aspects. In Proceedings of the workshop on Aspect-Oriented Programming at ICSE'01 (2001).
5. Harrison, W., and Ossher, H. Subject-oriented programming (A critique of pure objects). In OOPSLA 1993 Conference Proceedings, A. Paepcke, Ed., vol. 28 of ACM SIGPLAN Notices. ACM Press, Oct. 1993, pp. 411–428.
6. Kersten, M., and Murphy, G. Atlas: A case study in building a web-based learning environment using AOP. In Workshop Aspect-Oriented Programming at ECOOP'99 (June 1999). <http://trese.cs.utwente.nl/aop-ecoop99/>.
7. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. An overview of AspectJ. In Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01) (2001), Lecture Notes in Computer Science.
8. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J., and Irwin, J. Aspect-oriented programming. In Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97) (June 1997), vol. 1241 of Lecture Notes in Computer Science, Springer, pp. 220–242.
9. Lieberherr, K. Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns. PWS Publishing Company, Boston, 1996. <http://www.ccs.neu.edu/research/demeter/biblio/dem-book.html>.
10. Parnas, D. On the criteria to be used in decomposing systems into modules. Communications of the ACM 15, 12 (1972), 1053–1058.
11. Suzuki, J., and Yamamoto, Y. Extending UML with aspects: Aspect support the design phase. In Workshop Aspect-Oriented Programming at ECOOP'99 (June 1999). <http://trese.cs.utwente.nl/aop-ecoop99/>.