논·문 16-3-1

C2 아키텍처에서 Adapter패턴을 이용한 EJB 합성 기법

(An Approach to Composite Techniques of EJB using Adapter Pattern on the C2 Architecture)

정 화 영*, 송 영 재**
경희대학교 전자계산공학과
jmichael@hanmir.com*
yjsong@khu.ac.kr**

요 약

최근 소프트웨어 개발분야에서는 컴포넌트 기반 개발방법론(CBD)에 관한 효율성 및 필요성이 입증되면서 이를 도입하고자 활발히 연구되고 있다. 이에 따라, 컴포넌트들을 쉽게 조립 및 운용할 수 있는 합성방법들이 제시되고 있으며, 이들 중 C2(Chiron-2) 스타일 아키텍처는 GUI 방식을 따르는 쓰레드 기반의 메시지전달방법에 의한 컴포넌트 합성기법이라는 점에서 많은 관심을 받고 있다. 그러나, 서버측 컴포넌트 모델인 EJB(Enterprise Java Beans)의 경우, 쓰레드를 포함할 수 없고 직접적인 메소드 호출방식 따름으로써 쓰레드 기반의 메시지 호출방식에 의한 컴포넌트 합성을 위해서는 EJB를 위한 C2 프레임워크의 수정이 불가피하다.

따라서, 본 논문에서는 C2 스타일 아키텍처에서의 EJB 컴포넌트를 적용할 수 있는 기법으로서, 디자인 패턴을 이용한 쓰레드 기반의 컴포넌트 합성기법을 제안하고자한다. 즉, 디자인패턴 중 구조패턴의 하나인 Adapter패턴을 이용하여 EJB 컴포넌트를 C2 스타일 아키텍처에 적용할 수 있도록 하였다.

1. 서론

소프트웨어를 효율적으로 개발하기 위한 노력은 객체지향, 디자인 패턴, CBD등의 기법들을 제시하였다. 특히, 디자인 패턴은 객체지향 소프트웨어에서 경험을 활용하기 위해 제안된 기술이며[1], 소프트웨어 개발시의 반복적인 부분과 많은 문제점을 야기할 수 있는 프로그램의 오류부분을 분류하고 구조화함으로써 최적상태의 효율적인 구현을 가능하게 하였으며 재사용성을 높였다. 또한, 컴포넌트 기반 개발방법(CBD)은 소프트웨어를 하나의 단위로 보고 이를 합성하여 새로운 소프트웨어를 개발하는 방법으로서, 현재 많은 연구 및 도입되고 있다. 컴포넌트 기술은 소프트웨어 프로그래밍에서 하드웨어 개

발 환경처럼 소프트웨어 Plug-and-Play 방식으로 시스템을 구축하는 '합성을 통한 시스템 구축'으로의 전환을 목적으로 한다[2, 3]. 따라서, CBD를 효과적으로 지원하기 위해서는 응용 컴포넌트들이 서로 정확하게 결합하여 작동할 수 있는 아키텍처를 기반으로 컴포넌트의 생성과 합성작업이 이루어질 수 있어야한다[4].

이를 위하여, 컴포넌트 합성방법에서는 컴포넌트간의 인터페이스 불일치를 해결할 수 있어야하며, 독립적인 컴포넌트의 메소드 수정 없이 합성할 수 있어야 한다[5]. 아키텍처 기반기술 중, GUI방식인 쓰레드 기반의 메시지 호출방법의 컴포넌트 합성기법으로는 C2 스타일 아키텍처[6]를 들 수 있으며, 컴포넌트간의 직접적인 메소드 호출방식이 아닌 메시지 전달방식의 비 동기적인 상호작용을 지원하는 대표적인 구조라 할 수 있다. 그러나, 서버측 컴포넌트 모델인 EJB의 경우는 자체적으로 쓰레드를 호출 및 운용할 수 없으며[7] 직접적인 메소드 호출방식을 택하고 있어서 C2 스타일의 적용을 위해서는 수정이 필요하다.

따라서, 본 논문에서는 EJB 컴포넌트에 대한 C2 아키텍처의 적용기법을 제시하고자 한다. 이에 관한 효과적인 구성을 위하여 디자인패턴 중 구조화패턴에 속하는 Adapter 패턴을 이용하여 EJB와 C2 아키텍처 프레임워크 사이의 연결부분을 구성한다. 이는, C2 아키텍처 프레임워크 중 EJB가 적용될 수 없었던 컴포넌트 쓰레드부분을 Adapter 패턴을 이용함으로서, EJB를 위한 컴포넌트 쓰레드를 생성 및 운용하여 컴포넌트 요소의 수정 없이 C2 스타일 아키텍처에 적용 및 활용할 수 있도록 하였다.

