An Objective Look at Subjectivity

Brian Foote[1]
University of Illinois at Urbana-Champaign
foote@cs.uiuc.edu

15 August 1995

A POSITION PAPER
for the
OOPSLA '95 Workshop on
Subjectivity in Object-Oriented Programming

Subjectivity

Subject-Oriented Programming [Harrison & Osher 1993] is a proposed stylistic variant on the traditional object-oriented trinity of polymorphism, inheritance, and encapsulation (PIE). It emphasizes *subjective views* as an alternative to a proliferation, in monolithic hierarchies, of methods and states. By allowing distinct concerns to be accessed at different times, from different places, via distinct portals, subjectivity would seem to allow complex, evolving, dispersed phenomena to be modeled more naturally that the "classical model" might.

Alas, subjectivity is but a model, because today's popular object-oriented programming languages are simply not powerful enough to allow the mechanisms needed to support subjectivity to be constructed. Subjectivity would seem to require dispatching, consistency maintenance, dependency, and state management facilities that are beyond the reach of programmers in conventional object-oriented languages. It also requires a more flexible sense of object identity and mutability than traditional languages allow.

These are precisely the kinds of issues that are being address by researchers who are studying reflective object-oriented languages [Maes 1987]. A goal of this research is to allow mechanisms to support programming models like subjectivity to be built into existing languages. Since such languages are themselves built out of objects, these objects can serve as the basis for linguistic extensions, just as other objects can provide a basis for application-specific needs.

History

The idea that object-oriented programmers seek to model the world using complex constellations of cooperating objects is not a new one. For instance, [Stefik & Bobrow 1986] describes two mechanisms known to the LOOPS community for constructing such aggregates.

Composite Objects were groups of interconnected objects that were instantiated together; a recursive extension to the notion of object. A composite is defined by a template that describes the subobjects and their connections.

[1]Address: 1304 W. Springfield, Urbana, IL 61801, USA +(217) 333-3411, FAX +(217) 244-5876

Perspectives were independent, linked objects which each represent a view. Each perspective has its own namespace. Once can access any of these view names form each of the others. For example, given an object, one asked for its Traveler perspective using the name "Traveler". Perspectives employed instantiation on demand (in contrast with composites, in which all parts were created at initialization time). Additional views could be added as needed to any object.

The Common Lisp Object System (CLOS) provides facilities that allow users to define how inherited methods are composed. The Metaobject Protocol [Kiczales et. al. 1991] provides programmers with access to state, generic function dispatch, class definition and initialization, and additional method combination mechanisms.

Reflection and Smalltalk

Smalltalk-80 allows programmers to directly manipulate objects such as classes, contexts, and methods. (SELF provides similar facilities, though it has no classes.) Additional reflective facilities can be added if direct access to dispatching mechanism is provided [Foote & Johnson 1989]. This paper discussed how these mechanisms could be used to construct automatic mechanisms to dynamically forward and delegate messages to components, and provide multiple views of an object by allowing outer objects to reroute messages to inner objects that embody state.

The authors also noted that inheritance, which is (at least in Smalltalk) a static, per class mechanism, is often overused. Often, programmers are modeling dynamic, per-instance relationships instead. Since no mechanisms exist to support this sort of programming, programmers fall back on inheritance.

Open Languages and Subjectivity

Linguistic support for the following sorts of mechanisms should make building subjective objects easier:

Dispatching Mechanisms that permit messages to be dynamically routed, and that can themselves be easily changed, might allow an object to forward a part of its protocol to another object that manages its "true" state, or allow a message to be broadcast to several objects at once.

State Mechanisms that are distinct from other object functions could allow version management, and dependency mechanisms to be constructed, and could permit these mechanisms to be dynamically altered at runtime. Consistency maintenance, persistence and constraint support can be piggybacked on these mechanisms too.

Protocol objects could allow the designation of method and message groups that could be dynamically given special treatment en masse.

Methods and other objects that designate behavior might be designed so that they too provide dynamic loci for the augmentation of behavior.

Namespace, or scope objects can be manipulated to control the visibility of different views of an object.

References or *Handles* can be used to so that owners of different handles have different views and capabilities for the objects these objects refer to.

Identity issues can be dealt with by distinguishing functional and object-level notions of identity. These can be handled using different names (such as = and = =) or by making different versions of the identity operators visible under varying circumstances.

