제주교육대학교 논문집 (2002)
제31집 pp. 341~358

C2 아키텍처를 변형한 메시지 중앙처리 기반의 컴포넌트 활용 기법

An Approach to Application of Component Based on Message Central Processing Change the C2 Architecture

김종훈* 정화영** 김종진***

〈목 차〉

〈ABSTRACT〉

Recently, Software development method supported CBD is applied with many concern and is researched with part of application and composition based-on architecture effectively use it. Effectively, C2 architecture has been concern with the point of

* 제주교육대학교 컴퓨터교육과 조교수
** 예원대학교 전자상거래과 교수
*** 홍익대학교 컴퓨터공학과 박사과정

component composition method based-on message driven for supported GUI. But, In case of classified sequence in component and method call method in server component, component must be modified to apply it. Thus, In this paper, Message handling part with a part of C2 architecture change is locate in the message neither component not connector. So, Although method call method it can be composit and operate component for support Plug-and-Play without modification. Also, it's possible the more flexible message handling with parallel composition of component between message without classified sequence.

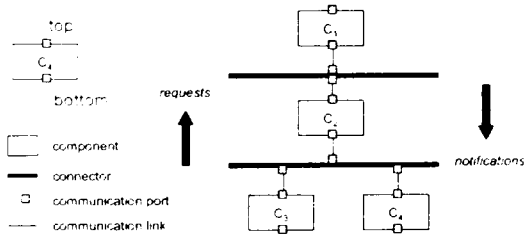
Ⅰ. 서 론

컴포넌트 기술은 소프트웨어 프로그래밍에서 하드웨어 개발 환경처럼 소프트웨어 Plug-and-Play 방식으로 시스템을 구축하는 '합성을 통한 시스템 구축'으로의 전환을 목적으로 한다[1, 2]. 따라서, CBD를 효과적으로 지원하기 위해서는 응용 컴포넌트들이 서로 정확하게 결합하여 작동할 수 있는 아키텍처를 기반으로 컴포넌트의 생성과 합성작업이 이루어질 수 있어야한다[3]. 이를 위하여, 컴포넌트 합성방법에서는 컴포넌트간의 인터페이스 불일치를 해결할 수 있어야하며, 독립적인 컴포넌트의 메소드 수정 없이 합성할 수 있어야 한다[12]. 아키텍처 기반기술은 UNIX의 Pipe-and-filter아키텍처와 일반적인 어플리케이션에서 오랜 기간에 사용된 Blackboard 아키텍처기술[4]에서부터 Style 기반의 Unicon[5], Aesop[6], C2[7], 시멘틱 모델 기반의 Wright[8], Rapide[9], Domain Spec 기반의 실시간 객체지향구조의 ROOM[10] 등을 들 수 있다. 이들 중, C2 스타일 아키텍처[11]는 컴포넌트간의 직접적인 메소드 호출방식이 아닌 메시지 전달방식의 비 동기적인 상호작용을 지원하는 대표적인 구조라 할 수 있다. 그러나, C2 스타일은 각 컴포넌트들간의 Top, Bottom을 이용한 Connector상의 계층적 구조를 갖음으로써 중간노드의 컴포넌트에 관한 결과가 요구될 때 바로 결과를 확인할 수 가없고, 여러 단계의 커넥터 Port를 통하거나 최하위계층의 컴포넌트에 관한 Notification을 통하여 확인된다. 또한, 컴포넌트간 메시지 전달을 위하여 In, Out Vector 2개와 각 컴포넌트마다 top, bottom 메시지 공간등 최소 4개의 메모리공간을 필요로 한다.

따라서, 본 논문에서는 C2 스타일을 변형하여 컴포넌트와 컴포넌트간의 직접적인 메시지 핸들링 방식이 아닌 하나의 메시지 공간에서 각 컴포넌트에게 해당 메시지를 전달하는 메시지 중앙처리방식(MCP : Message Central Processing)을 제안한다. 즉, 컴포넌트간의 계층적 구조가 아닌 메시지공간을 매개체로 한 병렬적 구조를 갖는다. 또한, 1개의 메시지 이름 공간(Message Name Vector)와 Request/Notification 메시지 공간 등 3개의 메시지공간만을 사용하도록 하였다. 또한, 본 활용기법 구현을 위하여 서버측 컴포넌트 모델인 EJB[13]를 택하였고, 각기 다른 EJB서버 2대의 시스템에 각 데이터베이스서버를 두었으며 이들을 핸들링하는 메인 서버 시스템 1대를 두는 멀티 서버 환경에서 구현 및 테스트되었다.