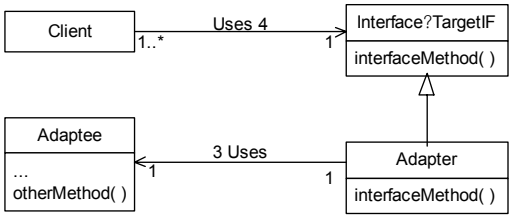
본 논문은 다음과 같이 구성되어 있다. 2장에서는 기존의 디자인 패턴중 Adapter 패턴과 C2 아키텍처에 의한 컴포넌트 합성기법에 대한 연구들을 살펴보고자 한다. 3장에서는 C2 아키텍처에 따른 EJB 컴포넌트의 합성에 관한 기존의 연구된 제안기법을 살펴본다. 이를 기반으로 4장에서 Adapter 패턴을 이용하여 본 논문에서 제시한 EJB컴포넌트 적용기법을 나타내며, 5장에서는 제안한 기법을 EJB 컴포넌트에 적용한 예제를 보임으로서, 본 논문이 제안한 기법이 실제 직접 적용될 수 있음을 보인다. 6장에서는 기존의 연구기법과 제안한 기법과의 차이점을 분석하고 제안된 기법이 효율적임을 나타낸다. 마지막으로 6장에서는 결론 및 향후 연구 방향으로 끝을 맺고자 한다.

2. 관련연구

2.1 Adapter 패턴

디자인 패턴은 소프트웨어 설계에 관한 추상화 형태의 해결방법으로서, 클래스의 일반적인 구조, 역할, 상호관계, 클래스들 사이의 책임에 대한 내용 등을 가진다. 이에 관한 활용성을 증가시키기 위해 Gamma[8]는 생성, 구조, 행위패턴으로 분류하고있다.

Adapter 패턴은 이들 중 구조화패턴에 속하며, <그림 1>과 같이 이미 제공되어 있는 것과 필요한 것 사이의 간격을 연결하는 패턴을 말한다[9].

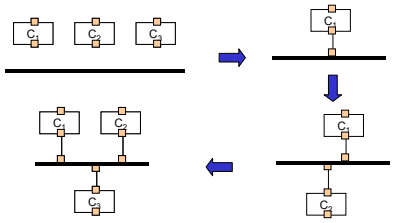


<그림 1> Adapter Pattern

TargetIF는 필요로 하는 기능을 제공하는 역할로서 Interface로 구성하며, Adaptee는 적합되는 측으로 이미 준비되어있는 기능을 나타내고 Adapter는 적합하는 측으로 Adaptee의 기능을 사용해서 Target역할을 충족시키는 부분을 나타낸다.

2.2 C2 아키텍처에 의한 컴포넌트 합성 기법

쓰레드 기반의 메시지 전달방식에 의한 독립적인 컴포넌트 합성기법으로서 대표적으로 C2 스타일 아키텍처를 들 수 있다. 이에 관한 기본원칙은 메시지기반의 컴포넌트간 통신, 멀티 쓰레드, 각 계층의 독립성, Message Routing Connector를 통한 컴포넌트들의 연결구성, GUI 소프트웨어 요구사항 지원 등이다. 또한, 분산 및 이 기종 환경, 분할된 주소공간을 갖지 않는 환경에서의 컴포넌트실행, 다중사용자 및 다중사용자 툴킷, 실행시간에서 변화될 수 있는 동적 구조 등에 적합하다[10]. C2 스타일의 컴포넌트 조립은 다음 <그림 2>에서와 같이 각각의 커넥터를 중심으로 C2 컴포넌트들을 추가, 삭제, 재연결 할 수 있다[11].

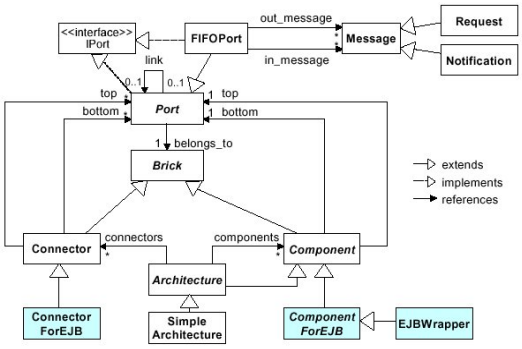


<그림 2> C2 컴포넌트 조립

3. C2 아키텍처에서 EJB합성을 위한 기존의 연구기법

C2 아키텍처에서는 EJB 적용에 맞지 않는 컴포넌트와 커넥터를 위한 쓰레드를 사용하고 있다. 또한, EJB 컴포넌트의 수정 없이 합성될 수 있어야한다.

