Runtime Software Architecture Based Software Online Evolution

Qianxiang Wang, Gang Huang, Junrong Shen, Hong Mei, Fuqing Yang
Institute of Software, School of Electronics Engineering and Computer Science
Peking University 100871 P.R.China.
{wqx, huanggang, meih}@cs.pku.edu.cn

Abstract

Runtime environment of software are becoming more and more dynamic and changeable, while pervasive computing and web services further this situation. Software systems are not only becoming larger, more complex, and also more rigid, which make it difficult to evolve software. This paper focuses on online evolution, more exactly, how to make online evolution process convenient and smart, with help of Runtime Software Architecture (RSA). Following issues are discussed in this paper: types of software environment changes, the incarnation of RSA, retrieval and manipulation of RSA, the relation between RSA and the runtime system, and a visual tool to show RSA, and make evolution process more easy and intuitionist.

Key Words: Runtime Software Architecture, Online Evolution.

1. Introduction

Internet, wireless network, web services, pervasive computing, grid computing, and so on and so forth, all of these concepts and technologies have made runtime environment of software systems much more open and changeable. Moreover, from the managerial perspective, because of the stricter requirements of return on investment and time-to-market, the incremental development paradigm becomes more prevalent. Software systems are becoming larger, more complex, and more rigid, that means, more difficult to control, maintain and evolve than ever before. Effective approaches should be developed to help to cope with changeability of software environment.

Although many approaches have been developed to make software evolution convenient, such as design pattern [4], AOP [12], most of them use stop-adjust-start mode. Few approaches focused on online evolution. It's time to give much effort on online evolution, with the coming age of network computing.

Software architecture is defined as the organizational structure of a software system including components, connections, constraints, and rationale. SA not only guides the development of software as a blueprint, but also plays an important role in at least six aspects of

software development: understanding, reuse, construction, evolution, analysis, and management [10]. Therefore, SA can exist in different phases of the whole software lifecycle [3]. Runtime Software Architecture (RSA) describes SA at runtime, which treats SA at design phase (DSA) as the centric and basic role, but adds much more runtime information to it. RSA can help capture the system's runtime status and feed back to make the running system more adaptive and controllable through certain mechanism. In other words, RSA can help software engineers and maintainers to understand, control, adapt and evolve the system, while keep the system running.

The rest of this paper is organized as follows: Section 2 analyzes types of software environment changing. Section 3 presents the RSA by defining what information is encapsulated in RSA and related operations. Section 4 describes the evolution process, especially on how to retrieve RSA from the runtime system and to change the states of the runtime system through manipulating RSA. Section 5 discusses how PKUAS supports the RSA-based evolution. Section 6 summarizes the contributions and identifies the future work of the paper.

2. Types of Changes

Environment changes that have impact to software can be divided into two kinds: changes on space dimension and changes on time dimension. The former focuses on variants between different software products, which are developed for different clients. Domain engineering, product line are main technologies that help software solve this kind changes [11]. The latter focuses on variants that same software must adapt with the time go ahead. Software maintenance, software evolution, and software adaptation are technologies that solve this kind of changes.

For software evolution, today we have much knowledge of what will change. The early research results were concluded by Swanson [13], which includes corrective, adaptive, and perfective. Recently, types of software evolutions have been proposed by Chapin [1], who lists 188 activities that belong to software maintenance and evolution. Some activities refer to

environment changes, some activities refer to architecture change, and some activities do not change software. The first classification is based on the results of evolution while the latter focus on the activities; both of them lack the consideration of the causes of evolution. Therefore, after carefully observed causes of software evolution, this paper proposed classifies of changes as figure 1. First, environment changes can be divided into two types: customer requirement and runtime environment. Then, changes of runtime environment can be divided into runtime platform changing and interactive context changing.

Software Environment		
Customer Requirement	Runtime Environment	
	Runtime Platform	Interactive Context
Process Function Constraints	OS DBMS Middleware Resource	Service-specific Context Client-specific Context

Figure 1. Types of Changes

(1) Customer Requirement Changing

Customer requirement changing includes:

- Process changing, i.e. adjusting of process flow, process unit deployment, etc.
- Function changing, i.e. function adding, removing, altering, correcting, etc.
- Constraints changing, i.e. security, reliability, UI, consistency, performance, adaptive data, etc.

