C2 스타일을 이용한 EJB 컴포넌트의 합성 방법

최 유 희[†] · 권 오 천[††] · 신 규 상[†††]

요 약

EJB(Enterprise JavaBeans)는 서버측 컴포넌트 모델로 소프트웨어 개발의 복잡도를 감소시키고 재사용성을 높여 주므로 소프트웨어 산업계는 현재 EJB 컴포넌트의 개발에 많은 관심을 가지고 있다. 그러나 특정 어플리케이션 시스템을 위해 제삼자에 의해 개발되어 tightly 결합된 EJB 컴포넌트를 plug-&-play 방식으로 조립하여 재사용하는 것은 쉽지 않다. 따라서 EJB 컴포넌트를 레고 블럭게임하듯이 쉽고 빠르게 조립하여 재사용할 수 있는 합성 방법에 대한 연구가 필요하다. 본 논문에서는 Chiron-2(C2) 스타일을 이용하여 EJB를 합성하는 방법에 대하여 설명한다. 먼저 EJB 합성을 위해 EJB를 지원하는 C2 아키텍처 프레임워크를 변경하고 변경된 프레임워크를 이용하여 EJB 합성을 위해 필요한 EJB wrapper를 생성하는 방법에 대하여 설명한다. 또한 여러 EJB 컴포넌트로 구성된 C2 아키텍처를 하나의 단일 EJB 컴포넌트로 사용할 수 있도록 하기 위한 합성 EJB를 생성하는 방법에 대하여 설명한다.

An Approach to Composition of EJB Components Using the C2 style

You-Hee Choi[†] · Oh-Cheon Kwon[††] · Gyu-Sang Shin[†††]

ABSTRACT

EJB(Enterprise JavaBeans) is the server-side component model and its purpose is to reduce the complexity of software development and to increase software reusability. Many concerns for development of EJB components have recently been raised. However, it is difficult to compose EJB components provided by third parties through the plug-and-play method. Therefore, the composition method by lego block styles is needed for EJB components. In this paper, we propose an approach to composition of EJB components using the C2 architectural style. In order to support EJB composition, we modified the general C2 architecture framework. We propose how to create EJB wrappers that can compose EJB components according to the C2 framework. Our approach also enables developers to create a new composite EJB that uses a C2 architecture which is composed of EJB components. To do this, we propose how to create a new composite EJB.

키워드 : 컴포넌트 기반 개발(CBD), 컴포넌트 기반 소프트웨어 개발(CBSD), 합성, 조립, EJB, C2

1. 서 론

컴포넌트 기반 소프트웨어 개발(Component-Based Software Development : CBSD)은 이미 만들어진 소프트웨어 컴포넌트들을 조립하여 큰 단위의 소프트웨어 시스템을 개발하는데 초점을 둔다[1]. 즉, 컴포넌트 기반 소프트웨어 개발의 관점에서 시스템 개발은 코드를 작성하는 대신에 기존의 소프트웨어 컴포넌트를 조립하는 것으로 대체되고 있다[2]. 이에 따라 코드 재사용성을 높이기 위한 컴포넌트 모델들과 이질적인 컴포넌트를 조립하기 위한 하부 구조를 지원하는 엔터프라이즈 어플리케이션 통합 기술(Enterprise Application Integration : EAI) 등을 개발하고자 하는 노력들이 많이 이루어지고 있다.

이러한 관점에서 서버측 컴포넌트 모델인 EJB[3]는 Java 프로그래밍 언어의 "Write Once, Run Anywhere" 개념을 따르는 것으로 한번 개발된 EJB 컴포넌트는 코드 수정이나 재컴파일없이 다양한 플랫폼에서 전개되어 재사용되도록 하자는 목적으로 제안된 것이다[4]. 즉, 각 EJB 컴포넌트는 원칙적으로는 재컴파일없이 다양한 플랫폼상에 전개될 수 있다는 것이지만 실제로는 제삼자에 의해 제공된 이질적인 EJB 컴포넌트를 plug-&-play 방식으로 조립하여 재사용하는 것은 쉽지 않다. 그 첫 번째 이유로는 실제로 다양한 vendor들과 이종의 하부 구조에 종속적이라는 것이고, 두 번째 이유로는 EJB 컴포넌트간의 상호작용이 가능하기 위해서는 직접적인 메소드 호출 로직이 필요하고 이에 따라 유연한 재구성 및 합성이 쉽지 않다. 그러나 첫 번째 이유는 하부 구조와의 밀접한 연관성을 줄이기 위한 방법으로 여러 미들웨어와 어

† 정 회 원 : 한국전자통신연구원 컴퓨터 · 소프트웨어연구소 S/W공학연구부 연구원
†† 정 회 원 : 한국전자통신연구원 컴퓨터 · 소프트웨어연구소 S/W공학연구부 책임연구원
††† 정 회 원 : 한국전자통신연구원 컴퓨터 · 소프트웨어연구소 S/W공학연구부 컴포넌트공학연구팀 팀장
논문접수 : 2001년 10월 5일, 심사완료 : 2001년 11월 6일

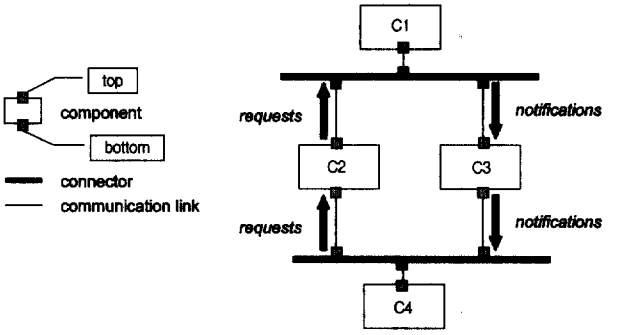
플리케이션 서버 vendor들에 의해 해결 방법이 모색되고 있지만 실질적인 EJB 컴포넌트의 합성 방법의 개발에는 많은 노력을 하고 있지 않다.

따라서 EJB 컴포넌트의 plug-&-play 방식으로 EJB 컴포넌트를 합성할 수 있는 방법에 대한 연구가 필요하다. 컴포넌트 합성을 위해 요구되는 사항은 독립적으로 개발된 컴포넌트들을 합성할 수 있어야 한다. 즉 컴포넌트간의 인터페이스 불일치를 해결할 수 있어야 한다. 또한 컴포넌트들의 유연한 재구성이 가능하여야 한다[5]. 이를 위해 각각의 독립적인 EJB 컴포넌트의 인터페이스를 수정하지 않고 합성할 수 있도록 합성하고자 하는 컴포넌트 간에 중간 매개체가 요구 된다. 이러한 중간 매개체는 독립적인 EJB 컴포넌트간의 상호작용을 기술해 주고 불일치하는 인터페이스를 맞춰주어야 한다. 이에 따라 컴포넌트의 기능과 컴포넌트들간의 상호작용을 분리하여 기술하는 소프트웨어 아키텍처 개념의 도입이 요구된다. 즉 소프트웨어 아키텍처의 주요 기본 구성요소인 컴포넌트, 커넥터, 포트, 링크 중에서 커넥터가 중간 매개체의 역할을 하기 위해 사용된다. 그리고 각 컴포넌트 합성 시점에 유연한 재구성을 위해 EJB 컴포넌트간의 직접적인 메소드 호출 방식 대신 메시지 전달방식의 비동기적인 상호작용 방식이 요구된다. 그러한 점에서 C2(Chiron-2) 아키텍처 스타일[6]은 메시지 전달 방식의 상호작용을 지원하는 대표적인 아키텍처 스타일이라고 할 수 있다. 따라서 메소드 호출 방식으로 밀접하게(tightly coupled) 상호작용하는 EJB 컴포넌트들에 대해 C2 스타일을 바탕으로 메시지 전달방식의 상호작용을 하는 중간 매개체를 두어 각 EJB 컴포넌트를 합성하는 방법을 제시한다. 또한 여러 EJB 컴포넌트들로 구성된 C2 아키텍처는 각 EJB 컴포넌트들이 합성되어 새로운 기능을 수행하게 된다. 궁극적인 컴포넌트의 합성 목적은 새로운 기능을 가진 합성된 컴포넌트를 만들고자 하는 것이므로 C2 아키텍처를 하나의 컴포넌트로 재사용 가능하여야 한다. 이를 위해 여러 EJB들로 구성된 아키텍처를 하나의 기능으로 수행하는 새로운 단일 EJB인 합성 EJB를 생성할 수 있도록 한다. 따라서 EJB들로 구성된 아키텍처를 대표하는 합성 EJB를 생성하는 방법을 제시한다.