따라서, 최유희[5]의 연구기법에서는 다음 <그림 3>과 같이 EJB 적용을 위한 C2 아키텍처 부분을 변경하였다. 즉, C2 아키텍처 프레임워크에서 컴포넌트와 커넥터의 메시지 핸들링을 위한 쓰레드 부분인 Component Thread와 Connector Thread를 각각 ComponentForEJB와 ConnectorForEJB로 변경하였다.



<그림 3> EJB 적용을 위한 기존의 C2 아키텍처 프레임워크 변경기법

또한, EJB 컴포넌트를 위한 중간 매개체 역할을 하는 EJBWrapper를 두어 EJB의 홈/원격 인터페이스를 가져와서 메소드를 호출하며, 그 결과를 메시지 형태로 변환하여 C2 아키텍처에 따라 합성할 수 있도록 하였다.

그러나, 이러한 기법은 EJB 적용을 위하여 컴포넌트 조립합성기법인 C2 아키텍처의 프레임워크 변경이 필요하였다. 또한, EJB 컴포넌트들을 위한 EJBWrapper가 별도로 필요하며, 이 부분에서 메소드 호출 및 결과를 메시지 형태로 변환하여야만 C2 아키텍처의 합성기법에 따라 운용이 가능하였다.

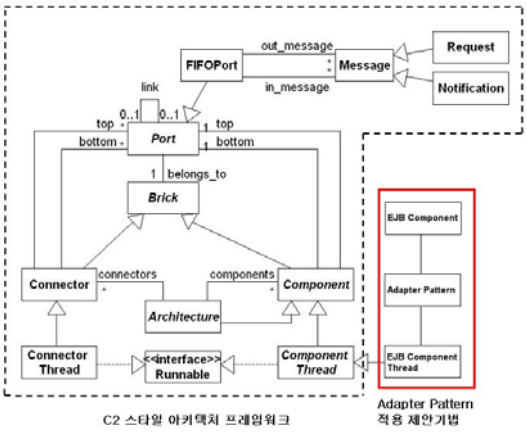
따라서, 본 논문에서는 기존의 C2 아키텍

처와 EJB를 수정하지 않으면서 C2의 합성기법에 따라 EJB 컴포넌트를 운용할 수 있는 Adapter 패턴의 적용기법을 제안하고, 예제를 통해 실제 적용될 수 있음을 나타내고자 한다.

4. C2 아키텍처에서 Adapter 패턴을 이용한 EJB 컴포넌트 합성 기법

쓰레드를 기반으로 한 메시지 호출방식의 컴포넌트 합성모델인 C2 스타일 아키텍처에서는, 쓰레드를 호출할 수 없으며 직접적인 메소드 호출방식의 서버 컴포넌트 모델인 EJB에서는 적용할 수 없었다. 따라서, 직접적인 메소드 호출방식을 메시지 호출방식으로 전환하여야하며 자체적인 쓰레드 호출이 가능하도록 변환이 필요하다.

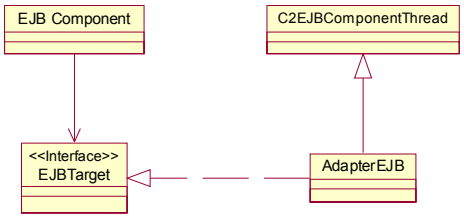
이를 위하여, 본 논문에서는 C2 아키텍처에서의 컴포넌트와 커넥터 쓰레드부분에서, EJB컴포넌트를 Adapter패턴을 이용하여 C2의 컴포넌트 요소에 바로 적용할 수 있도록 변경함으로서 커넥터 쓰레드는 수정 없이 사용하였다.



<그림 4> Adapter Pattern을 적용한 C2 아키텍처 프레임워크

따라서, EJB컴포넌트의 C2 아키텍처 컴포넌트 적용을 위하여, <그림 4>과 같이 Adapter 패턴을 이용하여 EJB 컴포넌트부분과 C2 프레임워크 사이의 중간 연결부분을 담당하도록 함으로서 EJB에서 사용할 수 없었던 ComponentThread부분을 연결하도록 하였다. 따라서, C2 프레임워크에서는 EJB 컴포넌트에 관하여 컴포넌트 쓰레드를 호출할 수 있으며, EJB 컴포넌트 부분은 소스의 수정 없이 Adapter 패턴에 의하여 C2 프레임워크에 적용될 수 있다.

