

# Analyzing Concerns used in Analysis/Design Techniques

*Tomoji Kishi and Natsuko Noda*

*System Design Department, NEC Electron Devices, NEC Corporation*

*e-mail kishi@aj.jp.nec.com & n-noda@cw.jp.nec.com*

## Abstract

In the field of advanced separation of concerns, most researches focus on programming levels. On the other hand, many analysis/design techniques utilize separation of concerns, as it is one of the common techniques in the software engineering field. Though the utilization of these concerns is slightly different from that in programming level techniques dealing with advanced separation of concerns, it is important to recognize what kinds of concerns are used in the analysis/design field, and to clarify the current issues. In this paper, we analyze the concerns in the analysis/design field, observe the current status of separation of concerns, and discuss issues and future research directions.

## 1. CONCERNS IN ANALYSIS/DESIGN TECHNIQUES

Separation of concerns is one of the basic techniques in software engineering fields, and many analysis/design techniques apply the technique. Though their utilization of concerns is not as same as that of advanced separation of concerns [2][3], it is important to analyze the existing concerns and clarify the difference for further researches. The followings are listings of typical concerns used in the analysis/design techniques.

Technique	Concerns	Supplementation
Scenario analysis	Scenario	Sequences of service
Use case analysis	Use case	Services to actors
Structured analysis/design	Function	
Statechart design	States	utilize super state and and/or decomposition
Object-oriented analysis/design (OOA/OOD)	Class Association	Encapsulation of data and functions Association can be class
Multi-task design	Thread	A kind of collaboration, sequence of functions
Database design	Entity	
Design for reusable assets	Function	Hot/frozen spots Functional commonality in time and space
Design pattern	Collaboration	Abstraction of recurring collaboration
Role-based design	Collaboration	Responsibility driven design, CRC method
Architectural design	Architectural view	Abstraction of software structure
AOA [4]/AOD [5]	Collaboration	Concerning quality attributes

**Table 1 Typical Concerns in Analysis/Design Techniques**

## 2. ANALYSIS OF CONCERNS

We analyze concerns listed in the previous chapter from the following points:

- Type of the concerns.
- Type of separation.
- Technique for composing concerns.

### 2.1 Type of Concerns

We have categorized concerns into three types, static structure, collaboration, and state.

- **Static Structure:** In Structured Analysis (SA), they adopt functional decomposition in module design. In the method, it is considered that each module provides functions, and in order to examine adequate module structure, they hierarchically decompose the function into smaller

functions. In this case, they eventually separate the static structure (module structure) of the software. Class in OOA/OOD and entity in database design fall into this category.

- **Collaboration:** In Role-based design, such as CRC method, they separate the collaboration of objects. Though, collaborations correspond to services or functions, they are different from static structures, as collaborations may share same objects and objects may have different role in each collaboration. Use case analysis, multi-task design, and design patterns are included in this category.
- **State:** In statechart design, they hierarchically separate states. State is the special abstraction of the software that has 'reactive ness', and they are neither the static structures nor collaborations.

## 2.2 Type of separation

We have categorized the ways separating them into four types, spatial, temporal, abstraction, and projection.

- **Spatial separation:** Static structure is usually separated spatially.
- **Temporal separation:** Collaborations may share same components and cannot separate spatially. They represent different functionality that is provided at logically different time and situation.
- **Abstraction:** Generalization is an abstraction of commonality of different classes. Patterns are abstraction of commonality of different collaborations.
- **Projection:** In AOA [4] and AOD [5], in order to examine each quality attribute, we pick up important factors that determine the quality attribute, and characterize services in terms of the factors ignoring unrelated factors. This operation clarifies the requirements on corresponding quality attributes. This is a kind of abstraction, similar to projection of database field.

## 2.3 Composition techniques

Though many concerns are used in analysis/design field, they do not always have specific composition techniques.