(2) Runtime Platform Changing

Runtime platform changing includes:

- Changing of operating system, which may includes hardware changing.
- Changing of DBMS.
- Changing of middleware.
- Changing of resource, such as hardware increase, failure, etc.

Although some customer requirements involve runtime environment, we prefer customer requirement as features that decided mainly by application, little affected by runtime environment, while runtime platform belongs to features of application's runtime environment.

(3) Interactive Context Changing

Interactive context changing includes context of two interacting ends: service-specific context and client-specific context.

- Service-specific context includes: service related data, communication bandwidth, physical location, number of connected clients, communication cost, etc;.
- Client-specific context includes: communication protocol, role, transaction, reply time, etc.

Some customer requirements also involve some features about interactive context. We refer interactive context to features that can hardly **forecast** by customer.

Custom requirements may cause the change of DSA which may **subsequently** changes RSA. Runtime platform and interactive Context are where RSA is retrieved from. Therefore, RSA is affected by all of the above changes. And with its reflection mechanism, RSA can, maybe partly, help to control the above three change types.

3. Approach Overview

3.1 What is RSA

In order to manage the running application system, we capture its main aspects that form the runtime model by RSA. There are three fundamental issues on RSA:

Figure 2. RSA and DSA

- The information encapsulated in RSA is retrieved from some part of the states of the runtime system. And in order to represent and manipulated RSA at runtime, RSA should be **incarnated** as a runtime entity, i.e. it should consist of a set of meta objects that encapsulate all of architectural information of the runtime system and allow access to such data at runtime.
- RSA is not only detected from the running system but also derived from the design and implementation information of the application, especially the SA in design phase (DSA). The initialization and starting of a certain application means the configuration and instantiate of the software release. Moreover, the achievements of and experiences with DSA, such as formal notations, graphical representation, reason and validation, may still play an important role at runtime. Thus RSA should be able to be transformed from DSA, and vice versa.
- The introduction of RSA is not only employed to model the running application system, we should use RSA to help manage and evolve the running system as well. In other words, the modification and configuration of the RSA model will lead to modify and configure corresponding states of the running system and thus more or less change the functionality or quality of the runtime system. So there exists causal connection between RSA and runtime system.

On the basis of such analysis on RSA, our approach of software online evolution should cope with 3 problems:

1) Incarnation of RSA: RSA plays a centric and basic role in our approach, so build a feasible model of RSA is very important. We encapsulate architectural information in a set of objects and investigate operations on this information.

2) Evolution implementation: Based on RSA, our evolution process contains following steps: RSA retrieval, RSA modification and reconfiguration, etc.

3) Supporting Infrastructure: Most RSA-based evolution and adaptation approaches should only be applied into proprietary systems. In other words, the approach needs the support of system software. In this paper, we will introduce our approach with the help of PKUAS, one J2EE application server that is developed by authors.

3.2 What's inside RSA

The incarnation of RSA includes the following entities: components, connections, and constraints.

- **Components.** A component can be defined in multi-aspects. The Player aspect defines interfaces that incorporate the type of the player (provide or request) and several method specifications, i.e., Method. The Attribute aspect designates the attributes the component will use in the interaction with others. Most of the excess architectural information of RSA over DSA is represented in the property aspects of the component, connection and architecture. And the semantic aspects use formal methods to model, or, at least, use natural languages to describe the behaviors and features of the elements.

- **Connections:** A Connection describes the relationship of two components, such as the communication style (by Local? by Procedure Call? or by message Transfer?), the communication protocol (IIOP? JRMP? SOAP?), and other attributes of connections (using SSL?).

- **Constraints:** The constraints describe additional information of the component, such as variety scope of: component attributes, security and transaction attributes of component interface's operation, throughout limit, client connection numbers, etc.

Compared with DSA, RSA increase much information of software system. Usually these kinds of information can not been decided during design time. For components, these kinds of information may include: number of connected clients, size of pooled instance, etc. For connection, these kinds of information may include: communication protocol, response time, etc. For constraints, these kinds of information may include: security attribute of some component method, transaction attribute of some component method, etc.

3.3 Operations on RSA

According to the characteristics of the encapsulated data, the operations provided by RSA can be categorized into the following:

- Access to data: All of the data encapsulated in RSA can be accessed through the operations named as "get<Data>", e.g..

