Separation of Concerns in Early Stage of Framework Development

Shin NAKAJIMA
NEC C&C Media Research Laboratories
4–1–1 Miyazaki, Miyamae-Ku, Kawasaki, 216-8555 JAPAN
(TEL) +81–44–856–2259, (FAX) +81–44–856-2233
nakajima@ccm.cl.nec.co.jp

1 Introduction

Object-oriented framework is a promising solution technology for improving reusability of software, where a framework is a reusable design of a program or a part of a program expressed as a set of classes [5][8]. Having recognized the importance of object-oriented framework, many methodologists propose design methods focusing on framework development. The methods include collaboration-based design [3][4][10], role-based design [14] and design patterns [6][8][13]. However, desiging well-organized frameworks is still an art.

Jackson points out two important issues in general on method and problem [7]; (1) methods cannot be panaceas (medicines that cure all diseases), and (2) very few problems can be decomposed into *homogeneous* structures. Because real world system is complex, the problem is decomposed into a set of subproblems. The subproblem is *heterogeneous* in the sense that it needs a different problem frame (a kind of structural pattern to solve the subproblem). In addition, methods should be related to a particular class of problem and thus give a sharply focused help in reaching a solution.

In developing object-oriented frameworks for a real world complex system, a method focused on *separation of concerns* is important at an early design stage. The method bridges the gap between the complex problem and existing object-oriented methods; the problem is one such that is decomposed into a set of heterogeneous subproblems (separating out concerns), and the object-oriented methods can handle only homogeneous world of objects.

This paper discusses a method that deals with *separation of concerns* at an early design stage. It is a summary of an experience [12] in developing object-oriented frameworks for an implementation of the OMG trading server [1]. The paper also identifies three further research topics.

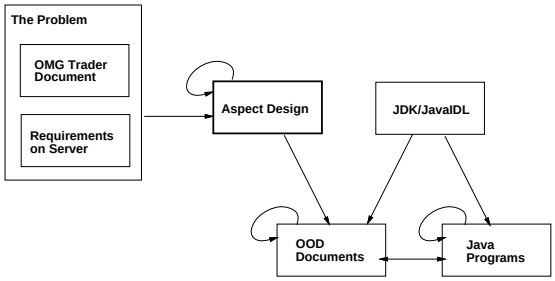

Figure 1: Design Process

2 *Separation of Concerns* Phase

Existing design methods for developing object-oriented frameworks are effective in general. The methods, however, have several drawbacks. (1) The design methods have their basis on the object-orientation, and explicitly assume that *object* is sole constituent of the system. (2) The methods provide only general guidelines of decomposing a whole problem into constituent objects, and mention no concrete hint for the decomposition. (3) The design pattern is a catalog of useful design idioms, but most of them are at a programming level and are thus not suitable for use at an early stage of the development.

Real world software such as the OMG trader server is a complex system. The target problem has various aspects[1] that are not amenable to *homogeneous* object-oriented modeling. Analyzing the target problem and decomposing it into a set of subproblems is the most important task.

Figure 1 summarizes the design process adapted in the present approach. The first step of the process (the aspect design phase) is identifying a set of distinct aspects in the problem to obtain a semi-formal description. The phase starts with analyzing both the OMG document and the system requirements. Using a specification technique best fitted for the characteristics of each aspect reaches aspect solutions. The solutions form a whole design artifact that is the input to the next phase. The phase (object-oriented design) makes use of existing methods such as collaboration-based design or design patterns. In the course of preparing the design document, some part of the framework is implemented incrementally in Java.

The following two characteristics of the aspect design seems well-known common practices; (1) decomposing a large complex problem into a set of manageable subpoblems to solve individually, and (2) seeing a target system from various viewpoints. For example, a top-down functional design approach deals

[1] The terminology, *aspect*, is borrowed from Kiczales et al[9] because viewing a software system consisting of many aspects is the common idea.

Aspect	Specification Technique
language	denotational semantics
policy	functional programming
algorithm	stream-style programming
common concept	abstract datatype
architecture	collaboration-based design
functional object	collaboration-based design

Table 1: Aspects and Specification techniques

with decomposition into procedures or processes. Its decomposition is homogeneous and hierarchical. OMT provides three *models* (object, dynamic, and functional), and promotes a method to describe the system behavior by using the three different models[15]. The model, however, represents a different viewpoint of a same entitiy, *object*.

On the other hand, the important characteristics of the aspect-centered design method is *heterogeneity*. *Aspect* is related to a subproblem that is further refined and elaborated to reach solution description individually. The subproblem needs not to be object-oriented, but is *heterogeneous* in the sense that each subproblem is associated with specification technique best fitted for its intrinsic nature.

Table 1 summarizes the identified aspects together with the accompanying specification techniques. Of the entries in Table 1, the meaning of policies and the query algorithm are described in terms of the stream-style functional programming model, and the constraint language is defined and elaborated by means of standard technique for defining language semantics. The common concepts such as `ServiceType` and `PropertyDefinition` provide basic vocabularies and are naturally modeled as abstract datatypes. Both the architecture and functional objects are refined and elaborated by using collaboration-based design methods [3][10]; collaboration diagrams or message sequence charts are used to analyze their interaction patterns so that the responsibility of each participant object is well defined. However, the control aspects of the resouce management requires detailed knowledge of the middleware solution used and the execution mechanism that the implementation language/library (Java and JavaIDL) provides.

