A Runtime Model for Multi-Dimensional Separation of Concerns

Awais Rashid, Ruzanna Chitchyan
Computing Department, Infolab21, Lancaster University, Lancaster LA1 4WA, UK
{awais | rouza} @comp.lancs.ac.uk

Abstract

Multi-dimensional separation of concerns techniques for aspect-oriented software development (AOSD) support symmetric representation and composition of various concerns in a system. In a multi-dimensional separation one can project any set of concerns on another set of concerns hence offering powerful modular and compositional reasoning abilities. This is in contrast with asymmetric approaches to AOSD which distinguish aspects from base concerns with the projections directed from aspects to base. However, to date, most multi-dimensional models only exist during analysis and design or, at best, as static models of programs – the multi-dimensionality disappears as the program models are statically composed. In this paper we highlight the challenges of maintaining a multi-dimensional model at runtime and discuss how these have been addressed in the Dynamic Hyperslices approach.

1. Introduction

Aspect-oriented software development (AOSD) techniques are generally classified into two broad categories:

- *Asymmetric*, e.g., [4, 7, 16, 22, 24], where there is a base model in a dominant decomposition, for instance, an object-oriented model, with an aspect model crosscutting the elements in the base model.
- *Symmetric or multi-dimensional*, e.g., [9, 13, 14, 25, 27], where there is no separation between base and aspects. All concerns are treated as first-class entities in the model with the ability to project or compose any set of concerns with any other set of concerns.

Multi-dimensional models are, of course, more powerful than their asymmetric counterparts. While in an asymmetric separation aspect composability is constrained by the composition semantics of the base decomposition model, there are no such restrictions in a multi-dimensional model. One is able to define powerful and expressive composition operators and utilise these as a basis to reason about and manipulate the relationships between concerns and the influences and dependencies they exert on each other – this has been termed *compositional reasoning* [21].

As far as asymmetric models are concerned, we find a number of techniques aiming to maintain first-class representations of such models at runtime. AspectJ [4], for instance, offers a meta-aspect protocol (MAP) with basic introspection support as well as load-time composition of aspects. Other systems such as JAC [17] and PROSE [18] maintain runtime representations of the aspect and base models and support runtime adaptation of both models.

In contrast, multi-dimensional models have so far mainly been utilised for analysis and design [10, 13, 14, 25]. Runtime representations of such models have been largely unexplored. The Hyperspaces approach [27] and its supporting tool Hyper/J [15] provide support for multi-dimensional models during design and implementation. However, the composition in Hyper/J is based on statically merging the various concern slices. As a result, the multi-dimensional separation is mapped on to a two-dimensional model (Java's object-oriented model) and the concerns in the analysis and design models (as well as the static program model) do not have counterparts at runtime. This, in turn, means that one cannot introspect or manipulate concerns in the multi-dimensional concern model at runtime for dynamically adapting the system. Any changes have to be made statically and the concerns recomposed to map onto the object-oriented model for execution.

In this paper, we tackle the issue of maintaining such a multi-dimensional model at runtime and supporting dynamic composition and adaptation of concerns in such a model. Such a multi-dimensional runtime model has the usual applications in supporting dynamic composition and reconfiguration in auto-adaptive systems, such as those deployed in ambient computing environments. However, more significantly, it helps improve traceability from multi-dimensional analysis and design models – which offer more powerful means for analysis of broadly-scoped properties in auto-adaptive systems [14] – to runtime. Section 2 discusses the challenges of maintaining a multi-dimensional model at runtime. Section 3 discusses how these challenges have been addressed in the Dynamic Hyperslices approach, that extends the notion of Hyperslices from [27] to runtime. Section 4 discusses some related work and Section 5 concludes the paper.

2. Challenges for Multi-Dimensionality at Runtime

There are three key challenges when attempting to maintain a multi-dimensional separation at runtime. We discuss each of these below.

Flattening a multi-dimensional model onto an OO model

In a multi-dimensional model concerns are not sliced on class boundaries. Instead they are often sliced on the basis of the high-level features in a system. For instance, *payroll* and *personnel* may be considered as two concerns in a multi-dimensional model (e.g., the Hyperslices model [27]) with fragments of the class *Person* appearing in both (cf. Fig. 1). Note that the same method getSalary( ) appears in $Person_{payroll}$ and $Person_{personnel}$ as both concerns have to be declaratively complete – both personnel and payroll may require access to salary information. In other cases, there may be different fragments of the behaviour of such "shared" method in each concern.

