ACM Computing Surveys **28**(4es), December 1996, http://www.acm.org/pubs/citations/journals/surveys/1996-28-4es/a148-aksit/. This article derives from a position statement prepared for the Workshop on Strategic Directions in Computing Research.

Separation and Composition of Concerns in the Object-Oriented Model

Mehmet Aksit
University of Twente, Department of Computer Science, The TRESE Project
P.O. Box 217, 7500 AE Enschede, The Netherlands
aksit@cs.utwente.nl, http://wwwtrese.cs.utwente.nl/~aksit/

Abstract: This is a position statement for the workshop on strategic directions in computing research held at MIT in June 1996.

Categories and Subject Descriptors: D.2.10 **[Software Engineering]** Design; D.3.2 **[Programming Languages]** Object-Oriented Languages; D.2.2 **[Software Engineering]** Tools and Techniques; D.2.7 **[Software Engineering]** Enhancement;

General Terms: Object-Oriented Programming

Additional Key Words and Phrases: software composition, application-domain concerns, composition filters

---

In the (conventional) OO model, the *separation of concerns* principle is supported basically in three ways:

1. By defining objects as the models of real-world concepts that are ``naturally'' separated from each other
2. By separating the concerns of providing an abstract object interface and its implementation
3. By grouping functions together around objects so that functions that are less related are structurally separated from one another

Composition of Concerns

To be able to construct complex software systems, the separate concerns must be put together with minimum effort. The OO model provides various ways in composing concerns together:

1. In the implementation part of an object, the structure and the behavior of the nested implementation objects can be composed under the definition of the encapsulating object;
2. Both inheritance and delegation mechanisms define composition of behavior. For example, in inheritance, the operations defined within a subclass is composed with the operations of its superclass(es);

3. By defining a set of protocols among cooperating objects, the software engineer may create more complex system structures provided that each component (or object) fulfills the protocol specification.

Application Domain Concerns

Application domains may define additional concerns. Consider, for example, an electronic mail object that provides operations for defining the sender, receiver and the content of the mail. In addition, various operations are defined to approve and deliver the mail to its destination. An important requirement here is that only the ``system objects'' are allowed to invoke the approve and deliver operations. In addition to the previously mentioned concerns, a mail object has therefore an additional concern: multiple views. Each mail object has a user view and a system view that restrict the operations of the mail object with respect to its clients.

To implement the mail object, each concern must be mapped to an OO concept. For example, views can be implemented as operations. The operation approve, for instance, can be executed if the operation checkSystemView returns True. This implementation, however, blurs the separation of the multiple-views concern because the views are then realized within the operations of objects.

The problem appears if we want to extend/modify the mail object. For example, we may further partition the user view as sender and receiver views. We may extend the sender and receiver views to group-sender and group-receiver views. We may want to give a warning message if the same operation is invoked for the same mail object twice. In almost all these cases, the OO model cannot express these extensions without redefining the previously defined mail object. This is because the mail object cannot implement multiple views on objects as a separate and composable concern.

Obviously, there can be many other concerns. For example, various synchronization constraints can be defined for the mail object. A request for send, for instance, can be delayed if the sender, receiver and the mail content have not been defined yet. Other possible concerns are for example, addressing the history information, evolution of behavior, coordinated behavior, security and reliability measures, real-time behavior, distribution aspects, etc. Since, the OO model may not separate these concerns adequately, the resulting programs are likely to be less adaptable and reusable. Several publications identified the composability problems for certain concerns such as atomic transactions [14], synchronization [22] and real-time specifications [5]. [16] discusses the problem of separation of concerns in OO modeling.

The proposed design patterns in [11] cannot solve these problems adequately because the composability features of the patterns are defined by the capabilities of the conventional OO model.

Extensions to the Object Model

One may define a composable model for a particular concern by identifying the orthogonal components and the composition operators for that concern. For example, the multiple-view problem can be modeled by representing views as explicit concepts and by defining accept

functions between views and the operations of the mail object. These additional concepts and operators have to be integrated with the OO model. This can be realized basically in two ways:

1. *Using special language constructs*: Many research proposals introduce new language constructs and/or computation models to tackle a given problem. For example, [18], [28], [15] and [19] introduce language constructs to model synchronization, real-time, coordinated behavior and multiple-view concerns, respectively. The language constructs introduced must be uniformly integrated with the composability features of the underlying object model. Otherwise, the resulting programs cannot be fully composable. In most publications, expressiveness is the major concern and the proposed language constructs are hardly composable.
2. *Meta-level programming*: Meta-level programming can be used to solve specific concerns at a meta-level. Reflection techniques can be used to keep different levels consistent with each other. For example, views and accept functions can be defined at a meta-level without modifying the basic structure of the mail object. A number of papers have presented meta-level solutions for various problems, such as distributed architectures [23], atomic transactions [26], concurrent programming [17], operating system structuring [30] and compiler design [20].The challenge here is to define reflection techniques that support, in addition to the basic OO concerns, a large number of possible application-defined concerns in a composable manner. Most research activities in this area do not address the composability issues of the proposed meta-level concerns.