Given a reflective perspective, subjective coalitions might be seen as dynamic collections of cooperating objects. One intriguing consequence of a subjective perspective is that the metaobjects distinguished in some metaarchtectures might be treatable as different subjective views of the same object. One view might expose computationally relevant information, while others could present various application-level views.

How to Build Views and Perspectives out of Reflective Parts

There are a number of strategies that might be employed to build views and perspectives, given that the a programming language is itself built out of objects.

One might construct multiple views using a dynamic coalition of different objects. Each object would correspond to a view. To keep the views consistent, mechanisms for managing the mutual constraints among objects must be put into place. In some cases, these might be coded explictly. However, it is more desirable that this be done more automatically. This requires that the metalevel dispatching strategies be late bound, and extensibly defined. Depending on the metalevel chosen, a variety of the parts enumerated above might play a role.

Perspectives (of the variety described by Ungar and Smith) present different metalevel challenges. Among the objects that might come into play are Namespaces, Methods, Handles, and truely virtual Virtual Machines.

Issues

I believe these three questions would be appropriate foci for the workshop:

- *What is subjectivity?*
- *How can languages support it?*
- *How do we implement it?*

What is subjectity?

A good number of the submissions to last year's workshop read like the results of a Rorschach test. It seemed clear that there was not a cohesive sense of exactly what subjectivity was, and hence, many participants seemed to project their own interests and requirements onto it. This is neither undesirable or unusual for an emerging area. It merely indicates that we are still at the beginning of a process through which a colletive understanding of our common premises and interests emerges.

At last years workshop, the two most distinct visions of what subjectivity was about where (not surprisingly) provided by the co-organizers. Smith and Ungar proposed *perspectives* as distinct projections of the entire object space that were orthogonal to the existing inheritance hierarchies. Ossher and Harrison espoused a model in which problems were modeled using different views of the same subject matter that could be composed as binaries. The New York school seemed to emphasis the utility of subjects as a modeling and design medium, while the San Francisco school focus on linguistic issues. It would seem that the two perspectives are not incompatable, but this remains to be demonstrated.

It would seem pruduent to re-examine questions like: What is subjectivity?, What isn't?, What is the relationship to objects? Does subjectivity replace objectivity? Does it refine it?

There is a need for more carefully defined "benchmark" problems so as to be able to gauge the relative effectiveness of proposed solutions.

How can langauges support it?

It is clear that a more detailed look at how subjective solutions should be expressed needs to be undertaken if this area is to progress. What languages and tools support subjectivity, or will support it? Can subjective programs be written in current languages? What changes might be made to existing languages to allow this? How might reflective languages help?

How can subjectivity be implemented?

The question of how to implement subjectivity, is of course, intertwined with the langauge issue, but raises extra-linguistic and sublinguistic issues as well. What is the role of preprocessors and tools? How can subjectivity be implemented efficiently? Can subjects be composed?

REFERENCES

[Foote & Johnson 1989]
Brian Foote and Ralph E. Johnson
Reflective Facilities in Smalltalk-80
OOPSLA '89 Proceedings
New Orleans, LA
October 1-6 1989, pages 327-335

[Foote 1991]
Brian Foote
Flexible Foundations and Movable Walls
OOPSLA '91 Workshop on
Reflection and Metalevel Architectures
Phoenix, AZ

[Foote 199x]
Brian Foote
An Object-Oriented Framework
for Reflective Metalevel Architectures
(in progress)

[Harrison & Osher 1993]
William Harrison and Harold Osher
Subject-Orietned Programming
(A Critique of Pure Objects)
OOPSLA '93 Proceedings
Washington, DC, pages 411-428

[Johnson & Foote 1988]
Ralph E. Johnson and Brian Foote
Designing Reusable Classes
Journal of Object-Oriented Programming
Volume 1, Number 2, June/July 1988
pages 22-35

[Kiczales et. al. 1991]
Gregor Kiczales, Jim Des Rivieres,
and Daniel G. Bobrow
The Art of the Metaobject Protocol
MIT Press, 1991

[Maes 1987]
Pattie Maes
Concepts and Experiments in
Computational Reflection
OOPSLA '87 Proceedings
Orlando, FL, October 4-8 1977 pages 147-155

[Stefik & Bobrow 1986]
Mark Stefik and Daniel G. Bobrow
Object-Oriented Programming:
Themes and Variations
AI Magazine 6(4): 40-62, Winter, 1986

[Ungar & Smith 1987]
David Ungar and Randall B. Smith
Self: The Power of Simplicity
OOPSLA '87 Proceedings
Orlando, FL, October 4-8 1977 pages 227-242