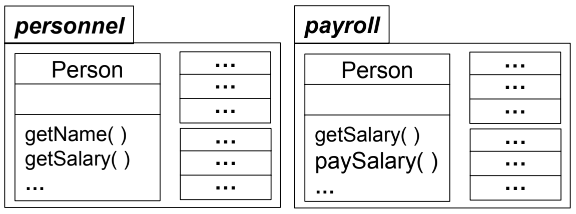

Fig. 1: Class fragments in a multi-dimensional slicing

Static flattening of such a model onto an OO model is easier as one can combine the various fragments of *Person* into one class definition which is then used for runtime representation of objects. In fact, this is what the Hyper/J tool does in a nutshell. However, if we wish to maintain the two concerns in our multi-dimensional separation at runtime along with the partial definitions of the class *Person* within each, then type safety issues come into play. How do we maintain two (or more) variations of the class *Person* at runtime? One way to achieve this is to use different packages to represent different concerns. However, then the *Person* class definitions in the two packages represent two different classes. But semantically all *Person* objects should belong to both class definitions. This implies that we need support for multiple classification of objects. Though such multiple classification can be simulated via inheritance, firstly, $Person_{personnel}$ is not a sub-type of $Person_{payroll}$ and vice versa, and secondly, this would lead to deep inheritance trees with high levels of fragility [12] as concern slices are dynamically adapted.

Nature of composition operators

Composition operators in multi-dimensional approaches are also driven by the above-mentioned need to flatten the model onto an OO model (or another model such as, functional programming, for that matter) for execution purposes[1]. When mapping to an OO model (for instance, in Hyper/J) the operators focus on *merging* the various class fragments based on certain syntax or semantics. In cases where a method appears in multiple class fragments, one can specify whether one of the various method implementations is to be used (i.e. other implementations are *overridden*) or if they are all to be used and executed in a specific order. While for runtime representation, we need to maintain the concerns and class fragments as first-class entities, at the same time, there is a need to realise the semantics of static merging, i.e. the concern compositions should yield the same result (behaviourally) during runtime composition as they would had static composition been used. This requires first-class representation of compositions so that they may be manipulated and adapted at runtime as the concern slices evolve and the *merge/over-ride* relationships change.

[1] No doubt that the ideal solution would be to design a language and runtime for multi-dimensional models. However, the need for backward compatibility and utilisation of existing programmer expertise drives the mapping onto an existing programming model for execution.

Concern interfaces and consistency preservation

If a multi-dimensional concern separation is to be maintained at runtime, it also needs to be clarified as to what is the *public interface* of a concern. Is the interface the union of the public interfaces of the various class fragments in the concern slice? For static reasoning this would be sufficient but if we choose this approach for runtime representation, then the interface of a concern will change each time there is a change to the public interface of one of the class fragments within it. While simpler consistency issues can be resolved automatically – additive changes being backward compatible and subtractive changes being forward compatible – the more arbitrary changes cannot be so easily resolved. Therefore, for runtime representation of concerns in a multi-dimensional separation, we need provided/required interfaces for each concern (in the same fashion as components [26]) that abstract over the public interfaces of the various class fragments.

3. The Dynamic Hyperslices Approach

The Dynamic Hyperslices approach, initially presented in [5, 6], aims to maintain a runtime representation of the Hyperspaces model from [27] for runtime adaptability and evolution of concerns (referred to as *hyperslices*). Below we discuss how we address the three issues highlighted in Section 2 in the design of the Dynamic Hyperslices approach. Interested readers are referred to [5, 6] for details on the implementation of the Dynamic Hyperslices approach.

Runtime representation of concerns

The Dynamic Hyperslices approach provides a meta-object protocol (MOP) that supports first-class representation of each concern (hyperslice) at runtime. It also supports first class representation of the concern compositions (as discussed below). The MOP supports introspection and adaptation of the concerns and their compositions at runtime. We deal with the issue of multiple classification by using a mechanism analogous to having versioned type semantics in object database environments [19, 20]:

- Each class A can have multiple representations of itself, $a_1, a_2, \ldots, a_n$, in the various concerns, $C_1, C_2, \ldots, C_n$, at runtime.
- Semantically, each a_i is equivalent to A, i.e. whenever an object of type A is required an object of type a_i can be substituted.
- Objects are instantiated per class fragment a_i and bound to the MOP representation of their instantiating fragment over their lifetime (unless the fragment is deleted in which case the object can be, if so desired, reclassified and rebound to another fragment of the same class). Method invocations and field accesses are delegated to the relevant object slices as dictated by the composition specifications.

