Software Design based on Architecture Conformance

Tomoji Kishi, Natsuko Noda

Software Design Laboratories, NEC Corporation,

(Igarashi Building) 11-5, Shibaura 2-chome,
Minato-ku, Tokyo 108-8557, JAPAN
TEL +81 3 5476 1089, FAX +81 3 5476 1113,
e-mail kishi@ccs.mt.nec.co.jp

Abstract

Non-functional properties such as performance are determined by many factors and it is quite difficult to understand the relationships between these factors and the properties involved. Instead of completely understanding the nature of non-functional properties, we are examining a design method for finding a proper way to attain the required non-functional properties, in which we actually measure the properties of some modules in the system, and utilize the knowledge gained by the measurement in design activity. Since the properties of modules are easily influenced by its usage and environment, we have to be careful to ensure that the modules show the desired properties. In other words, we can observe that modules impose constraints (e.g. proper usage) on other software modules for the attainment of desired properties. We define the software architecture based on these constraints and introduce the notion of architecture conformance and then discuss how to use conformance in our design activities.

Keywords
software architecture, architecture conformance, non-functional properties, design method, layered system

1. INTRODUCTION

In software development, it is important to meat demands with respect to such non-functional properties as performance. Though studies in the software engineering field have brought us various design techniques, most of them mainly focus on

functional aspects or some specific non-functional properties, especially reusability or extensibility.

We are currently studying a design method in which required non-functional properties are attained based on the knowledge of the properties of software modules gained by actual measurement. In the method, we introduce and apply the concept of software architecture conformance (architecture conformance). In this paper, we outline the idea mainly focusing on one of the non-functional properties, performance.

2. BACKGROUND

We have developed a railway-signaling system (software for an interlocking device) in which safety must be carefully considered [Kis96]. In designing the system, we have to realize not only service functions (interlocking functions) but also safety functions (functions that detect errors and invoke error-handling mechanisms). In order to facilitate the development of safety functions, we have applied layered architecture, in which the lower layer provides basic mechanisms for safety functions.

As performance (such as response time of error detection) is crucial for safety functions, we tried to design the lower layer to show enough performance for the safety of interlocking service. However, we cannot guarantee that the final safety functions (that are realized in the upper layer based on the lower layer) show expected performance, because the performance of the functions in the lower layer is easily influenced by the usage of the functions. For example, the sensitivity of a watchdog timer (a safety function that periodically checks whether or not tasks are activated correctly) is influenced by the correlation between the interval of the monitored task and the interval of the error checking (Fig.1).

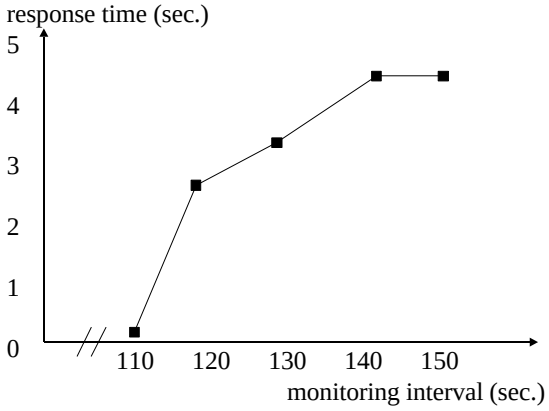

Figure 1. Relationship between Response Time and Monitoring Interval

One of the common techniques to estimate the performance of the system is to measure the performance of some parts of the system and conjecture the final performance. In the above case, we could measure the performance of the functions in the lower layer, and estimate the final performance. However, as we have observed above, relationship between the performance of the part of the system and that of final system is not obvious.

In this paper, we discuss on the software architecture from a performance perspective, and examine the conditions that have to hold, when we expect that a part of the system show certain performance when they put into the entire system. We also introduce the notion “architectural conformance” and outline a design technique based on the conformance.

3. NON-FUNCTIONAL PROPERTIES AND ARCHITECTURE

3.1 Non-functional Properties

Non-functional properties are features of a system not covered by its functional description [Bus96]. Though non-functional properties include a variety of properties, in this paper we focus on such run-time properties as performance. As our primary concern is how the run-time non-functional properties relate to the services provided by the system, we will state "performance of the system with respect to the service" and "performance of the service".

3.2 Serviceable Module Set

In order to clarify the relationship between a service and the modules that provide the service, we define a "serviceable module set". A serviceable module set for service S is a set of modules that are functionally necessary to realize service S. In other words, service S is provided by the collaboration of these modules. In this paper, M(S) denotes a serviceable module set for S.

