A Direction for Research on Virtual Machine Support for Concern Composition

Harold Ossher
IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10590, USA
+1-914-784-7975

ossher@us.ibm.com

ABSTRACT
This position paper suggests research directions in the area of virtual machines supporting aspect-oriented capabilities in the context of object-oriented languages**.**

Categories and Subject Descriptors
D.3.4 [**Programming Languages**]: Processors - *Run-time environments.*

General Terms
Languages.

Keywords
Aspect-oriented software development, separation of concerns, dynamic software composition and decomposition.

1. INTRODUCTION
This brief position paper suggests research directions I consider important in the area of virtual machines (and perhaps intermediate languages) supporting aspect-oriented capabilities in the context of object-oriented languages. Rather than a detailed model or solution, I'll suggest a broad class of runtime models that I believe would be fruitful to explore and to use as a basis for innovative virtual machine concepts.

Section 2 discusses some key requirements, and section 3 discusses the proposed class of runtime models.

2. REQUIREMENTS
The fundamental problem to be addressed is the provision of suitable primitives for adding or removing concerns. One example is primitives for binding and unbinding or enabling and disabling advice. This is valuable, but not the whole story. In general, primitives supporting concern composition and decomposition are the goal.

Workshop VMIL '07, March 12-13, 2007 Vancouver, BC, Canada.

I will consider here just two key requirements I believe such primitives should satisfy: *naturalness* and *safety*.

2.1 Naturalness
Object-orient practitioners are used to thinking about a *sea of objects* at runtime, sending messages to one another. This is very natural to them, and it models the real world effectively. It is much less natural to aspect-oriented practitioners, however, because it forces them to think about translating all aspect-oriented effects into object-oriented effects. Concerns are not first class, and operations on them, like composition and decomposition, are meta-operations, introducing an additional level of complexity. This shows through particularly unpleasantly during debugging. Developers are confronted with the woven runtime objects, often involving mangled names are glue code, rather than with runtime elements that match their source code, such as separate, clean objects and aspects.

I argue that one of the key elements of research into virtual machine support for concerns is exploration of good, natural runtime models. In section 3 I discuss a general class of runtime models I believe are especially promising.

2.2 Safety
The naturalness issue, as well as other considerations like those that apply to virtual machines and JITs in general, imply that VM support for composition can be preferable to pre-execution weaving even in the absence of dynamic AOSD, in which composition and decomposition can be initiated on demand at run time. The safety considerations described here apply primarily to the dynamic case, though even VM-based implementations of essentially static composition must beware of the pitfalls.

Addition or removal of a concern requires, in general, coordinated changes to many objects or classes. A key element of safety is therefore *atomicity*: when a concern is added or removed, all its elements must be added or removed as an (effectively) atomic operation. If it is interleaved with execution, various semantic problems can arise. For example, two pieces of advice in an aspect might be written to cooperate, where one computes a value and stores it in an aspect-local variable, and the other uses it. The pointcuts might be such that the producer advice always executes before the consumer. If the aspect is added in such a way that the consumer advice is added, then some execution is allowed to happen, then the producer is added, it is possible that the consumer advice will be triggered before the producer.

A related issue is ensuring that composition or decomposition do not disrupt the running program. For example, suppose that at some instant a method that has both before and after advice asso-

ciated with it is executing. The before advice has thus already executed, and might expect the after advice to do some cleanup, perhaps as critical as committing a transaction. If the concern containing both pieces of advice is removed at that particular instant, what happens to execution of the after advice? If the advice is simply removed and not executed, semantic problems result, even if the removal is atomic.

These sorts of problems can be dealt with in a variety of ways. For example:

- Assuming that the primitives will be generated only by a compiler in a manner that is guaranteed to be safe.
- Some special support for currently-executing code, along the lines that JITs use to allow optimization of methods without disrupting current executions.
- Suitable interlocks within the VM.

Another element of safety has to do with proper handling of additional state introduced by a concern. This is essentially the same as the data migration problem in databases. When new state is introduced, it must be initialized, perhaps in a fixed way, perhaps based on the existing state. If a concern is removed and later reinstated, is it expected that the state values previously computed be retained?

This position paper does not provide solutions to these problems. My point here is that these issues are crucial, and need to be at least considered in all work on VM support for aspect-oriented technology.