Ⅱ. 관련연구

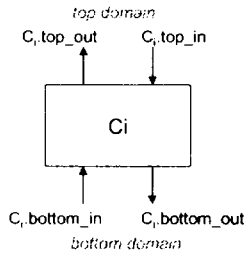
C2 스타일 아키텍처의 기본원칙은 메시지기반의 컴포넌트간 통신, 멀티 쓰레드, 각 계층의 독립성, Message Routing Connector를 통한 컴포넌트들의 연결구성, GUI 소프트웨어 요구사항 지원 등이다. 또한, 분산 및 이기종 환경, 분할된 주소공간을 갖지 않는 환경에서의 컴포넌트실행, 다중사용자 및 다중사용자 툴킷, 실행시간에서 변화될 수 있는 동적구조 등에 적합하다[4]. C2 스타일의 기본구조는 〈그림 1〉과 같이 각 계층을 기반으로 메시지의 상호작용을 하도록 되어있다.



〈그림 1〉 C2 스타일 아키텍처의 기본구조

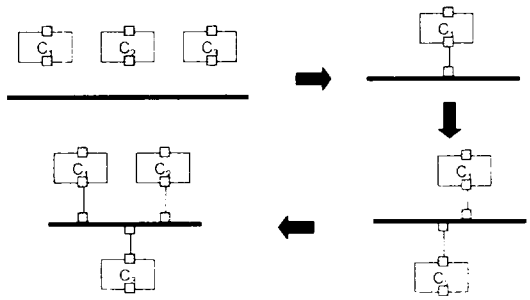
각 계층은 컴포넌트와 커넥터 2개 형식의 블록으로 이루어져 있으며, 각 컴포넌트와 커넥터에서 top port 와 bottom port를 가지고 있다. 이를 통한 메시지교환은 메시지 라우팅 장치와 같은 역할을 하는 커넥터를 통하여 전달하는데 C2 합성규칙에서는 다

음과 같은 방법을 따른다. 즉, 단일 커넥터의 bottom port는 컴포넌트의 top port와 연결될 수 있고, 단일 커넥터의 top port는 컴포넌트의 bottom port와 연결될 수 있다. 또한, 단일 커넥터에 접속되는 컴포넌트와 커넥터의 수는 한계가 없다. 이에 따른, 메시지의 전달방식은 다음 〈그림 2〉에서와 같이 컴포넌트의 top port에서 상위계층의 컴포넌트로 보내는 Request 메시지와 상위계층 컴포넌트의 bottom port로부터 전달되는 Notification 메시지를 받을 수 있다. 따라서, 각 계층의 컴포넌트들은 독립적이며 최하위계층의 컴포넌트의 Notification 메시지를 통하여 최종결과를 확인할 수 있다.



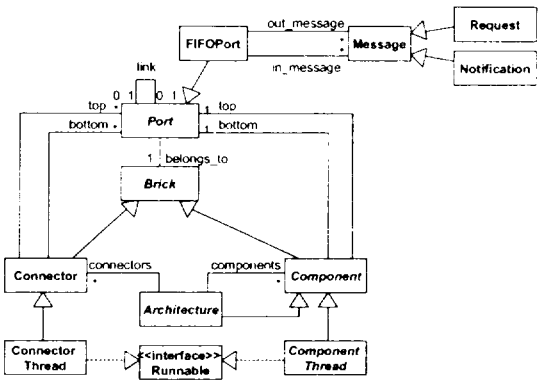
〈그림 2〉 C2 컴포넌트 도메인

C2 스타일의 컴포넌트 조립은 〈그림 3〉에서와 같이 각각의 커넥터를 중심으로 C2 컴포넌트들을 추가, 삭제, 재연결 할 수 있다[14].



〈그림 3〉 C2 컴포넌트 조립

또한, 이를 활용하기 위한 프래임워크의 구조는 〈그림 4〉와 같이 Port를 통하여 전달받은 메시지는 해당 커넥터와 컴포넌트로 전달된다. 이를 위하여, 각 커넥터와 컴포넌트에서 Thread를 동작시켜 메시지를 확인하고 있다.