- Modification of the data without changing the state of the runtime system: Some information in RSA is derived from DSA to keep consistency and cannot be retrieved from the runtime system. The maintainers can modify such data to make RSA more concrete and accurate.

- Modification of the data that leads to changes of the state of the runtime system: Most of the data encapsulated in RSA is causally connected with the state and behavior of the runtime systems. For example, when changing the name of the implementation class of a component as ComponentDef.properties.put ("ejb-class","..."), the current implementation class will be replaced by the new one identified by the second parameter.

4. Detailed Approach

As shown above, the RSA consists of components, connections and constraints of software, so the retrieval and manipulation of RSA also have 3 correspondent parts. And both of them are implemented based on the reflective ability of PKUAS.

4.1 The Structure of AS

Currently, PKUAS implements all of the functionalities specified in [4]. The structure of PKUAS involves:

- **Application.** One application usually composes of multiple modules, such as procedures, classes, or components. Modules interact with each other, by way of method invocation or message interchanging. Application software runs over system software such as an Operating System, DBMS. A component is an independent module that undertakes some specific functions in software systems. Although most people focus on component implementation, component specification is another indispensable part of a component [10]. Component implementation is the code that implements the functions of a component. Component specification describes how to assemble, use, and manage the component. The application that runs on an application server can contain multiple components. The relationship between components and applications is similar with the relationship between employees and enterprise.

● **Component Container.** Component container is the main mechanism in an application server that supports component technology. Container gives component independent running environment, which makes development and management of components more feasible and convenient. As one middle layer located between component and component's client, container is responsible for the connection between component and client, life cycle management of components, coordination of transaction for components, maintenance of component **persistence**, and identification authentication of clients, etc., while client is unaware of the existence of containers. Some of above functions are implemented with support of common services, and thus developers can concentrate their **attention** to the main functions of applications.

● **Common service.** The concept of service becomes mature with the development of the client-server computing model. Belonging to the system level, common services are used to support common features of different applications. Common services undertake most of application server's functions. For example, communication service is responsible for locating the server, transaction service is responsible for assuring system's consistence, and security service is responsible for authenticating the client and controlling the access to server, etc.

In the above entities, EJBs are components in RSA, and the container holds most of the states related to the Components. The communication service provides Connections and holds most of the states related to the Connections. The services except communication hold some states related to the Constraints, such as method permission and transaction attribute.

4.2 Retrieval of RSA

The retrieval of RSA is performed during following activities: application deployment, application un-deployment and application re-deployment, and the re-configuration of services. Such strategy can minimize the effect of representation of RSA on the system performance.

Retrieving Components

A J2EE application is packaged and deployed as an archive with suffix of ".war", ".ear" or ".jar". Such archives typically contain interfaces and implementations of EJBs, deployment descriptors of EJBs and the application. The deployment descriptor describes the structure of EJBs, their external dependencies and the application assembly information, specifying how to compose individual EJBs into an application [4]. The data encapsulated in Components can be retrieved from the

deployment package, which are held and operated by the EJB containers, as shown in Figure 3.

RSA Data	Elements in Deployment Package
Name of <i>Component</i>	<ejb-name> in <module>
Name of the provide player of <i>Component</i>	<home> and <remote> or <local-home> and <local> in <session> or <entity>
Name of the request player of <i>Component</i>	<home> and <remote> in <ejb-ref>; <local-home> and <local> in <ejb-local-ref>
Attributes of <i>Component</i>	<env-entry>, <resource-ref>, <cmp-field> and <primkey-field>
Properties of <i>Component</i>	<ejb-class>, <session-type>, <persistence-type>, <prim-key-class>, <transaction-type>, <reentrant>, <security-role-ref>, <security-role>.

Figure 3. Retrieving Components from the deployment package

Retrieving Connections

Most connection information can be retrieved from deployment descriptors of the application, which declared components and resources that are required by component.

There are three typical communication protocols for EJBs, including RMI-IIOP (Remote Method Invocation – Internet Inter-ORB Protocol), RMI-JRMP (Jave Remote Method Protocol) and EJBLocal. RMI is the **default** distributed object model of Java. The RMI-IIOP supports interactions between EJBs and other CORBA-compliant systems. And the RMI-JRMP supports interactions between EJBs and other RMI-compliant Java systems. As RMI supports interactions between EJBs deployed in the same or different JVM with the pass-by-value semantics, the EJBLocal supports interactions between EJBs **collocated** in the same JVM with the pass-by-reference semantics and at a much lower cost.