다음 <그림 5>는 이를 위한 EJB 컴포넌트와 C2 아키텍처 적용을 위한 컴포넌트 쓰레드 사이의 구성을 나타낸다.



<그림 5> C2 프레임워크 적용하기 위한 Adapter Pattern

즉, Client는 EJB 컴포넌트가 되며, TargetIF부분은 필요한 구성요소로 변경하기 위한 인터페이스로서 EJBTarget부분이 된다. 즉, EJBTarget은 EJB 컴포넌트를 C2 아키텍처의 컴포넌트에 매칭 시키기 위한 인터페이스를 구현한다. 또한, Adapter부분은 EJBAdapter가 담당하며 인터페이스를 실제로 구현하여 EJB 컴포넌트가 C2아키텍처의 컴포넌트로서 작동하도록 변환한다. Adaptee는 C2EJBComponentThread 부분이며 적용하고자하는 C2 아키텍처의 ComponentThread를 상속받은 EJB 쓰레드이다. EJB 컴포넌트를 C2 아키텍처를 위한 컴포넌트로 변환하기 위

한 인터페이스인 EJBTarget은 다음과 같이 구현하였다.

```
public interface EJBTarget {
public abstract void create();
public abstract void start();
public abstract void handle(Request 또는 Notification);
}
```

즉, 쓰레드를 실행하기위한 start() 메소드와 메시지를 커넥터를 통하여 주고 받는 Handle() 메소드로 구성된다. 따라서, Request는 커넥터를 기준으로 합성된 컴포넌트들 중에서 하위 컴포넌트가 상위 컴포넌트에게 호출을 요청하는 메시지며, Notification은 상위 컴포넌트가 하위 컴포넌트에게 결과를 통지하는 메시지이다. 이러한 인터페이스를 실제로 구현하는 부분인 AdapterEJB는 다음과 같다.

```
public class AdapterEJB extends C2EJBComponentTh
    read implements EJBTarget {
   public AdapterEJB () { }
   public void create() { super.create(); }
   public void start() { super.start(); }
   public void handle(Request 또는 Notification) {
            super.handle(Request 또는 Notification); }
   }
```

이는, AdapterEJB부분이 C2 아키텍처의 ComponentThread 형태인 C2EJBComponentThread를 상속받으며, 인터페이스인 EJBTarget을 구현하고 있다.

또한, C2EJBComponentTread는 C2 프레임워크에 있는 ComponentThread를 그대로 상속받아 다음과 같이 구현한다.

```
public class C2EJBComponentTread extends Compon-
    entThread {
   public AdaCustomerBeanHand () { }
   public void create () { super.create (); }
   public void start () {
      super.start ();
      Request Method Message 등록 ;
      Request Message 전송 ;
   }
   public synchronized void handle(Request Message)
   {
      Request Message 생성 및 전송 ;
   }
   public synchronized void handle(Notification
Message) {
      Notification Message 생성 및 통지 ;
   }
}
```

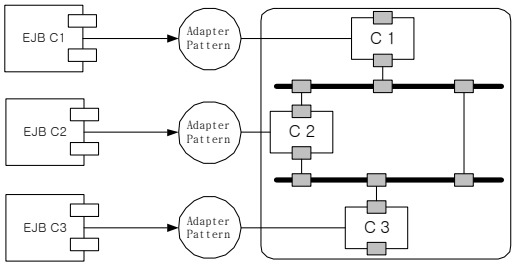
Adapter 패턴에 따라 재구성된 각 컴포넌트는 다음과 같이 합성하여 운용되어진다.

```
public C2EJB()
{
 new 컴포넌트_1 생성 ;
          :
 new 컴포넌트_n 생성 ;
 new 커넥터_1 생성 ;
          :
 new 커넥터_n 생성 ;
 addComponent (생성된 컴포넌트 이름);
          :
 addConnector (생성된 커넥터 이름);
 weld (합성할 컴포넌트_1, 합성할 커넥터_1);
 weld (합성할 커넥터_1, 합성할 컴포넌트_2);
          :
 start(); //메시지 운용 시작
 }
```

즉, 합성할 컴포넌트와 커넥터 객체를 생성한 후, addComponent()와 addConnector()를 이용하여 생성된 객체들을 C2 아키텍처에 등록시킨다. 이들을 weld() 메소드를 이용하여 합성순서에 따라 컴포넌트와 커넥터를 연결시키고 메시지 운용시작 쓰레드를 실행하면 C2 아키텍처에 의한 컴포넌트 합성이 완성되고 이를 운용할 수 있다.

이에 관하여 다음 <그림 6>은 각 EJB 컴포넌트들이 Adapter 패턴에 의하여 재구성

되어 C2 스타일 아키텍처에 따라 합성된 구성을 나타낸다.



<그림 6> C2 스타일 아키텍처에서의 EJB 컴포넌트 합성을 위한 Adapter 패턴 적용

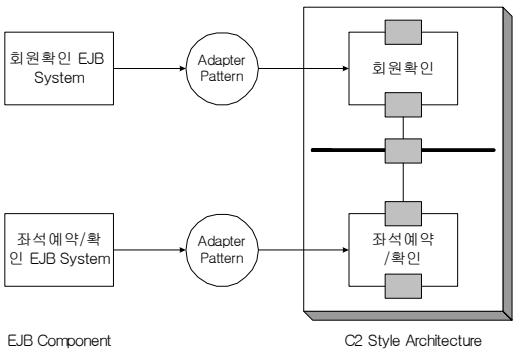
각 EJB 컴포넌트 EJB C1, EJB C2, EJB C3에 대하여 Adapter Pattern을 적용해서 C2 스타일 아키텍처의 컴포넌트 C1, C2, C3에 대응되도록 한다.

따라서, 두 개의 커넥터 사이에 구성되어 있는 컴포넌트 C1, C2, C3는 커넥터 쓰레드와 컴포넌트 쓰레드에 의하여 해당 포트를 통한 EJB 컴포넌트 메소드 호출 메시지를 전달받고 처리하며 이를 다음 컴포넌트에 전달할 수 있다.

5. 예제 시스템 적용

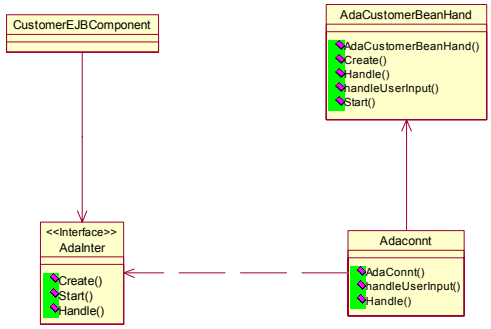
예제 시스템에서는 <그림 7>에서와 같이 회원확인 EJB 컴포넌트와 회원의 좌석예약 및 결과 EJB 컴포넌트를 C2 스타일 아키텍처에 따라 합성하였다. 이를 위하여, J2SDK 1.3과 EJB를 위한 J2SDKEE1.2.1 환경에서 구현하였으며, 회원 데이터베이스는 Cloudscape 데이터베이스엔진을 활용하였다.

회원확인 EJB System은 Adapter Pattern에 의해 회원확인 C2 스타일 아키텍처를 위한 회원확인 컴포넌트로, 좌석예약 EJB System은 C2 스타일 아키텍처를 위한 좌석예약/확인 컴포넌트로 각각 변화된다. 이들 사이에 커넥터를 두어 합성함으로서 C2 스타일 아키텍처 모델을 구성하였다.



<그림 7> 예제 시스템 구성도

이를 위하여, 회원확인 컴포넌트를 위한 Adapter Pattern 부분은 다음 <그림 8>과 같이 구성하였다. 즉, 기존의 EJB 컴포넌트인 CustomerEJBComponent 부분을 C2 스타일 아키텍처에서 사용 가능한 AdaCustomerBeanHand 부분으로 변환하였다.



<그림 8> 회원확인을 위한 Adapter Pattern

EJBComponent에서 Adapter에 의해 변환 적용할 인터페이스인 AdaInter부분은 다음과 같이 구현하였다.

```
public interface AdaInter {
```

```
    public abstract void create(String _name);
    public abstract void start();
    public abstract void handle(Request r);
    public abstract void handle(Notification n);
  }
```