〈그림 4〉 C2 스타일 아키텍처 프레임워크

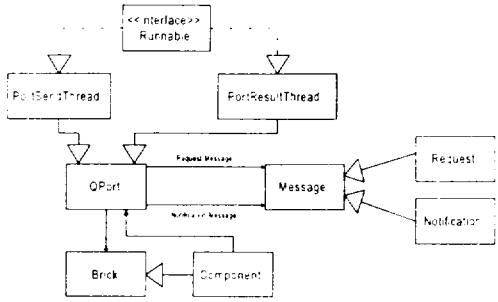
Ⅲ. C2 아키텍처를 응용한 메시지 중앙처리 방식의 컴포넌트 활용 기법

1. C2 아키텍처를 변형한 메시지 중앙 처리방식의 구조

컴포넌트의 조립 및 활용은 기존의 C2 스타일처럼 계층적인 형태의 컴포넌트 순차구조로 체계적인 틀을 구성하고 있지만, 실제로는 컴포넌트간 비 순차구조의 형태뿐만 아니라 순차구조로 된 조립형태에서도 중간의 컴포넌트에서 중간결과를 도출한 후에 다음의 구조를 진행할 경우가 많다. 또한, Plug-and-Play 조립방식을 지원하기 위하여 기존의 컴포넌트에 관한 수정작업이 없이 조립 가능하여야한다. 특히, 서버측 컴포넌트 모델인 EJB의 경우 컴포넌트간의 직접적인 메소드 호출방식을 택하고있으며 EJB에서 Thread를 시작, 중지, 재시작 할 수 없으므로 기존의 C2스타일을 적용하기 위해서는 많은 수정이 필요하다. 즉, 메소드 호출방식을 메시지호출방식으로 전환하여야하고 Component Thread와 Connector Thread에서는 이를 대체할 다른 방법이 필요하다. 따라서, 본 논문에서는 위 문제를 해결하기 위한 방안으로 C2스타일 프레임워크를 다음 〈그림 5〉와 같이 변형하였다.

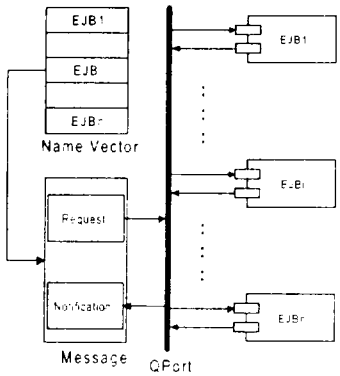
이에 따라, 컴포넌트들간의 메시지 핸들링은 컴포넌트나 커넥터의 단계가 아닌 메시지 부분에서 Thread를 통하여 자체 핸들링 하도록 하였다. 즉, 컴포넌트의 메시지는 QPort를 통하여 Message영역에 저장되며 PortSendThread와 PortResultThread부분에서

핸들링 함으로써 기존의 컴포넌트에 관한 수정 없이 조합 및 운용할 수 있다.



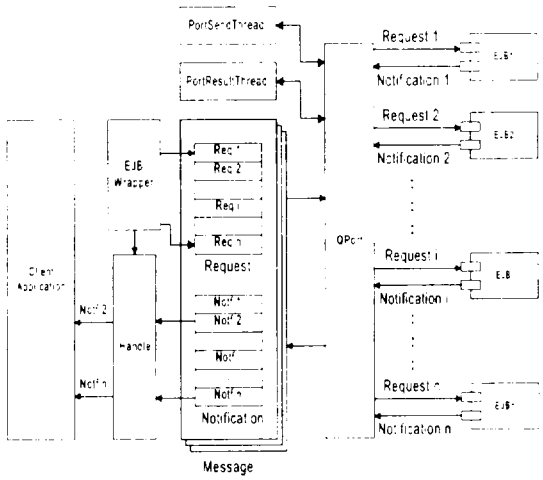
〈그림 5〉 C2 스타일 아키텍처를 변형한 제안된 프레임워크

또한, 중간단계의 컴포넌트 결과확인이 필요할 경우 다음 〈그림 6〉에서와 같이 핸들링되는 메시지의 이름공간(Name Space)을 두어 이를 식별하고 그 결과를 확인할 수 있도록 하였으며, 다음의 메시지를 진행할 수 있도록 하였다.



〈그림 6〉 C2 스타일 아키텍처를 변형한 제안된 메시지 중앙저리방식

〈그림 7〉에서는 본 논문에서 제시된 기법을 활용하여 각 컴포넌트를 조립하고 이를 운용하는 프레임워크를 나타낸다.



