컴포넌트 합성에서 커넥터의 메시지 스케줄링 알고리즘

The Message Scheduling Algorithm of Connector in the Software Composition

정 화 영*
HWa-Young Jeong

요 약

컴포넌트 기반 소프트웨어 개발에서, 컴포넌트 모듈간의 인터페이스는 매우 중요하다. 대부분의 기존 방법들은, 커넥터가 원격 프로시저 호출(RPC)이나 이벤트 호출에 의해서 둘 또는 그 이상의 컴포넌트들 사이의 모든 통신 채널을 담당한다. 그러나 이들 처리방법은 컴포넌트가 커넥터를 통하여 다른 컴포넌트에게 많은 요청을 보낼 때 한계를 가진다. 즉, 커넥터에서 다중 요청을 처리할 수 있는 보다 효율적인 인터페이스 방법이 필요하다. 본 논문에서는 커넥터에서 메시지 큐를 이용한 상호작용 스케줄링 알고리즘을 제안하였다. 이를 위하여 메시지를 일시적으로 저장 및 가져오도록 운영하는 메시지 버퍼를 사용하였다.

Abstract

In the component based software development, it is very important to interface between modules of component. Almost of existing method, Connectors are deal with all communication channels between two or more components/interfaces by RPC(Remote procedure call) and event call. But these process has limits when component send a lot of request call to other component through connector. That is, we need more efficient interface method that connector can process multi request call. In this paper, I propose interaction scheduling algorithm using message queue in the connector. For this purpose, I use message buffer which operate to save and load message temporarily.

☞ keywords: CBD, Component composition, Process scheduling algorithm

1. Introduction

Software reliability engineering is an important aspect of many system development efforts, and consequently there has been a great deal of research in this area [1]. Component Based Development(CBD)[2,3,4] technique designs and embodies component, independent unit parts module, or completes whole system by assembling and composing the existing developed component with connector or interface. Software Architectures(SAs) are recognized to be a powerful tool in meeting these requirements. Also, it emerged to structure complex software systems, exploiting commonalities in specific domains organizations and providing a high-level system description. Software architecture is described in terms of components, connectors, and their configurations. From operational aspect of CBD and SAs point of view, all the approaches agree that dynamic aspects are orthogonal to all the views and they allow for analysis and validation of architectural choices. The most important thing is how to interface between components efficiently. In this question, we have to consider the factors as following words. What would a special purpose connector specification model look like? Can connectors be reused not just to compose

* 종신회원 : 경희대학교 교양학부 조교수
hyjeong@khu.ac.kr
[2007/03/27 투고 - 2007/04/05 심사 - 2007/08/19 심사완료]

components into (sub)systems, but also to compose more complex connectors? What composition operators are necessary and sufficient to allow connector composition? Is there a set of primitive connectors out of which "wall interesting or useful" connectors can be constructed by those connector composition operators? How can one characterize interesting and useful in this context?[13].

In this paper, I would like to address these questions and show it process as a language for compositional construction. So I propose message based interface process method between software components, efficiently. In this research, request call and response result of component treat as a message in the connector. This architecture has two threads and one port which are in charge of transmission in and out. For efficient handling of message, I use message buffer which operate to save and load message temporarily. Proposal method has been successfully evaluated and operated on application in this paper.

2. Component compositions

2.1 A software component

Traditional software development methods focus on software correctness, introducing performance issues later in the development process. But, in recent years, increasingly systems are implemented as compositions of independently-developed components that must be integrated into working systems using various interaction mechanisms, compose with one another, modify, and maintain[9]. Component software emerged from object-oriented programming as a way to apply compositional engineering to the construction of complex software[5]. At the moment various commercial vendors provide components as building blocks for industrial systems, trying to establish component marketplaces[6]. Components provide a higher level of design abstraction than objects. It can be composed together by binding required to provide services to form a higher-level component. The structure of a composition is in the focus of software architectures[7]. In the component composition, a component can interact with its environment through operations at identified access points, called interfaces. The visibility of the interfaces of a sub-component, in and out of the enclosing component, is determined by the controller of the enclosing component[11].

