Reuse Software Architecture through Dynamic Composition

Liang ZaoQing[1], Ying Shi[1,2], Cao Rongzeng[3], Jia XiangYang[1], and Zhang Tao[1]
1 State Key Lab of Software Engineer, Wuhan University, Wuhan,430072,
P.R. China, xt_lzq@126.com,
WWW home page: http://www.sklse.org.cn
2 ComputerScienceSchool,Wuhan University, Wuhan, Hubei,430071,
P.R. China, yingshi@whu.edu.cn,
WWW home page: http://www.cs.whu.edu.cn
3 Business Optimization Team, IBM China Research Laboratory, Beijing,
P.R. China, caorongz@cn.ibm.com,
WWW home page: http://www.ibm.com/cn/

Abstract: With the mature technology of software component and software architecture, software reuse has achieved significant progress. However, dynamic reusing software architecture such as in the condition of changing requirements under non-stopping condition at runtime remains a challenge. Reflection is the ability that a system performs the computing about itself. Based on reflection, in this paper, a reflective architecture RSA4RDC was constructed to support architecture dynamic reuse, and also the design framework, primitives, and process were illustrated.

1 Introduction

Software reuse refers to the process during similar or identical software products used repeatedly in multiple software producing or process. With the emergence of software architecture, the component technology came out and become mature, software reuse has achieved significant progress.

However, dynamic reusing of coarse-grained elements, such as software architecture, still does not have much. Traditionally, reusing software architecture is done at design phase: according to the requirement of the system, designers reuse the architectural constituents by incorporating the existing elements into the architecture. However, with respect to the systems which are non-stopping, having change

* Research supported by the National Nature Science Foundation of P.R. China under No. 60473066 and Young Outstanding Talent Foundation of Hubei Province, P.R. China under No. 2003ABB004.

Please use the following format when citing this chapter:

ZaoQing, L., Shi, Y., Rongzeng, C., XiangYang, J., Tao, Z., 2006, in International Federation for Information Processing, Volume 205, Research and Practical Issues of Enterprise Information Systems, eds. Tjoa, A.M., Xu, L., Chaudhry, S., (Boston:Springer), pp.297-305.

requirements, and have to evolve continuously [1], the design phase and running phase are overlapping. Traditional reuse method is insufficient to deal with such condition.

The common way to deal with the system with continuous evolving architecture is to define the possible evolving strategy at the design phase of architecture. At run-time, the system will execute the architecture evolution automatically while the evolving conditions are satisfied. But such approach could not be used in the system with changing requirement at run-time. Because the changing requirement is not predictable, it could not be considered at design phase.

Focus on these issues, this paper proposed a design and reuse architecture method at run-time, which called reuse software through dynamic composition. The research work was based on reflection mechanism; it provided an approach to reuse architecture constituents by manipulating architecture. Based on reflective, it uses the meta-information about the architecture to provide the designers and tools with the operation and process of reusing architecture.

In this article, we constructed architecture---RSA4RDC (Reflective Software Architecture for Reusing SA through Dynamic Composition) to support dynamic composition reuse of the architecture at design time, provided a framework and process of reuse through dynamic composition in RSA4RDC, in this article, a simple case was given to illustrate on how to use the framework and process..

This paper is organized as follows: Section 2 describes the model of reflective architecture for reuse through dynamic composition, specifies the elements of architecture. Section 3 illustrates the framework, the operation primitives, and process for reuse. In section 4, an example is given to illustrate how to use the framework and process to gain a new system in runtime. And in section 5, a conclusion is given by comparing some relative research work.

2 RSA4RDC

2.1 Reflective Software Architecture for reuse

In order to redesign the architecture at run-time, we need explicit data structure to represent and control of the run-time architecture of the system. Explicitly representing the design information of architecture could support the architecture design at run-time. For this end, our approach is to construct a reflective architecture for reuse through dynamic composition: RSA4RDC.

Reflection is the ability that a system performs the computing about itself [2]. In a software system, reflection has enabled a program to access the internal structure and behavior of itself, and to manipulate its own structure and modify its own behavior in runtime. Thus software system could adapt itself to the changing conditions, which has more flexibility for the system.