Concern composition

For first-class representation of the concern compositions (and the merge/over-ride semantics of such compositions) we introduce the notion of a *composition connector*, which combines the concept of a connector in software architecture [2] with the notion of a filter in the Composition Filers approach [1]. Our composition connectors differ from a normal architectural connector in that they connect partial class representations and not complete classes or components. Furthermore, they not only match provided/required services and specify roles for connected concerns, but also support dynamically updateable composition strategies to build up functionality of coarser-grained concerns from primitive hyperslices. The filters associated with a composition connector intercept incoming and outgoing messages to a composite hyperslice and dispatch them to the relevant primitive hyperslice as per the composition strategy. Interested readers are referred to [5] for more details of the structure of a composition connector as well as realisation of the various composition operators (such as *merge*) for runtime composition. An illustrative example of runtime adaptation using the Dynamic Hyperslices approach is also provided in [5].

Concern interfaces

As mentioned above, the composition connectors carry out the composition based on provided/required interfaces. The provided/required interfaces are similar to those well-known in the components-based development space. The provided interface of a hyperslice declares the methods it exposes to other hyperslices (these methods may or may not have a one-to-one correspondence with those of its constituent class fragments). The required interface, on the other hand, specifies what methods a hyperslice expects from those it is to be composed with to carry out its tasks in line with the composition strategy in the composition connector. Note that connectors have the ability to reflect on the internals of their immediately connected hyperslices while still keeping these internals hidden from all other connectors and hyperslices.

As such impact of the changes to the internals of a hyperslice is largely contained within that hyperslice, unless an internal change triggers the need to change a provided interface (which may in turn trigger the need to change a required interface elsewhere). The dynamic adaptation capabilities of the MOP extend to such changes. Since such information is contained in the connectors, usually simpler adaptations (e.g., renaming) can be made by creating a *bridge* at the connector level instead of adapting the actual concern interfaces. Subtractive changes are only allowed if the specific method in the concern interface is not being used by any other concern. This is achieved by maintaining a counter for references from required interfaces within the connector, with method deletion from a provided interface allowed only if the counter is zero.

4. Related Work

The work presented in this paper strongly relates to the notion of dynamic aspect-oriented programming (AOP) techniques, e.g., [17, 18]. As mentioned earlier, the key distinction is that we focus on providing a runtime representation and adaptation mechanism for a multi-dimensional separation of concerns approach while most dynamic AOP techniques focus on asymmetric models.

In fact, our work can be seen as an attempt to bring component-based and aspect-oriented approaches closer but with focus on multi-dimensional concern models. The notion of provided/required interfaces for our concerns at runtime is similar to what is employed in CaesarJ [16], which utilises the concept of provided/required interfaces and role-based composition as in component-based software development. Similarly, the introspection and adaptation of hyperslices through our composition connectors is analogous to open component models such as OpenCOM [8]. Finally, our composition connectors compose partial class representations which is similar to mixin-based composition of concerns [3, 23].

5. Conclusion

In this paper, we have highlighted three key challenges for maintaining a multi-dimensional separation of concerns model at runtime and dynamic introspection and adaptation of concerns in such a model. These challenges pertain to multiple classification due to the mapping of concerns slices onto an OO model; first-class representation of concern compositions and the merge/over-ride semantics used in them; and well-defined interfaces for concerns at runtime. We have discussed how these issues have been addressed in the Dynamic Hyperslices approach through the multiple classification and dynamic reclassification mechanisms employed in environments with versioned type systems, use of composition connectors and specifying clear provided/required interfaces for concerns.

One of the key advantages of the approach is the ability to map the multi-dimensional analysis and design models onto the system runtime. This, in turn, facilitates derivation of proof obligations from the analysis and design models [11] which can themselves be represented as a concern in the Dynamic Hyperslices model. Such proof obligation concerns can monitor and enforce constraints (hence aiding verification and validation) to ensure that the runtime representations meet the constraints and trade-offs specified by the analysis and design models. Another key strength of the approach is the direct correspondence of runtime entities with user views, hence facilitating the mapping of requirements-level changes on to the running system.

6. References

[1] M. Aksit, L. Bergmans, and S. Vural, "An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach", European Conference on Object-Oriented Programming (ECOOP) 1992, Springer LNCS 615, pp. 372-395.

[2] R. Allen and D. Garlan, "A Formal Basis for Architectural Connection", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 6, No. 3, pp. 213-249, 1997.

[3] S. Apel, T. Leich, and G. Saake, "Aspectual Mixin Layers: Aspects and Features in Concert", International Conference on Software Engineering (ICSE), 2006, ACM, pp. 122-131.

[4] "AspectJ Project", http://www.eclipse.org/aspectj/, 2006.