2. 관련 연구

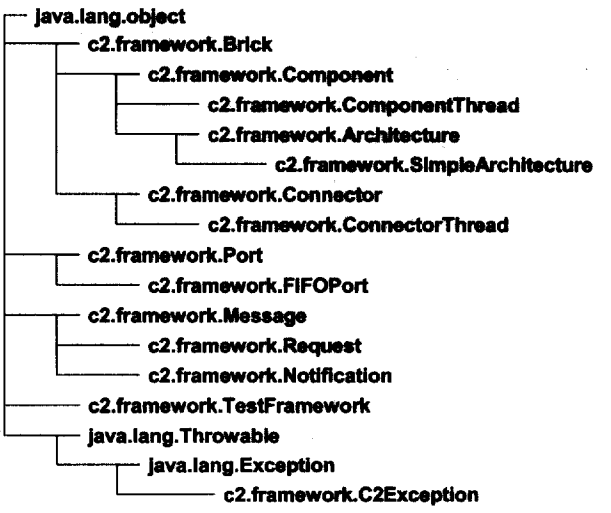
2.1 C2 스타일

C2 스타일은 메시지 기반의 계층적 아키텍처 스타일로서 특정 컴포넌트 패키징 방법이라기 보다 상위 레벨의 시스템 합성 이슈와 기본적으로 관계가 있다. C2 스타일이 컴포넌트 합성에 적합하다고 할 수 있는 중심 개념으로는 메시지 기반의 상호작용과 하위 계층에 대한 독립성이라고 할 수 있다. (그림 1)에서 C2 스타일의 기본 요소는 컴포넌트와 커넥터로서 top과 bottom을 가진다.



(그림 1) C2 아키텍처와 메시지 흐름

컴포넌트의 top은 단일 커넥터의 bottom과 연결될 수 있고 컴포넌트의 bottom은 단일 커넥터의 top과 연결될 수 있다[6]. 하나의 커넥터에 연결될 수 있는 컴포넌트와 커넥터의 수는 제한이 없다. 각 컴포넌트들간의 상호작용은 메시지를 교환함으로써 이루어지고 각 컴포넌트의 top과 bottom에 메시지를 교환하는 인터페이스가 명시된다. 컴포넌트의 top은 상위 컴포넌트로 전달하는 request의 집합과 상위 컴포넌트로부터 전달받는 notification의 집합으로 구성된다. 컴포넌트의 bottom은 하위 컴포넌트로부터 전달받는 request의 집합과 하위 컴포넌트로 전달하는 notification의 집합으로 구성된다. 즉 각 컴포넌트는 top을 통해 상위 컴포넌트와 상호작용하고 bottom을 통해 하위 컴포넌트와 상호작용한다. 그리고 하위계층에 대한 독립성의 원리에 의해 하위 계층의 컴포넌트를 독립적으로 대체할 수 있고 또한 하위 계층의 컴포넌트에 직접적으로 결합되지 않으므로 재사용성이 증가된다[6]. 이러한 C2 개념을 지원하기 위한 객체 지향 컴포넌트와 메시지 프레임워크는 (그림 2)와 같이 자바의 12개 클래스로 구성된다. 이를 이용하여 C2 아키텍처를 자바로 구현할 수 있다. 이와 같은 C2 스타일의 특징을 바탕으로 해서 독립적인 EJB 컴포넌트를 합성하기 위한 방법에 적용될 수 있다.



(그림 2) C2 아키텍처 프레임워크

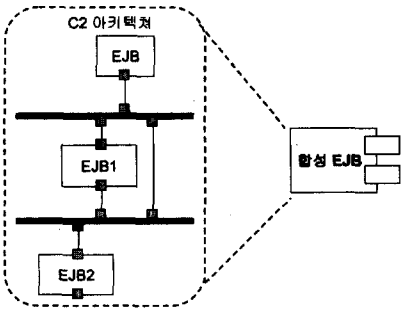
2.2 C2 스타일을 이용한 JavaBeans 컴포넌트 합성 방법

JavaBeans 컴포넌트 모델에 대해 C2 아키텍처 스타일에 따라 컴포넌트 합성 방법을 제시한 연구로서 bean을 C2 컴포넌트와 커넥터로 형성하기 위해 두 가지 방법을 제공한다[7,8]. 첫 번째로는 C2 아키텍처 프레임워크에서 제공되는 클래스들의 서브 클래스를 정의하는 것과 두 번째로는 C2 wrapping 방법을 사용해서 bean을 wrapping하는 것이다. 첫 번째 방법에 의해 생성된 bean은 다른 아키텍처 스타일을 사용하는 아키텍처에서의 재사용성이 없다는 단점이 있다. 두 번째 방법은 C2 wrapper를 생성하는 것으로 기존의 bean을 C2 컴포넌트로 맵핑하거나 변경하지 않고 bean의 속성, 메소드, 이벤트를 추출하여 bean의 인터페이스를 형성한다. bean은 이벤트를 이용해서 상호작용하고 이벤트들이 C2 아키텍처에서 request와 notification 메시지로 맵핑되도록 한다. bean은 C2 스타일에서 정의되고 있는 일반적인 wrapping 모델에 따라 C2 컴포넌트로 wrapping된다. 그러나 이 방법의 한계는 Javabeans 컴포넌트 모델에만 적용할 수 있다는 것이다. 그 이유는 JavaBeans 컴포넌트 모델이 GUI 컴포넌트의 재사용을 위한 컴포넌트 모델이라는 점이다. GUI 컴포넌트의 특징은 비즈니스 로직을 위한 컴포넌트와는 달리 개발자가 코드를 직접 작성하지 않고 정해진 속성, 메소드, 이벤트 편집의 형태로 GUI 컴포넌트의 합성이 가능하다. 그러나 EJB 컴포넌트의 경우 개발자가 작성한 비즈니스 로직에 대해 다양한 메소드들이 정의될 수 있고 이로 인해 다양한 개발자에 의해 작성된 EJB 컴포넌트를 합성하기 위해서는 각 EJB 컴포넌트간의 불일치할 수 있는 인터페이스 타입과 형태를 맞춰 주기 위한 다른 wrapping 방법이 요구된다.

3. C2 스타일을 이용한 EJB 컴포넌트의 합성 방법

3.1 C2 스타일을 이용한 EJB의 합성 구조

본 연구에서 EJB 합성의 목적은 각 독립적인 EJB를 plug-&-play 방식으로 합성하여 만들어지는 하나의 아키텍처를 바탕으로 궁극적으로는 새로운 단일 EJB인 합성 EJB를 생성하기 위한 것이다. (그림 3)은 EJB 합성의 형태와 합성 EJB의 관계를 개념적으로 나타낸 것이다.