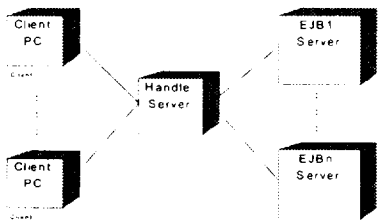
〈그림 7〉 제안된 메시지 중앙처리방식의 컴포넌트 조합 프레임워크

본 기법에서 EJB Wrapper는 조합될 컴포넌트들의 인터페이스 정보를 갖고있으며, 이를 통하여 각 컴포넌트들의 요구 메소드들을 초기화한다. Handle에서는 Wrapper에서의 컴포넌트 정보들을 기반으로 컴포넌트들의 호출 메소드들을 Message공간의 Request영역과 Name Vector에 적재한다. 적재된 Request Message들은 PortSendThread를 통하여 확인되고 각 컴포넌트의 조립순서에 따라 메소드 메시지들을 호출하며, 확인된 결과 메시지들은 Notification에 적재한다. PortResultThread는 Notification에 적재된 메시지들을 확인하고 Handle을 통하여 요구된 컴포넌트의 결과를 사용자에게 나타낸다. 〈그림 7〉의 경우 컴포넌트들의 조합순서 EJB1, EJB2,...EJBi,...EJBn에 관하여 최종결과인 EJBn이전에 EJB2단계에서 결과를 확인하고자 할 경우 각 컴포넌트들의 결과메시지 공간인 Notification에서 해당 컴포넌트의 결과를 가져올 수 있으며 이후 최종결과인 EJBn단계의 결과를 사용자에게 나타낸다. 따라서, 본 논문에서 제안된 메시지 중앙처리방식을 적용할 경우 각 컴포넌트들의 인터페이스정보 및 조립순서들을 Wrapper에 등록시키고 이를 통하여 메시지단계에서만 각 컴포넌트들의 통신을 핸들링 함으로써 컴포넌트의 수정 없이 조합 및 운용이 가능하다. 또한, 조립순서에서 중간단계의 컴포넌트에 대한 결과를 확인하고자할 경우 이를 Name Vector와 Notification 메시지를 통하여 확인이 가능하며, 계속 진행하여 최종단계의 컴포넌트에 대한 결과를 도출할 수 있다.

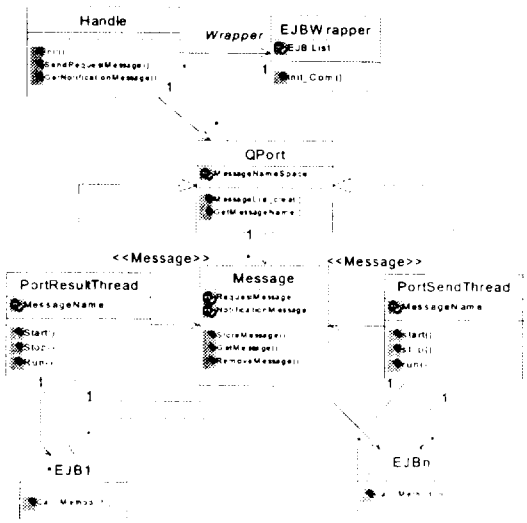
2. 메시지 중앙처리방식의 조립 및 활용에 관한 모델링

본 기법의 활용을 위한 모델링은 UML을 이용하였다. 이에 따라, Deployment Diagram은 〈그림 8〉과 같이 나타내며 각 EJB 서버를 별도로 두는 다중서버환경을 이용할 수 있다.

또한, 컴포넌트의 조합 및 운용을 위한 메시지 핸들링 기법은 다음 〈그림 9〉와 같이 EJBWrapper에서 조립 컴포넌트를 초기화 한 후, Handle에서 각 컴포넌트의 호출 메소드를 Request 메시지로 변환하여 저장하고 실행된 Notification 메시지를 Client에 나타낸다

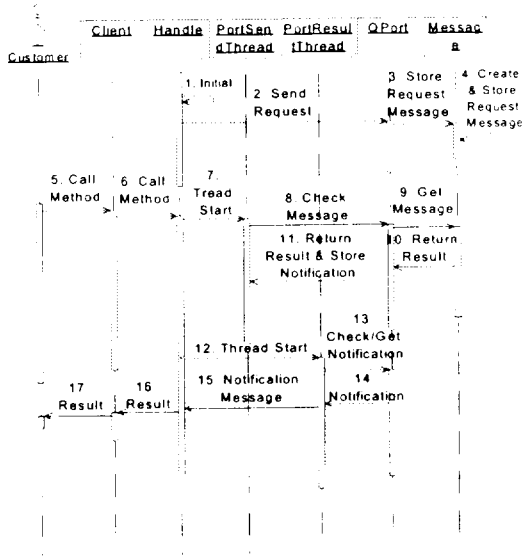


〈그림 8〉 Deployment Diagram



〈그림 9〉 Class Diagram

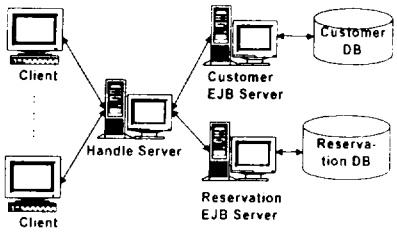
이를 구체적인 각 단계별로 처리흐름을 나타내면 〈그림 10〉의 Sequence Diagram과 같다.