3.3 Architecture Determined by Serviceable Module Sets

Since the properties of software modules are easily influenced, we have to use the modules carefully in order to ensure that they show the desired properties. Observing the situation from the reverse side, modules impose constraints on other software modules for the attainment of desired properties. Based on these constraints, we define the software architecture, in which serviceable module sets for services and for sub-services are considered as components, and the constraints imposed on other serviceable module sets as connectors.

Consider the situation in which service S is realized using sub-services SL*i* (*i* =1,2,…m), and S is used by the realization of upper services SU*j* (*j*=1,2,…n). In this case, the architecture determined by M(S) is defined as follows:

Architecture determined by M(S) = {
(a1) intended non-functional properties of M(S),
(a2) structural constraints imposed on all M(SU*j*) (*j*=1,2,…n),
(a3) conditions imposed on all M(SU*j*) (*j*=1,2,…n), to achieve (a1),
(a4) expected non-functional properties of each M(SL*i*) (*i* =1,2,…m)
}

(a1) is the intended properties, such as sensitivity of error detection.

(a2) is the structural constraints imposed on modules that use the service S, namely all M(SU*j*) (*j*=1,2,…n). In order to provide service S to some modules, there must be interactions between the modules and M(S). Although there may be multiple ways of interacting to provide service S, the non-functional properties of S may be changed by the way they are used. The constraints include: (1) constraints on static structure (i.e. constraints on the roles of each module participates in the interaction), and (2) constraints on dynamic structure (i.e. constraints on the partial ordering of each interactions, such as protocol or calling sequences) [Gam95].

(a3) is conditions that are necessary to be satisfied by M(SU*j*) (*j*=1,2,…n) for the expected non-functional properties (a1) to be attained. In the case of error detecting functions, conditions on monitoring intervals correspond to these conditions. Typically, these conditions are determined by the data or knowledge gained by actual measurement of the properties of S.

(a4) is the expected properties assigned to each M(SL*i*) (*i* =1,2,…m). As S is realized using SL*i* (*i* =1,2,…m), attainment of (a1) depends on whether or not each M(SL*i*) (*i* =1,2,…m) can really attain these expected properties.

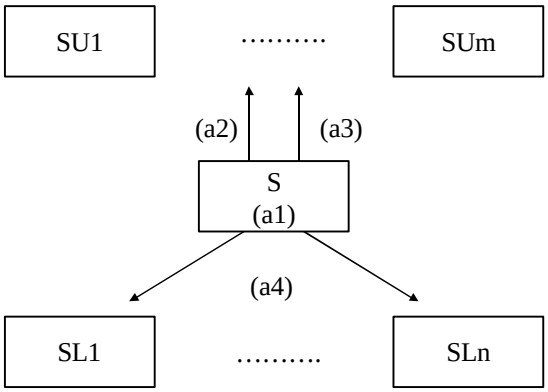

Figure 2. Architecture Defined by a Serviceable Module Set

3.4 Architecture Conformance

We can consider that designing non-functional properties means designing modules in which architecture determined by serviceable module sets for every service is consistently systematized. In order to discuss the design, we introduce the notion of "architecture conformance". Because each serviceable module set imposes constraints on other serviceable module sets, the conformance is defined for each serviceable module set as the conformity to each constraint imposed on the serviceable module set.

Architecture conformance of serviceable module set for S is defined as follows:

Architecture Conformance of M(S) = {
(c1) conformity to each (a2) imposed by M(SL*i*) (*i* =1,2,…m),
(c2) conformity to each (a3) imposed by M(SL*i*) (*i* =1,2,…m),
(c3) consistency between (a1) and (a4) as required by M(SU*j*) (*j*=1,2,…n)
}

(c1) represents whether M(S) keeps to each structural constraint imposed by M(SL*i*) (*i* =1,2,…m).

(c2) represents whether M(S) keeps to constraints imposed by M(SL*i*) that are necessary to attain the intended properties of each SL*i* (*i* =1,2,…m).

(c3) represents that the intended properties of M(S) are consistent with the expectations of M(SU*j*) (*j*=1,2,…n).

4. DESIGN BASED ON ARCHITECTURAL CONFORMANCE

In this chapter, we outline the design method using an example.

4.1 Layered Architecture and Serviceable Module Sets

In this example, the software has to provide communication service S1. As reliability is required of the service, we use error detecting functions to invoke an error-handling mechanism if an error occurs. In order to realize S1, we use sub-service S11 (communication service) and sub-service S12 (error detecting service). We use services provided by real-time OS (RTOS), S111 (basic communication service), S121 (task service) and S122 (interrupt service), to realize the service S11 and S12. RTOS is an existing platform and is not the design target.

