Reflective Composition: the declarative composition of roles to unify objects, roles, and aspects [poster session]

Conference or Workshop Item

How to cite:


For guidance on citations see FAQs.

Version: Accepted Manuscript

Link(s) to article on publisher’s website:

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online’s data policy on reuse of materials please consult the policies page.
**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. ROLES AS A BASIS FOR OBJECT-ORIENTATION

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).

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 instantaneous 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.

2. LOGIC META-PROGRAMMING AND GENERALISED COMPOSITION

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 roles 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.

3. **ROLES, LMP AND ASPECTS**

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’.

4. **CONCLUSIONS**

Roles have both philosophical and practical benefits as a foundation for organizing objects. Logic meta-programming can be combined with this approach to abstract out composition and inheritance-like relationships. The resulting framework, known as Reflective Composition, leads to flexibility, a good foundation for re-use, potentially reduced redundancy and the subsumption of aspects. Reflective Composition has been implemented in Squeak Smalltalk. Various prototype applications have been implemented using this version of RC.

5. **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.

6. **REFERENCES**