〈그림 10〉 Sequence Diagram

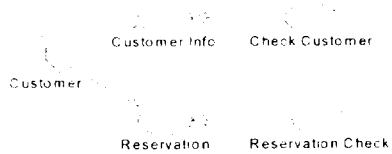
Ⅳ. 사례적용

본 논문에서 제안된 기법을 이용하여 간단한 예제를 적용하여 보았다. 적용된 시스템 환경은 각 서버에 대하여 Windows 2000 Server에서 JDK1.3.1과 J2SDKEE1.2.1을 이용하여 구현 및 테스트되었다. 또한, 〈그림 11〉과 같이 각 시스템의 데이터베이스 서버는 각 서버에 따라 별도로 두었으며 J2SDKEE에 내장된 Informix사의 Cloudscape를 사용하였다.

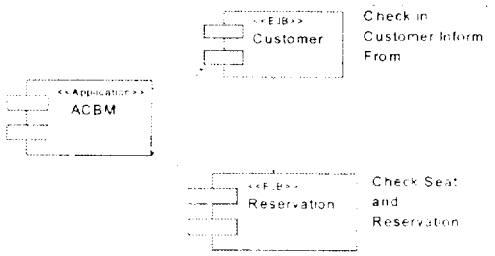


〈그림 11〉 좌석 예약 시스템 배치도

예제 구현 시스템은 고객정보를 검색하는 CustomerEJB와 해당고객의 좌석예약상황을 알아보는 ReservationEJB로 각각 구현되었으며 두 EJB시스템 사이에 Handle서버를 두어 컴포넌트들을 조합 및 운용하였다. 즉, 조합된 EJB정보는 Handle서버에 두었으며 이를 통하여 원격지의 Customer 서버 시스템과 Reservation 서버 시스템을 각각 두어 활용함으로써 3대의 다중서버 시스템환경으로 구현하였다. 이에 따른 사용자 요구사항을 다음 〈그림 12〉와 같이 나타내며, 컴포넌트 Diagram은 〈그림 13〉과 같이 나타낸다.

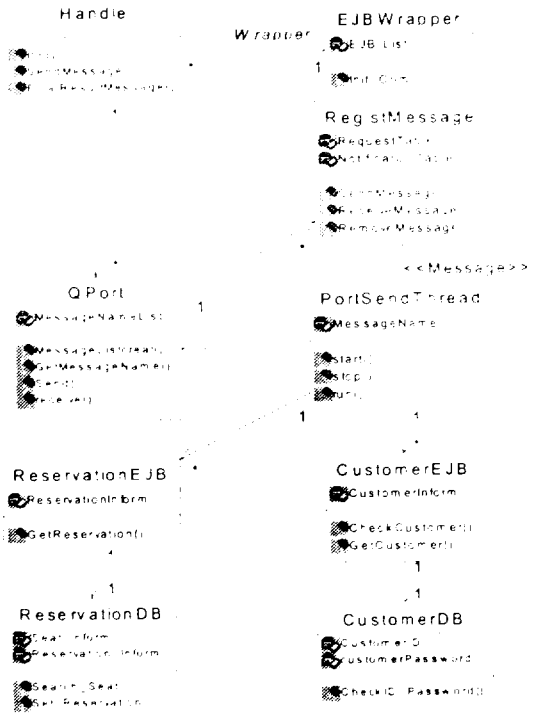


〈그림 12〉 Usecase Diagram

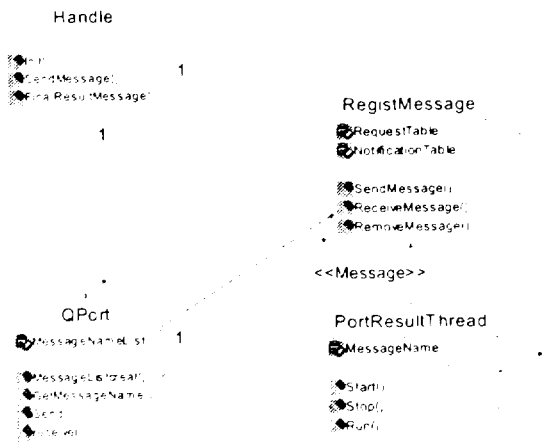


〈그림 13〉 Component Diagram

또한, 데이터베이스를 포함한 각 클래스들의 연관관계는, 〈그림 14〉에서는 조립된 컴포넌트들의 Request 메시지처리 및 Notification 메시지적재까지를 나타내며, 〈그림 15〉는 적재된 Notification 메시지 적재를 나타낸다.