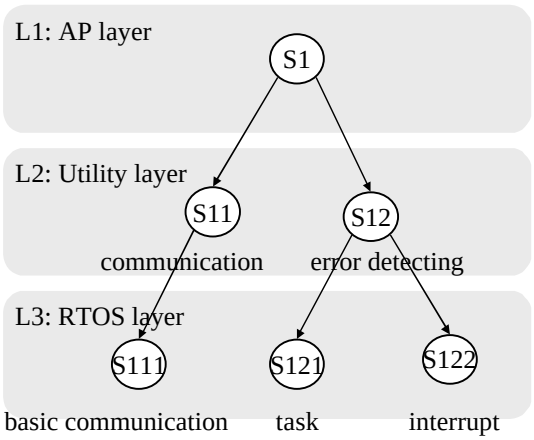

Figure 3. Serviceable Module Set and Layers.

In the figure, the circle labeled *s* indicates m(*s*), modules in serviceable module set for *s* belong to the layer that provide the service *s*. Here, M(S1) includes m(S1), m(S11), m(S12), m(S111), m(S121) and m(S122), while M(S12) includes m(S12), m(S121) and m(S122). When we design L1 (AP layer), we have to design m(S1), and when we design L2 (utility layer), we have to design m(S11) and m(S12).

4.2 Constraints between Layers

As discussed in 3.3, we can define the constraints between the layers, in which (A1), (A2), and (A3) of a layer correspond to (a1), (a2), (a3), and (a4) of a serviceable module set (Table 1).

Table 1. Constraints between Layers.

L1 (AP layer)	(A1) intended properties	(a1) of M(S1)
	(A3) to L2 (Utility layer)	(a4) of M(S1)
L2 (Utility layer)	(A1) intended properties	(a1) of M(S11) (a1) of M(S12)
	(A2) to L1 (AP layer)	(a2) and (a3) of M(S11) (a2) and (a3) of M(S12)
	(A3) to L3 (RTOS layer)	(a4) of M(S11) (a4) of M(S12)
L3 (RTOS layer)	(A1) intended properties	(a1) of M(S111) (a1) of M(S121) (a1) of M(S122)
	(A2) to L2 (Utility layer)	(a2) and (a3) of M(S111) (a2) and (a3) of M(S121) (a2) and (a3) of M(S122)

Since L1 (AP layer) is a topmost layer of the system to be designed, we do not consider the (A2) conditions ((a2) and (a3) of M(S1)). If the client of this software is also the software, these conditions may be the usage of the software. On the other hand, L3 is the platform of our design, we do not consider the (A3) conditions ((a4) of M(S111), (a4) of M(S121) and (a4) of M(S122)).

If we expect that the sensitivity of error detection in S12 is less than 10 ms, the upper layer have to keep the conditions (a2) of M(S12) (structural conditions defined as Fig.4), and (a3) of M(S13) that may be defined as “the interval of monitored task must be 100 ms”.

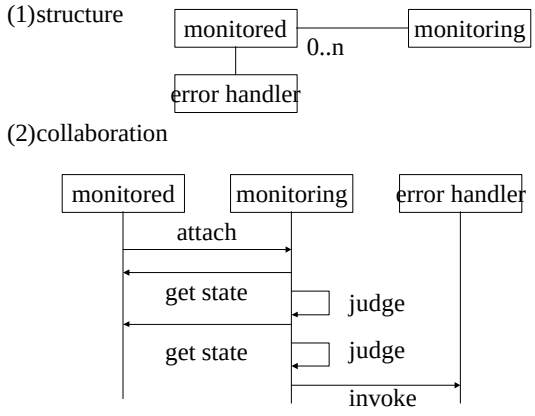

Figure 4. Structural Constraints of S12.

In the design process, we try to clarify these conditions and make each set of modules to keep these conditions, and eventually we may expect that the final system shows the desired properties.

5.CONCLUSIONS

In this paper, we have examined the method for designing non-functional properties utilizing the notion of architecture conformance. How to define conditions, especially (a3) conditions, and how to test the conformance needs more elaboration.

6. REFERENCES

[Bus96] Buschmann, F., et al., Pattern-Oriented Software Architecture - A System of Patterns, Wiley, (1996).
[Cle96] Clements, P.C., et al., Software Architecture: An Executive Overview, Component-Based Software Engineering - Selected Papers from the Software Engineering Institute, IEEE (1996).
[Gam95] Gamma, E. et al., Design Patterns: Elements of Reusable Object-Oriented Design, Addison-Wesley, Reading, Mass., (1995).
[Kis96] Kishi, T., et al., Development of Safety Software based on Software Architecture, SIGSE, Vol.96, No.112, IPSJ, 1996 (in Japanese).
[Sha96] Shaw, M., et al., Software Architecture, Perspectives on an Emerging Discipline, Prentice-Hall, (1996).