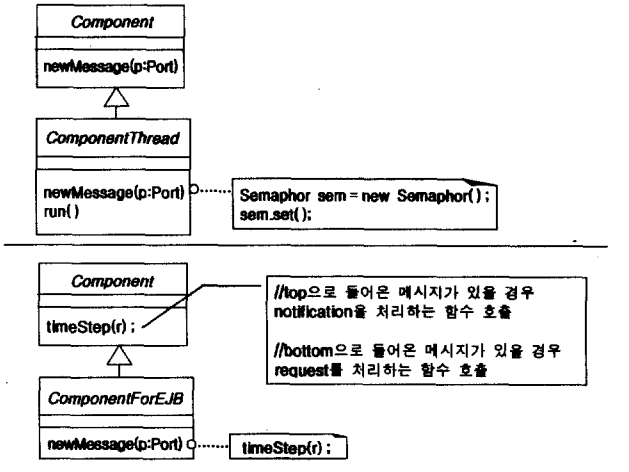


(그림 3) C2 아키텍처와 합성 EJB의 관계

(그림 3)에서 보듯이 각 독립적인 EJB 컴포넌트가 C2 스타일의 형태에 따라 C2 컴포넌트로 맵핑되고 커넥터와 연결된다. 각 EJB 컴포넌트의 상호작용은 C2 스타일의 규칙에 따라 메시지를 주고 받으면서 이루어진다. 이렇게 구성된 아키텍처에 대해 하나의 단일 합성 EJB를 생성하게 되고 합성 EJB의 내부에는 여러 EJB들이 아키텍처의 형태에 따라 상호작용하도록 구성되어 있는 것으로 볼 수 있다.

C2 스타일을 이용한 이러한 형태의 실질적인 EJB 컴포넌트의 합성을 위해 EJB 컴포넌트의 메소드 호출 방식이 메시지 전달 방식의 C2 스타일을 따르는 상호작용 방식으로의 수정이 요구된다. 그러나 EJB 컴포넌트를 plug-&-play 방식으로 합성하기 위해서는 EJB 컴포넌트의 수정 없이 합성될 수 있어야 한다. 따라서 이러한 두 가지 요구 사항을 만족시키기 위해 EJB 컴포넌트를 C2 컴포넌트로 맵핑하는 접속 코드가 요구되고 이를 지원하는 프레임워크가 필요하다.

이를 위해 C2 아키텍처의 Java 구현을 지원하는 기존 C2 아키텍처 프레임워크[6]와의 비교가 필요하다. 기존 C2 아키텍처 프레임워크에서 EJB와 맞지 않는 부분은 쓰레드(thread)에서 수행되는 컴포넌트 또는 커넥터를 위해 사용되는 'Component Thread'와 'Connector Thread'이다. 그 이유는 EJB 명세[3]에 의해 EJB는 쓰레드를 시작, 중지, 재시작할 수 없다고 명시하고 있기 때문이다. 따라서 'Component Thread'와 'Connector Thread'가 EJB의 명세에 위배되는 것으로 'Component Thread'와 'Connector Thread' 클래스는 EJB 합성을 위한 C2 아키텍처 프레임워크에서 제외되어야 한다. 기존 C2 아키텍처 프레임워크에서 이 클래스들이 하는 역할은 컴포넌트의 메시지가 도착하면 처리할 수 있도록 계속 쓰레드로 동작하면서 메시지가 도착하였는지 확인하는 역할을 하는 것으로 이를 대체할 수 있는 새로운 클래스가 요구된다. (그림 4)는 기존 C2 아키텍처 프레임워크의 'ComponentThread' 클래스에서 새로운 메시지가 도착했을 때 호출되는 'newMessage

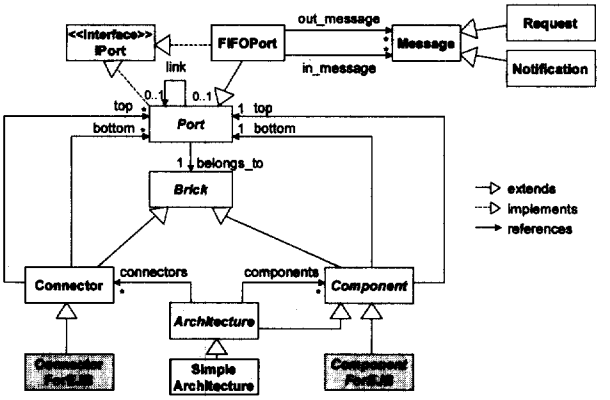


(그림 4) 'ComponentThread' 클래스와 'ComponentForEJB' 클래스 비교

( )'와 EJB 합성을 위한 아키텍처 프레임워크의 'Component-ForEJB' 클래스에서 새로운 메시지가 도착했을 때 호출되는 'newMesage( )'에 대한 형태를 보여 준다.

(그림 4)에서 기존 C2 아키텍처 프레임워크는 새로운 메시지가 왔을 때 단지 저장만 한 후 적당한 시간마다 메시지를 처리하게 된다. 그러나 EJB의 세션 빈(Session Bean)과 엔터티 빈(Entity Bean)은 동기적인 메시지 소비자(Consumer)로서 메시지를 보낸 후 결과를 받아야 하므로 바로 새로운 메시지를 처리하도록 되어야 한다. 따라서 'ComponentFor-EJB' 클래스는 'newMessage( )' 메소드 내에서 'Component' 클래스의 'timeStep( )' 메소드를 호출하여 컴포넌트의 top 인터페이스와 bottom 인터페이스로부터 메시지를 가져와서 메시지 처리 로직을 호출한다.

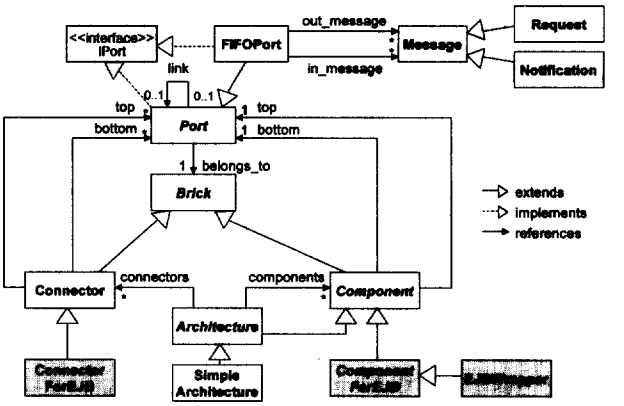
따라서 (그림 5)와 같이 EJB 합성을 지원하기 위해 변경된 C2 아키텍처 프레임워크에 각 컴포넌트와 커넥터에 대하여 새로운 클래스인 'ComponentForEJB'와 'ConnectorForEJB' 클래스를 'Component'와 'Connector' 클래스의 서브 클래스로 둔다. 이렇게 변경된 아키텍처 프레임워크에 의해 아키텍처 상에 존재하는 컴포넌트와 커넥터는 'ComponentForEJB'와 'ConnectorForEJB'의 서브 클래스가 되어야 한다.



(그림 5) 변경된 C2 아키텍처 프레임워크

그러나 C2 아키텍처에서 컴포넌트로 맵핑될 기존의 EJB 컴포넌트는 'ComponentForEJB'의 서브 클래스로 구현되어 있지 않다. 따라서 EJB 컴포넌트들을 수정하지 않고 C2 아키텍처 프레임워크를 만족하는 형태의 합성을 위해서는 'ComponentForEJB'의 서브 클래스가 되는 중간 매개체가 요구된다. 이를 위해 (그림 6)과 같이 C2 아키텍처 프레임워크에 'ComponentForEJB'의 서브 클래스인 'EJBWrapper' 클래스를 두어 각 EJB 컴포넌트가 합성될 때 'EJBWrapper' 클래스의 서브 클래스이면서 특정 EJB컴포넌트에 맞는 EJB wrapper가 구현될 수 있도록 한다.