2.2 Connector interaction for request call and response

A connector is an abstraction capturing communication/interaction between components, clearly separating communication from the business logic of the components. Connectors can address several software development issues ranging from application distribution including data transfer, conversion and support for various middleware, to interface adaptation and access coordination[7]. By referring to architectural elements components and connectors, [10] make use of the definitions as Weakly-Closed System, Closed System, Weakly-Open System, Open System. Almost of this method handle the request message with FIFO method according to role composition of connector[8]. In the basic architecture of component composition system[12], coordination means connector interaction and autonomous component system means software component include business logics. Composite system send request call to component system and received response process(or service) result from component system. At this time, coordination(or connector) perform interaction between composite system and component system by RPC call and FIFO method. [14] was described that Components are composite objects with ports, interacting with one another either through event triggers or method invocations. Each component can have the following types of ports:

- Publish Port to publish events.
- Subscribe Port to subscribe to events.
- Receptacle to issue method invocations.
- Facet to accept method invocations.

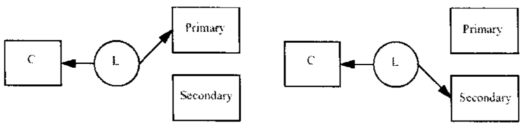


Figure 1. The connector extension for multiplex connection.

Wright has been studied as like the structure in Figure 1 for the multiplex connection of components[15]. In this structure, the requesting service process was performed by connector playing a role as like channel in the occurrence of request of component. Also, In the [16] research, they study the multiple choice and process the in side of the dynamic software architectures, the component instead of web service.

3. Connector's interaction process scheduling in the component composition

3.1 Design of connector interaction architecture

In this research, I designed connector interaction architecture for efficient handling between composite system and component. Composite system and each component have two ports, In port for request call and Out port for response result. This architecture for message handling in connector has two thread processes and messages as shown Figure 2.

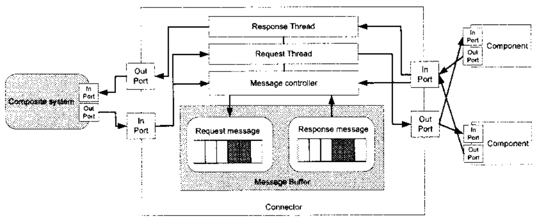


Figure 2. Proposed architecture of connector interaction

In this method, composite system send request call to connector for component process. Message controller in connector receives this request and command request Message Buffer to save request call information as a message. And next time, Message controller operates to start Request Thread for request call transmission to component. Finally, Request Thread is able to send composite system's request call to component with request message. In side of Response process of connector interaction, this process is analogous with request call process handling. When component finish the process(or business logic) successfully, it send to acknowledge response call to Message controller in connector. Then Message controller operates to start Response Thread. Response Thread sends response result of component process with response message to composite system.

3.2 Interaction process scheduling algorithm in connector

According to above this architecture, I can implement the interaction algorithm in connector. Composite system which was described total system construct algorithm, as shown Figure 3. In this construction, Composition() perform to link connected component to composite system using linkcomponent() function in connector.

```
Class CompositeSystem
  Algorithm main()
  Begin
    Call InitialSystem()
    Call Binding()
    Call ServiceStart()
  End
  Algorithm InitialSystem()
  Begin
    Search for sub component
    Create home interface object of component
    Create remote interface object of component
  End
  Algorithm Composition()
  Begin
    ∀comp : connected component,
        for i←0 to {number of connected component}
        do
            Call Connector.linkcomponent(this, call
            component, comp)
  End
  Algorithm ServiceStart()
  Begin
    Component service request start to connector
  End
```

Figure 3. Composite system algorithm

Figure 4 shows the connector include interaction algorithm between composite system and connected components. Actually, roles() perform to link connected component using Bind() and create RequestThread and ResponseThread for message handling. RequestThread and ResponseThread operate message handling from MessageBuffer. MessageController and MessageBuffer has inherited connector. MessageController perform to save request message to message buffer and order to start RequestThread and ResponseThread. MessageBuffer perform to save and load message, request and response.