Composition of Concerns

Apart from our work, a number of publications have recently appeared to address composability problems in object-oriented modeling [25] [24] [10].

We have extended the conventional OO model with the concept of composition filters (CFs). Each message that arrives at an object (or sent from an object) is subject to evaluation and manipulation by the CFs of that object. CFs can be attached to objects expressed in different languages, such as C++ or Smalltalk [13] [9], and therefore allow extensions of programs written in different languages; the conventional OO model can be used to implement the basic concepts such as the mail object, and each additional concern can be expressed as a CF.

Several different filter types have been defined in the past. For example, [1] illustrated how both inheritance and delegation can be simulated using filters. In [2] filters were introduced for defining reusable transactions. Language-database problems were addressed in [3]. In [4], filters were used to abstract coordinated behavior among objects. The application of composition-filters for real-time specifications appears in [5], and composing synchronization and real-time specifications is discussed in [8].

Since different types of CFs show a similar structure, we have been investigating more primitive compositional structures than those provided by the CF model. We modeled the currently defined CFs using so-called message manipulators [27]. Message manipulators define logical operators for the received (and sent) messages to (or from) an object. These operators are for example, AND, Conditional-AND, OR, Conditional-OR, and Sequential manipulators. Each manipulator can be further decomposed by using sub-manipulators, etc. A

``terminal manipulator" is composed of a constraint checker and accept and reject handlers. Constraint checkers are applied to messages. The accept and reject handlers are first-class objects and represent the semantics of different concerns. Our conclusion here is that defining logical message composition operators with extensible semantics is a promising way to compose together separated concerns.

We believe that the concept of composition of different concerns must be also applied during the software development process. Propogation patterns [21], for example, separate the concern of defining algorithms and class structures from each other. During software development, the so-called software artifacts are generated in various formats, from informal textual information to executable object-oriented programming concepts. Composability of design models requires explicit representation of software artifacts in a composable way. In our recent work [7], we have applied fuzzy-logic-based techniques to represent and compose various software artifacts. In contrast to deterministic object-level compositions, we found the fuzzy-logic-based reasoning techniques more appropriate for representing design level concerns because fuzzy logic can deal with design uncertainties.

The so-called software architecture definition languages (ADLs) [29] are used to model and structure higher-level design concepts. Most architecture definition languages, however, do not adequately address the issue of evolution and composition of different architectural concepts [12]. In this direction, we are currently defining an ADL based on the concept of composition of specializations of knowledge domains [6].

References

[1] M. Aksit and A. Tripathi. Data Abstraction Mechanisms in Sina/ST, Proc. of the OOPSLA '88 Conference, ACM SIGPLAN Notices, Vol. 23, No. 11, November 1988, pp. 265-275.

[2] M. Aksit, J.W. Dijkstra and A. Tripathi. Atomic Delegation: Object-oriented Transactions, IEEE Software, Vol. 8, No. 2, March 1991, pp 84-92.

[3] M. Aksit, L. Bergmans and S. Vural. An Object-Oriented Language-Database Integration Model: The Composition-Filters Approach, Proc. of the ECOOP '92 Conference, LNCS 615, Springer-Verlag, 1992, pp. 372-395.

[4] M. Aksit, K. Wakita, J. Bosch, L. Bergmans and A. Yonezawa. Abstracting Object-Interactions Using Composition-Filters, In Object-based Distributed Processing, R. Guerraoui, O. Nierstrasz and M. Riveill (eds), LNCS 791, Springer-Verlag, 1993, pp 152-184.

[5] M. Aksit, J. Bosch, W. v.d. Sterren and L. Bergmans. Real-Time Specification Inheritance Anomalies and Real-Time Filters, Proc of the ECOOP '94 Conference, LNCS 821, Springer Verlag, July 1994, pp. 386-407.

[6] M. Aksit, F. Marcelloni, B. Tekinerdogan, C. Vuijst and L. Bergmans. Designing Software Architectures as a Specializations of Knowledge Domains, University of Twente, Memoranda Informatica 95-44, December 1995.

[7] M. Aksit and F. Marcelloni. Minimizing Quantization Error and Contextual Bias Problems of Object-Oriented Methods by Applying Fuzzy-Logic Techniques, University of Twente, 1996.