이를 실제 구현한 Adapter부분인 Adacon nt에서는 C2 아키텍처의 ComponentThread 부분에서 처리하는 AdaCustomerBeanHand 의 메소드들을 호출하도록 하였다. 따라서, 다음과 같이 각 메소드들이 직접 처리하지 않고 부모 클래스가 메소드에 관한 처리를 하도록 하였다.

```
public class AdaConnt extends AdaCustomerBea-
      nHand implements AdaInter {
  public AdaConnt (String _name)
      { super (_name); }
  public void create(String _name)
      { super.create(_name); }
  public void start() { super.start(); }
  public void handle(Request r) { super.handle(r); }
  public void handleUserInput (Request r)
      { super.handleUserInput(r); }
  public void handle(Notification n)
      { super.handle(n); }
}
```

C2 아키텍처에서 ComponentThread부분을 상속받은 AdaCustomerBeanHand는 Adapter에 의해 다음과 같이 구현하였다.

```
public class AdaCustomerBeanHand extends
ComponentThread {
  public AdaCustomerBeanHand (String _name)
    { create (_name); }
  public void create (String _name)
  { super.create (_name, FIFOPort.classType()); }
  public void start ()
  {
    super.start ();
    r = new Request ("Request 호출 메시지");
    r.addParameter ("id", "Amadeus");
    r.addParameter ("pass", "1111");
    r.addParameter ("Command", "Check");
    send (r);
  }
  public synchronized void handle(Request r)
  {
    handleUserInput (r); //Request 메시지 분류
    send(r); //Request 메시지 전송
  }
  void handleUserInput (Request r)
  {
    if (메시지 분류)
      if ("Check" 메시지인 경우))
      {
      new_r = new Request("Check Message 생성");
      new_r.addParameter("Check 변수 등록);
      send (new_r); //메시지 전송
      } else if ("Get" 메시지인 경우) {
        new_r = new Request("Get Message 생성");
        new_r.addParameter("Get 변수 등록);
        send (new_r); //메시지 전송
      }
    }
  }
  public synchronized void handle(Notification n)
  {
    new_n = new Notification("Notification
                              메시지 생성");
    send(new_n); //메시지 전송
  }
}
```

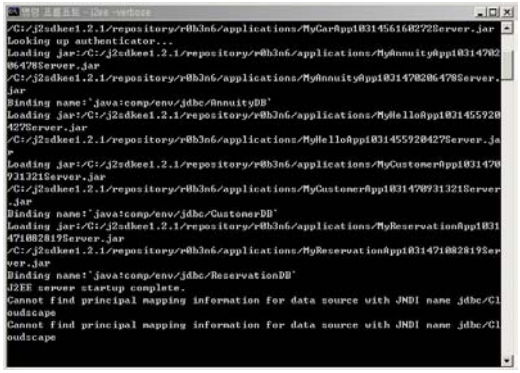
이와 같이 방법으로 좌석예약 및 확인 EJB 컴포넌트도 구현하였다. 따라서, 회원확인 EJB Adapter와 좌석예약 및 확인 EJB Adapter를 BindingBus 커넥터에 연결하여 <그림 7>과 같이 합성하면 다음과 같다.

```
AdaConnt CustHnd = new AdaConnt("Customer");
AdaReserv ReservHnd =
        new AdaReserv("Reserve");
ConnectorThread binding_bus =
        new ConnectorThread("BindingBus");
addComponent (CustHnd);
addComponent (ReservHnd);
addConnector (binding_bus);
```

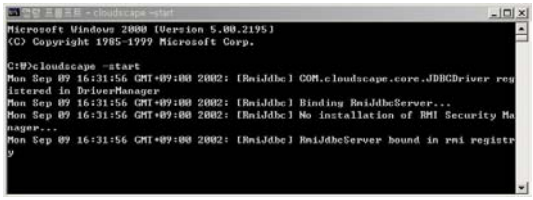
```
weld (CustHnd, binding_bus);
weld (binding_bus, ReservHnd);
start();
```

즉, 회원확인 EJB Adapter 객체인 CustHnd와 좌석예약/확인 EJB Adapter인 ReservHnd를 각각 생성하고, 생성된 컴포넌트를 연결하는 커넥터 binding_bus도 생성한다.

이를, C2의 addComponent와 addConnector 메소드를 통하여 C2아키텍처에 컴포넌트와 커넥터를 등록하고 Weld 메소드를 통하여 생성된 컴포넌트와 커넥터를 연결하며 합성시킨다. 이를, start() 메소드에 따라 쓰레드를 동작시키면 회원확인 컴포넌트는 회원확인 요구 메소드 호출 메시지를 받아서 처리한 후 그 결과를 커넥터에 전달하고, 커넥터는 해당 회원의 예약/확인 요구 메소드 호출 메시지를 받아서 좌석예약컴포넌트에 전달하여 그 결과를 얻음으로서 C2 스타일 아키텍처에 의한 컴포넌트 합성 결과를 얻을 수 있었다. 이에 따라, 실제 적용하여 구현된 결과는 다음과 같다.