이에 따라 각 EJB 컴포넌트를 위한 중간 매개체 역할을 하는 EJB wrapper는 EJB의 메소드를 호출할 수 있도록 홈/원격 인터페이스를 가져 오고 EJB의 메소드를 사용하는 C2 request/notification에 대해 해당 EJB의 메소드를 호출하고 그 결과를 다시 메시지 형태로 변환하여 top과 bottom을 통해 전달하도록 함으로써 C2 스타일에 따라 plug-&-play 방식으로 합성될 수 있도록 한다.



(그림 6) 'EJBWrapper' 클래스가 추가된 C2 아키텍처 프레임워크

3.2 EJB 합성을 위한 EJB wrapper의 형태

EJB wrapper는 EJB 컴포넌트를 C2 컴포넌트로 맵핑하기 위해서 EJB 컴포넌트의 메소드 호출 방식의 상호작용을 메시지 방식의 상호작용으로 변환해 주기 위해 필요한 접속 코드이다. EJB wrapper는 각 EJB 컴포넌트의 메시지 처리 로직과 전개된 서버에 따라 홈/원격 인터페이스를 찾는 로직으로 구성된다. EJB 컴포넌트가 C2 아키텍처상의 컴포넌트로 맵핑되었을 때 가지는 인터페이스는 top 인터페이스와 bottom 인터페이스로 구성되고 EJB간의 상호작용은 top과 bottom 인터페이스를 통해 이루어진다. EJB wrapper의 메시지 처리 로직은 top으로부터 notification을 받거나 bottom으로부터 request를 받을 경우에 대하여 각 EJB가 C2 아키텍처 상에서 어떻게 합성되느냐에 따라 여러 가지 형태로 다양하게 존재할 수 있다. 메시지를 받아서 해당 EJB의 메소드가 호출되고 그 결과값을 top이나 bottom으로 메시지를 이용하여 전달할 수 있고 단순한 전달자 역할만을 하여 받은 메시지를 다시 다른 컴포넌트에게 전달하는 역할을 할 수 있다. 따라서 EJB wrapper의 request/notification 처리 로직은 각 EJB 컴포넌트가 수행하는 기능뿐만 아니라 아래와 위 계층에 어떤 컴포넌트가 존재하느냐에 따라서 다양하게 구현된다. EJB가 합성될 때의 아키텍처 구조상에서 상위 EJB의 기능을 사용할 필요가 있을 때 요구되는 메소드가 top 인터페이스의 request와 notification으로 구성된다. 그리고 합성시 아키텍처의 구조상에서 하위 계층의 EJB 기능에 의해 사용되는 메소드가 bottom 인터페이스의 request와 notification으로 구성된다.

EJB wrapper의 request 처리 로직은 C2 컴포넌트의 구조상 하위 계층에서 오는 request 즉 bottom 인터페이스에 정

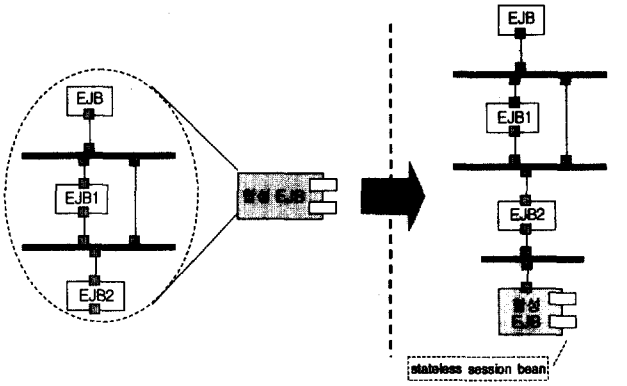
의된 request를 다루는 로직이 들어가게 되는 부분이다. 하위 계층으로부터 받은 request가 상위 EJB의 기능이 요구되는 request일 경우 상위 계층의 request를 생성하여 전달하고 notification을 받는 형태가 된다. 이러한 상위 계층의 컴포넌트에 전달하는 request와 notification이 해당 컴포넌트의 top 인터페이스의 request와 notification으로 정의된다. 또한 하위 계층으로부터 받은 request가 해당 EJB의 기능을 필요로 하는 경우 해당 EJB의 메소드를 호출하고 그 결과값에 대해 하위 계층에 notification을 생성하여 전달하거나 상위 계층에 request를 생성하여 전달하는 로직이 포함된다. 이러한 경우 하위 계층에 대한 notification이 해당 컴포넌트의 bottom 인터페이스의 notification으로 정의되고 상위 계층에 대한 request가 top 인터페이스의 request로 정의된다.

EJB wrapper의 notification 처리 로직은 C2 컴포넌트의 구조상 상위 계층에서 오는 notification 즉 top 인터페이스에 정의된 notification을 다루는 로직이 들어가게 되는 부분이다. 상위 계층으로부터 받은 notification에 따라 받은 notification의 결과를 이용하여 상위 계층에 새로운 request를 생성하여 전달하거나 하위 계층에 notification으로 전달하는 형태가 될 수 있다. 이러한 경우 생성되는 request는 top 인터페이스의 request로 정의되고 생성되는 notification은 bottom 인터페이스의 notification으로 정의된다. 또한 상위 계층으로부터 받은 notification의 결과를 이용하여 해당 EJB의 메소드를 호출하고 하위 계층에 notification을 생성하여 전달하거나 상위 계층의 request를 생성하여 전달하는 로직이 포함된다. 이러한 경우 생성되는 notification은 bottom 인터페이스의 notification으로 정의되고 생성되는 request는 top 인터페이스의 request로 정의된다. 이러한 구조를 기반으로 하여 EJB의 홈/원격 인터페이스와 관계없이 새로운 C2 request/notification을 추가로 정의 할 수 있다.

다음으로 EJB가 가지고 있는 메소드를 사용하기 위해서는 EJB의 클라이언트 관점에서 각 EJB의 홈/원격 인터페이스를 찾는 로직이 필요하다. 각 EJB는 서버와 전개 방법에 따라 EJB의 홈 인터페이스를 찾는 등의 일련의 EJB 연결 부분이 다르게 구현된다. 따라서 EJB 연결 부분은 제삼자가 제공한 EJB를 특정 서버에 전개할 수 있도록 하기 위해서 각 EJB wrapper마다 다르게 구현될 수 있도록 한다. 결과적으로 EJB wrapper를 바탕으로 각각의 독립적인 EJB들을 C2 아키텍처상에서 합성하여 독립적인 EJB들의 기능을 다양한 형태로 쉽게 조합하고 새로운 기능을 수행할 수 있도록 한다.

3.3 합성 EJB 생성

여러 EJB 컴포넌트로 구성된 아키텍처를 하나의 단일 EJB 컴포넌트로 사용할 수 있도록 합성 EJB를 생성한다. (그림 7)은 여러 EJB 컴포넌트로 구성된 아키텍처와 합성 EJB의 관계를 나타낸 것이다.



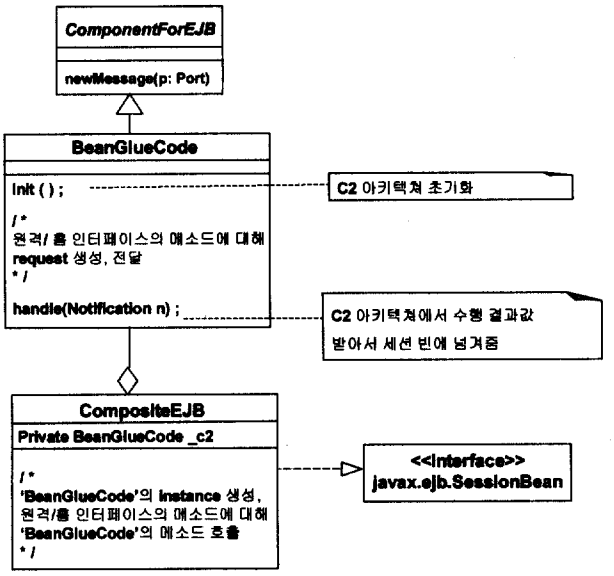
(그림 7) C2 아키텍처와 합성 EJB의 관계

(그림 7)에서 보듯이 세 개의 EJB 컴포넌트(EJB, EJB1, EJB2)로 구성된 C2 아키텍처를 하나의 단일 EJB인 합성 EJB로 생성한다는 것은 하위 계층으로부터 request를 받는 형태의 C2 아키텍처의 구조상 최하위 계층 컴포넌트의 bottom 인터페이스에 정의된 request를 사용하는 EJB를 생성하는 것으로 볼 수 있다. 이러한 관점에서 여러 EJB가 합성되어 구성된 C2 아키텍처의 최하위 계층 컴포넌트에 연결되어 전체 C2 아키텍처의 기능을 사용하는 stateless 세션 빈 형태의 새로운 EJB인 합성 EJB를 생성한다. 합성 EJB의 홈/원격 인터페이스는 최하위 계층 컴포넌트의 bottom 인터페이스의 request와 notification을 다양한 조합으로 결합한 새로운 메소드로 구성될 수 있다. 이렇게 합성 EJB와 C2 아키텍처를 연결해 주기 위해서는 또 하나의 중간 매개체가 요구된다. 이 때 생성되는 중간 매개체는 EJB 컴포넌트가 C2 아키텍처에서 합성될 때 사용되는 EJB wrapper와는 다른 성격을 가지는 것으로 C2 아키텍처와 합성 EJB 의 stateless 세션 빈 사이에서 접속해 주는 역할을 하므로 세션 빈 접속 코드라고 할 수 있다. 세션 빈 접속 코드는 접속하고자 하는 C2 아키텍처의 형세(topology) 정보를 이용하여 컴포넌트와 커넥터 연결 관계를 형성하여 주고 세션 빈의 원격 인터페이스에 정의된 메소드에 대해 세션 빈 대신에 해당 request를 생성해 주고 notification을 받아서 결과값을 넘겨주는 역할을 한다.

(그림 8)은 합성 EJB를 위한 세션 빈 접속 코드인 'BeanGlueCode'와 합성 EJB의 세션 빈인 'CompositeEJB'의 구조를 나타낸 것이다.

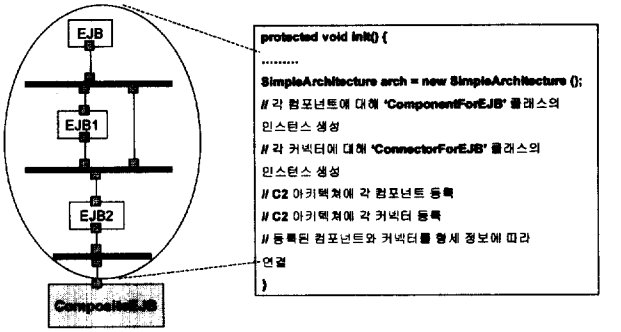
(그림 8)에서 'BeanGlueCode'는 'init( )' 메소드를 통해 C2 아키텍처를 초기화해 주고, 생성된 C2 아키텍처상에 합성 EJB의 홈/원격 인터페이스의 메소드에 따라 request를 생성하고 전달하는 역할을 한다. 'BeanGlueCode'의 'handle( )' 메소드는 notification의 형태로 C2 아키텍처로부터 전달되는 결과값을 합성 EJB 세션 빈에 넘겨준다. 합성 EJB 세션 빈인 'CompositeEJB'의 홈/원격 인터페이스 메소드에 대한 구현은 'BeanGlueCode'의 인스턴스를 생성하여 C2 아키텍처를 형성하는

'init( )' 메소드를 호출하는 부분과 생성된 인스턴스를 이용하여 합성 EJB의 홈/원격 인터페이스의 메소드를 실제로 구현하는 'BeanGlueCode'의 메소드를 호출하는 부분으로 구성된다.



(그림 8) 세션 빈 접속 코드와 합성 EJB간의 관계

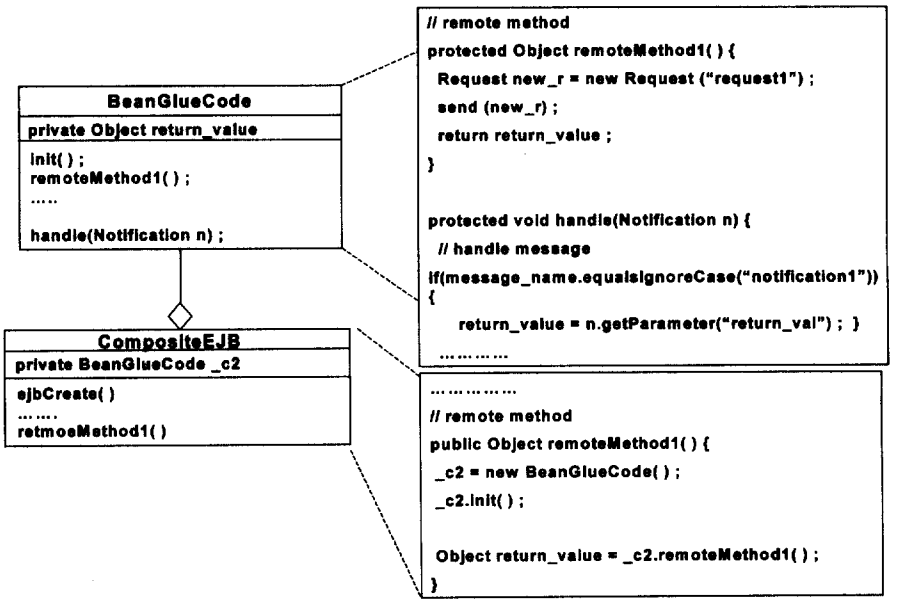
(그림 9)는 'BeanGlueCode'의 'init( )' 메소드의 세부 구성에 대해 나타낸 것이다. (그림 9)에서 보듯이 'BeanGlueCode'의 'init()' 메소드에서는 현재 합성 EJB로 생성될 C2 아키텍처의 형세 정보를 바탕으로 각 컴포넌트와 커넥터의 인스턴스를 생성하고 컴포넌트와 커넥터를 연결해 주는 로직을 구현한다. 이 때 'BeanGlueCode' 클래스도 하나의 컴포넌트로 간주하여 C2 아키텍처의 최하위 계층 컴포넌트 인스턴스와 'BeanGlueCode' 클래스 인스턴스를 임의의 커넥터에 의해 연결하는 로직이 포함된다.



(그림 9) 세션 빈 접속 코드의 'init()' 메소드 구성

또한 합성 EJB의 세션 빈에서는 홈/원격 인터페이스의 메소드에 대해 세션 빈 접속 코드의 대응되는 메소드를 호출하고 세션 빈 접속 코드의 메소드에서는 대응하는 request를 생성하여 전달하고 결과값을 받아서 합성 EJB의 세션 빈에 전달하는 로직이 요구된다.

(그림 10)은 합성 EJB인 'CompositeEJB'의 원격 인터페이스 메소드인 'remoteMethod1( )'에 관련된 세션 빈 접속 코드인 'BeanGlueCode'와 'CompositeEJB'의 구현을 나타낸 것이다. (그림 10)에서 보듯이 'BeanGlueCode'에서 'CompositeEJB'의 메소드 호출이 C2 아키텍처상의 적절한 request인 'request1'으로 맵핑되도록 합성 EJB의 홈/원격 인터페이스 메소드인 'remoteMethod1'에 대해 해당되는 request를 생성해 주고 전달하는 로직을 가진다. 그리고 합성 EJB의 'remoteMethod1'과 같이 결과값을 요구하는 메소드에 대해서는 해당 request에 대한 수행 결과가 요구되므로 대응되는 no-



(그림 10) 합성 EJB와 세션 빈 접속코드의 원격 인터페이스 메소드 구현 예

tification인 'notification1'의 파라미터 형태로 전달된 결과값을 해당 속성값(return_value)에 설정하는 로직을 가진다. 'CompositeEJB'의 각 메소드에서는 C2 아키텍쳐에 대한 연결 역할을 하는 'BeanGlueCode'의 인스턴스를 생성하고 C2 아키텍쳐를 형성해 주는 'init( )' 메소드를 호출하고 대응되는 메소드들을 호출하도록 구현된다.

