Reflective Composition: the Declarative Composition of Roles to Unify Objects, Roles, & Aspects

Simon Holland
Department of Computing
The Open University
Milton Keynes MK7 6AA
United Kingdom
+44 1098 653148
s.holland@open.ac.uk

ABSTRACT
As bases for object-orientation, both class-based and prototype-based organization have limitations. We argue that roles have significant benefits as a foundation for organizing objects. We further argue that these benefits can be realised most flexibly using logic meta-programming. Additional benefits from this approach are to reduce redundancy and subsume aspects.

Categories and Subject Descriptors
D.1.5 [Object Oriented Programming].

General Terms
Design, Experimentation, Languages, Theory.

Keywords
Roles, Logic Meta Programming, Role Models, Composition, Generative Programming, Aspects.

1. REFLECTIVE COMPOSITION

There are practical and philosophical problems with both classes and prototypes as organising mechanisms for object-orientation [1-5]. Class-based organisation has well-known limitations in dealing with rapidly evolving situations [2,6]; prototype-based organisation, though highly flexible, can be undisciplined without additional organising principles [5,7,8]. We argue that an approach to object orientation based fundamentally on roles has the potential for significant benefits; conceptual, methodological, and practical [1,3,9,10]. Abstract arguments in favour of role-modeling are well-known, but have to some extent been muted in their force by difficulties in implementing role-based object mechanisms without introducing new problems [9,11,12]. We consider two such problems, and discuss ways of avoiding them, leading to a new approach to organising objects using role-based principles, known as Reflective Composition (RC).

This paper appeared as Holland, S. (2004) Reflective Composition: the Declarative Composition of Roles to Unify Objects, Roles, and Aspects. In OOPSLA Companion 2004: pp 224-225. John M. Vlissides, Douglas C. Schmidt (Eds.): Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada. ACM 2004, ISBN 1-58113-833-4.

The first problem, typically encountered in any approach to implementing role-based programming is the problem of *object schizophrenia [9,11]*. Informally, this problem can be outlined as follows. When modeling a domain using a role-based approach, two kinds of entity are encountered: *roles* and *instantiable objects*. The economical approach is to model both kinds of entity as objects. However, a problem then arises with object identity, as follows. When an object plays one or more roles (which may interact with each other) then typically (though not always) it is inappropriate from the point of view of domain modeling for each role to have its own identity, as seen by objects external to the containing object. To address this problem, Reflective Composition uses a sub-object/ super object approach, also used by others such as Bardou [11]. In effect, this provides a facility to coalesce an aggregation of sub-objects into a super-object, after which the sub-objects have no separate object identity.

The second principal problem addressed by Reflective Composition is more general. Loosely speaking, the problem is that, as role-based models become larger, they can become difficult to organise and re-use. In order to fully realise, in a scaleable way, the flexibility and expressivity that role-based organisation makes possible, Reflective Composition uses logic meta-programming [13,14] to factor out the definition of all composition relationships (both inheritance relationships, in the broadest sense, and aggregation [15]). One way of viewing this is to say that composition relationships are factored out into a *separate aspect* - though this has nothing to do with the claim that RC unifies role with aspects – this property arises in a different way, as described below.

In order to allow role composition to be factored out cleanly, and to facilitate the maximum flexibility and minimal redundancy in the re-use of roles, this aspect is expressed by a *declarative, reflective, logic meta-programming (LMP) system*, which manipulates composition relationships between parameterised roles [14,16]. A particular LMP program used for this purpose in a given domain is known as a *declarative role composition map* or *role map*. The resulting role maps may be read as abstracted descriptions of a role-based model of the domain in question. Note that this use of LMP has no connection with the composition rules of Ossher et al. [17]. The associated method code describing detailed behaviour is typically relatively less complicated than code that has to deal explicitly with

composition relationships. A system of *aliases* loosely equivalent to *directed resends* in *Self* are used as a mechanism for composing behaviour.

For fully expressive role-modeling power in arbitrary domains, it is not enough to have the capacity to model a single role hierarchy at a time – it is necessary to be able to model role *polyarchies* – arbitrarily overlapping hierarchies with role nodes or subtrees of roles in common. Declarative role composition maps of the kind noted above make directed acyclic graphs of this kind relatively straightforward to model in a disciplined way. In particular, it is straightforward to control sharing and replication in composed structures with an arbitrarily fine granularity. The ability to model role polyarchies directly, coupled with the logic meta-programming approach to composition relationships give Reflective Composition two interesting properties. Firstly, these properties allow code redundancy to be reduced, in principle, to a minimum. In fact, depending on the definition of code redundancy used, there does not seem to be any obvious theoretical limit to the removal of redundancy using this approach. Secondly, because the LMP control of composition relationships allows overlapping role hierarchies to be effectively switched on and off, this provides a relatively simple and straightforward way of implementing declaratively quantified aspect oriented programming[18]. With this perspective, it becomes reasonable to think of the terms *role* and *aspect* as interchangeable for many purposes, without any 'tyranny of the primary decomposition'.