Firstly, in some cases, composition does not occur or composition is obvious. In functional decomposition of SA, when we are to decompose a module into smaller modules, we carefully define the interfaces between them, namely define the data flows between them utilizing data flow diagram. Therefore, to compose decomposed modules into one is obvious. Secondly, in some cases, separation and composition is examined spirally, and each concern is not examined completely in separate. In object-oriented design, we have to design both static class structure and dynamic state structure. As these two aspects relate each other, we cannot design them separately, and we check the consistency between them iteratively.

However, in some cases, we do consider each concern almost separately. For example, we define collaboration for each service almost separately. In this case, the composition of each collaboration is done by manually add the requirements from each collaboration; namely, if a class has different roles, R1 and R2 in different collaborations, the class must have methods to play the both roles.

In the field of state transition design, there is an interesting technique that is generation of code from state transition definition. There are several way of implementing state diagram as code, such as having transition matrix as data, or having transition as function, and so on. Each implementation has different characteristics (i.e. some are good for memory size, others are good for performance). We can observe that defining the state transition itself (behavior of application), and defining the implementation style (quality attributes of implemented software) is different concern, and they are combined using code generator.

Based on the above observation, we, so far, categorize the composition techniques as follows:

- **Iteration:** iteratively examined and composed.
- **Simple gathering:** usually done by manually not by mechanically.
- **Ad hoc techniques depend on related concerns:** such as code generation from specific design information.

## 3. DISCUSSION

As we have analyzed in the previous chapter, we can say that separation of concerns are one of the common techniques in analysis/design fields, but the utilization is slightly different from that in programming level techniques dealing with advanced separation of concerns [3][6][7]. Roughly speaking,

they separate each concern not so completely, and in many cases, they iterate the process, examining each concern and compose them together, repeatedly. The main reason of the differences is that, in analysis/design field, we have weak formalism; namely, modeling techniques such as UML are basically for human communication. We do have weak tool-support (except drawing or simple data manipulation), and we manually separate and compose concerns. Another reason is that as analysis and basic design is more semantic-dependent and strategic, it is difficult to compose them mechanically.

Of course, there are some situations, in which stricter and more powerful separation of concerns are required. For example, we cannot examine every necessary collaborations at the same time, because systems have to provide many services. Even now, we depict many collaborations using sequence diagrams for each service separately. The composition of these different collaborations has to make each object consistent to every sequence diagrams, by hand.

From architectural point of views [1], it is not so straightforward to make them consistent. In considering collaborations, our prime concern is realizing required functionality. However, collaboration also determines many quality attributes, such as performance and reliability. We also have to be careful to find out the sharing of sub-collaborations between different collaborations, in order to make the system architecture consistent, extensible, and reusable. Current analysis/design techniques have not enough resolution for considering these aspects adequately.

AOA/AOD is one of the approaches to the problem. We separate the requirements on each quality attribute from functionalities, and consider required architectural techniques/styles separately. This technique is based on our observation that many architectural techniques/styles are stored to solve specific issues, such as to improve performance and to make extensibility.

As conclusion, we propose that, in order to make more powerful separation of concern in analysis/design field, we have to:

- Separately examine each quality attribute and functions.
- Separately examine the architectural techniques/styles for required quality attributes.
- Introduce stronger formalism or modeling techniques to handle these concerns.

## REFERENCE

- [1] Bass, L., et.al.: Software Architecture in Practice, Addison-Wesley, 1998.
- [2] Harrison, W. and Ossher, H.: Subject-oriented programming (a critique of pure objects), In Proceedings of the Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA), Sep. 1993.
- [3] Kiczales, G., et.al.: Aspect-Oriented Programming, In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Jun. 1997.
- [4] T.Kishi and N.Noda: Aspect-Oriented Analysis for Product Line Architecture, The First Software Product Line Conference (SPLC1), 2000.
- [5] Noda, N. and Kishi, T.: On Aspect-Oriented Design – An Approach to Designing Quality Attributes -, In proceedings of the 6<sup>th</sup> Asia-Pacific Software Engineering Conference (APSEC '99), Dec. 1999.
- [6] Tarr, P., et.al: *N* Degrees of Separation: Multi-Dimensional Separation of Concerns, Proceedings of the International Conference on Software Engineering (ICSE 21), May 1999
- [7] <http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm>