Enterprise JavaBeans(EJB)에서 효율적인 CBD 컴포넌트 설계 기법

(An Effective Method to Design CBD Components in Enterprise JavaBeans (EJB))

김 수 동 † 민 현 기 †† 이 진 열 ††† 김 성 안 †††
(Soo Dong Kim) (Hyun Gi Min) (Jin Yeal Lee) (Seong An Kim)

요 약 Enterprise JavaBeans (EJB)는 컴포넌트 기반 소프트웨어 개발 (Component-based Development, CBD)을 위해 보급된 환경이다. 컴포넌트는 재사용 단위의 복합 객체이지만, EJB는 객체단위의 작은 컴포넌트이므로 다수의 엔터프라이즈 빈을 복합하여 큰 단위 재사용을 지원해야 한다. 따라서 EJB를 이용하여 컴포넌트를 상세 설계 및 구현하기 위한 구체적이고 실용적인 기법이 필요하다. 본 논문에서는 CBD의 구성요소를 EJB를 이용하여 어떻게 설계하는지의 지침과 기법을 제시한다. 단일, 복합, 화이트, 블랙 박스, 다중 인터페이스, 가변성을 지원하는 EJB 환경의 컴포넌트 설계 및 구현 기법을 제시한다. 사례 연구를 이용하여 제시한 설계 기법을 보이고, CBD 컴포넌트의 특징과 본 논문의 기법을 비교하여 검증한다. 결론적으로 EJB 환경에서 컴포넌트의 재사용성, 활용성, 이식성을 더욱 증가 시킬 것이다.

키워드 : 컴포넌트 기반 개발(CBD), Enterprise JavaBeans(EJB), 컴포넌트의 구성 요소, EJB 컴포넌트 설계, EJB 가변성 설계, EJB 인터페이스 설계

Abstract Enterprise JavaBeans (EJB) has been accepted for supporting Component-Based Development (CBD). A component is a large-grained reuse unit consisting of several objects; however, an enterprise bean in EJB is a unit of atomic object and so multiple enterprise beans should be composed to support larger-grained reuse. Therefore, we need practical methods for designing and implementing components with EJB. In this paper, we propose instructions and techniques for designing CBD elements with EJB constructs. That is, we define methods for designing and implementing single and composite components, white- and black-box components, multiple interfaces, and variability mechanism in EJB platform. We evaluate the proposed method by performing a case study and comparing the characteristics of CBD components with the method. Consequently, the method is supposed to improve reusability, applicability, portability of components in EJB platform.

Key words : Component-based Development (CBD), Enterprise JavaBeans (EJB), Component Metamodel, EJB Component Design, Variability Design, Interface Design

1. 서 론

어플리케이션 개발기술의 하나인 CBD 기술은 효율적인 재사용 기술이며 산업계에서 보편적으로 사용되는 패러다임이다. 컴포넌트는 소프트웨어의 재사용 부품으로써 단일 객체보다 단위가 크고 특정 소프트웨어를 위한 가변적인 부분들을 포함하여 재사용성이 객체보다 좋다. 따라서 컴포넌트 기반 개발 환경에서 소프트웨어를 개발하는 것은 소프트웨어의 재사용 측면에서 비용과 노력을 줄이는 결과를 가져온다. 산업계에서는 J2EE의 엔터프라이즈 자바 빈즈(Enterprise JavaBeans, EJB)[1]를 컴포넌트 개발 환경을 위한 최적의 솔루션으로서 채택하고 있다.

그러나 EJB 기반 환경에서 컴포넌트를 만들기 위한

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@ssu.ac.kr
†† 정 회 원 : 숭실대학교 컴퓨터학과 연구원
hgmin@otlab.ssu.ac.kr
††† 학생회원 : 숭실대학교 컴퓨터학과
jylee@otlab.ssu.ac.kr
sakim@otlab.ssu.ac.kr
논문접수 : 2005년 10월 18일
심사완료 : 2005년 11월 15일

효율적인 설계모델과 구체적이고 체계적인 구현 기법에 대한 연구가 필요하다. 컴포넌트는 특성에 따라 그룹화된 여러 개의 인터페이스를 가지고 있다. 컴포넌트의 단위는 객체를 한 개만 포함하고 있는 단일 컴포넌트와 2개 이상의 객체를 포함하는 복합 컴포넌트가 있다. 컴포넌트를 사용하기 위해서는 사용자의 목적에 맞게 특화를 시켜야 한다. 본 논문에서는 컴포넌트의 구성 요소인 컴포넌트의 객체모델, 인터페이스, 가변성, 워크플로우를 EJB를 활용한 설계 기법을 제시한다. 또한 컴포넌트를 개발하기 위한 주요 장치인 선택형, 플러그인 기법과 컴포넌트 조립 장치를 제시한다.

본 논문의 2장 기반 연구에서는 EJB 2.0의 구성요소와 각 구성요소간의 기능적인 역할과 상호작용에 대해서 설명한다. 그리고 패밀리 멤버간의 서로 다른 기능을 가지는 컴포넌트의 가변성과 이를 특화 할 수 있는 기법에 대해서 소개한다. 3장에서는 컴포넌트의 구성요소로서 컴포넌트의 객체모델과 인터페이스, 가변성, 워크플로우를 설명한다. 4장에서는 앞서 소개한 컴포넌트의 요소를 EJB에서 지원하기 위한 컴포넌트 설계기법을 제안한다. 5장에서는 사례연구를 통해 제안된 기법을 타당성을 검증하고 6장에서는 CBD의 요소와 본 논문의 설계 기법을 비교하여 평가한다.

2. 기반 연구

2.1 EJB 2.0

엔터프라이즈 자바 빈즈(Enterprise JavaBeans, EJB)는 엔터프라이즈 환경에서 분산 컴포넌트 어플리케이션을 개발하기 위한 자바 기반의 서버 측 컴포넌트 아키텍처이다[1]. EJB를 사용함으로써 복잡한 분산 컴포넌트 프레임웍을 개발자가 직접 구현 하지 않고 확장성, 신뢰성, 보안성이 좋은 어플리케이션을 작성할 수 있다. 그러므로 개발자는 서버 측의 컴포넌트 및 어플리케이션 개발이 용이하다[2]. EJB에서의 엔터프라이즈 빈(Enterprise Bean)은 분산 객체를 지원하기 위해 자바 객체 한 개에 홈 인터페이스와 컴포넌트 인터페이스를 제공한다. 그러므로 엔터프라이즈 빈은 객체 단위의 작은 컴포넌트이다.

엔터프라이즈 빈의 인터페이스는 홈 인터페이스(Home Interface)와 컴포넌트 인터페이스(Component Interface)로 구성된다. 홈 인터페이스는 엔터프라이즈 빈의 라이프 사이클을 관리하고, 컴포넌트 인터페이스를 참조 할 수 있게 한다. 컴포넌트 인터페이스는 엔터프라이즈 빈의 기능을 제공하기 위한 인터페이스이다. EJB에는 비즈니스 로직을 제공하는 세션 빈, 영속성을 제공하는 엔티티 빈, 비동기 메시지를 위한 메시지 드리븐 빈이 있다. 세션 빈은 세션이 연결된 동안 빈의 상태를 관리하지 않는 무상태 세션 빈(Stateless SessionBean)과 세션이 연결된 동안 빈의 상태를 관리하는 상태 유지 세션 빈(Stateful SessionBean)이 있다.

2.2 컴포넌트 가변성

컴포넌트는 사용자의 공통적인 요구사항인 공통성(Commonality)을 가지고 있다. 컴포넌트의 가변성(Variability)은 이러한 공통적인 요구사항에서 각 어플리케이션마다 약간씩 다른 것을 의미한다[3]. 가변점(Variation Point)은 컴포넌트를 사용하는 멤버들간에 서로 다른 부분인 가변성이 발생되는 곳이다. 이러한 가변점(Variation Point)은 특정 어플리케이션을 위한 설정 값인 가변치(Variant)로 설정되어야 사용할 수 있다[4].

컴포넌트 내부에 가변성의 식별에 따라 두 개의 범위로 나눌 수 있다[5]. 분석 단계에서 가변치들이 모두 식별이 가능하면 Close 영역이라 한다. 반면에 분석 단계에서 모든 가변치를 식별 할 수 없어 컴포넌트가 구현된 후에 사용자에 의해 정의되어야 하면 Open 영역이라 한다[6]. 고객의 요구사항에 맞게 컴포넌트의 가변점이 가변치로 설정되어야 하는데 이것을 특화(Customization)라고 한다.

3. 컴포넌트의 구성요소

본 장에서는 컴포넌트의 구성요소를 그림 1에서와 같이 UML을 이용하여 메타 모델로 정의한다. 컴포넌트는 클래스, 인터페이스, 가변성, 워크플로우로 이루어진다. 컴포넌트는 시그네처와 제약사항을 명세한 인터페이스를 구체화한다. 인터페이스를 구체화하기 위해 컴포넌트는 하나 또는 여러 개의 복합 클래스로 구현된다. 컴포넌트는 워크플로우에 따라 컴포넌트 내부의 객체들과 외부 컴포넌트에 메시지가 전달된다. 컴포넌트는 화이트박스와 블랙박스 컴포넌트가 있다[7]. 컴포넌트들과 객체들을 조합하여 어플리케이션이 생성된다.

3.1 컴포넌트의 객체모델

컴포넌트의 정적 구조를 이루는 객체모델은 하나의 클래스 또는 여러 개의 클래스들과 이들간의 관계로 정의된다. 이러한 컴포넌트의 객체모델은 컴포넌트의 크기(Granularity), 가시성(Visibility), 데이타 영속성(Data Persistence), 메소드 동기화(Method Synchronization) 등 크게 4가지 측면으로 구분 할 수 있다. 첫째, 컴포넌트의 크기는 컴포넌트 내부의 객체의 수에 따라 단일(Single) 객체와 복합(Complex) 객체로 구분된다. 둘째, 컴포넌트의 가시성은 컴포넌트의 내부가 외부에 공개되어 내부에 직접적으로 접근이 가능한지 여부에 따라 화이트박스와 블랙박스로 구분된다. 세 번째, 컴포넌트의 영속성은 컴포넌트가 관리하는 데이타가 영속적으로 관리가 필요한지 여부를 말한다. 넷째, 메소드 동기화는

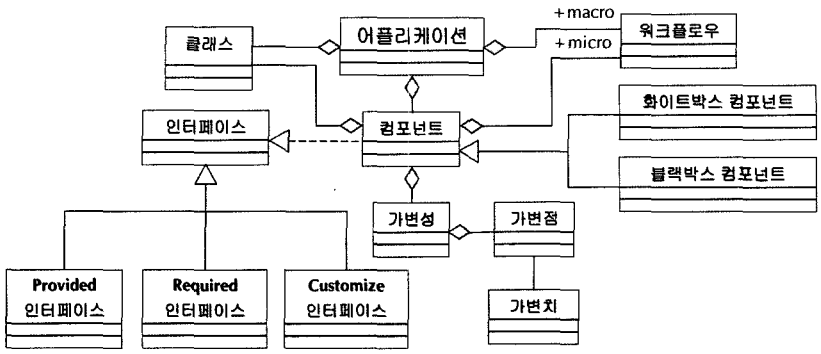


그림 1 컴포넌트의 메타 모델

컴포넌트 내부의 비즈니스 로직의 결과값이 동기화가 요구되는 경우와 컴포넌트가 이벤트 기반으로 실행되고 결과값의 동기화가 요구되지 않은 경우로 구분된다.

3.2 컴포넌트 인터페이스

컴포넌트의 인터페이스는 *Provided, Required, Customize* 인터페이스의 세 가지 유형으로 정의한다[4]. 이러한 인터페이스는 그림 2와 같이 표현된다. 인터페이스는 시그네처와 제약사항을 명세한다. 인터페이스 규약에 따라 컴포넌트가 구현된다. *Provided* 인터페이스는 컴포넌트가 제공하는 기능군을 정의한다. 이는 컴포넌트의 주요 기능을 결정하는 인터페이스로 일반적인 의미의 컴포넌트 인터페이스를 가리킨다.

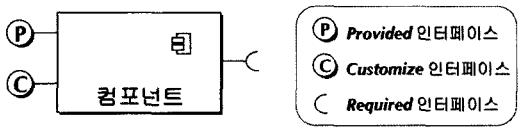


그림 2 컴포넌트 인터페이스

Required 인터페이스는 컴포넌트 실행 시 요구되는 다른 컴포넌트의 기능군을 명세 한다. 물리적인 인터페이스를 할당하지 않고 명세서를 제공하여 컴포넌트간 의존 관계를 명시한다. 즉, 실제 물리적인 컴포넌트 인터페이스의 구현으로 표현되지 않는다.

Customize 인터페이스는 컴포넌트 내부의 가변점에 어플리케이션에서 요구하는 가변치를 설정하기 위한 인터페이스이다. *Customize* 인터페이스는 컴포넌트를 수정하지 않고 컴포넌트를 특화시킬 수 있기 때문에 확장성을 제공한다. 컴포넌트는 이러한 확장성을 제공하기 위한 특화 기법을 컴포넌트 내부에 구현되어야 한다. *Customize* 인터페이스를 통해 컴포넌트 사용자는 컴포넌트를 자신의 목적에 맞게 설정 한 후에 사용자는 컴포넌트의 기능은 Provided 인터페이스를 통해 컴포넌트 사용자에게 제공된다.

3.3 가변성

객체지향 컴포넌트의 가변성을 지원하기 위한 특화 기법은 선택형(Selection), 플러그인(Plug-in) 기법이 있다[4]. 선택형 기법은 이미 식별된 가변치를 컴포넌트에 포함시킨 후에 내부의 가변치를 선택하는 기법이다. 플러그인 기법은 모든 가변치를 컴포넌트에 포함시킬 수 없으므로 외부의 가변치를 컴포넌트에 설정하는 기법이다.

3.3.1 선택형 기법

선택형 기법은 컴포넌트의 개발 시 어플리케이션에 따라서 변할 수 있는 가변적인 요소를 미리 식별함으로써 컴포넌트 구매자가 식별된 가변치 중에서 해당 어플리케이션에 맞는 가변치를 선택하여 특화 시킬 수 있는 기법이다.

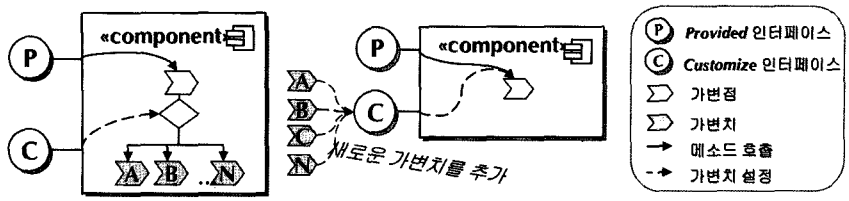
선택형 기법은 그림 3(a)에서와 같이 컴포넌트 내부에서 N개의 가변치 중 하나를 선택하기 위해서 *Customize* 인터페이스와 N개의 가변치를 나타낸 것이다. 선택형 기법에서는 가변점을 갖는 메소드에 switch-case문을 사용하여 가변치들을 구현한다. 구현된 가변치는 *Customize* 인터페이스를 통해 선택된다.

3.3.2 플러그인 기법

플러그인(Plug-in) 기법은 *Customize* 인터페이스를 통해 컴포넌트 외부의 가변치를 할당하기 위해서 사용된다. 어플리케이션에서 요구하는 가변치를 구현한 후에 컴포넌트로 전달하고 이를 컴포넌트 내부에서 사용한다. 이러한 방법으로 컴포넌트는 각 어플리케이션에 맞게 특화될 수 있다. 그림 3(b)는 사용자에 의해서 요구되는 가변치 A, B, C 그리고 N등과 같은 여러 가변치를 컴포넌트 외부로부터 *Customize* 인터페이스를 통해서 동적으로 가변치를 추가할 수 있다.

3.4 워크플로우

워크플로우는 메시지들의 호출 순서이다. 워크플로우는 매크로 워크플로우(Macro Workflow)와 마이크로 워크플로우(Micro Workflow)로 구성된다[8]. 매크로 워



(a) 선택형 기법 (b) 플러그인 기법

그림 3 컴포넌트 특화 기법

크플로우는 여러 컴포넌트가 조립(Assemble)되거나 조합(Composite)된 컴포넌트간의 메시지 흐름을 말한다. 이 워크플로우는 컴포넌트의 인터페이스를 통해서 컴포넌트들간의 필요한 메시지를 주고 받는다. 마이크로 워크플로우는 컴포넌트 내부의 객체들간의 메시지 흐름을 말한다.

3.5 컴포넌트 조립 장치

컴포넌트의 조립(Assembly) 장치는 컴포넌트의 획득이나 개발을 통해 수집된 컴포넌트들을 조립하여 특정 어플리케이션을 생성할 수 있도록 도와주는 장치이다. 이때, 획득된 컴포넌트를 어플리케이션에 맞도록 특화할 수 있는 장치가 적용될 수 있으며, 수집된 컴포넌트 외에 특정 어플리케이션에 한정된 추가적인 장치가 사용될 수 있다.

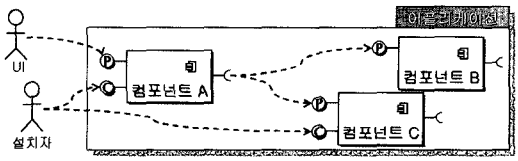


그림 4 컴포넌트 조립을 통한 어플리케이션 생성

그림 4는 컴포넌트들간의 기능을 요구하고 제공하는 관계를 보여준다. 컴포넌트 A는 컴포넌트 B의 *Provided* 인터페이스에서 제공하는 기능성을 필요로 한다. 이는 설계상에서 컴포넌트 A의 *Required* 인터페이스로 표현된다. 그리고 컴포넌트 A는 컴포넌트가 필요로 하는 기능성이 컴포넌트 C에서 제공하는 기능성과 어느 정도 일치하지만 완전히 부합하지 않는 경우, 관리자는 컴포넌트 C에서 제공하는 *Customize* 인터페이스를 이용하여 컴포넌트 C를 특정 어플리케이션에 맞도록 특화한다. 단, 컴포넌트 C가 컴포넌트 A에서 필요로 하는 기능성을 제공할 수 있는 특화장치를 제공하는 경우에만 *Customize* 인터페이스를 사용한 특화가 가능하다.

4. EJB의 컴포넌트 기반 설계 기법

본 장에서는 3장의 컴포넌트 기반 설계의 구성요소를 EJB 환경에 적용하기 위하여 학계에서 논하는 CBD와 EJB 플랫폼에서의 컴포넌트 설계의 차이점을 식별하고 이들간의 차이를 해소하기 위한 컴포넌트 설계 기법을 제시한다.

4.1 컴포넌트 설계와 EJB에서의 컴포넌트 설계와의 차이점

본 절에서는 컴포넌트 설계의 요소와 EJB에서의 컴포넌트 설계와의 차이점을 식별한다. EJB가 컴포넌트 플랫폼으로서 컴포넌트 설계 요소를 제공하고 있는 부분과 제공하지 않는 보완해야 하는 부분을 식별함으로써 이어지는 절에서 이러한 차이를 줄이기 위한 설계 기법을 제시할 것이다.

4.1.1 컴포넌트 객체 모델

*EJB에서는 하나의 엔터프라이즈 빈이 객체 단위의 작은 컴포넌트이므로 CBD의 큰 단위의 재사용을 지원하지 못한다*는 컴포넌트 크기에서의 차이점이 있다. 따라서 이러한 컴포넌트 크기의 문제를 해결할 수 있는 큰 단위 재사용을 지원하기 위한 설계 기법이 필요하다.

EJB에서 단일 엔터프라이즈 빈이 컴포넌트의 단위인 경우 화이트박스 컴포넌트 설계는 지원되지 않으며 큰 재사용 단위를 제공하기 위하여 다중 엔터프라이즈 빈을 컴포넌트의 단위로 여길 시 블랙박스 컴포넌트 설계를 직접적으로 지원하지 않는다는 가시성의 차이점이 있다. 이는 엔터프라이즈 빈을 반드시 컴포넌트 인터페이스를 통해서 접근해야 하는 EJB의 명세와 다중 엔터프라이즈 빈을 구성한 컴포넌트에 대한 기법을 제시하지 않는 EJB의 명세에 의해 발생하는 차이점이다.

영속성 서비스를 위해서는 엔티티 빈을 이용하여야 한다. 세션 빈 또는 메시지 드리븐 빈에서의 데이타베이스 관련 로직의 처리는 가능하지만 이는 영속성 서비스가 아니며 실제 객체-관계 매핑을 통한 영속성 서비스를 이용하기 위해서는 반드시 엔티티 빈을 사용하여야 한다. 이는 가장 많이 사용되고 있는 영속성 관리 장치인 관계형 데이타베이스의 사용을 통한 객체와 관계형 데이타베이스와의 차이점을 해소하기 위한 EJB의 특징으로 학계의 CBD 이론과는 차이가 있다.

비동기 서비스를 이용하기 위해서는 메시지 드리븐

빈을 이용하여야 한다. 메소드 동기화 처리 시 비즈니스 로직을 처리하는 세션 빈과 데이타 영속성을 관리하는 엔티티 빈은 비동기 메시지를 처리하지 못한다. EJB는 특정 엔터프라이즈 빈의 형태에 따라 역할을 구분 짓고 있다. 때문에 메소드 동기화 여부에 따른 컴포넌트의 구성 요소가 변경된다. 이는 CBD 이론에서는 나타나지 않는 EJB의 특징이다.

4.1.2 인터페이스

EJB의 엔터프라이즈 빈은 오직 하나의 인터페이스만을 제공한다. 하나의 컴포넌트는 다수의 인터페이스를 구현할 수 있다. 하지만 EJB에서 컴포넌트의 단위인 엔터프라이즈 빈은 하나의 *Provided* 인터페이스만을 가질 수 있다[1]. 홈 인터페이스와 컴포넌트 인터페이스를 제공하므로 두 개의 인터페이스를 제공하는 것으로 생각할 수 있지만 하나의 컴포넌트 인스턴스의 기능을 제공하기 위한 인터페이스는 오직 컴포넌트 인터페이스뿐이다.

4.1.3 컴포넌트 가변성

EJB는 컴포넌트 특화를 위한 장치를 지원하지 않는다. 컴포넌트를 다수의 어플리케이션에서 재사용하기 위한 컴포넌트 특화 기법은 컴포넌트 설계에서 중요한 요소로 인식되고 있다[10]. 명세상의 제약에 의해 EJB의 엔터프라이즈 빈간의 상속을 통한 메소드 재정의(Override)가 가능하지 않기 때문에 컴포넌트를 특화 하기 위해서는 별도의 장치가 필요하다[2]. 하지만 EJB는 컴포넌트를 해당 어플리케이션의 요구사항에 맞도록 특화할 수 있는 장치를 제공하고 있지 않다.

4.2 컴포넌트의 객체모델 설계 기법

본 절에서는 컴포넌트의 객체모델을 컴포넌트의 크기, 가시성, 데이타 영속성, 메소드 동기화 측면의 4가지로 표 1과 같이 구분한다. 표 1에서 사용되는 표기법은 다음과 같다. '<>'는 컴포넌트를 '→'는 빈들간의 메시지 전달 관계를 '|'는 옵션을 의미하고, 'SB'는 세션 빈을 'EB'는 엔티티 빈을 'MB'는 메시지 드리븐 빈을 의미한다. 예를 들면, $<SB_1 \rightarrow EB_1>$의 의미는 세션 빈과 엔티티 빈으로 구성된 컴포넌트이며, SB_1 세션 빈에서 EB_1 엔티티 빈으로 메시지를 전달하는 패턴을 표현한다.

첫째, 컴포넌트를 크기(**Granularity**) 측면으로는 단일(Single) 컴포넌트와 복합(Complex) 컴포넌트로 구분한다. 즉, 컴포넌트의 객체모델이 단일 객체로 구성된 컴포넌트와 다중 객체로 구성된 컴포넌트로 구분한다. EJB에서 단일 컴포넌트는 하나의 엔터프라이즈 빈에 해당되며, 복합 컴포넌트는 관련된 여러 엔터프라이즈 빈들을 관장하는 중재자(Mediator) 빈을 정의하여 설계

표 1 컴포넌트의 종류에 따른 EJB에서의 컴포넌트 설계 기법

사례	크기	가시성	영속성	메소드 동기화	EJB에서의 컴포넌트 설계 기법
1	단일	화이트박스	비영속	동기	- 단일 화이트박스 컴포넌트의 설계 기법은 EJB 에서 지원하지 않음.
2				비동기	
3			영속	동기	
4				비동기	
5		블랙박스	비영속	동기	- $<SB_1>$, $<SB_2>$, …, $<SB_n>$
6				비동기	- $<MB_1>$, $<MB_2>$, …, $<MB_n>$
7			영속	동기	- $<EB_1>$, $<EB_2>$, …, $<EB_n>$ - $<SB_1 \rightarrow EB_1>$, $<SB_2 \rightarrow EB_2>$, …, $<SB_n \rightarrow EB_n>$
8				비동기	- $<MB_1 \rightarrow EB_1>$, $<MB_2 \rightarrow EB_2>$, …, $<MB_n \rightarrow EB_n>$
9	복합	화이트박스	비영속	동기	- $<SB_1 \rightarrow SB_2 \mid SB_3 \rightarrow SB_4 \rightarrow SB_5>$, …
10				비동기	- $<MB_1 \rightarrow SB_1 \mid MB_2 \rightarrow SB_2 \rightarrow SB_3>$, …
11			영속	동기	- $<SB_1 \rightarrow SB_2 \rightarrow EB_1 \mid SB_2 \rightarrow EB_1 \mid EB_2>$, …
12				비동기	- $<MB_1 \rightarrow EB_1 \mid MB_2 \rightarrow SB_1 \rightarrow EB_2>$, …
13		블랙박스	비영속	동기	- $<SB_1 \rightarrow SB_2 \rightarrow SB_3 \mid SB_1 \rightarrow SB_4>$, … - SB_1의 인터페이스만을 공개
14				비동기	- $<MB_1 \rightarrow SB_1 \rightarrow SB_2 \mid MB_1 \rightarrow SB_3>$, … - 하나의 MB만을 가진다. - 모든 SB의 인터페이스는 공개하지 않는다.
15			영속	동기	- $<SB_1 \rightarrow SB_2 \rightarrow EB_1 \mid SB_1 \rightarrow EB_2>$, … - SB_1의 인터페이스만을 공개
16				비동기	- $<MB_1 \rightarrow EB_1 \mid MB_1 \rightarrow EB_2>$, ... - $<MB_1 \rightarrow SB_1 \rightarrow EB_1 \mid MB_1 \rightarrow SB_2 \rightarrow EB_2 \mid MB_1 \rightarrow EB_3>$, … - 하나의 MB만을 가진다. - 모든 SB의 인터페이스는 공개하지 않는다.

할 수 있다.

둘째, 컴포넌트의 가시성(**Visibility**) 측면에서 화이트박스와 블랙박스 컴포넌트로 구분한다. 화이트박스 컴포넌트는 클라이언트가 컴포넌트 내부의 요소에 직접 접근하여 기능을 제어할 수 있는 컴포넌트를 말하며 블랙박스 컴포넌트는 클라이언트가 공개된 인터페이스를 통해서만 컴포넌트의 기능에 접근할 수 있으며 내부의 요소에는 직접 접근할 수 없다.

셋째, 데이타 영속성(**Data Persistence**) 측면에서는 영속성 관리가 필요하지 않은 경우와 영속성 관리가 필요한 경우로 구분할 수 있다. EJB에서는 영속성 관리가 필요하지 않고 비즈니스 로직의 처리를 관리하기 위해서 세션 빈을 사용하여 설계 가능하며 영속성 관리가 필요한 경우 엔티티 빈을 이용하여 설계 가능하다.

넷째, 메소드 동기화(**Method Synchronization**) 측면에서는 결과값의 동기화가 요구되는 경우와 컴포넌트가 이벤트 기반으로 실행되며 결과값의 동기화가 요구되지 않는 경우로 구분할 수 있다. EJB에서는 동기 메시지 처리를 위해서 세션 빈을 사용하여 설계가능하며 비동기 메시지 처리를 위해서 메시지 드리븐 빈을 사용할 수 있다.

사례1부터 사례4의 단일 화이트박스 컴포넌트는 EJB에서 지원하지 않는다. EJB에서 내부의 빈에 접근하기 위해서는 반드시 컴포넌트 인터페이스를 통하도록 명세한다. 이는 사용자가 정의한 엔터프라이즈 빈 이외에 미들웨어의 기능성을 추가하기 위한 EJB에서의 특징이다.

사례5부터 사례8의 단일 블랙박스 컴포넌트는 객체모델에서 하나의 클래스를 EJB를 이용해 설계하는 것이다. 사례5와 사례6은 영속성 서비스가 필요하지 않으므로 동기 메시지인 경우 단일 세션 빈을 이용하고 비동기인 경우 단일 메시지 드리븐 빈을 이용하여 설계할 수 있으며 이는 EJB에서 제공하는 기본 컴포넌트 설계 기법이다. 사례7의 경우 클라이언트가 영속성 서비스에 직접 접근할 수 있도록 하나의 엔티티 빈을 이용한 설계와 클라이언트가 비즈니스 로직을 처리하는 세션 빈을 통해 접근하여 해당 호출을 영속성 서비스를 제공하는 엔티티 빈에 위임하도록 설계할 수 있다. 사례8의 경우 EJB에서는 엔티티 빈을 비동기로 호출하는 것은 지원되지 않으므로 비동기 서비스를 제공하는 메시지 드리븐 빈에 메시지를 보내 해당 메시지를 엔티티 빈에 위임하도록 설계한다.

사례9부터 사례12의 복합 화이트박스 컴포넌트는 다수의 클래스로 구성된 객체모델을 EJB를 이용해 화이트박스 컴포넌트로 설계하는 것이다. 사례9와 사례10의 경우 다수의 세션 빈을 그룹하고 각각의 세션 빈의 인터페이스를 공개한다. 비동기 서비스의 경우 그 진입점이 비동기 메시지 처리를 위한 메시지 드리븐 빈이 되도록 설계하고 다수의 메시지 드리븐 빈과 다수의 세션 빈을 그룹하여 컴포넌트를 설계한다. 이때 비동기 서비스를 유지하기 위해 세션 빈의 인터페이스는 공개하지 않는다. 사례11과 사례12는 영속성 서비스를 필요로 하므로 사례9와 사례10의 경우에 영속성 서비스 처리를 위한 엔티티 빈을 추가하여 설계한다. 사례11에서 엔티티 빈이 직접 호출 가능하다는 것과 사례12에서 엔티티 빈의 인터페이스 역시 공개하지 않는다는 특징을 가진다.

사례9부터 사례12의 복합 블랙박스 컴포넌트는 다수의 클래스로 구성된 객체모델을 EJB를 이용해 블랙박스 컴포넌트로 설계하는 것이다. 이는 사례9와 사례12의 설계와 동일하지만 컴포넌트 내부를 공개하지 않고 외부에는 좀 더 큰 기능성(Coarse Grained Functionality)을 제공하기 위해서 동기 메시지인 경우 J2EE 패턴 중 Session Facade 패턴[9]을 적용한 하나의 세션 빈을 이용하여 그룹된 컴포넌트에 대한 접근점을 제공하고, 비동기 메시지인 경우 동기 메시지의 경우와 유사하게 메시지 드리븐 빈을 외부에 대한 접근점으로 제공한다. 동기 서비스인 경우 외부의 세션 빈의 인터페이스만을 외부에 공개하고 비동기 서비스인 경우 메시지 드리븐 빈을 통해 접근하게 되는 내부의 빈들에 대한 인터페이스를 제공하지 않는다. 컴포넌트의 디플로이먼트 디스크립터에 <ejb-client-jar>를 명시해 컴포넌트의 클라이언트에게는 공개 하고자 하는 인터페이스만을 공개할 수 있다.

4.3 컴포넌트 인터페이스 설계 기법

본 장에서는 컴포넌트의 세 가지 인터페이스인 *Provided* 인터페이스, *Required* 인터페이스, *Customize* 인터페이스를 EJB 기반의 설계 기법을 제시한다.

4.3.1 Provided 인터페이스 설계

컴포넌트의 기능성을 제공하는 *Provided* 인터페이스는 EJB에서 외부에 기능을 제공하기 위하여 사용하는 컴포넌트 인터페이스로 설계할 수 있다. 그림 5와 같이 EJB 컴포넌트의 *Provided* 인터페이스의 설계는 단일 컴포넌트와 복합 컴포넌트인 두 가지 경우로 나눌 수 있다. 단일 컴포넌트인 경우 내부 빈의 홈 인터페이스와 컴포넌트 인터페이스가 외부에 공개할 기능을 정의한다.

복합 컴포넌트인 경우 외부에 제공하고자 하는 컴포넌트의 기능성을 정의한 빈의 홈 인터페이스와 컴포넌트 인터페이스를 외부에 공개하고 다른 내부의 빈들은 외부의 클라이언트에게 공개되지 않으므로 인터페이스는 로컬 홈 인터페이스와 로컬 인터페이스로 정의한다. 클라이언트에게 제공하는 인터페이스는 클라이언트가 컴포넌트와 동일한 컨테이너에서 동작하지 않는 경우를 고려하여 반드시 리모트 홈 인터페이스와 리모트 인터

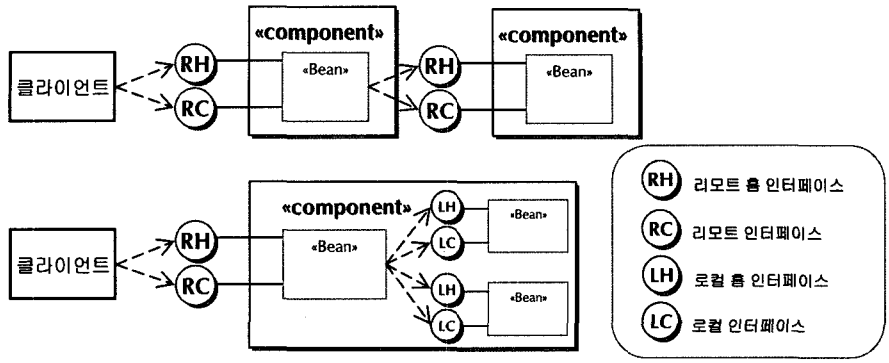


그림 5 EJB 컴포넌트의 Provided 인터페이스

페이스를 제공해야 하며 동일한 컨테이너에서 동작 시에 얻을 수 있는 성능상의 이점을 얻기 위해 로컬 홈 인터페이스와 로컬 인터페이스를 추가적으로 제공할 수 있다.

4.3.2 Required 인터페이스 설계

Required 인터페이스는 컴포넌트가 필요로 하는 기능을 명세하며 컴포넌트간의 의존관계를 보여주는 역할을 한다. 즉, 컴포넌트의 *Required* 인터페이스는 실제 EJB의 인터페이스로 직접 구현되는 인터페이스가 아니다. EJB에서 *Required* 인터페이스는 해당 컴포넌트에서 필요한 다른 컴포넌트의 *Provided* 인터페이스의 사용관계를 표현한다. 이러한 사용관계는 그림 6과 같이 명세서를 통하여 보여줄 수 있다.

Required 인터페이스 명세서

1. 컴포넌트 이름 : Membership Manager
2. JNDI 이름 : ejb.MemberMgr
3. 인터페이스 이름 : MembershipHome.class, Membership.class
4. 기능성
   - 사용자 등록
     사용자의 정보를 저장한다.
   - ...
5. *필요한 오퍼레이션*(optional)
   - registerMember (id : String, pw : Integer, ...) : void
   - ...
6. 관리될 데이터
   - 아이디, 패스워드, 이름, 주소, 전화번호
   - ...
7. 제약 사항
   - EJB 컴포넌트(BEA Web Logic 8.1 이상)

그림 6 Required 인터페이스 명세서의 예

Required 인터페이스는 외부 컴포넌트와의 의존관계를 명세한다. 외부 컴포넌트를 사용하는 컴포넌트는 *Required* 인터페이스에 명세된 컴포넌트의 홈 인터페이스와 컴포넌트 인터페이스를 얻은 후에 엔터프라이즈 빈에 메시지를 전달 한다. 따라서 외부 컴포넌트의 홈 인터페이스와 컴포넌트 인터페이스의 자바 클래스 파일이 필요하다. 그러므로 이러한 사용자에게 제공할 클래스를 묶은 Jar 파일을 제공해야 한다. 따라서 *Required* 인터페이스 명세서는 컴포넌트를 조립할 때에 필요한 인터페이스를 묶어서 컴포넌트 사용자에게 제공하는데 유용하다.

4.3.3 Customize 인터페이스 설계

컴포넌트의 가변성은 컴포넌트의 내부에 구현된다. *Customize* 인터페이스에 의해 가변치가 엔터프라이즈 빈에 설정되면, 내부 클래스가 가변치의 설정에 맞게 서비스한다. 우리는 컴포넌트의 인터페이스를 *Provided*와 *Customize* 인터페이스로 분리하였다. *Customize* 인터페이스는 사용자에게 기능을 제공하는 역할이므로 *Provided* 인터페이스와 동일하다. 하지만 쓰이는 용도가 다르다.

Provided 인터페이스는 컴포넌트의 기능을 컴포넌트 사용자인 빈 조립자(Bean Assembler)에게 제공하기 위해 사용되는 API이며, *Customize* 인터페이스는 빈 설치자(Bean Deployer)등과 같은 관리자에게 컴포넌트를 특화하기 위해 제공되는 API이다. 만약 같은 인터페이스에 두 종류의 오퍼레이션들이 모두 제공된다면, 일반 사용자가 사용하는 인터페이스에 관리자에게만 제공되는 오퍼레이션까지 포함되게 된다. 따라서 *Provided*와 *Customize* 인터페이스가 분리되어 설계 및 제공되어야 한다.

4.4 컴포넌트 가변성 설계 기법

본 절에서는 4.1.3절에서 식별한 컴포넌트 가변성 설계에서의 차이점을 해결하기 위한 EJB 플랫폼에서의 컴포넌트 가변성 설계 기법을 제안한다. 본 절에서 제안한 가변성 설계 기법을 이용하여 설계한 컴포넌트는 어플리케이션을 조립하는 사용자가 *Customize* 인터페이스를 이용하여 특화한 후 지속적으로 가변치에 대한 정보를 저장하고 있으며 실제 어플리케이션 이용자에게

Customize 인터페이스는 공개되지 않기 때문에 의도하지 않은 *Customize* 인터페이스의 호출을 막을 수 있다.

EJB 플랫폼은 컴포넌트 특화를 위한 장치를 제공하지 않기 때문에 컴포넌트 특화를 위해서는 별도의 장치를 이용해야 한다. 본 논문에서는 컴포넌트 내부의 엔터프라이즈 빈들을 특화하는 *Customize* 인터페이스와 특화 기법을 관리하는 별도의 엔터프라이즈 빈과 가변치 설정 정보를 저장하기 위한 일반 자바 객체(Plain Old Java Object, POJO)를 이용하는 기법을 제안한다.

이러한 두 개의 엔터프라이즈 빈과 자바 객체를 이용하여 그림 7과 같이 컴포넌트의 가변성을 설계할 수 있다. 가변성을 설계하기 위한 엔터프라이즈 빈인 *VariabilityBean*의 컴포넌트 인터페이스인 *Variability*를 *Customize* 인터페이스로 제공한다. *Variability*를 통해 가변치를 선택하고 *VariabilityBean*은 선택된 가변치를 지속적으로 유지하기 위해 *Variation* 클래스의 *static* 필드에 저장한다. 이는 가변치를 관리하는 *VariabilityBean*은 *java.io.Serializable* 인터페이스를 구현하여 직렬화되어 컨테이너에 의해 풀링(Pooling)되기 때문이다. 엔터프라이즈 빈의 *static* 필드는 직렬화되어 풀링될 수 없기 때문에 지속적인 값을 유지할 수 없다. 이러한 이유로 엔터프라이즈 빈의 *static* 필드의 사용은 EJB 명세에서 권장하고 있지 않다[1]. 가변치가 설정되어 컴포넌트가 배포된 이후 컴포넌트의 클라이언트는 *Loan* 인터페이스를 통해 엔터프라이즈 빈인 *LoanBean*에 오퍼레이션을 호출한다. *LoanBean*의 *foo1()* 오퍼레이션은 자바 클래스 *Variation*으로부터 선택된 가변치 값을 읽어와 해당 가변치에 따른 오퍼레이션을 수행한다.

그림 7을 통해 제시한 EJB에서의 컴포넌트 가변성 설계 기법은 컴포넌트의 내부에 설정된 가변치를 선택하는 기법이고, 이 외에도 컴포넌트 외부의 가변치를 컴포넌트에 플러그인하는 기법과 외부 프로파일을 이용한 컴포넌트 설계 기법이 있다.

4.5 EJB에서의 컴포넌트 조립 기법

본 절에서는 어플리케이션을 개발하기 위해 EJB 컴포넌트를 조립하는 기법을 그림 8과 같이 제안한다. CBD에서의 컴포넌트 조립은 관련된 컴포넌트들을 목표 어플리케이션의 아키텍처의 정의에 따라 배치하고, 이들 간의 관계를 설정하는 과정이다. EJB는 3-티어(Tier) 클라이언트-서버 모델을 가정한 미들웨어 표준으로서, 서버 측면의 2개 티어 컴포넌트를 정의하고 있다. 각 계층은 크게 클라이언트 계층, 비즈니스 계층, 데이타 계층으로 구분된다.

*클라이언트 계층*은 모델의 데이타를 JSP나 서블릿을 통해서 사용자에게 표현하고 사용자 인터페이스를 통해 사용자의 요구사항을 서버 측으로 보내는 역할을 한다. 이 계층에서 사용자가 원하는 기능을 서버로부터 요구하고자 할 때 사용자는 서버 측의 인터페이스를 통해서 세션 빈의 해당 기능을 수행한다.

*비즈니스 로직 계층*은 클라이언트에 서비스를 제공하기 위한 알고리즘 및 워크플로우를 제공한다. 이를 위해 로직 계층은 데이타 계층의 엔티티 빈을 호출하기 위해 세션 빈으로 설계 된다. 또한 비동기 서비스를 위해 메시지 드리븐 빈으로 설계된다. 이 계층에서는 세션 빈에서 비즈니스 수행 중 필요한 정보를 데이타 계층의 엔티티 빈을 통해 데이타 베이스나 레거시 시스템으로부터 필요한 데이타를 얻어서 기능을 수행하고 데이타 베이스에 저장한다.

*데이타 계층*은 엔티티 빈을 통해 데이타 베이스 및 레거시 시스템을 이용해서 데이타의 생성, 수정, 삭제,

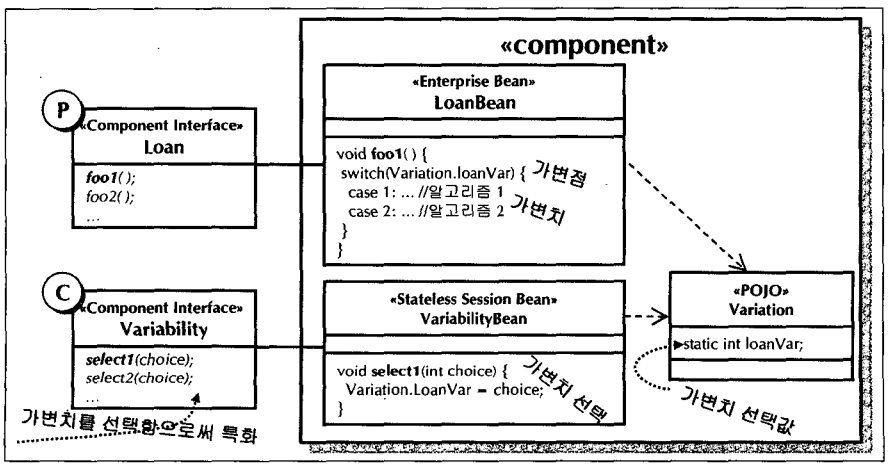


그림 7 선택형 가변성 설계 기법

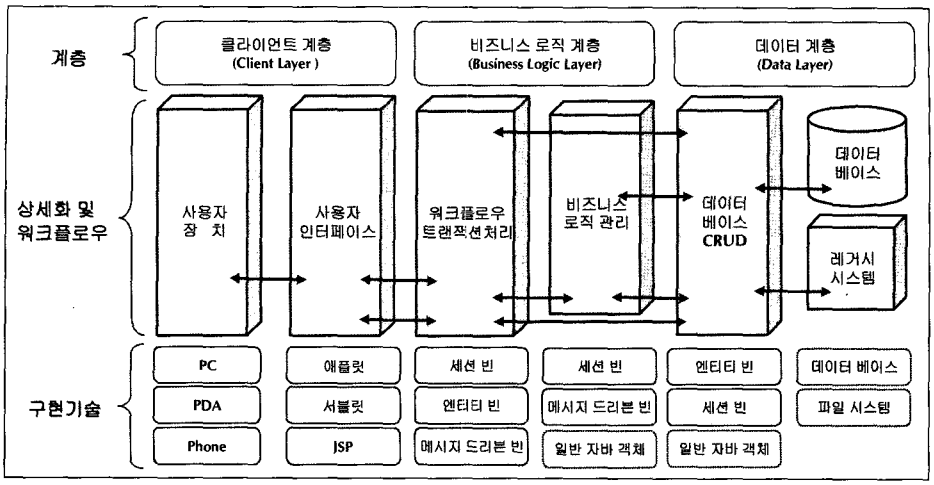


그림 8 EJB에서 컴포넌트 조립을 위한 3 티어 정의

조회를 수행한다. 이러한 데이타 계층으로서의 엔티티 빈은 클라이언트가 세션 빈에 접근하여 비즈니스를 수행한 후 해당 데이타를 영속적인 장치에 저장하고자 할 때 사용된다.

따라서 각 계층에 해당하는 빈들의 상호작용은 3.5절에서 설명한 컴포넌트의 조립 장치와 대응된다. 이는 EJB로 컴포넌트를 설계 시 다수의 컴포넌트가 조립되어 어플리케이션을 생성하는 것처럼 각 빈들의 조합으로 하나의 컴포넌트 혹은 어플리케이션의 생성이 가능하다는 것을 의미한다. 그림 8은 3개의 계층에 대한 설명을 기반으로 각 계층의 상세화 및 워크플로우와 이에 해당되는 구현기술 그리고 관련된 빈의 종류를 상세화 및 이를 구현하기 위한 기술 별로 분류한 것이다.

5. 사례연구

본 장에서는 제시한 CBD기반의 EJB의 설계 기법들을 사례연구를 통해 적용함으로써 기법의 적합성을 보인다. 사례연구의 도메인은 도서관 관리 시스템이며 제시한 설계기법을 적용하여 도서대출 컴포넌트를 설계하고 구현한다. 컴포넌트 개발을 위한 EJB 서버는 BEA사의 WebLogic Server 8.1 서비스팩5를 이용하였다.

5.1 도메인 요구사항 명세

도서대출 컴포넌트의 요구사항은 도서의 대출, 반납, 연기, 대출정보 검색의 도서대출 기능과 대출과 연체에 따라 발생하는 과금을 처리하는 회계관리 기능이다. 또한, COTS 컴포넌트로의 판매를 목적으로 특정 어플리케이션에 한정된 컴포넌트가 아니라 다수의 도서대출 시스템에서 재사용 가능한 컴포넌트를 필요로 한다. 도서 대출은 시스템의 성격에 따라 유료 또는 무료로 동작할 수 있어야 한다.

5.2 CBD 컴포넌트 설계

5.1절의 도메인 요구사항을 그림 9와 같이 UML 2.0의 복합 구조 다이어그램(Composite Structure Diagram)을 사용하여 CBD 컴포넌트로 설계하였다. 도서대출 컴포넌트는 도서대출 기능을 담당하는 *Loan*과 회계관리 기능을 담당하는 *Account*로 구성된다. 또한, 위의 요구사항을 만족하기 위해 *Loan*과 *Account*는 컴포넌트를 특화 할 수 있는 장치를 포함한다.

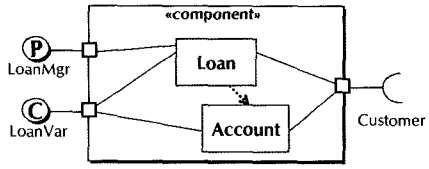


그림 9 CBD 컴포넌트 설계

5.3 EJB에서 컴포넌트 상세설계

도서대출 컴포넌트는 컴포넌트의 기능 설계와 컴포넌트의 가변성 설계로 나눌 수 있다. CBD 컴포넌트 설계를 그림 10과 같이 EJB를 위한 컴포넌트로 상세 설계하였다. 제시한 기법은 기능 설계와 가변성 설계를 컴포넌트 내부에서 역할에 따라 분리함으로써 비즈니스 로직의 설계는 가변성의 설계와 결합되지 않고 낮은 결합도를 유지할 수 있다. 또한 인터페이스를 기능을 제공하는 *LoanMgr*과 가변성을 위한 *VarMgr*로 분리하여 *VarMgr*은 어플리케이션 개발자에게는 제공하지 않고, 빈 관리자만 사용할 수 있다.

그림 10의 컴포넌트 설계모델에서 컴포넌트의 기능성 설계는 컴포넌트의 기능군을 정의하는 *Provided* 인터페이스인 *LoanMgr*, 대출을 관리하는 *LoanBean*, 과금

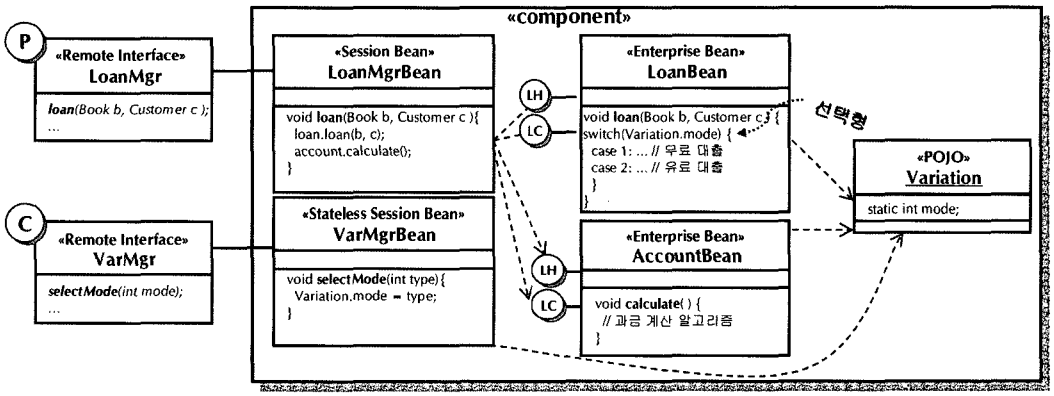


그림 10 도서대출 컴포넌트 상세 설계

과 관련된 회계처리를 담당하는 *AccountBean*, *LoanBean*, *AccountBean*을 묶어 큰 기능성을 제공하기 위한 *LoanMgrBean*으로 구성된다. 도서대출 컴포넌트를 블랙박스 컴포넌트로 상세 설계 하기 위해서 *LoanBean*과 *AccountBean*은 *LoanMgrBean*을 통해서 접근 가능하게 하며 외부에 인터페이스를 공개하지 않고 로컬 홈 인터페이스와 로컬 인터페이스를 사용하는 내부 요소로 설계한다.

가변성 설계는 가변성을 설정하기 위한 *Customize* 인터페이스인 *VarMgr*과 가변성 설정을 처리하는 *VarMgrBean*, 설정된 가변치를 저장하는 자바 클래스 *Variation*로 구성된다. 컴포넌트의 비즈니스 로직을 처리하는 부분은 설정된 가변치를 이용하기 위하여 *Variation*과 결합관계를 가질 뿐 가변성을 처리하는 다른 부분과는 결합관계를 갖지 않는다.

5.4 EJB에서 인터페이스 구현

도서대출 컴포넌트의 인터페이스는 기능성을 제공하기 위한 *Provided* 인터페이스와 특화하기 위한 *Customize* 인터페이스를 설계 및 구현하였다. *Required* 인터페이스는 4.3.2절에서와 같이 명세할 수 있으며 본 사례연구에서는 제외하였다.

그림 11과 같이 *Provided* 인터페이스와 *Customize* 인터페이스를 EJB의 엔터프라이즈 빈의 컴포넌트 인터페이스를 통하여 구현하였다. EJB 명세에 별도의 *Customize* 인터페이스가 존재하지 않으므로 컴포넌트 인터페이스를 이용하여 *Customize* 인터페이스를 구현하였다.

```
// Provided 인터페이스 구현
public interface LoanMgr extends javax.ejb.EJBObject {
   ...
   public void loan(Book b, Customer c) throws java.rmi.RemoteException;
   ...
}
// Customize 인터페이스 구현
public interface VarMgr extends javax.ejb.EJBObject {
   public void selectMode(int mode) throws java.rmi.RemoteException;
}
```

그림 11 도서대출 컴포넌트의 인터페이스

5.5 EJB에서의 컴포넌트 구현

5.2절에서 논의했던 것과 같이 도서대출 컴포넌트는 크게 비즈니스 로직의 처리를 위한 엔터프라이즈 빈과 가변성 처리를 위한 엔터프라이즈 빈과 자바객체로 구성된다. 먼저, 가변성 로직의 구현은 가변성을 담당하는 *VarMgrBean*의 컴포넌트 인터페이스인 *VarMgr*의 가변성 설정 오퍼레이션인 *selectMode*를 사용자가 호출하면 가변성을 처리하는 *VarMgrBean*은 해당 가변치 설정을 지속적으로 유지하기 위해 *Variation* 자바 클래스에 *static* 필드에 저장한다. 구현은 그림 12와 같다.

다음으로, 컴포넌트의 비즈니스 로직의 구현에서 *LoanMgrBean*의 컴포넌트 인터페이스인 *LoanMgr*의 비즈니스 오퍼레이션을 호출하면 *LoanMgrBean*은 해당 오퍼레이션의 수행을 위해 내부의 빈들에 해당 비즈니스 로직의 처리를 위임한다. 또한, *LoanMgrBean*은 워크플로우를 조정하는 역할을 한다. 이때 *LoanBean*과 *AccountBean*은 해당 오퍼레이션을 수행하는데 필요한 어플리케이션의 요구사항에 맞게 설정된 가변치 정보를

```
// VarMgrBean – 가변치 관리(선택, 설정)
public class VarMgrBean implements SessionBean {
        ... // EJB 필요 메소드들

          public void selectMode(int mode) {
            Variation.mode = mode;
          }
}
// Variation – 가변치 설정 저장
public class Variation {
   public static int mode;
}
```

그림 12 가변성 구현

```
// LoanMgrBean – 외부에 공개한 기능 수행. 워크플로우 제어.
public class LoanMgrBean implements SessionBean {
        ... // EJB 필요 메소드들
        public void loan(Book book, Customer customer) {
    loan.loan(book, customer)
    account.calculate();
        }
}
// LoanBean - 대출 관련 비즈니스 로직
public class LoanBean implements SessionBean {
        ... // EJB 필요 메소드들

        public void loan(Book book, Customer customer) {
                switch(Variation.mode) { //선택된 가변치 정보 이용
                        case 1: // 무료 대출
                        case 2: // 유료 대출
                }
        }
}
// AccountBean – 회계관련 비즈니스 로직
public class AccountBean implements SessionBean {
        ... // EJB 필요 메소드들

    public void calculate() {
       // 과금 계산 알고리즘
    }
}
```

그림 13 비즈니스 로직 구현

Variation 자바 클래스로부터 얻어온다. 구현은 그림 13과 같다.

본 사례 연구를 통해 순수 EJB 아키텍처에서 제공하지 않는 블랙 박스 형식의 복합 컴포넌트인 Loan 컴포넌트를 구현하기 위해 본 논문에서 제시한 설계 기법을 이용하여 EJB 기반의 컴포넌트를 설계하였다. 또한 도서 대출 알고리즘에 대한 가변성을 갖는 컴포넌트를 설계하여 구현하였다. 따라서 컴포넌트 최종 사용자는 큰 단위의 재사용이 가능하며, 가변성 기법을 사용하여 컴포넌트의 소스 수정 없이 사용자가 원하는 대출 방식과 연체 계산 방법으로 Loan 컴포넌트를 특화 할 수 있었다.