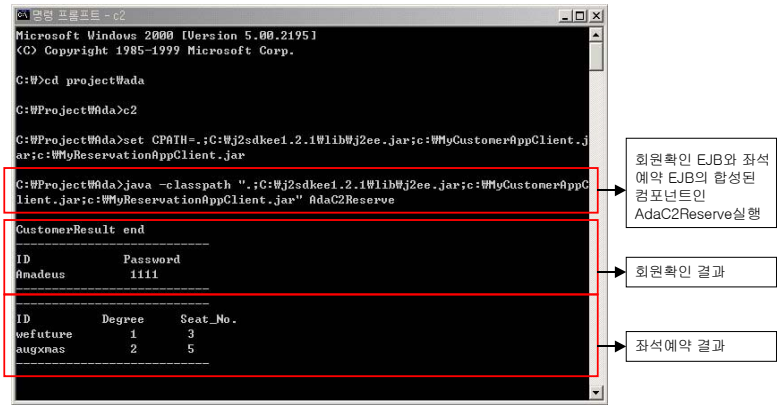


<그림 9> J2EE 서버의 기동



<그림 10> Cloudscape 엔진 기동

즉, 위 <그림 9>과 같이 EJB서버의 가동을 위한 J2EE 서버의 기동을 하며, 다음 <그림 10>은 데이터베이스를 위하여 J2EE에 내재된 Cloudscape 엔진의 기동 모습을 나타낸다.



<그림 11> Adapter 패턴에 의해 합성된 EJB 컴포넌트 실행 결과

이에 따라, EJB 서버와 데이터베이스 서버를 기반으로 C2 스타일 아키텍처의 합성 기법에 의한 회원확인 및 좌석예약 EJB 예제시스템의 실행결과는 다음 <그림 11>와 같다. 이는, 회원관리 EJB 컴포넌트인 MyCustomerAppClient.jar과 좌석예약 EJB 컴포넌트인 MyReservationAppclient.jar을 합성한 AdaC2Reserve를 실행함으로서 회원확인 결과와 좌석예약결과를 나타낸다.

6. 결과 및 분석

쓰레드 기반의 메시지호출방식의 컴포넌트 합성기법인 C2 아키텍처는 쓰레드를 호출할 수 없는 EJB의 경우 적용할 수 없었으나 최유희[5]의 연구에서는 그 구조를 변경하면서 적용 및 운용할 수 있었다. 그러나, 이는 EJB를 적용하기 위하여 C2아키텍처 프레임워크의 구성요소에서 컴포넌트와 커넥터부분의 변경필요하며, EJB의 메소드 직접적인 호출방식을 메시지 호출방식으로 전환 및 운용하기 위하여 EJBWrapper가 필요하였다. 따라서, 기존의 연구방법에서는 각각의 EJB 특성에 따라 C2아키텍처에서 변경이 필요한 ConnectorForEJB와 ComponentForEJB에 대한 충분한 고려가 필요하다. 또한, EJB 컴포넌트들의 메소드 호출 메시지 부분을 담당하고있는 EJBWrapper에서도 합성되는 컴포넌트가 많을 경우와 컴포넌트들 사이의 메시지 전달이 많을 경우 프로세스에 부하가 많이 걸릴 수 있다. 또한, 모든 컴포넌트들의 메시지 교환을 담당함으로서 EJBWrapper부분의 프로세스 오류 발생시 합성된 전체 시스템의 실행정지로 이어질 수 있다.

이에 비하여, 본 논문의 제안기법에서는 기존의 EJB 컴포넌트와 C2 아키텍처 모두 수정이 필요 없다. 따라서, EJB 컴포넌트를 적용하기 위하여 추가적인 수정작업이 없이 EJB컴포넌트들을 C2아키텍처에 그대로 합성함으로서, 쉽고 간단하게 쓰레드를 기반으로 하는 독립적인 메시지 교환이 가능하다. 또한, 각각의 EJB 특성에 따라 C2 아키텍처의 적용을 위하여 구조 변경에 관한 고려를 하지 않아도 된다. 본 제안기법에서는 이에 관하여 각기 다른 프로세스 특성을 갖는 EJB라 할지라도 일반적인 Adapter 패턴의 규약을 따름으로서 C2 아키텍처에 따른 EJB 컴포넌트 합성에 규칙을 갖도록 하였다.