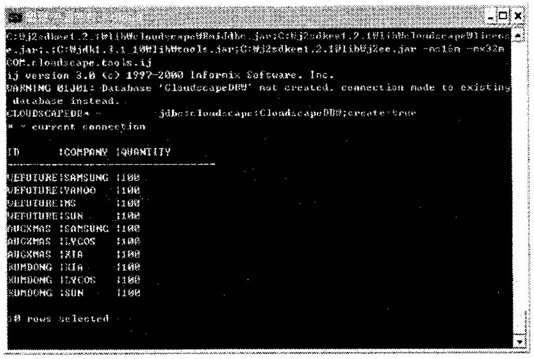
```
Class Connector
  Algorithm roles()
  Input parameter => object composite system,
        object call component, string component name
  Begin
    Bind(composite system, call component)
    Create lookup table with component name
    Create RequestThread and ResponseThread
  End
  Algorithm RequestThread()
  Begin
    Wait for component ready
    ∀compName : component name in lookup table
        for i←0 to {number of connected component}
        do
            ComponentName = Search compName
    ComponentService =

MessageBuffer.GetRequestMessage(ComponentName)
    Send ComponentService to component
  End
  Algorithm ResponseThread()
  Begin
    MessageBuffer.ResponseMessage(response result
        object)
    Wait for Composite system ready
    ∀compName : component name in lookup table
        for i←0 to {number of connected component}
        do
            ComponentName = Search compName
    ComponentResult =

MessageBuffer.GetResponseMessage(ComponentName)
    Send ComponentResult to composite system
  End

Class MessageController extends Connector
  Algorithm Start()
  Begin
    MessageBuffer.RequestMessage(request call object)
    Start super.RequestThread
    Wait for Acknowledge response call
    if(receive Acknowledge response call from
    component)
    then
```

```
      Start super.ResponseThread
  End

Class MessageBuffer extends Connector
  Algorithm RequestMessage()
  Input parameter => object request call object
  Begin
    Check message queue status
    Save request message to message queue
  End
  Algorithm GetRequestMessage()
  Input parameter => string request component name
  Output parameter => Object request call object
  Begin
    Search for request call object with request
            component name which in lookup table
    Return request call object
  End

  Algorithm ResponseMessage()
  Input parameter => object response result object
  Begin
    Check message queue status
    Save response message to message queue
  End
  Algorithm GetResponseMessage()
  Input parameter => string response component name
  Output parameter => Object response call object
  Begin
    Search for response object with response
            component name which in lookup table
    Return response call object
  End
```

Figure 4. Interaction process scheduling algorithm in connector

4. Application

In this section, I applied a simple application that uses proposal techniques. For this application, I use Java language(EJB: Enterprise Java Beans) on Windows XP. Figure 5 shows response result after check stock list in which stock process component perform. Quantity means a residuary stock of each company.



```
C:₩j2sdkee1.2.1₩lib₩cloudscape₩RmiJdbc.jar;C:₩j2sdkee1.2.1₩lib₩cloudscape₩licens
e.jar;.;C:₩jdk1.3.1_10₩lib₩tools.jar;C:₩j2sdkee1.2.1₩lib₩j2ee.jar -ms16m -mx32m
COM.cloudscape.tools.ij
ij version 3.0 (c) 1997-2000 Informix Software, Inc.
WARNING 01J01: Database 'CloudscapeDB' not created, connection made to existing
 database instead.
CLOUDSCAPEDB* -          jdbc:cloudscape:CloudscapeDB;create=true
* - current connection

ID        |COMPANY |QUANTITY
-------------------------------------
WEFUTURE|SAMSUNG |100
WEFUTURE|YAHOO   |100
WEFUTURE|MS      |100
WEFUTURE|SUN     |100
AUGXMAS |SAMSUNG |100
AUGXMAS |LYCOS   |100
AUGXMAS |KIA     |100
KUMDONG |KIA     |100
KUMDONG |LYCOS   |100
KUMDONG |SUN     |100

10 rows selected
```

Fig 5. Response result after check stock list process

5. Conclusion

It is important that connector interaction operate request and response call between connected components, efficiently. In this paper, I propose interaction process scheduling algorithm which used in connector. For efficient handling of request and response call, I used two messages, request and response call, in message buffer. Also, for a message handling, I used two Thread process, RequestThread and ResponseThread. Message controller in connector performs to control message buffer and Thread process. More flexible of components is possible due to parallel structure of components with message in the middle and not having sequenced hierarchical structure. For application example, I implement and apply proposal method. In application, target system was stock management system. At this result, this system operates customer and stock service, successfully.