6. 평 가

본 장에서는 CBD의 특징과 본 논문에서 제시한 EJB 환경을 적용한 컴포넌트 설계 기법을 비교한다. 여러 문헌에서 기술된 컴포넌트의 특징은 다음과 같다. Szyperski는 컴포넌트의 모듈성과 조립을 위한 아키텍처를 주장했다[11]. D'Souza의 Catalysis에서는 특화와 *Provided* 인터페이스와, *Require* 인터페이스를 설명했다[12]. Heineman과 Council은 표준 컴포넌트 모델의 중요성과 컴포넌트의 규약(Contract)인 인터페이스를 강조했으며, 도메인에 종속적인 큰 단위의 비즈니스 컴포넌트를 설명하였다[7]. 또한 컴포넌트는 여러 규약을 만족시킬 수 있으므로, 여러 인터페이스를 구현하여 하나의 컴포넌트를 만들 수 있다. 즉, 컴포넌트는 복수의 인터페이스를 가질 수 있다[13].

이를 근거로 한 평가 항목은 표 2와 같다. 본 논문은 8가지의 컴포넌트 특징을 적용하기 위해 EJB를 이용한 설계 기법을 제시하였다. 본 논문에서 제안한 EJB 기반의 컴포넌트 설계 기법은 EJB 아키텍처를 준수하고, 복합 컴포넌트의 설계를 가능 하게 하며, 블랙 박스 형식의 컴포넌트 설계가 가능하도록 하였다. 또한 다수의 컴포넌트 인터페이스를 제공 할 수 있으며 컴포넌트 특화 기법을 위한 설계 기법도 제공하였다. 또한 사례 연구를 통해 가변성이 있는 CBD 컴포넌트를 EJB를 이용하여 설계 및 구현하였다. 따라서 본 논문의 설계 기법을 적용하여 EJB 아키텍처를 활용한 고품질의 컴포넌트를 생산 할 수 있다.

7. 결 론

산업계와 학계 모두 CBD의 중요성이 인식되고 있다. CBD 기술은 효율적인 재사용 기술이며 대형 시스템 개발 시 사용되고 있다. EJB는 엔터프라이즈 환경에서 분산 컴포넌트 어플리케이션을 개발하기 위한 자바 기반의 서버 측 컴포넌트 아키텍처이다. EJB는 분산 서비스, 트랜잭션, 영속성, 보안을 미들웨어 차원에서 제공하여 개발자의 노력을 줄일 수 있기 때문에 CBD를 위한 최적의 플랫폼으로 인식되어 왔다.

CBD 컴포넌트를 만들기 위해 EJB를 활용한 EJB 컴

표 2 CBD 컴포넌트의 특징과 본 논문의 설계 기법 적용 평가

비교 대상 / 컴포넌트 특징	EJB를 이용한 본 논문의 CBD 적용 사항
모듈화 (단일, 복합 컴포넌트)	객체단위의 EJB 단일 빈들을 사용하여 EJB 기반의 복합 컴포넌트로의 설계 가능
표준 컴포넌트 모델	EJB를 사용하므로 표준 EJB 아키텍처를 사용
Provide 인터페이스	EJB의 컴포넌트 인터페이스를 활용하여 지원
Required 인터페이스	UML의 Required 인터페이스 개념을 적용한 명세 지원
Customize 인터페이스	EJB 인터페이스를 이용한 구현 기법 제시
특화	EJB에서 가변성을 지원하기 위한 3가지 기법을 제시
다수의 인터페이스 지원	빈에는 한 개의 인터페이스만 있지만, 복합 컴포넌트를 제공하여 여러 개의 인터페이스 제공 가능
컴포넌트 조립	EJB는 한 개의 빈 수준의 아키텍처를 제시하지만, 본 논문에서는 어플리케이션 수준의 조립 패턴 제시

포넌트 설계 기법이 필요하다. 본 논문에서는 컴포넌트는 특성에 따라 그룹화된 여러 개의 인터페이스를 EJB 컴포넌트에서 지원 가능하게 하였다. 기본적으로 한 개의 객체 단위인 엔터프라이즈 빈을 복합 컴포넌트로 설계 하는 기법을 제시하였다. 또한 EJB 컴포넌트를 특화하기 위한 기법도 제시하였다.

본 논문에서는 CBD의 요소를 만족시키기 위해 EJB를 활용하여 설계하는 기법을 제시하였다. CBD 사례연구를 통해 본 논문의 기법의 적용 내용을 보이고, CBD 컴포넌트의 특징과 비교하였다. 제시한 설계 및 구현 기법을 이용하여 EJB 환경에서 EJB 컴포넌트를 재사용성, 활용성, 이식성을 더욱 증가 시킬 것으로 기대한다.

참 고 문 헌

[1] DeMichiel, L., *Sun Microsystems, Enterprise JavaBeans™ Specification, Version 2.1*, Sun Microsystems, pp.1-635, 2002.

[2] Roman, E., *Mastering Enterprise JavaBeans Third Edition Wiley*, 2005.

[3] Muthig, D. and Atkinson, C., "Model-Driven Product Line Architectures," *SPLC2 2002, LNCS Vol. 2379*, pp.110-129, 2002.

[4] Kim, S., Min, H., and Rhew, S., "Variability Design and Customization Mechanisms for COTS Components," *Lecture Notes in Computer Science Vol. 3480*, pp.57-66, May, 2005.

[5] Sinnema, M., "COVAMOF: A Framework for Modeling Variability in Software Product Families," *LNCS 3154*, pp.197-312, 2004.

[6] Kim, S. Her, J, and Chang, S., "A Theoretical Foundation of Variability in Component-based Development," Information and Software Technology, Vol. 47, pp.663-673, July, 2005.

[7] Heineman, G. and Councill, W., *Component-Based Software Engineering*, Addison Wesley, 2001.

[8] Manolescu, D.A., and Johnoson, R.E., "A Micro Workflow Framework for Compositional Object-Oriented Software Development," Workshop on the Implementation and Application of Object-Oriented Workflow Management Systems II, OOPSLA, 1999.

[9] Alur, D., Crupi, J., and Malks D., *Core J2EE Patterns 2nd*, Prentice Hall, 2003.

[10] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison-Wesley, 2001.

[11] Syzperski, C., *Component Software, Second Edition*, Addison Wesley, 2002.

[12] D'Souza, D. and Wills, A. C., *Objects, Components, and Frameworks with UML*, Addison Wesley Longman, Inc. 1999.

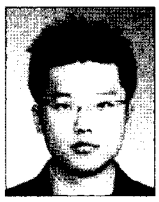
[13] Object Management Group (OMG) *Unified Modeling Language: Superstructure, Version 2.0*, ptc/03-08-02, 20

김 수 동

1984년 Northeast Missouri State University 전산학 학사. 1988년/1991년 The University of Iowa 전산학 석사/박사. 1991년~1993년 한국통신 연구개발단 선임연구원. 1994년~1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월~현재 숭실대학교 컴퓨터학부 부교수. 관심분야는 객체지향 S/W공학, 컴포넌트 기반 개발(CBD), 제품계열 공학(PLE), 모델 기반 아키텍처(MDA), 시스템 온 칩(SoC)

민 현 기

1999년 건양대학교 전자계산학과 공학사 2001년 숭실대학교 컴퓨터학과 공학석사 2005년 숭실대학교 컴퓨터학과 공학박사 2005년 3월~현재 숭실대학교 전임연구원. 관심분야는 컴포넌트 기반 개발(CBD), 제품계열 공학(PLE), 모델 기반 아키텍처(MDA)



이 진 열

2005년 삼육대학교 컴퓨터과학과 이학사 2005년 3월~현재 숭실대학교 석사과정 관심분야는 컴포넌트 기반 개발(CBD), 제품계열 공학(PLE)

김 성 안

2005년 서울산업대학교 컴퓨터공학과 공학사. 2005년 3월~현재 숭실대학교 석사과정. 관심분야는 컴포넌트 기반 개발(CBD), 소프트웨어 패턴