그리고, EJBWrapper와 같이 한 프로세스에서 모든 컴포넌트 메시지를 핸들링하지 않고 각각의 EJB를 Adapter에 의해 변경하여 C2 아키텍처의 컴포넌트 요소에 사용하면서 메시지의 교환은 C2 아키텍처의 형식을 그대로 따를 수 있었다. 즉, 커넥터와 Adapter에 의해 변경 적용된 컴포넌트들은 자체적인 쓰레드에 의해 메시지를 교환할 수 있었다

7. 결론 및 향후 연구 방향

GUI 기반의 컴포넌트 합성기법인 C2 스타일 아키텍처에서는 쓰레드를 기반으로 컴포넌트와 커넥터를 통한 메시지 교환방식을 택하고 있다. 따라서, 쉽고 효율적이며 시각적인 컴포넌트의 합성기법으로 알려져 있다. 그러나, 쓰레드를 호출할 수 없는 직접적인 메소드 호출방식을 택하고 있는 EJB 컴포넌트에서는 이를 활용할 수 없었다.

따라서, 본 논문에서는 C2 스타일 아키텍처에서 쓰레드를 호출할 수 없는 EJB 컴포넌트의 적용기법을 제안하였으며, 실제 예제 시스템을 구현하여 적용된 결과를 확인하였다. 이를 위하여, 디자인 패턴중 구조화 패턴의 하나인 Adapter 패턴을 적용하여 EJB컴포넌트에서 쓰레드를 호출할 수 있도록 전환

하였다. 따라서, C2 스타일 아키텍처에서 EJB 컴포넌트 쓰레드를 호출할 수 있었으며 EJB 컴포넌트를 합성하여 실제 구현됨을 확인할 수 있었다.

향후 연구과제로는 각 컴포넌트를 C2 EJB 쓰레드 적용을 위한 개별 Adapter 패턴이 아닌, 보다 일반화되고 구조화된 단일 Adapter 패턴에 의한 핸들링이 필요하다.

참고문헌

[1] C. Marcos, M. Compos, and A. Pirotte, "Reifying Design Patterns as Metalevel Constructs", *Electronic Journal of Sadio*, 2(1) pp.17-19, 1999.

[2] 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.

[3] P. C. Clements, "From Subroutines to Subsystem : Component-Based Soft ware Development", *Component Based Software Engineering, IEEE CSPress*, 1996.

[4] 신동익 외6인, "C2 스타일의 아키텍처 기술을 지원하는 ADL 지원도구의 개발", 한국정보처리학회 논문지 Vol. 8-D, No 6. 2001.

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

[6] The C2 Style, http://www.ics.uci.edu/pub/arch/c2.html, *Information and Computer Science, University of California*, Irvine.

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

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns-Elements of Reusable Object-Oriented Software", *Addison-Wesley*, 1995.

[9] Adapter Diagram : Overview of Design Patterns, Patterns in Java Volume 1: A Catalog of Reusable Design Patterns Illustrated with UML, *http://www.mindspring.com/~mgrand/pattern_synopses.htm*, 2002

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

[11] 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.

정화영

1991년~1994년 경희대학교 전자계산공학과 (공학석사)

1998년~2000년 한남대학교 컴퓨터공학과(박사과정)

2000년~현재 경희대학교 전자계산공학과(박

사과정)
2000년~현재 예원대학교 정보경영학부 교수
관심분야 : 소프트웨어공학, OOP/S, S/W 재사용, 컴포넌트기반 개발 방법론.

송영재

1969년 인하대학교 전기공학과(공학사)
1976년 일본Keio University 전산학과(공학석사)
1979년 명지대학교 대학원 졸업(공학박사)
1971년~1973년 일본 Toyo Seiko 연구원
1982년~1983년 미국 Univ. of Maryland 전산학과 연구교수
1985년~1989년 IEEE Computer society 한국지부회 회장
1984년~1989년 경희대학교 전자계산소장
1976년~현재 경희대학교 컴퓨터공학과 교수
1993년~1995년 경희대학교 교무처장
1996년~1998년 경희대학교 공과대학장
1998년~2001년 경희대학교 기획조정실장
2002년~현재 경희대학교 산업정보대학원 원장
관심분야 : 소프트웨어공학, OOP/S, CASE 도구, S/W 개발도구론, S/W 재사용, 컴포넌트기반 개발 방법론.