Retrieving Constraints

Most constraint information can be retrieved from deployment descriptors of EJB, by section of "assembly-descriptor", which usually includes information of "method permission", or "container-transaction". Some time "security-role" can also be used to declare other constraints on security.

4.3 Manipulation of RSA

With the help of RSA, we can adapt software by manipulation of RSA, which may lead to manipulations of components, connections, or constraints.

Manipulating Components

Component manipulation includes following actions: add or remove component, change component implementation, change constraints of components, etc. In PKUAS, the instance manager is responsible for the lifecycle management of EJBs, such as creating, releasing, passivating and activating the instances. The context manager holds all of the data related to EJBs, including

the states derived from the deployment package and runtime context.

The modification of the Components may enable PKUAS to perform the following activities:

- To add or remove Component will make PKUAS load or unload the corresponding EJB.
- To modify the value of the “ejb-class” in Properties of Component will make PKUAS update the corresponding EJBs with new implementation classes, including blocking the incoming invocations, buffering the old instances until its execution or transaction are over, creating new instances of the new implementation and copying the unchanged attributes from the old instances.
- To modify the constraints of component will make PKUAS reconfigure the corresponding containers, such as the size of the instance pool and the rule to passivate the instances.

More details of adding, removing, updating and reconfiguring EJBs in PKUAS can be found in [9].

Manipulating Connections

The communication service of PKUAS is implemented with a flexible interoperability framework that can add, modify and delete communication protocols embedded in PKUAS. PKUAS interoperability framework separates the interoperability aspect from the container in order to enable EJBs to interoperate with other components through multiple protocols, such as IIOP, JRMP and SOAP. The stub and skeleton are responsible for the transformation between the invocations of RMI and the messages of the underlying interoperability protocol. The connection and acceptor transform the messages between the interoperability protocol and the underlying transport protocol and manage the connections. The transport is responsible for sending and receiving messages via the underlying transport protocol. And PKUAS naming service can publish and retrieve different interoperability addresses with the integration of the naming services specific to the interoperability protocols. Currently, PKUAS allows EJBs to interoperate with other components through IIOP, JRMP, SOAP and EJBLocal.

Then, the modification of the Connections will enable PKUAS to perform the following activities:

- To add or remove the Connection will make PKUAS reconfigure the communication service by means of adding or removing the corresponding interoperability protocol.
- To add the Connection will make PKUAS allocate a new acceptor to the “to” component and register the interoperability address of the new acceptor into the naming service. Then the “from” component can invoke the “to” component through the new connection. This operation is very useful in the integration or evolution. For example, to add a new Connection of the

RMI-JRMP type allows the legacy Java program to invoke an EJB, and to add a new Connection of the RMI-SOAP type enables an EJB to act as a Web Service.

- To remove the Connection will make PKUAS reclaim the corresponding acceptor from the “to” component and remove its interoperability address from the naming service. Then the “from” component cannot invoke the “to” component through the corresponding connection any more.

- To modify the value of the elements in properties of connection will make PKUAS reconfigure the corresponding acceptor or connection, including the size of the message buffer, the transport semantics and the maximum number of the concurrency connections.

Manipulating Constraints

In order to add or remove functions other than the standard functions defined in [4] of the EJB container, we employ the design pattern of interceptor to dynamically insert extra functions before and after the invocation of the implementation of EJB. Typically, the services used by the container should have corresponding interceptors, such as the security interceptor, the log interceptor and the transaction interceptor, as shown in Figure 2. Thus, to modify the value of the elements in component that relate to the usage of the services will enable PKUAS to add, remove or update the corresponding interceptors. For example, changes of the “transaction-type” will change the behavior of the transaction interceptor. The manipulation of the constraints related to the configuration of services is under development.

5. Experiment and Visual Tool

In this section, we present an example of evolution that illustrates the evolution process and the reflect ability of PKUAS. The example is only about the evolution of components of “electrical-shop” application (eShop) via PKUAS with support of RSATool, a visual tool that was designed for our approach. And the evolution of constrains and Connections is much the same.

eShop is a J2EE application running on PKUAS, which is a simple B2C application allowing customers to purchase computer products from Internet. And we may use RSATool to model its Runtime software architecture in Figure 5.