For future work, I will use this architecture to enhance software system reliability using software architecture models. That is, I have to consider many other situations as component process problem itself and interface problem in connector.

References

[1] Genaina Rodrigues, David Rosenblum, and Sebastian Uchitel, “Using Scenarios to Predict the Reliability of Concurrent Component-Based Software Systems”, FASE 2005, LNCS 3442, 2005.

[2] Stojanovic Z., A. Dahanayake, “Components and Viewpoints as integrated Separations of Concerns in system Designing”, International Conference on Aspect-Oriented Software Development, April, 2002.

[3] Miguel Goulão, “CBSE: a Quantitative Approach”, Proceeding of ECOOP 2003, 2003.

[4] Murali Sitaraman, Timothy J. Long, Etc., “A formal approach to component-based software engineering: education and evaluation”, Proceedings of the 23rd International Conference on Software Engineering, IEEE Computer Society, 2003.

[5] A. Bailly, M. Clerbout, and I. Simplot-Ryl. “Component Composition Preserving Behavioural Contracts Based on Communication Traces”. Proc. of Tenth International Conference on Implementation and Application of Automata (CIAA 2005), LNCS 3845, 2005.

[6] Andreas Speck, Elke Pulvermüller, Michael Jerger, Bogdan Franczyk, “Component Composition Validation”, International Journal of Applied Mathematics and Computer Science 12(4), December, 2002.

[7] Stanislav Višnovský, “Modeling Software Components Using Behavior Protocols”, Ph.D Thesis, Department of Software Engineering of Charles University, 2002.

[8] Ioannis Georgiadis, “Self-Organising Distributed Component Software Architecture”, University of London, Ph.D Thesis, 2002.

[9] Bridget Spitznagel and David Garlan, “A Compositional Formalization of Connector Wrappers”, Proceedings of the 2003 International Conference on Software Engineering, 2003.

[10] Mauro Caporuscio, Paola Inverardi, and Patrizio Pelliccione, “Formal Analysis of Architectural Patterns”, First European Workshop on Software Architecture (EWSA 2004). St Andrews, Scotland, UK. May, 2004.

[11] E. Bruneton, T. Coupaye, and J. B. Stefani, “Recursive and Dynamic Software Composition with Sharing”, Seventh International Workshop on Component-Oriented Programming (WCOP02), June, 2002.

[12] Ricardo de MendonÃ§a da Silva, Paulo Asterio de C. Guerra, and CecÃ-lia M. F. Rubira, “Component Integration using Composition Contracts with Exception Handling”, Object-Oriented Technology: ECOOP 2003 Workshop, LNCS 3013, July, 2003.

[13] Farhad Arbab, "Coordination for Component Composition", Electronic Notes in Theoretical Computer Science, FACS 2005, 2005.

[14] Zonghua Gu, Kang G. Shin, "Model-Checking of Component-Based Event-Driven Real-Time Embedded Software", Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, IEEE Computer society, 2005.

[15] Robert Allen, Rémi Douence, and David Garlan, "Specifying and Analyzing Dynamic Software Architectures", Proc. of 1998 Conference on Fundamental Approaches to Software Engineering, 1998.

[16] Jeremy S. Bradbury, James R. Cordy, Juergen Dingel, Michel Wermelinger, "A Survey of SelfManagement in Dynamic Software

Architecture Specifications", Proc. of the International Workshop on Self-Managed System, ACM. 2004.

◐ 저 자 소 개 ◑

정 화 영(HWa-Young Jeong)
1991년 목원대학교 수학교육과 졸업(학사)
1994년 경희대학교 전자계산공학과 공학석사
2004년 경희대학교 전자계산공학과 공학박사
2000~2005년 예원예술대학교 게임영상학부/정보경영학부 조교수
2005~현재 경희대학교 교양학부 조교수
관심분야 : 소프트웨어 공학, 컴포넌트 조립/합성, 웹 기반 교육.
E-mail : hyjeong@khu.ac.kr