3. SEA OF FRAGMENTS

Crosscutting concerns, by their very nature, affect many scattered elements in the dominant decomposition of the system. In the case of object-oriented systems, many objects and/or classes are affected when a new concern is added (or removed). The behavior of some of their methods is enhanced or overridden. Some additional state or methods might be added. This is generally implemented, and often defined, by *weaving*: transforming the underlying language constructs. Even if weaving is provided as a VM operation, this diminishes the first-class nature of concerns: though they may be first-class groupings, their elements get absorbed into the language's runtime representation and do not remain as first-class elements participating in execution.

A concern consists of a number of fragments, such as pieces of advice or members to be introduced. There is also a specification, within the concern or separate, of how the fragments are to be associated with objects, such as the pointcuts at which advice is to be applied, and the objects or classes into which new members are to be introduced. It would seem natural that, once the concern is added to a running system, its fragments remain as elements of the runtime representation. This is equally true of approaches like Hyper/J [7], which explicitly has a fragment-composition model, and approaches like AspectJ [5], which does not: each AspectJ aspect is a fragment consisting, in turn, of advice and intertype declaration fragments, and it is more natural to have the aspect and its sub-fragments as part of the runtime representation than to have all or part of them "woven away."

This suggests a runtime model consisting of fragmented (or faceted) objects. In this sort of model, an object has a unique identity, but it is not usually a single chunk of memory with a single table of associated methods. Instead, it is a constellation of a number of fragments, each of which is a single chunk of memory with a single table of associated methods (modulo inheritance from or delegation to other objects), but not necessarily with its own identity. These fragments combine to determine the behavior of the object, delegating to one another according to the manner in which they are combined. Bill Harrison and I analyzed the various ways in which such fragments can interact in an earlier paper [1].

There are many ways to realize this sort of model. For example, one can think of it in terms of groups of co-operating objects rather than as objects as constellations of fragments, as we did in that analysis [1]. It is along the lines of dynamic role models [6], Object Teams [2], and subjective objects [8]. My objective here is not to offer a model in detail, but to claim that exploration of such models is an especially fruitful direction for research in virtual machine support for aspect-oriented languages.

How natural are these models in fact? It is hard to beat the simplicity and real-world evocativeness of object models. I believe fragmented-object models come close, however, because they mirror the real-world situation of objects acquiring or losing characteristics (roles) during their lifetimes. When a single person marries, s/he acquires some new behaviors and characteristics of married people, while still remaining the same person. If s/he divorces, s/he loses these. Everyone understands this in the real world. The goal is to come up with ways of surfacing the concepts of fragmented-object models to developers that mirror the natural situation as much as possible. This might not seem like an important issue for a virtual machine, and perhaps it is not in the guts, but it is important that the runtime model that developers need to understand and debug be simple and natural.

3.1 Operations

Fragments are thus associated with, or part of, objects. They are also associated with concerns. Indeed, a concern is a set of fragments, possibly along with instructions about how to compose those fragments into the system (or the instructions might be separate, a model I prefer). Another way to think of this is that the runtime model is a multi-dimensional sea or space of fragments, and concerns are hyperslices [10, 7]. The actual composition involves establishing the appropriate connections between fragments and objects. Removing a concern involves breaking such connections.

This leads to initial thoughts about VM operations at two levels: the individual object and fragment level, and the concern level. I postulate an approach in which the fragment-level operations in isolation are not safe, but they may only be used as part of larger concern-level operations, which are safe.

3.1.1 Fragment-level operations

Operations are needed to:

- Create a new object. This is really just creation of a unique identity. For the object to acquire state or behavior requires the association of fragments. Of course, most object creation is likely to be accompanied by immediate association of at least one fragment.
- Associate a fragment with an object, establishing one of a number of possible delegation relationships between them [1]. If the fragment contains state, there must be a

means to initialize it as part of this operation (perhaps just to *null*).

- Remove a fragment from an object, breaking the delegation relationships.

A fragment could be a single method (piece of advice) that is associated with an existing method of the object in such a way that the new fragment executes before (or after or around) the original method. Alternatively, the fragment could contain some state to be added to the object, and perhaps some methods to manipulate it. Addition or removal of fragments thus covers the important case of binding and unbinding advice, in addition to introduction of new state and behavior.

3.1.2 Concern-level Operations

At a minimum, operations are needed to:

- Add a concern into the running system
- Remove a concern from the running system

These operations are realized in terms of a number of fragment-level operations, and the VM must enable them to be performed atomically. It must also, as noted above, have a means of ensuring that execution is not corrupted: there must be an approach to handling the situation where execution is currently in progress within an existing fragment with which some new fragment becomes associated or dissociated.