Introduction reflection mechanism into architecture, a reflective architecture which could perform the computing about itself is constructed [3 - 5]. RSA4RDC is a kind of reflective architecture that used for dynamic reuse; it split a system into two parts explicitly: one is base-level architecture which executes business process; the

other is meta-level architecture which monitors base-level architecture. By using the reflection, the whole software system could dynamically changes at run-time. So we also could design the system through dynamic composition architecture constitute. Figure 1 represents the structure of RSA4RDC.

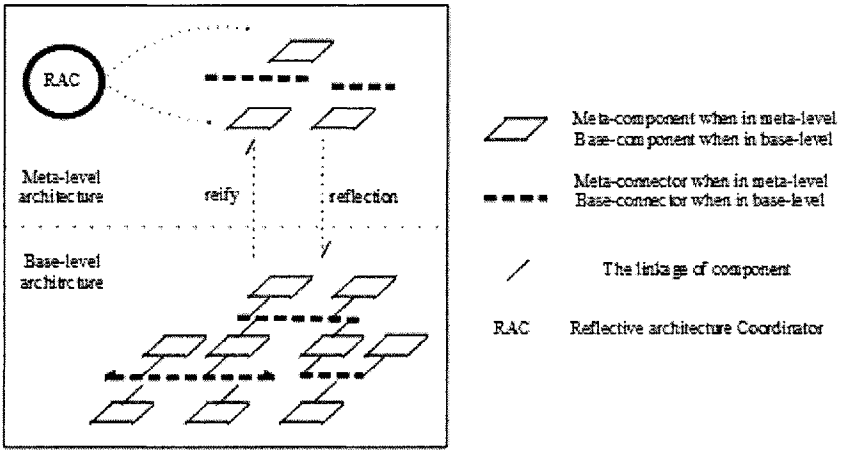

Fig. 1. The structure of RSA4RDC

Comparing to the non-reflective architecture, component/connector in non-reflective architecture has their counterpart in the base-level of RSA4RDC, and meta-component/meta-connector in meta-level of RSA4RDC. The meta-information of the base-level architecture could support the management and manipulation of the reuse process of base-level architecture. Therefore at the design-time, by separating the concerns, it could help us to do the design at different abstractions and simplify the design and promote reusability.

2.2 RSA4RDC base-level architecture

The base-level architecture is the space of implement business requirement, which is about the assignment of responsibility of the system. It is composed of components and connectors, component is the basic unit to implement the functions, while connector is an explicit entity which binds the component together and plays the role of coordination between components [6].

2.3 RSA4RDC meta-level architecture

The meta-level architecture is the space of monitor base-level architecture, which could obtain the base-level architecture information such as constitute, topology, linkage and runtime information, and also could new, delete, replace of constitute. It provides designers with the information when they are reusing the architecture and

its constituents at architectural design phase, and also provide architect and tools with a standard interface to use base-level architecture.

The meta-level architecture is composed of meta-component, meta-connector and RAC (Reflective Architecture Coordinator), meta-component and meta-connector are composed of two parts: first, they represent the meta-information of the elements in base-level. The meta-information includes basic meta-information and purpose-specific meta-information. (1) Basic meta-information: describing the basic information about elements in base-level, such as the id, name, methods etc of component and connector. (2) Purpose-specific meta-information (such as reuse, evolution, refinement): in meta-level architecture, elements in meta-level can not only monitor the run-time properties of the elements in base-level, but also manipulate the adding, deleting and replacing of the elements in base-level, thus controlling the evolution, refinement and reuse of the system. Towards these purposes, extended meta-information is needed, these purpose-specify meta-information should be represent in them. Second, change the base level architecture such as new, delete and replace of them based on some strategy.

RAC is used to describe the reflect/reify strategy between base-level and meta-level: the base-level architecture and meta-level architecture in the reflective architecture are mutually interactive and interacted, and their interaction should be guided by certain strategy, such as under what condition the meta-component sends the modification or replace operation to base-level. It receives the request that modifying the design at run-time from the user, and then modifies the meta-level architecture, and finally reflects the resulting meta-level to the base-level architecture.