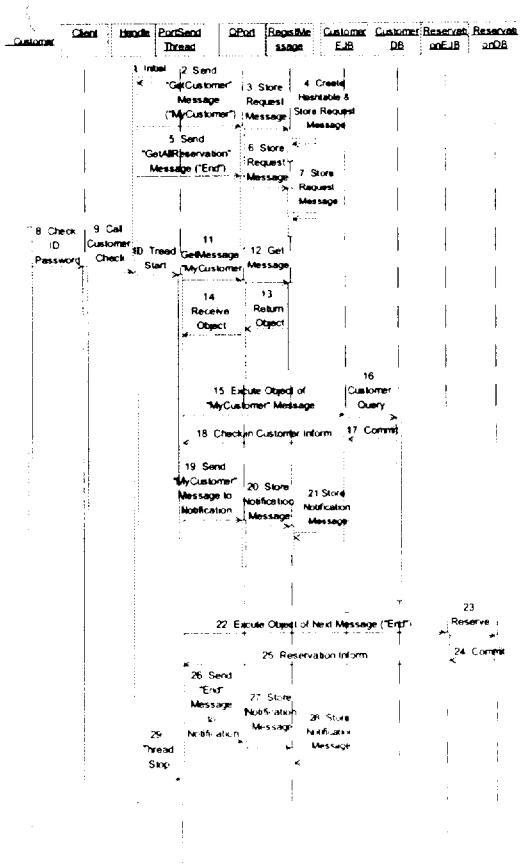


〈그림 14〉 Request 메시지 처리 Class Diagram

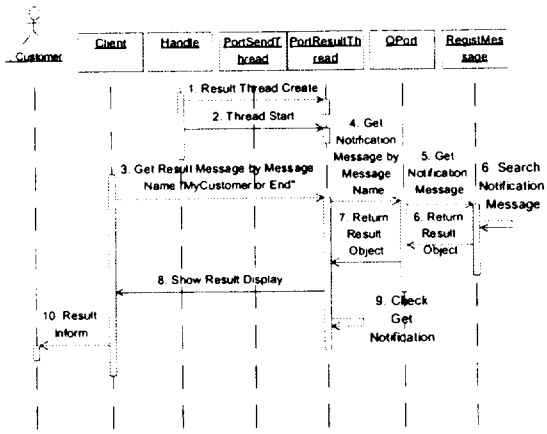


〈그림 15〉 Notification 메시지 처리 Cass Diagram

이에 따라, 각 단계의 처리순서는 Request 메시지처리 부분은 〈그림 16〉에서 Notification 메시지 처리부분은 〈그림 17〉과 같이 각각 나타낸다.