To provide full-scale, first-class status for concerns, and to enable composition, additional operations would be valuable, such as:

- Compose concerns "on the side" so that a collection of concerns can be added or removed as a unit.
- Form a concern from a collection of fragments.
- Locate join points based on queries.

Exploring the set of appropriate operations is one of the research topics in this area.

3.2 Language model

It seems convenient, and perhaps essential, to describe the associations between fragments and objects in terms of delegation. There have been long-standing debates between delegation and inheritance, and although they have been shown to have a degree of formal equivalence [9], they do have different convenience characteristics. Classes and inheritance are often preferred because of the static typing they provide, and yet Ungar and Smith, in their landmark paper on Self, made powerful arguments for the power and simplicity of delegation [11]. Work on Self also showed that the performance penalty often attributed to flexible delegation approaches can be greatly reduced [4, 12, 3]. The introduction of fragments into the mix seems to me a force in the direction of delegation, and I think a sea-of-fragments model would be best defined and implemented in terms of delegation. This does not mean abandoning all the value of higher-level organizational and typing structures, such as classes and interfaces; I believe there is potential for realizing much or all of their benefit in layers above the underlying delegation layer.

4. CONCLUSION

This brief position paper raised the important requirements of naturalness and safety for virtual machine support for aspect-oriented capabilities, construed broadly. It also suggested a sea-of-fragments runtime model based on delegation as a promising approach to explore.

5. ACKNOWLEDGMENTS

Many thanks to Bill Harrison, Peri Tarr, David Ungar and Adrian Colyer for many enjoyable and insightful discussions over the years, from which emerged many of the insights expressed in this position paper. Thanks to the anonymous reviewers for some most helpful comments.

6. REFERENCES

[1] William Harrison and Harold Ossher, "Member-Group Relationships Among Objects." AOSD '02 Workshop on Foundations Of Aspect-Oriented Languages (FOAL).

[2] Stephan Herrmann, "Object Teams: Improving Modularity for Crosscutting Collaborations." Proc. of Net.ObjectDays, Erfurt, 2002.

[3] Urs Hölzle and Ole Agesen, "Dynamic vs. Static Optimization Techniques for Object-Oriented Languages." Theory and Practice of Object Systems 1(3), 1995.

[4] Urs Hölzle, Craig Chambers, and David Ungar, "Optimizing Dynamically-Typed Object-Oriented Programming Languages with Polymorphic Inline Caches." In *ECOOP `91 Conference Proceedings*, Geneva, Switzerland, July, 1991. Published as Springer Verlag LNCS 512, 1991.

[5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, Jeffrey Palm and William G. Griswold. "An Overview of AspectJ." In *Proc. 15th European Conference on Object-Oriented Programming*, pp. 327-353, 2001.

[6] Bent Bruun Kristensen and Kasper Osterbye, "Roles: conceptual abstraction theory and practical language issues." Theory and Practice of Object Systems 2(3), 1996.

[7] Harold Ossher and Peri Tarr. "Using Multi-Dimensional Separation of Concerns to (Re)Shape Evolving Software." *CACM* 44(10): 43–50, October 2001.

[8] Randall B. Smith, David Ungar: A Simple and Unifying Approach to Subjective Objects. Theory and Practice of Object Systems 2(3), 1996.

[9] Lynn Andrea Stein, Henry Lieberman, David Ungar, "A Shared View of Sharing: The Treaty of Orlando." In Object-Oriented Concepts, Databases and Applications, ACM Press/Addison-Wesley, 1989.

[10] Peri Tarr, Harold Ossher, William Harrison, and S. M. Sutton, Jr., "N degrees of separation: Multi-dimensional separation of concerns." In *Proceedings of the 21st International Conference on Software Engineering (ICSE '99),* 107–119, IEEE, May 1999.

[11] David Ungar and Randall B. Smith, "Self: The Power of Simplicity." In *OOPSLA '87 Conference Proceedings*, pp. 227-241, Orlando, FL, October, 1987.

[12] David Ungar, Randall B. Smith, Craig Chambers, and Urs Hölzle, "Object, Message, and Performance: How They Coexist in Self." Computer, 25(10), October, 1992, pp. 53-64.