Membranes for AOP: From Vision To Practice

Ismael Figueroa *
PLEIAD Laboratory - Computer Science Department (DCC) -University of Chile
ifiguero@dcc.uchile.cl

Categories and Subject Descriptors: D.3.3 [Programming Languages]: Language Constructs and Features

General Terms: Languages, Design

Keywords Aspect-oriented programming, membranes, execution levels.

1. Introduction

Due to the scattering of crosscutting concerns through programs, weaving typically requires aspects to have a global view of computation. This hampers modularity [1, 6–9]. The problem is addressed either by limiting the *scope* of aspects [3, 11, 12]; or by *protecting* software units from advising [1, 6–9]. Moreover, aspects with a global view of computation make it difficult to compose aspects while keeping coherent semantics, because aspect computation can be observed by all aspects, allowing for infinite regressions when aspects capture their own computation. Recently, Tanter developed *execution levels* [12] as a means to structure computation and avoid infinite regressions by default. As a continuation to his work, Tanter *et al.* proposed programmable membranes [13] to structure computation and control aspect scoping. However, their proposal is focused more on the ideas than in practical aspects of membranes. We propose a roadmap for developing membranes for AOP more concretely.

2. Programmable Membranes for AOP

We briefly summarize programmable membranes for AOP, as proposed by Tanter *et al.* [13]. Membranes [2] are *permeable* structures in which computations can take place, providing a notion of locality and hierarchy. Programmable membranes provide membranes with some computing power to control its own permeability. Membranes for AOP define three dimensions: first, membranes act as *dynamic* containers of computation. Also, aspects are *registered* in membranes. Second, a membrane can bind to another membrane to *advise* it. This controls aspect scoping, because join points emitted by computation inside an advised membrane can only be seen by aspects in the corresponding advising membranes. Finally, because membranes are programmable, they control join point propagation between the environment and the membrane aspects.

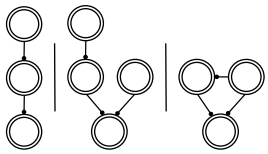

Figure 1. Topological scoping. (a) tower; (b) tree; (c) DAG (adapted from [13]).

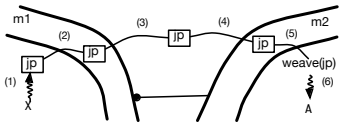

Figure 2. From join point emission to aspect weaving (adapted from [13])

Flexible Aspect Scoping. Bindings between membranes define a directed graph with membranes as vertices, and the advising relation as lollypop arrows; allowing flexible topologies of computation. In fact, membranes subsume execution levels, as shown in Figure 1(a). Other topologies are trees (Figure 1(b)) or DAGs (Figure 1(c)).

Join Point Propagation and Weaving Membranes control the flow of join points through its programmable layers. Figure 2 shows membranes m1 and m2, where m2 advises m1. Computation X happens in m1 (1), which emits a join point jp. m1 decides whether to relay jp to its outer environment or not (2). If m1 propagates jp to m2, then jp is tagged with its destination (3). From its side, m2 is waiting for join points addressed to it (4), it finds jp and decides whether to

* Funded by a CONICYT-Chile Doctoral Scholarship

AOSD'12, March 25-30, 2012, Potsdam, Germany.
ACM 978-1-4503-1222-6/12/03.

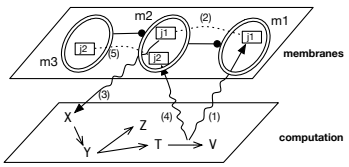

Figure 3. Membranes in action in a level-like setting (from [13])

accept it or not (5). If m2 accepts jp, then jp is woven with aspect A, registered inside m2.

Membranes and Computation. The membrane that contains current computation is a property of the execution flow of the program, not of (static) code artifacts. This way, a same piece of code can emit a join point inside a membrane and, at another time in execution it can emit a join point inside another membrane. Figure 3 shows membranes in an execution-levels-like setting. The membranes plane contains membranes m1, m2 and m3 such that m3 advises m2, and m2 advises m1, as denoted by the lollypop arrows. Computation initiates inside m1, so when computation flows from T to V, join point j1 is emitted inside m1 (1). Consequently, j1 is exposed to aspects registered in m2 (2). Then, an aspect woven in m2 executes computation X (3), which emits a join point j2 when the execution flow goes from T to V (4). Because computation is inside m2, j2 propagates from m2 to m3 (5), though both j2 and j1 were generated by the same code.