[8] L. Bergmans and M. Aksit. Composing Synchronization and Real-Time Constraints, University of Twente, Memoranda Informatica 95-41, (to be published in Journal of Parallel and Distributed Computing September 1996), December 1995.

[9] W. van Dijk and J. Mordhorst. CFIST, Composition Filters in Smalltalk, Graduation Report, HIO Enschede, The Netherlands, May 1995.

[10] I. Forman, S. Danforth and H. Madduri. Composition of Before/After Metaclasses in SOM, Proc. of the OOPSLA '94 Conference, ACM SIGPLAN Notices, Vol. 29, No. 10, October 1994, pp. 427-439.

[11] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[12] D. Garlan, R. Allen and J. Ockerbloom. Architectural Mismatch or, Why it's Hard to Build Systems out Existing Parts, Proc. of the 17th. Int. Conf. on Software Engineering, April 1995.

[13] M. Glandrup. Extending C++ Using the Concepts of Composition Filters, M.Sc. Thesis, University of Twente, November 1995.

[14] R. Guerraoui. Atomic Object Composition. Proc of the ECOOP '94 Conference, Springer-Verlag, 1994, pp. 118-138.

[15] I. Holland. Specifying Reusable Components Using Contracts, Proc. of the ECOOP '92 Conference, LNCS 615, Springer-Verlag, 1992, pp. 287-308.

[16] W. Hursch and C. Lopes. Separation of Concerns, Northeastern University, February 1995.

[17] Y. Ichisugi, S. Matsuoka and A. Yonezawa. A Reflective Object-Oriented Concurrent Language Without a Run-Time Kernel, Int. Workshop on New Models for Software Architecture'92, Reflection and meta-Level Architecture, Yonezawa & Smith (eds), November 1992, pp. 24-35.

[18] D.G. Kafura and K.H. Lee. ACT++: Building a Concurrent C++ with Actors, J. of Object-Oriented Programming May/June 1990, Vol. 3, No. 1, pp. 25-37.

[19] H. Kiessling and U. Kruger. Sharing Properties in a Uniform Object Space. Proc. of the ECOOP'95 Conference, LNCS 952, Springer Verlag, 1995, pp. 424-448.

[20] J. Lamping, G. Kiczales, L. Rodriguez and E. Ruf. An Architecture for an Open Compiler, Int. Workshop on New Models for Software Architecture'92, Reflection and meta-Level Architecture, Yonezawa & Smith (eds), November 1992, pp. 95-106.

[21] K. Lieberherr. Adaptive Object-Oriented Software the Demeter Method with Propagation Patterns., PWS Publishing Company, 1995.

[22] S. Matsuoka and A. Yonezawa. Inheritance Anomaly in Object-Oriented Concurrent Programming Languages, in Research Directions in Concurrent Object-Oriented Programming, (eds.) G. Agha, P. Wegner and A. Yonezawa, MIT Press, April 1993, pp. 107-150.

[23] J. McAffer. Meta-level Programming in CodA, Proc. of the ECOOP'95 Conference, LNCS 952, Springer Verlag, 1995, pp. 190-214.

[24] P. Mullet, J. Malenfant and P. Cointe, Towards a Methology for Explicit Composition of MetaObjects, OOPSLA'95 Conference Proceedings, ACM Sigplan Notices, Vol. 30, No. 10, October 1995, pp. 316-330.

[25] O. Nierstrasz & D. Tsichritzis (eds), Object-Oriented Software Composition, Prentice Hall, 1995.

[26] R. Stroud and Z. Wu, Using Metaobject Protocols to Implement Atomic Data Types, Proc. of the ECOOP'95 Conference, LNCS 952, Springer Verlag, 1995, pp. 168-189.

[27] C. Stuurman. Techniques for Defining Composition-Filters Using Message Manipulators, M.Sc. Thesis, University of Twente, August 1995.

[28] K. Takashio and M. Tokoro. DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems, Proc of the OOPSLA '92 Conference, ACM SIGPLAN Notices, Vol. 27, No. 10, October 1992, pp. 276-294.

[29] S. Vestal. A Cursory Overview and Comparision of Four Architecture Description Languages, Honeywell Technology Center, Minneapolis, February 1993.

[30] Y. Yokote. The Apertos Reflective Operating System: The concept and its Implementation, Proc of the OOPSLA'92 Conference, ACM SIGPLAN Notices, Vol. 27, No. 10, October 1992, pp. 414-434.

Last modified: Wed Feb 26 07:29:39 EST 1997
Mehmet Aksit *aksit@cs.utwente.nl*