[5] R. Chitchyan and I. Sommerville, "Composing Dynamic Hyperslices", Workshop on Correctness of Model-based Software Composition (held with ECOOP), 2003, pp. 29-36.

[6] R. Chitchyan, I. Sommerville, and A. Rashid, "A Model for Dynamic Hyperspaces", Workshop on Software Engineering Properties of Languages for Aspect Technologies (held in conjunction with AOSD), 2003.

[7] S. Clarke and E. Baniassad, *Aspect-Oriented Analysis and Design: The Theme Approach*: Addison-Wesley, 2005.

[8] G. Coulson, G. S. Blair, P. Grace, A. Joolia, K. Lee, and J. Ueyama, "OpenCOM v2: A Component Model for Building Systems Software", IASTED Software Engineering and Applications (SEA), 2004.

[9] W. Harrison, H. Ossher, S. M. Sutton, and P. L. Tarr, "Supporting Aspect-Oriented Software Development with the Concern Manipulation Environment", *IBM Systems Journal*, Vol. 44, No. 22, pp. 309-318, 2005.

[10] M. Kande, "A Concern-Oriented Approach to Software Architecture": PhD Thesis, EPFL, Switzerland, 2003.

[11] S. Katz and A. Rashid, "From Aspectual Requirements to Proof Obligations for Aspect-Oriented Systems", International Conference on Requirements Engineering (RE), 2004, IEEE Computer Society, pp. 48-57.

[12] L. Mikhajlov and E. Sekerinski, "A Study of The Fragile Base Class Problem", European Conference on Object-Oriented Programming (ECOOP), 1998, Springer LNCS 1445, pp. 355-382.

[13] A. Moreira, J. Araujo, and A. Rashid, "A Concern-Oriented Requirements Engineering Model", International Conference on Advanced Information Systems Engineering (CAiSE), 2005, Springer LNCS 3520, pp. 293-308.

[14] A. Moreira, A. Rashid, and J. Araujo, "Multi-Dimensional Separation of Concerns in Requirements Engineering", International Conference on Requirements Engineering (RE), 2005, IEEE Computer Society, pp. 285-296.

[15] H. Ossher and P. L. Tarr, "Hyper/J: multi-dimensional separation of concerns for Java", International Conference on Software Engineering, 2001, ACM, pp. 734-737.

[16] "CaesarJ", http://caesarj.org/, 2006.

[17] R. Pawlak, L. Seinturier, L. Duchien, and G. Florin, "JAC: A Flexible Solution for Aspect-Oriented Programming in Java", 3rd International Conference on Meta-Level Architectures and Separation of Concerns (Reflection), 2001, Springer LNCS 2192, pp. 1-25.

[18] A. Popovici, A. Frei, and G. Alonso, "A Proactive Middleware Platform for Mobile Computing", ACM/IFIP/USENIX International Middleware Conference, 2003, Springer LNCS 2672, pp. 455-473.

[19] A. Rashid and N. Leidenfrost, "Supporting Flexible Object Database Evolution with Aspects", International Conference on Generative Programming and Component Engineering (GPCE), 2004, Springer LNCS 3286, pp. 75-94.

[20] A. Rashid and N. Leidenfrost, "VEJAL: An Aspect Language for Versioned Type Evolution in Object Databases", Workshop on Linking Aspect Technology and Evolution (held in conjunction with AOSD), 2006.

[21] A. Rashid and A. Moreira, "Domain Models are NOT Aspect Free", Proceedings of MoDELS/UML, 2006, Springer (Accepted to Appear).

[22] A. Rashid, A. Moreira, and J. Araujo, "Modularisation and Composition of Aspectual Requirements", International Conference on Aspect-Oriented Software Development (AOSD), 2003, ACM, pp. 11-20.

[23] Y. Smaragdakis and D. S. Batory, "Implementing Layered Designs with Mixin Layers", European Conference on Object-Oriented Programming (ECOOP), 1998, Springer LNCS 1445, pp. 550-570.

[24] D. Stein, S. Hanenberg, and R. Unland, "Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design", International Conference on Aspect-Oriented Software Development (AOSD), 2006, ACM, pp. 15-26.

[25] S. M. Sutton and I. Rouvellou, "Modeling of Software Concerns in Cosmos", International Conference on Aspect-Oriented Software Development (AOSD), 2002, ACM, pp. 127-133.

[26] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*: Addison-Wesley-Longman, 1998.

[27] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. Sutton, "N Degrees of Separation: Multi-Dimensional Separation of Concerns", International Conference on Software Engineering (ICSE), 1999, ACM, pp. 107-119.