3. From Vision to Practice

The general vision on membranes for AOP is too wide and general, and there is no real assessment of its capabilities and drawbacks. Our contribution is a roadmap to go from the vision to the practice. Execution levels [12] can be considered as a reduced instance of membranes, but they have a restricted linear topology, which cannot express all interesting scenarios. In contrast, the membrane model is too unrestricted. For instance, it is not clear in which cases infinite regression is avoided. Also, there is no formalization of membrane semantics. We want to establish a proper tradeoff between the execution levels settings and the full membrane model, and further explore the membrane model. The milestones of our roadmap are:

Consolidate Execution Levels. We focus on formal properties of execution levels because they are a simple membrane instance on which to base our work. We are interested in avoiding infinite regression in as most cases as possible, and extrapolate those results to the general case of membranes.

Aspect Loops. Intuitively, aspect regression happens because an aspect *sees its own computation*. We will formalize this intuition and find out in what cases we can avoid loops with membranes.

Membrane Calculus. There is a lack of a formal calculus for membranes on which to reason about. We will develop a membrane calculus as a basis for our theoretical work.

Crosscutting Membranes. Computation can happen inside one or more membranes. This introduces aspect loops. We will explore how membranes can be composed and how to deal with crosscutting membranes while avoiding loops.

Membrane Topologies. There is a need to determine what topologies are appropriate to structure aspect programs, establishing their properties with relation to aspect behavior.

Modular Reasoning. We want to develop a notion of *module* to enable modular reasoning. We plan to develop an encoding similar to Open Modules [1].

Membrane-based Languages. MAScheme [13] and PHANtom [4] are the only membrane-based languages to date. We plan to implement language prototypes of different membrane models.

4. Current and Future Work

Now we are focused on the three first points of our roadmap. Regarding the consolidation of execution levels, the author collaborated with Tanter and Tabareau to formalize aspect loops. This work was recently submitted for peer review. Additionally, based on recent work by Tabareau [10], we implemented a monadic aspect weaver [5] in which aspect semantics are parameterized by monads. In this framework, we express execution levels as a monad, and we plan to add support for membrane semantics. Future work will focus on the remaining points of our roadmap.

References

[1] J. Aldrich. Open modules: Modular reasoning about advice. In A. P. Black, editor, *ECOOP 2005*.

[2] G. Boudol. A generic membrane model (note). In C. Priami and P. Quaglia, editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 208–222. Springer Berlin / Heidelberg, 2005.

[3] C. Dutchyn, D. B. Tucker, and S. Krishnamurthi. Semantics and scoping of aspects in higher-order languages. *Science of Computer Programming*, 63(3):207–239, Dec. 2006.

[4] J. Fabry and D. Galdames. Phantom: a modern aspect language for pharo smalltalk. In IWST'11.

[5] I. Figueroa, É. Tanter, and N. Tabareau. A practical monadic aspect weaver. In *FOAL 2012*.

[6] W. G. Griswold, K. Sullivan, Y. Song, M. Shonle, N. Tewari, Y. Cai, and H. Rajan. Modular software design with crosscutting interfaces. *IEEE Software*, 23(1):51–60, 2006.

[7] M. Inostroza, É. Tanter, and E. Bodden. Join point interfaces for modular reasoning in aspect-oriented programs. In *ESEC/FSE 2011, New Ideas track*.

[8] B. C. d. S. Oliveira, T. Schrijvers, and W. R. Cook. EffectiveAdvice: discplined advice with explicit effects. In *AOSD 2010*.

[9] F. Steimann, T. Pawlitzki, S. Apel, and C. Kästner. Types and modularity for implicit invocation with implicit announcement. *ACM Transactions on Software Engineering and Methodology*, 20(1):Article 1, June 2010.

[10] N. Tabareau. A monadic interpretation of execution levels and exceptions for AOP. Accepted for publication at *AOSD'12*.

[11] É. Tanter. Expressive scoping of dynamically-deployed aspects. In *AOSD 2008*.

[12] É. Tanter. Execution levels for aspect-oriented programming. In *AOSD 2010*.

[13] É. Tanter, N. Tabareau, and R. Douence. Exploring membranes for controlling aspects. Technical Report TR/DCC-2011-8, University of Chile, June 2011.