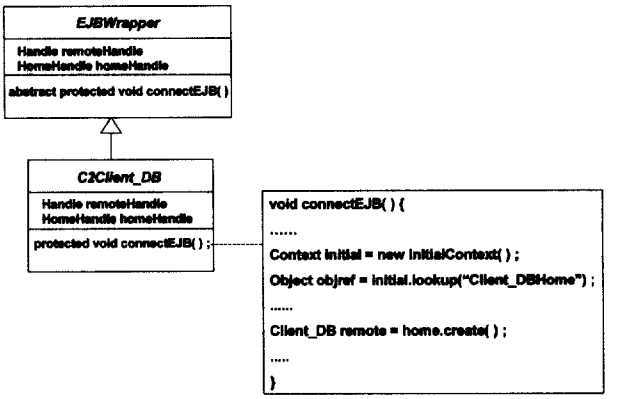
4. 사례 적용

4.1 C2 스타일을 이용한 EJB 합성

본 절에서는 앞에서 제안한 합성 방법을 간단한 예제에 적용한 것에 대해 기술한다. 예제 적용은 본 방법과 병행해서 개발되고 있는 EJB 컴포넌트 조립 도구를 이용한다. EJB 컴포넌트 조립 도구는 EJB 컴포넌트를 drag-&-drop 방식으로 조립할 수 있도록 하고 명세 편집기를 통하여 작성된 명세를 바탕으로 EJB wrapper와 합성 EJB를 자동 생성해 준다. 이 도구를 이용하여 적용할 예제는 사용자 정보를 이용하여 비만도를 측정하는 기능을 수행하기 위해 두 EJB를 합성한 아키텍쳐다. 예제 아키텍쳐는 사용자의 정보를 삽입, 삭제, 갱신, 선택 등의 기능을 수행하는 'Client_DB' EJB 컴포넌트와 키와 몸무게 정보를 바탕으로 비만도를 계산하는 기능을 수행하는 'Checkfat' EJB 컴포넌트를 합성하여 사용자의 정보를 바탕으로 비만도를 계산하는 기능을 수행한다.

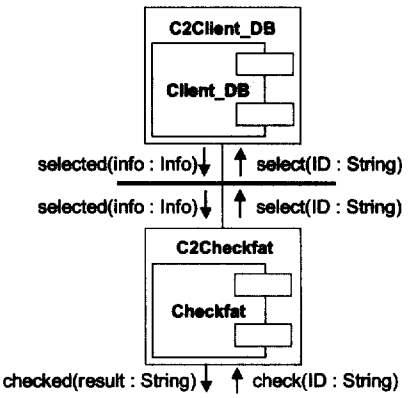
이러한 기능 수행을 위해 'Client_DB' EJB 컴포넌트의 기능은 'Checkfat' EJB 컴포넌트에게 비만도를 계산하는데 필요한 사용자 정보를 제공하는 역할을 하고 'Checkfat' EJB 컴포넌트의 기능은 'Client_DB' EJB 컴포넌트로부터 사용자 정보를 얻어 와서 비만도를 계산하는 역할을 한다. 각 EJB 컴포넌트의 역할에 따라 'Client_DB' EJB 컴포넌트는 C2 아키텍쳐 상에서 최상위 컴포넌트로 합성되어야 하고 이를 C2 아키텍쳐상에서 wrapping하는 역할을 하는 'C2Client_DB' EJB wrapper는 top 인터페이스를 가지지 않고 bottom 인터페이스만을 가진다. 'C2Client_DB' EJB wrapper는 하위 계층에서 전달되는 request에 대해 'Client_DB' EJB 컴포넌트의 메소드를 호출하고 하위 계층의 컴포넌트에게 notification을 생성하여 전달하는 로직을 가진다. 'Checkfat' EJB 컴포넌트는 C2 아키텍쳐 상에서 최하위 컴포넌트로 합성되어야 하고 이를 C2 아키텍쳐 상에서 wrapping하는 역할을 하는 'C2Checkfat' EJB wrapper는 top 인터페이스와 bottom 인터페이스를 가지고 있어 각각에 대해 request/notification 처리 로직을 가진다.

이렇게 구성된 아키텍쳐에 따라 각 EJB wrapper에서는 먼저 EJB 컴포넌트의 홈/원격 인터페이스를 가져 오는 로직이 수행되어야 한다. (그림 11)은 'C2Clinet_DB' EJB wrapper에서 J2EE 서버에 전개된 'Client_DB' EJB 컴포넌트의 홈/원격 인터페이스를 가져 오는 로직에 대한 구현 예로서 EJB 컴포넌트 조립 도구의 어플리케이션 서버 정보를 바탕으로 자동 생성되는 부분이다.



(그림 11) 'C2Client_DB' EJB wrapper의 'connectEJB( )' 메소드 구현 예

각 서버에 따라 EJB 컴포넌트의 홈/원격 인터페이스를 가져 오는 로직이 다르므로 (그림 11)에서처럼 C2 아키텍쳐 프레임워크의 'EJBWrapper' 클래스에서는 해당 로직이 구현된 메소드인 'connectEJB( )' 메소드를 추상 메소드로 정의하고 각각의 특정 EJB wrapper에서 전개된 서버에 따라 'connectEJB( )'에 대한 구현을 함으로써 합성되는 EJB 컴포넌트를 다양한 서버에 전개될 수 있도록 한다.



(그림 12) 'check' request에 대한 메시지 흐름도

(그림 12)는 최하위 컴포넌트에 대한 EJB wrapper인 'C2-Checkfat'에서 bottom 인터페이스를 통해 사용자의 ID 정보를 입력으로 하여 비만도를 체크하는 기능을 수행하고자 하는 'check' request가 들어왔을 때에 대한 메시지 흐름을 나타낸 것이다. (그림 12)에서 'check' request가 들어 오면 파라미터값으로 들어온 사용자 ID 값을 이용하여 비만도를 계산하는데 필요한 사용자 정보를 얻어 와야 한다. 이를 위해 'C2Client_DB' EJB wrapper에 'select' request를 생성하여 top 인터페이스를 통해 전달한다. 'select' request를 받은 'C2Client_DB' EJB wrapper에서는 사용자 정보를 얻어오기

위해 'Client_DB' EJB 컴포넌트의 해당 메소드인 'select( )' 메소드를 호출하여 결과값으로 사용자 정보를 얻어온다. 그리고 사용자 정보를 파라미터로 하여 'selected' notification을 생성하여 bottom 인터페이스를 통해 전달한다. 'C2Checkfat' EJB wrapper에서는 top 인터페이스를 통해 'selected' notification을 받아서 파라미터로 넘어온 사용자 정보를 이용하여 'Checkfat' EJB 컴포넌트의 비만도 계산 메소드인 'check( )'를 호출하여 결과를 얻어와서 결과값을 파라미터로 하여 'checked' notification을 생성한 후 bottom 인터페이스를 통해 전달한다.