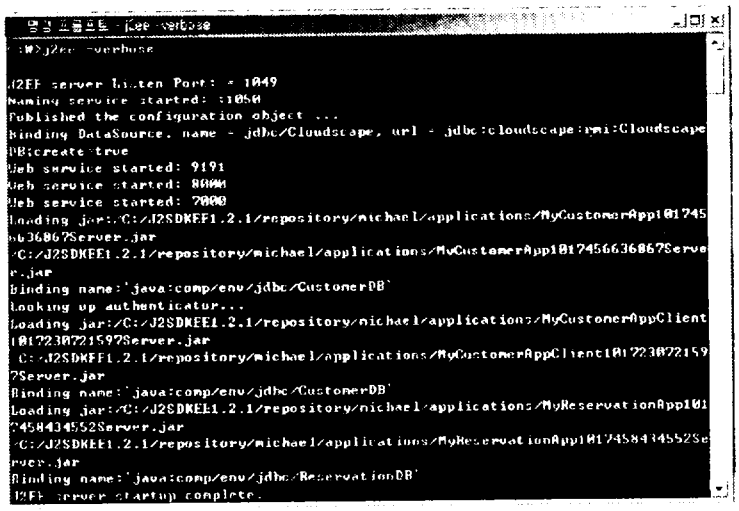


〈그림 16〉 Request 메시지 처리 Sequence Diagram

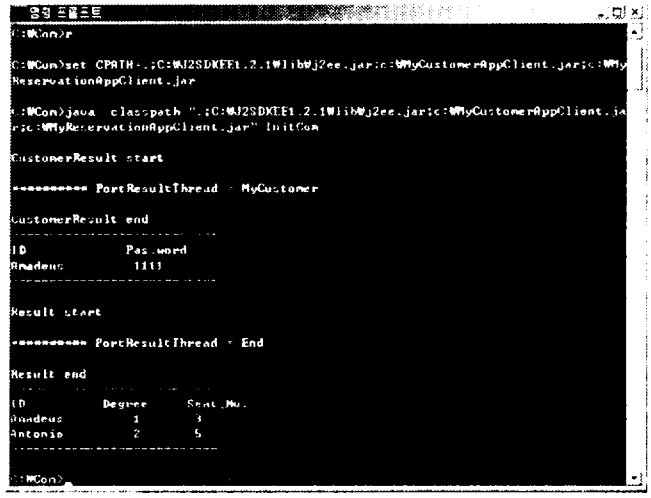


〈그림 17〉 Notification 메시지 처리 Sequence Diagram

본 예제 시스템의 실행을 위하여 먼저 J2EE Server를 〈그림 18〉과 같이 서버측에서 가동하였으며, 시스템 가동이후 Client측의 최종결과인 좌석 예약 상황의 화면은 〈그림 19〉와 같다.



〈그림 18〉 J2EE Server 가동



〈그림 19〉 좌석예약 시스템 결과

V. 비교 및 평가

본 제안 기법은 C2아키텍처를 변형하여 메시지의 핸들링 부분을 컴포넌트와 커넥터가 아닌 메시지부분에서 처리하도록 하였다. 따라서, 본 기법과 사례적용을 통하여 〈표 1〉에서와 같은 기본 구조적인 차이점을 들 수 있다.

〈표 1〉 기본 구조의 차이점

(단위 : 수량)

Item	C2아키텍처	본 제안 기법
Component	2	0
Connector	2	0
Message	1	1
Port	2	3
Message처리를 위한 Thread	2	2

즉, 컴포넌트와 커넥터부분은 각각의 Thread를 포함하여 2개씩이나 본 제안기법에

서는 메시지 핸들링부분을 메시지에 둠으로써 컴포넌트부분을 수정하지 않는다.

또한, Port부분에서 C2아키텍처의 경우 메시지의 창구 역활만 함으로써 Port와 FIFOPort를 두고있으나, 본 제안기법은 메시지부분으로부터의 처리까지 담당함으로 QPort, PortSendThread, PortResultThread 등 3개의 Port가 필요하다. 그러나, 메시지처리를 위한 Thread의 수는 본 기법과 C2아키텍처 모두 같다. 이에 따라, 실제 본 제안기법과의 운용상의 차이점은 다음 〈표 2〉와 같이 나타난다.

〈표 2〉 운용상의 차이점

특 징	C2아키텍처	본 제안 기법
구 조	컴포넌트간 계층적 순차구조	Message List를 통한 컴포넌트간 병렬구조
컴포넌트간 순차적인 메시지처리	가능	가능
컴포넌트간 병렬적인 메시지처리	여러 단계의 처리필요	쉬움
조립된 중간단계의 컴포넌트 결과확인	여러 단계의 처리필요	쉬움
메시지 저장소	4개	3개

컴포넌트간 병렬적 처리부분이나 중간단계의 컴포넌트 결과 확인시 C2아키텍처의 경우 컴포넌트의 Notification메시지를 직접확인하지 못하고 커넥터의 Port를 이용하여 몇 개의 단계를 거쳐야만 확인이 가능하다. 그러나, 본 기법에서는 메시지 단계에서 바로 핸들링이 가능하다. 또한, 메시지 저장소 부분에서는 C2아키텍처의 경우 in/out Vector 2개와 Request/Notification Message 2개가 필요하지만, 본 기법의 경우 Message Name Vector, Request/Notification Message 등 3개만이 사용된다. 따라서, 많은 Message공간의 활용은 시스템의 처리 속도를 저하시키며 Process의 복잡성을 가져온다.