The RSATool allows administrators to manage and evolve the applications deployed in PKUAS through representation and manipulation of RSA. Figure 5 shows the main window of RSATool, representing RSA of eShop. The location of the visual elements is very important to make the representation clearer and more understandable. Such location information cannot be retrieved from the runtime system and should be

Figure 5. eShop RSA shown in RSATool

complemented through the drag and drop of the visual elements by the administrator. After the “Save View” command in the “View” menu is executed, RSATool will save the location information and retrieve it the next time.

Now we can evolve the eShop system via RSATool. For example, we may select a component in RSATool, and then change its attributes. The RSATool will then pass our modification information to the RSA entity, which will in the last change the runtime application. The RSATool is the graphic representation of RSA Entity; while RSA Entity is the model of the physical running application. Their model and reflect mechanisms are provided by PKUAS. Detailed information can be found in [9].

6. Conclusion and Future Work

There has been lots of research on software evolution, to make software more adaptive and controllable. The much more dynamic and open runtime environment requires new technique to maintain software based on runtime software architecture (RSA), which is proposed by this paper. This paper addresses the RSA-based approach for software online adaptation and evolution, especially on three fundamental issues on our approach: incarnating RSA as a runtime entity, retrieving RSA from the runtime system, and enforcing the manipulation of RSA that may change the state of the runtime system. In addition, the infrastructure of our approach, PKUAS is also covered.

Till now, we have already established a J2EE application server (PKUAS) and made it support RSA retrieval and reflection. Furthermore, to make our approach more convenient to use, we have developed some visual tools, such as application deployment & management tool, RSATool and etc.

As a **novel** technique, RSA and RSATool require many more studies and experiments. More operations on RSA causally connected with the underlying runtime system need to explore and implement. And we will pay more attention on the performance, reliability and security of the causal-connection to put it into practice.

This effort is sponsored by the Key Project of National Science Foundation Program of China under Grant No. 60233010, National Science Foundation Program of China under Grant No. 60103001, National Science Fund for Distinguished Young Scholars of China under Grant No. 60125206, and the National Key Basic Research and Development Program (973) under Grant No. 2002CB312003.

7. References

- [1] Ned Chapin, Joanne E. Hale, Khaled Md. Khan, Juan F. Ramil, Wui-Gee Tan, Types of software evolution and software maintenance
- [2] Feng Chen, Qianxiang Wang, Hong Mei, Fuqing Yang. An Architecture-Based Approach for Component-Oriented Development. Proceedings of COMPSAC 2002.
- [3] Christine Hofmeister, Roboert Nord, Dilip Soni, Applied Software Architecture. Addison-wesley, 2000.
- [4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, “Design Patterns Elements of Reusable Object-Oriented Software”, Addison Wesley, 1995.
- [5] Sun Microsystems. Enterprise JavaBeans Specification, V1.1, Final Release. <http://java.sun.com/j2ee>, December 1999.
- [6] Sun Microsystems. Java 2 Platform, Enterprise Edition Specification, V1.2, Final Release. <http://java.sun.com/j2ee>, December 1999.
- [7] Nenad Medvidovic and Richard N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transaction On Software Engineering, Vol. 26, No. 1, p70-93, January 2000.
- [8] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
- [9] Qianxiang Wang, Feng Chen, Hong Mei, Fuqing Yang. Using Application Server To Support Online Evolution. Proceedings of International Conference on Software Maintenance (ICSM2002), 3-6 October 2002, Montréal, Canada.
- [10] David Garlan: Software Architecture: a Roadmap. ICSE – Future of Software Engineering Track 2000: 91-101
- [11] Guillermo Arango] Ruben Prieto-Diaz. “Domain Analysis Concepts and Research Directions. in Domain Analysis and Software System Modeling, R. Prieto-Diaz and G. Arango, eds., IEEE Computer Society Press, Los Alamitos, California, 1991, pp. 9-32
- [12] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin, “Aspect-Oriented Programming”, ECOOP’97 1997.
- [13] Swanson EB. The dimensions of maintenance. Proceedings 2nd International Conference on Software Engineering. IEEE Computer Society: Long Beach CA, 1976; 492-497.