3 The Framework and Process for Reuse through Dynamic Composition

In order to meet the requirements such as non-stopping, changing requirements and continuously evolving, based on reflection, we propose a framework and process for reuse a RSA4RDC through dynamic composition of large coarse element such as component and connector, Figure 2 is design framework for dynamical reuse.

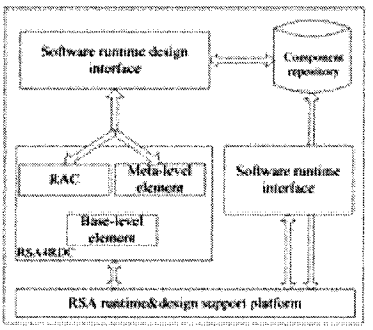

Fig. 2. Dynamic Reuse Frameworks

3.1 Reuse Design Framework

This reuse design framework differs with common software because that by using reflection, software is designed at run-time, and the design is taking effect at run-time and finally verified.

This framework includes following parts: (1) a running interface of the program: it can be executed directly during the reuse process, and the resulting architecture can be verified in the running interface, (2) a design tool: through this tool, designer can design the running architecture, (3) reflective architecture of software system: it is RSA4RDC designed by design tool and verified by running interface; it is the output of the design/verification activity, (4) a component repository: it provides reusable components and connectors, as well as the design information of these elements, (5) a runtime and design support platform: it provides a infrastructure for the design and running of RSA4RDC system; translates the operation of the design into the RSA4RDC and takes effect, and then executes the new system.

3.2 Reuse Design Primitive

In the reuse design framework, we define some primitives for design. Below are the main design primitives for meta-architecture for add, delete, replace of meta-elements:

(1) find(element, condition): the operation is to retrieval component or connector which conforms to the input condition from component repository. "element" is component or connect, "condition" prefer to the condition that element satisfy, and this operation return all results that coincidence with "condition".

(2) add(element): This operation is to add meta-components and meta-connectors into RSA4RDC system.

(3) delete(element): This operation is to delete meta-component or meta-connector that no long used from RSA4RDC system, after this operation, the argument "element" will no long exist in system, and the link to other element will be unlinked.

(4) replace(oldelement, newelement): This operation is used to replace an oldelement with newelement.

RSA4RDC meta-level architecture monitor base-level component, connector and the linkage between them, below are the main design primitives for monitor the base-element of meta elements:

(5) instance: instances a component or connector, this means add new element in base-level.

(6) delete: delete an element not used in base-level architecture.

(7) changeMethod: modify element method in base-level.

(8) link: link an element to another that existed in base-level.

(9) unlink: unlink a linkage that existed in the base-level.

(10) addStrategies():This operation is to add evolution strategy according to business requirement. A RSA4RDC is a system that has evolution ability according to evolution strategies, the strategies can be predefined when the system first design,

and also can be added when the system is running because the needs of business requirement. For example, in a CS system, when concurrency Clients number exceeds a given amount, the QoS (Quality of Service) is decreased. In order to increase QoS, we can add a strategy below to gain the aim.

While (S.activestate=T and S.responsetime>10s) replace(S, S_{fast})

(11) execute: Only through this operation can all design activities above take into effect. After this, we can get a new redesign RSA4RDC system.

3.3 Reuse Design Process

In order to get a new system in runtime, designer adds, deletes, replaces components, connectors, links and evolution strategies in design interface, the process mainly includes:

(1) Retrieval reusable element. It is the activity that retrieval and obtains component and connector needed from component repository through tools in design time.

(2) composition/modify/verify software system.

In this project, we develop a simple platform that provides dynamic composition design in runtime. Figure 3 is a screen-snapshot of the design interface: in this platform, we could add/delete/replace meta-component, meta-connector, define the meta-relation of elements between base-level and meta-level, and the linkage of components and connectors, add the evolution strategies of software system. After finish the design of system, click the execute button, the design platform can execute the design. And the system can dynamic take into effect in runtime. In section 5, we use the interface to design a new CS system.

