Reducing Aspect-Base Coupling through Model Refinement

Aswin van den Berg[1], Thomas Cottenier[1,2], Tzilla Elrad[2]

[1] Motorola Software Group, Motorola
1303 E. Algonquin Rd, 60196 Schaumburg, IL, USA
[2] Concurrent Programming Research Group, Illinois Institute of Technology,
3100 S. Federal Street, 60696 Chicago, IL, USA

{aswin.vandenberg, thomas.cottenier}@motorola.com
{cotttho, elrad}@iit.edu

Abstract. Aspect-Oriented Programming languages allow pointcut descriptors to quantify over the implementation points of a system. Such pointcuts are problematic with respect to independent development because they introduce strong mutual coupling between base modules and aspects. This position paper addresses the aspect-base coupling problem by defining pointcut descriptors in terms of abstract views of the base module. These abstract views should be towards the architectural viewpoints of the system under development.

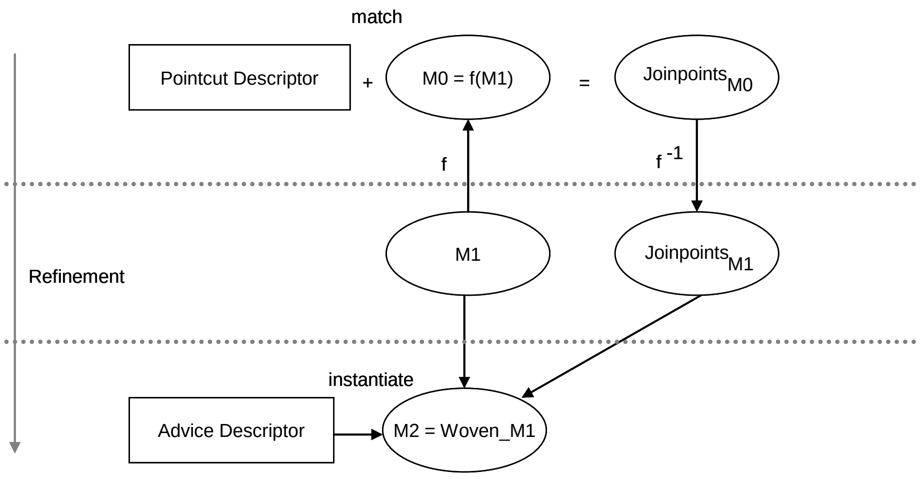
Since the inception of Aspect-Oriented Software Development (AOSD) in 1997, it has been known that Aspect-Oriented Programming (AOP) languages introduce strong coupling between base modules and aspects. AOP languages allow pointcut descriptors to refer directly to the implementations of modules to capture joinpoints, points where aspects inject behavior through advices. This practice is problematic with respect to modularity and independent development. Aspects need fine-grained control over the modules they advice and, vice versa, the advised modules need to be aware of those aspects. Therefore, both aspect and base module become hard to evolve independently.

There are three main research directions in addressing this aspect-base coupling problem. The first direction of research advocates restricting the expressiveness of aspects by forfeiting the obliviousness of modules [1][2][3]. A second approach favors investigating alternative ways to modular reasoning in the presence of aspects. In [4], the authors argue that a global analysis of the system configuration is required before the interfaces of the system modules can be determined. A third direction of research focuses on methods that allow pointcut descriptors to be defined at a higher level of abstraction, in terms of the program semantics [5]. Our work with Motorola ***WEAVR*** in [6] introduces pointcut descriptors that can infer implementation joinpoints from higher level descriptions. This paper proposes an approach to AO modeling that is integrated with a model refinement approach with the purpose to reduce the aspect-base coupling.

Let M1 be the current refinement of a software system. We show five requirements for moving towards our goal:

1. There needs to be an abstract view M0 of the refinement M1 of the system under development that is sufficiently describing the behavior of its specification towards a particular architectural viewpoint.
2. There needs to be a precise definition of what it means that a refinement is realizing an architectural view. This realization can be described by a well-defined mapping f from the refinement M1 to the view M0.

3. The development process/tool needs to enforce that the refinement of the view is actually realizing the view. That is, the process/tool needs to enforce the realization invariant M0 = f(M1).
4. Define pointcut descriptors in terms of the view M0. The matching produces a set of joinpoints in M0 (denoted by Joinpoints_M0)
5. Translate these joinpoints in terms of the refinement M1 and instantiate the advice at corresponding points in M1. The resulting woven model M2 is more refined than M1 because it has a new concern incorporated in it.



Since the pointcut descriptor is written in terms of M0 it is completely independent from the refinements that are introduced in M1. Since M0 is an abstract view towards an architectural viewpoint it is not a view that is dependent on the pointcut descriptor. And because M0 is not dependent on the pointcut descriptor it follows that also M1 is not dependent on it. Therefore there is no aspect-base coupling between the aspect and the refinements introduced from M0 to M1.

References

1. Aldrich, J. Open Modules: Modular Reasoning about Advice. In Proceedings of the 19th European Conference on Object-Oriented Programming, Glasgow, Scotland, LNCS 3586, pp. 144-168, Springer, 2005
2. Griswold, W.G., Shonle, M., Sullivan, K., Song, Tewari, N., Cai, Y., Rajan, H.: Modular Software Design with Crosscutting Interfaces. IEEE Software, 23:1, pp. 51–60, IEEE Computer Society, 2006
3. Gybels, K., Brichau, J.: Arranging Language Features for More Robust Pattern-Based Crosscuts. In proceedings of the International Conference on Aspect-Oriented Software Development, , Boston, USA, pp 60–69, ACM Press, 2003.
4. Kiczales, G., Mezini, M.: Aspect-Oriented Programming and Modular Reasoning. In proceedings of the International Conference on Software Engineering, St. Louis, USA, pp 49–58, ACM Press, 2005
5. Ostermann, K., Mezini, M., Bockisch, C.: Expressive Pointcuts for Increased Modularity. In Proceedings of the 19th European Conference on Object-Oriented Programming, Glasgow, Scotland, LNCS 3586, pp. 214-240, Springer, 2005
6. Cottenier, T., van den Berg, A., Elrad, T., Joinpoint Inference from Behavioral Specification to Implementation, In Proceedings of the 21st European Conference on Object-Oriented Programming (ECOOP), Berlin, Germany, 2007.