An implementation of Reflective Composition is noted, and various application*s* that have been modeled in this implementation are considered. Related approaches are noted.

2. ACKNOWLEDGMENTS

The work reported here is based on, and aims to continue, the work of the late Henrik Gedenryd on this approach to object organization [19]. Thanks to Henrik Gedenryd for numerous discussions about his work. Thanks to Kris Gybels and Benedict Heal for very useful questions and observations. Thanks to Michael Jackson, Bashar Nuseibeh, Leonor Barroca, Robin Laney and Patrick Hill for generous and useful comments.

3. REFERENCES

[1] Reenskaug, T., Working with Objects: The OORAM Software Engineering Method. 1995, Greenwich, Connecticut: Manning Publications Co. 420.

[2] Taivalsaari, A., On the notion of Inheritance. ACM Computing Surveys, 1996. 28(3): p. 438-479.

[3] Steimann, F., On the Representation of Roles in Object Oriented and Conceptual Modelling. Data & Knowledge Engineering, 2000. 35(1): p. 83-106.

[4] Lieberman, H., Using prototypical objects to implement shared behaviour in object-oriented systems. SIGPLAN Notices, 1986. 21(11): p. 214-223.

[5] Ungar, D. and R.B. Smith, Self, the Power of Simplicity. Lisp and Symbolic Computation, 1991. 4(3): p. 45-55.

[6] Scharli, N., et al. Traits: Decomposable Units of Behaviour. in ECOOP 2003 European Conference on Object-Oriented Programming. 2003: Springer Verlag.

[7] Smith, R.B. and D. Ungar, A Simple and Unifying Approach to Subjective Objects. Theory and Practice of Object Systems, 1996. 2(3): p. 161-178.

[8] Chambers, C., et al., Parents are shared parts of Objects: Inheritance and encapsulation in Self. Lisp and Symbolic Computation, 1991. 4(3): p. 207-222.

[9] Kendall, E.A., Role Model Designs and Implementations with Aspect Orineted Programming. OOPSLA, 1999.

[10] Eco, E., The search for the perfect language (Ricerca della lingua netta cultura europa). 1995, Oxford: Blackwell.

[11] Bardou, D. and C. Dony, Split Objects: a disciplined use of delegation within objects. ACM SIGPLAN Notices - Proceedings of 11th ACM Sigplan Conference on Object-oriented programming, Systems, Languages and Applications, 1996. 31(10): p. 122-137.

[12] Gottlob, G., M. Schrefl, and B. Rock, Extending object-oriented systems with roles. ACM Transaction on Information Systems, 1996. 14(3): p. 268-296.

[13] Gybels, K., Using a logic language to express cross-cutting through dynamic joinpoints. Proceedings of Second German Workshop on Aspect-Oriented Software Development. Technical Report IAI-TR-2002-1, 2002.

[14] Czarnecki, K. and U. Eisenecker, Generative Programming: Methods Techniques and Applications. 1999, Addison Wesley: Reading, MA.

[15] Lopez, C.V. and W.L. Hursch, Separation of Concerns. College of Computer Science, NorthEastern University, Boston, MA., 1995.

[16] Filman, R.E. and D.P. Friedman. Aspect Oriented Programming is Quantification and Obliviousness. in Workshop on Advanced Separation of Concerns, OOPSLA 2000. 2000. Minneapolis.

[17] Ossher, H., et al., Subject-Orineted Composition Rules. OOPSLA 95 ACM SIGPLAN, 1995. 30(10): p. 235-250.

[18] Kiczales, K. Aspect Oriented Programming. in ECOOP 97 Proceedings of European Conference on Object Oriented Programming. 1997: Springer Verlag.

[19] Gedenryd, H., Beyond Inheritance, Aspects and Roles: A unified Scheme for Object and Program Composition. Department of Computing, Open University Technical Report TR 2002/09, 2002.

[20] Gedenryd, H., Holland S. and Morse, D.R. Meeting the Software Engineering Challenges of Interacting with Dynamic and Ad-hoc Computing Environments. Department of Computing, Open University Technical Report TR 2002/08, 2002.