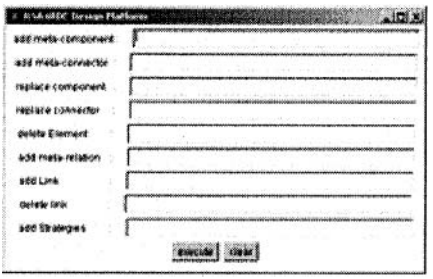

Fig. 3. Reuse Design Snapshot

4 Case study

Considering a CS system, we define C is a client that asks server S to calculate an input formula. And we let the origin system is a simple CS system which can only do addition and subtraction operations. User requests calculation form C and S provide

the calculation service, and send answer back to C. Figure 4 and Figure 5 is the runtime snapshot of the simple CS system.

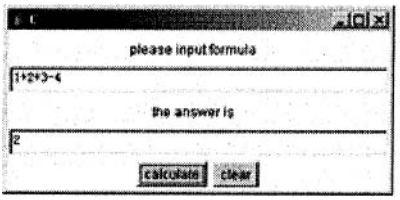

Fig. 4. C Component Runtime Interface

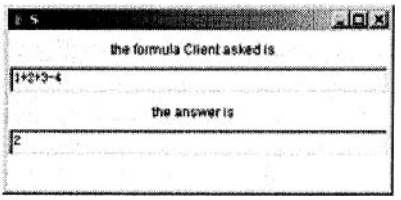

Fig. 5. S Component Runtime Interface

In non-stopping runtime, customer requests to add the security communication between S and C component, and he also wants server providing multiplication and division function. Figure 6 is the CS base-level architecture transition graphic.

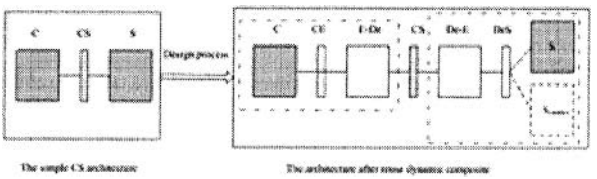

Fig. 6. The CS Base-Level Architecture Transition

As the security, we add an E-De component which can encrypt the request formulas, and decipher the answer which server have calculated, a De-E component which decipher the request formulas which encrypted by E-De, and encrypt the answer which S or $S_{complex}$ component calculate. And we also need add CE and DeS connector which connect C, E-De and De-E, DeS respectively. As the complex calculation includes multiplication and division operations request includes multiplication and division, we need to add $S_{complex}$ component to the system. And also we need to add evolution strategy that replace S component with $S_{complex}$ when the input formulas includes multiplication and division operations. In order to attain the new system, In RSA4RDC design platform, the design activities is below:

In add meta-component input box, we input metaE-De, metaDe-E;

In add meta-connector input box, we input metaCE, metaDeS;

In add meta-relation input box, we input CE, metaCE; E-De, metaE-De; De-E, metaDe-E; DeS, metaDeS;

In add Link input box, we input C, CE; CE, E-De; E-De, CS; CS De-E; De-E, DeS, DeS, S;

In add delete link input box, we input C, CS; CS, S;

In add Strategies input box, we input "if(C.formulas.includes(multiple) or C.formulas.includes(division), then replace(S, $S_{complex}$) ".

At last, we click the execute button, and the system runtime&design platform can take the design into effect in runtime: through the reflection mechanism, meta-

component and meta-connector can reflect into base-level, and create new component E-De, De-E, $S_{complex}$ and connector CE, DeS, the link can change into the new linkage, in the end, the CS software architecture changes into a new CS system with evolution ability and security function.

5 Related works

Dynamic software architecture is a hot area in software engineer academic and industrial community. In this field, there are some relative research works below:

Pila [7] is a reflective ADL with dynamic ability, it enforces compositionality. And it has been designed with formal languages based on process algebras, especially CCS [8], Pilar defines s set of primitives to make use of the reflective capabilities. The basic primitives are Avatar(α), Reify(ρ), Destroy(δ). In Pilar, connectors are just as meta-level components. And a component is defined as either a collection of interfaces or a configuration of component instances (composite component). Each interface is segmented in ports, defined as the interaction points published by the component. Each component has at least one interface.