(그림 13)은 'C2Client_DB' EJB wrapper의 request 처리 로직의 일부로서 본 조립도구의 명세 편집기를 이용하여 작성된 명세를 이용하여 자동 생성된 로직의 일부분이다. (그림 13)에서의 로직은 request가 'select'인 경우 request에서 파라미터를 얻어 와서 'Client_DB' EJB의 원격 인터페이스를 통해 해당 메소드인 'select' 메소드를 호출하고 notification을 생성하여 하위 계층에 전달한다.

```
public synchronized void handle(Request r)
{
  .....
  Client_DB remoteClient = (Client_DB) getRemoteInterface( ) ;
  .....

  if (message_name.equalsIgnoreCase("select"))
  {
    try
    {
      //get parameter
      ..............
      Info info = remoteClient.select(param_value) ;

      // generate a notification
      n = new Notification ("selected") ;
      ..............
      send(n) ;
    }
    ........
    .......
  }
```

(그림 13) 'C2Client_DB'의 'select' request 처리 로직

(그림 14)의 왼쪽 상자는 'C2Checkfat' EJB wrapper의 'check' request 처리 로직의 일부분이고 오른쪽 상자는 'C2-Checkfat' EJB wrapper의 'selected' notification 처리 로직의 일부분으로 (그림 13)에서와 같이 자동 생성된 로직의 일부분이다.

(그림 14)에서 'C2Checkfat' EJB wrapper의 request 처리 로직에서는 request가 'check'인 경우 'Client_DB' EJB의 'select( )' 메소드를 이용하여 사용자의 정보를 얻어 와야 한다. 이를 위해 'C2Client_DB' EJB wrapper에서 요구하는 request의 형태로 'select' request를 생성하여 전달한다. (그림 14)의 'C2Checkfat' EJB wrapper의 notification 처리 로직에서는 notification이 'selected'인 경우 'select( )' 메소드의 호출된 결과가 파라미터로 전달되므로 전달된 notification의 파라미터를 'Checkfat' 메소드의 파라미터에 맞도록 변환하는 과정을 거쳐 'Checkfat'의 실제 메소드를 호출한다.

```
public synchronized void handle(Request r)
{
  .....
  .....
  if (msg_name.equalsIgnoreCase("check"))
  {
    .....
    new_r = new Request ("select") ;
    .....
    send (new_r) ;
  }
  .....
}
```

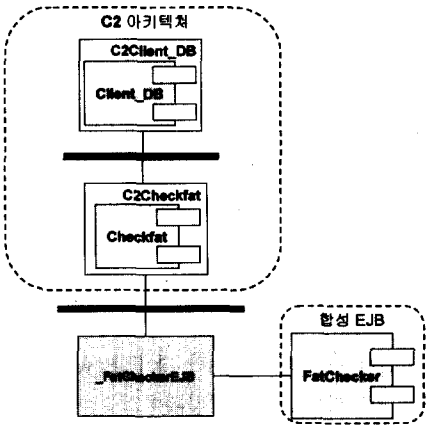
```
public synchronized void handle(Notification n)
{
  .....
  Checkfat remoteCheck
        = (Checkfat) getRemoteInterface( ) ;
  .....
  if (msg_name.equalsIgnoreCase("selected"))
  {
    try {
      // get parameters
      .....
      int i = remoteCheck.check(d1, d2) ;

      .....
      // generate a notification
      n = new Notification ("checked") ;
      .....
      send(n) ;
    }
    .....
  }
  .....
```

(그림 14) 'C2Checkfat' EJB wrapper의 request와 notification 처리 로직의 일부분

4.2 합성 EJB 생성

본 절에서는 이전 절에서 합성된 C2 아키텍처를 하나의 합성 EJB로 생성하는 과정에 대한 예를 기술한다. 합성 EJB와 C2 아키텍처의 구조는 (그림 15)와 같다. (그림 15)에서 'FatChecker'는 두개의 EJB로 구성된 C2 아키텍처의 기능을 가지는 합성 EJB에 해당되고 '_FatCheckerEJB'는 'Fat-Checker' EJB의 메소드에 대해 해당 request/notificaiton을 생성하여 전달하는 세션 빈 접속 코드에 해당된다. 합성 EJB의 홈/원격 인터페이스의 메소드는 C2 아키텍처를 구성하고 있는 'Client_DB' EJB와 'Checkfat' EJB가 가지고 있는 메소드와 독립적으로 정의 될 수 있고 두 EJB의 인터페이스 변경에 영향을 받지 않는다. 이러한 독립적인 합성 EJB 생성을 위해 '_FatCheckerEJB'는 'FatChecker' EJB의 메소드 호출에 대해 request/notification으로 변환하여 C2 아키텍처에 전달해 주는 중간 매개 역할을 하도록 구현된다.



(그림 15) 합성 EJB와 C2 아키텍처 관계 예

(그림 16)의 (a), (b)는 (그림 15)의 'FatChecker' EJB와

'_FatCheckerEJB'와의 관계를 나타낸 것으로 내부적으로 'FatChecker' EJB의 stateless 세션 빈인 'FatCheckerEJBBean'이 세션 빈 접속코드인 '_FatCheckerEJB'와 관련된다.

```
(a)
public class FatCheckerEJBBean
        implements SessionBean {
 ..........
 private _FatCheckerEJB _f;
 ..........
 public void ejbCreate( )
 {
  .....
}
//remote interface methods
public int check(String id)
{
   ..................
   int t = _f.check(id) ;
   return t ;
}
```

```
(b)
public class _FatCheckerEJB
     extends c2.framework.ComponentForEJB {
 ..........
 protected void init ( )
 {
   ..........
   // C2 아키텍쳐 초기화
   .....
 }

protected int check(String id)
 {
   in = 0 ;
   Request new_r = null ;
   new_r = new Request ("check ") ;
   ..........
   send (new_r) ;
   return in ;
 }
 ..................
```

(그림 16) 'FatCheckerEJBBean'과 '_FatCheckerEJB' 구현 예

(그림 16 (a))에서 'FatCheckerEJBBean'의 원격 인터페이스 메소드인 'check( )'에 대한 구현은 C2 아키텍처에 request/notification을 생성하고 전달하는 역할을 하는 '_FatChecker-EJB'의 해당 메소드인 'check( )'를 호출하는 로직으로 구현된다. (그림 16 (b))의 'init( )' method에서는 'FatChecker' EJB를 위한 C2 아키텍처를 생성하고 'check( )' method에서는 'FatCheckerEJBBean'의 'check( )'에 대해 대응되는 request인 'check'를 생성해서 전달한다.

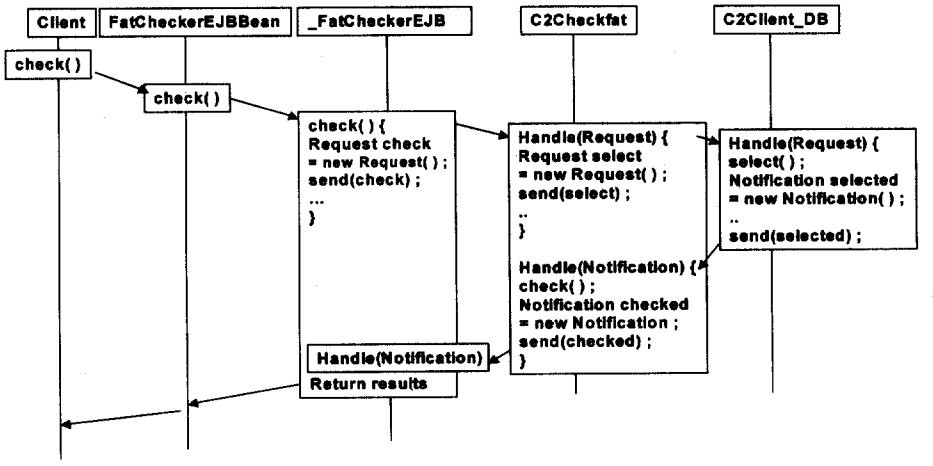
(그림 17)은 합성 EJB인 'FatChecker' EJB의 'check( )' 메소드를 호출했을 때의 'FatCheckerEJBBean', '_FatCheckerEJB', 'C2Checkfat', 'C2Client_DB'의 호출 관계를 나타낸 것이다.