3 Discussions

The present approach to *separation of concerns* is based on an idea of identifing a set of distinct aspects in the target problem. The aspect solutions form a clear input specification to the framework design phase where existing methods such

as collaboration-base design and design patterns are employed (Figure 1).

The aspect solution description is concise and thus expected to ease future maintenance.[2] For example, sixteen functions for the algorithm abstractly describes design of about 30 Java classes, and about thirty lines of the denotational style language description becomes more than 4K lines of Java codes including classes for abstract syntax tree and visitor skeletons.

The aspect solution shows clear relationships between the design descriptions at different levels. First, the stream-style description does not have a large gap with the OMG document, and, thus, both descriptions are quite traceable. It is easy to perform conformance checking during the design phase. Second, collaboration, which is the most important view of frameworks, is basically a set of global interaction patterns and requires a concise notation for grasping the global flow of control. Algorithm description using the functional programming style is a good candidate for such a representation.

Three research areas can be identified relating to the proposed method; (1) aspect discovery, (2) checking integrity of all the aspect solutions (aspect weaving!!), and (3) mapping pattern.

A hard part of the proposed method is lack of systematic methodology to discover appropriate aspects in a given problem. Since each aspect is accompanied with a specific specification technique, knowing a lot of specification techniques is helpful in identifying aspect in the problem. Accumulating various specification techniques and experience with their application to system development is one of the future directions.

The idea of aspect-centered design is essentially decomposing a whole problem into a set of manageable subproblems to solve individually. Each aspect has its own notation such as functional-style descriptions or message sequence charts, and is amenable to validate separately. On the other hand, the design of the overall system requires to integrate all the solution descriptions. And this is difficult when each aspect solution uses a different notation. Therefore, currently the integration is done only through manual reviews during the design phase of object-oriented framework.

Concerning the issue on the notation, two approach would be possible; (a) establishing relationship between different notations, and (b) providing a homogeneous notation. An example of the first approach is recent activities on assigning rigorous semantics to various UML diagrams [2]. UML, however, is based on the object-orientation and also not adequate for compact algorithm descriptions. For the second approach, formal notation such as algebraic specification language would be a candidate. It is because the language is powerful enough to cover aspect solution descriptions, from functional programming descriptions and abstract datatypes to message sequence charts[10][11]. Further research is necessary for the notation.

[2] The size is not constant because we maintain and update the program code periodically. The information here is only meant to illustrate the system size.

The proposed development process involves designing object-oriented frameworks by using the aspect solutions as the input specification. The design activity is essentially elaboration of the solution, where identifying participant objects and assigning them responsibility are important. Currently the activity (mapping a fragment of the aspect solutions to *object* solution) is carried out in a heuristic manner and does not have any systematic basis. Accumulating *mapping patterns* to have a catalogue would be helpful in the process.

References

[1] OMG : CORBAservices, Trading Object Service Specification (1997).

[2] The precise UML group (http://www.cs.york.ac.uk/puml/)

[3] Beck, K. and Cunningham, W. : A Laboratory for Teaching Object-Oriented Thinking, Proc. OOPSLA'89, pp.1-6 (1989).

[4] Carroll, J.M. (ed.) : *Scenario-Based Design*, John Wiley & Sons 1995.

[5] Deutsch, L.P. : Design Reuse and Frameworks in the Smalltalk-80 Programming System, in *Software Reusability vol.2 (Biggerstaff and Perlis, ed.)*, pp.55-71, ACM Press 1989.

[6] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. : *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley 1994.

[7] Jackson, M. : *Software Requirements & Specifications*, Addison-Wesley 1995.

[8] Johnson, R. : Documenting Frameworks using Patterns, Proc. OOPSLA'92, pp.63-76 (1992).

[9] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. : Aspect-Oriented Programming, Proc. ECOOP'97, pp.220-242 (1997).

[10] Nakajima, S. and Futatsugi, K. : An Object-Oriented Modeling Method for Algebraic Specifications in CafeOBJ, Proc. ICSE'97, pp.34-44 (1997).

[11] Nakajima, S. : Using Algebraic Specification Techniques in Development of Object-Oriented Frameworks, to appear in Proc. World Congress on Formal Methods'99 (1999).

[12] Nakajima, S. : Aspect-Centered Design of Object-Oriented Frameworks, to appear in Trans. IPS Japan.

[13] Pree, W. : Meta Patterns – A Means for Capturing the Essentials of Reusable Object-Oriented Design, Proc. ECOOP'94, pp.150-162 (1994).

[14] Riehle, D. and Gross, T. : Role Model Based Framework Design and Integration, Proc. OOPSLA'98, pp.117-133 (1998).

[15] Rumbaugh, J., Blaha, M., Premeriani, W., Eddy, F., and Lorensen, W. : *Object-Oriented Modeling and Design*, Prentice-Hall 1991.