Ⅵ. 결 론

본 논문에서는 C2아키텍처의 구조를 변형하여 메소드 호출방식의 컴포넌트에서도 쉽게 활용할 수 있으며, 컴포넌트간 병렬 조립구조를 갖는 환경에 맞도록 고안하였다. 즉, C2아키텍처 구조에서의 메시지 핸들링부분을 컴포넌트와 커넥터가 아닌 메시지부분으로 옮기고, In/Out Vector를 Message Name Vector 하나만으로 운용함으로써 컴포넌트의 수정이 없이 조립될 수 있는 Plug-and-Play를 지원하였다. 따라서, 메소드 호출방식의 서버컴포넌트의 경우 C2아키텍처를 적용하려면 컴포넌트와 커넥터부분의 수정이 불가피하였으나, 본 제안기법에서는 메시지의 핸들링을 QPort부분에서 담당함으로써 컴포넌트의 수정이 필요 없이 조립 및 활용이 가능하였다. 또한, C2아키텍처와 같은 순차적인 계층 조립구조를 갖지 않고 메시지를 사이에 둔 병렬적인 컴포넌트의 조립구조를 갖음으로써 컴포넌트간 유연한 조립 및 활용이 이루어질 수 있었다. 따라서, 컴포넌트의 순차적인 조립순서에 관계없이 컴포넌트의 실행결과를 확인할 수 있었으며, 조립순서의 마지막부분에서도 그 결과를 확인할 수 있었다. 이에 따라, 본 논문의 사례적용부분에서는 Cloudscape를 이용한 데이터베이스서버를 둔 각 EJB서버 시스템 사이에 Handle서버를 둠으로써 컴포넌트간 조립 시스템을 구현 및 테스트하였다.

앞으로 보다 효과적인 컴포넌트 조립기법 및 활용을 위하여 본 기법과 C2아키텍처 구조를 같이 적용하는 방안이 필요하며, 조립에 관한 효과적인 검증방안을 기반으로 보다 효율적인 컴포넌트간 핸들링부분이 필요하다.

❖ 참 고 문 헌 ❖

F. Brosard, D. Bryan, W. Kozaczynski, E. S. Liongorari, J. Q. Ning, A. Olafsson, and J. W. Wetterstrand, "Toward Software Plug-and-Play", in Proc. of the 1997 Symposium on Software Reusability, 1997.

P. C. Clements, "From Subroutines to Subsystem : Component-Based Software Development", Component Based Software Engineering, IEEE CSPress, 1996.

신동익 외 6인, "C2 스타일의 아키텍처 기술을 지원하는 ADL 지원도구의 개발", 한국정보처리학회 논문지 Vol. 8-D, No 6, 2001.

Taylor, R. N., Medvidovic, N., Anderson, K. M., Whitehead, E. J., Jr., Robbins, J. E., Nies, K. A., Oreizy, P. and Dubrow, D. L., "A Component-and Message-Based Architectural Style for GUI Software", IEEE Transactions on Software Engineering, Vol.22. No.6., June, 1996.

M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, Abstractions for Software Architecture and Tools to Support Them, IEEE Transactions on Software Engineering, Vol. 21, No. 4, April 1995, pp. 314-335

D. Garlan, R. Allen, and J. Ockerbloom, Exploiting Style in Architectural Design Environments, Proceedings of SIGSOFT 94 Symposium on the Foundations of Software Engineering, Dec. 1994

N. Medvidovic, P. Oreizy, and R. N. Taylor, Using Object-Oriented Typing to Support Architectural Design in the C2 Style, Proceedings of the 4th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE4), San Francisco, CA, Oct. 1996

R. Allen and D. Garlan, Formalizing Architectural Connection, Proceedings of 16th Int Conference on Software Engineering, Sorrento, Italy, May 1994

D. C.Luckham and J. Vera, An Event-Based Architecture Definition Language, IEEE Transactions on Software Engineering, Vol. 21, No. 9, Sept. 1995, pp. 717-734

B. Selic, G. Gullekson, and P. T. Ward, Real-Time Object-Oriented Modeling, John Wiley & Sons, Inc., 1994

The C2 Style, http://www.ics.uci.edu/pub/arch/c2.html, Information and Computer

Science, University of California, Irvine.

최유희, 권오천, 신규상, "C2 스타일을 이용한 EJB 컴포넌트 합성방법", 한국정보처리학회논문지, Vol. 8-D, No 6. 2001.

Sun Mircosystems Inc, " Enterprise Java Beans Specifications", at URL:http://java.sun.com

Nenad Medvidovic and Richard N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", IEEE Transactions on Software Engineering, Vol. 26, No. 1, January 2000, pp. 70-93