(그림 17)에서 클라이언트가 'FatChecker' EJB의 메소드인 'check( )'를 호출했을 때 'FatCheckerEJBBean'의 'check( )'에 의해 '_FatCheckerEJB'의 'check( )'가 호출된다. 'check( )' 메소드내에서는 'C2Checkfat'의 request인 'check'를 생성해서 전달하고 'C2Checkfat'의 request 처리 로직은 'check' request에 대해 'C2Client_DB'의 'select' request를 생성해서 전달한다. 그리고 'C2Client_DB'의 request 처리 로직에서는 'select' request에 대해 'Client_DB' EJB의 'select( )' 메소드를 호출하고 결과값을 'selected' notification으로 전달한다. 'C2Checkfat'의 notification 처리 로직은 'selected' notification을 받아서 선택된 값들을 얻어와서 'Checkfat' EJB의 'check()' 메소드를 호출한다. 그 결과에 대해 'checked' notification을 생성하여 전달하고 마지막으로 '_FatCheckerEJB'는 값을 받아서 'FatCheckerEJBBean'에게 전달한다. 이러한 형태로 각 독립적인 EJB들을 수정없이 합성하여 새로운 EJB 컴포넌트를 생성할 수 있다.

5. 결 론

본 논문에서는 C2 스타일을 기반으로 EJB 컴포넌트를 합성하는 방법에 대하여 설명하였다. 제안한 방법은 C2 스타일에 따라 EJB 컴포넌트들을 합성하기 위해 EJB wrapper를 생성하고 여러 EJB 컴포넌트들로 구성된 아키텍처를 하나의 단일 EJB 컴포넌트로 사용하기 위해 합성 EJB를 생성한다. 제안한 방법에서는 EJB 컴포넌트에 대해 EJB wrapper를 생성함으로써 수행 환경에 관계없이 다양한 EJB 컴포넌트들이 plug-&-play 방식으로 합성될 수 있도록 한다. 또한 합성된 EJB 컴포넌트들의 홈/원격 인터페이스의 메소드들을 바탕으로 추가적인 C2 request와 notification을 정의할 수 있도록 하고 합성 EJB를 생성함으로써 여러 EJB 컴포넌트로 구성된 C2 아키텍처를 하나의 EJB 컴포넌트로 재사용 가능하게 한다. 결과적으로 본 논문에서 제안하는 방법을 통해 개발자들은 여러 EJB 컴포넌트들을 합성해서 새로운 기능을 가진 EJB 컴포넌트를 만들 수 있다.

본 연구와 관련하여 현재 진행되고 있는 작업으로는 EJB 컴포넌트를 plug-&-play 방식으로 조립하기 위한 C2 ADL (Architecture Description Language)을 정의하였고 이를



(그림 17) 'FatChecker' EJB의 'check()' 메소드 호출시의 호출 관계

바탕으로 본 논문에서 제안한 방법을 지원하는 시각적 합성 도구인 EJB 컴포넌트 조립도구의 프로토타입을 구현하였다. 이 도구를 이용하여 EJB 컴포넌트들은 C2 아키텍처상에 drag-&-drop 방식으로 합성될 수 있고 정의된 ADL을 바탕으로 명세를 작성할 수 있도록 한다. 그리고 작성된 명세를 바탕으로 자동으로 EJB wrapper 코드를 자동으로 생성하여 주고 합성 EJB 또한 자동으로 생성해 준다. 또한 본 EJB 컴포넌트 조립도구 개발과 병행해서 EJB 컴포넌트 생성 도구의 프로토타입도 구현하였다. EJB 컴포넌트 생성 도구는 영역 모델러, 컴포넌트 모델러, 코드 생성기, DB 연계기, 응용 컴포넌트 추출기로 구성된다.

향후 연구 방향으로는 좀더 실질적인 EJB 컴포넌트를 이용하여 복잡하고 큰 시스템에 적용하여 본 방법의 문제점을 보완하고 JSP(Java Server Pages)와 같은 클라이언트 프로그램과 EJB 컴포넌트로 구성된 C2 아키텍처를 결합하여 하나의 독립적인 어플리케이션을 생성하는 방법에 대한 연구가 필요하다. 또한 본 방법의 보완으로 EJB 컴포넌트 조립도구 기능의 지속적인 반영이 이루어질 것이고 본 도구에서 현재 지원되고 있는 어플리케이션 서버인 J2EE와 Weblogic 뿐만 아니라 WebSphere, GemStone/J등으로 지원범위를 확대해 나갈 것이다.

참 고 문 헌

[1] SEI, "Component-Based Software Development/COTS Integration," 1997.

[2] L. Wilkes, "Application Connection," CBDi Forum Journal, September, 1999.

[3] Sun Microsystems Inc., "Enterprise JavaBeans Specifications," at URL : http : //www.javasoft.com

[4] Oreizy, P., Medvidovic, N., Taylor, R. N., and Rosenblum, D. S., "Software Architecture and Component Technologies : Bridging the Gap," Digest of the OMG-DARPA-MCC Workshop on Compositional Software Architectures, Monterey, CA, January, 1998.

[5] Mezini, M., Seiter, L. & Lieberherr, K., "Software Architectures and Component Technology : The State of the Art in Research and Practice," Kluwer, 2000.

[6] 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, pp.390-406, June, 1996.

[7] Rosenblum, D. S. and Natarajan, R., "Supporting architectural concerns in component-interoperability standards," IEE Proceedings- Software, Volume : 147 Issue : 6, pp.215-223, Dec. 2000.

[8] Natarajan, R. and Rosenblum, D. S., "Merging Component Models and Architectural Styles," Proceeding of the Third International Software Architecture Workshop(ISAW-3), November, 1998.

최 유 희

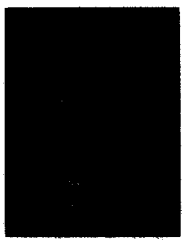
e-mail : yhchoi@etri.re.kr

1999년 부산대학교 컴퓨터공학과(학사)

2001년 부산대학교 컴퓨터공학과(석사)

2001년~현재 한국전자통신연구원 컴퓨터·소프트웨어연구소 S/W공학연구부 연구원

관심분야 : 컴포넌트 기반 소프트웨어 개발 방법론, 소프트웨어 아키텍처, 분산 컴포넌트, 소프트웨어 재사용 등임.



권 오 천

e-mail : ockwon@etri.re.kr

1994년 영국 Teesside 대학교 대학원 S/W 공학과 공학석사

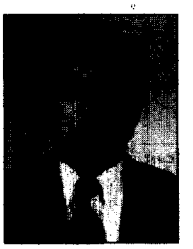
1998년 영국 Durham 대학교 대학원 전산과학과공학 박사

1991년 미국 RTP IBM 연구소 객원 연구원

1993년 시스템공학연구소 선임연구원

현재 한국전자통신연구원 컴퓨터·소프트웨어연구소 S/W공학연구부 책임연구원

관심분야 : 재사용, CBD, Product Line, Web Services 기술 등



신 규 상

e-mail : gsshin@etri.re.kr

1981년 성균관대학교 통계학과 학사

1983년 서울대학교 대학원 계산통계학과 이학석사

2001년 충남대학교 대학원 컴퓨터과학과 이학 박사

1983년 시스템공학연구소 연구원

1987년 시스템공학연구소 선임연구원

1997년 한국전자통신연구원 책임연구원

현재 한국전자통신연구원 컴퓨터·소프트웨어연구소 S/W공학연구부 컴포넌트공학연구팀 팀장

관심분야 : CASE, S/W 컴포넌트 기술, Web Services 기술, 멀티미디어 시스템