ArchWare [9] applies an innovative approach to the architecture-centric model-driven engineering of software systems that sets the "ability to evolve" as its central characteristic. Evolution arises in response to changes to requirements as well as to run-time feedback. In order to achieve this goal, ArchWare develops an integrated set of architecture-centric languages, such as ArchWare Architecture description language, Architecture Analysis Language, Architecture Refine Language. And it also develops an IDE, which includes support formal architecture description, analysis and refine, and also code generation.

Darwin [10] is a language for describing hierarchic configurative structures. It is used to describe and control the configuration structure of a program. It provides the ability of dynamic aspects of configuration as well as static aspects. A Darwin architecture can be used both to compose component implementations to build a system and/or compose LTS (labelled transition systems) specifications of component behaviour for system property analysis.

PKUAS [11] is design and implementation for a reflective component operating platform, it based on componentized structure, PKUAS introduces SA as the global view of the whole reflective system. As a J2EE-compliant application server, it can reflect both the underlying platform and EJB components. It provides management tool and evaluates ability through reflective middlewares.

Comparing to these works above, differently, we mainly concern on software architecture dynamic reuse design and analysis. Through define RSA4RDC---a reflective architecture which can provide dynamic composition ability, combining with reflective mechanism for Reflective software architecture, we can retrieval reusable element from repository, composite them together and change evolution strategies from a running system dynamically. By this mean, we provide a solve method for designing and analysis large, complex, non-stopping and continual requirement changes system.

References:

1. CERT 2004 Research Annual Report, http://www.cert.ort/research. CERT Software Engineering Institute, Carnegie Mellon University.
2. P. Maes, Concepts and experiments in computational reflection, In the Proceedings of the 2nd Annual Conference on object-Oriented Programming Systems, Languages and Applications (OOPSLA '87), (Orlando, FL, Oct. 1987).
3. G. T. Sullivan, Aspect-Oriented Programming Using Reflection and Metaobject Protocols, *Communications of the ACM* **44**(10), 95-99 (2001).
4. F. Tisato, A. Savigni, W. Cazzola, and A. Sosio , Architectural Reflection Realising Software Architectures via Reflective Activities, EDO 2000, LNCS 1999, pp. 102-115, Springer-Verlag Berlin Heidelberg 2001.
5. M. Ancona, W. Cazzola, G. Dodero, and V. Gianuzzi, Channel Reification: A Reflective Model for Distributed Computation, In the Proceedings of IEEE International Performance Computing, and Communication Conference (IPCCC' 98) 98CH36191, pp. 32-36, Phoenix, Arizona, USA.
6. M. Shaw and D. Garlan, *Software Architecture Perspectives on an Emerging Discipline*, (Prentice-Hall, 1996).
7. C. Cuesta, P. de la Fuente, and M. Barrio-Solárzano, Dynamic Coordination Architecture through the use of Reflection., In the Proceedings of the ACM 2001. Symposium on Applied computing, Las Vegas, Nevada, USA, pp: 134 - 140, 2001.
8. R. Milner, *A Calculus of Communicating Systems* (Springer-Verlag, Berlin Heidelberg New York 92/1980).
9. F. Oquendo, B. Warboys, R. Morrison, R. Dindeleux, F. Gallo, H. Garavel, and C. Occhipinti, ARCHWARE: Architecting Evolvable Software, EWSA 2004, LNCS 3047, (Springer-Verlag Berlin Heidelberg, 2004), pp. 257–271.
10. J. Magee , N. Dulay , S. Eisenbach ,and J. Kramer, Specifying Distributed Software Architectures, Proceedings of the 5th European Software Engineering Conference, p.137-153, September 25-28, 1995.
11. G. Huang, Q-X. Wang, H. Mei, and F-Q. Yang, Research on Architecture-Based Reflective Middleware, *Journal of Software* **14**(11), 1819-1826 (2003).