EJB와 COM+ 결합을 위한 모델기반 컴포넌트 변환 기법

(A Component Transformation Technique based on Model for Composition of EJB and COM+)

최 일 우 † 신 정 은 †† 류 성 열 †††
(IlWoo Choi) (Jungun Shin) (SungYul Rhew)

요 약 현재 EJB(Enterprise Java Beans), COM+(Component Object Model+)등의 서로 다른 컴포넌트 참조 모델(Component reference model)을 기반으로 한, 상이한 컴포넌트 시스템 간 통합(Integration)에 대한 새로운 기술들이 제기되고 있다.

동일한 컴포넌트 플랫폼에서 컴포넌트 간 운용은 소스레벨의 결합(Composition)을 통해 이루어진다. 그러나 상이한 컴포넌트 플랫폼의 경우, 유사 도메인 컴포넌트임에도 불구하고 컴포넌트 간 결합은 불가능한 실정이다.

본 논문에서는 상이한 컴포넌트 플랫폼 즉, EJB와 COM+ 컴포넌트 간의 결합 문제를 모델기반의 컴포넌트 변환 기법으로 해결 하였다.

EJB, COM+ 컴포넌트 간 결합을 위해 각 참조모델을 비교, 분석하여 구현 독립적(Implementation Independent)인 가상 컴포넌트 모델(Virtual Component Model)과 상호 변환을 위한 구현 테이블(Implementation Table)을 제시하였다.

가상 컴포넌트 모델과 구현 테이블을 참조, 각 구현 모델을 가상 컴포넌트 모델로 일반화하거나 가상 컴포넌트 모델링을 통하여 플랫폼의 구현 독립적인 가상 컴포넌트 모델을 작성하고, 선택적으로 EJB와 COM+로 변환 가능하게 한다.

상이한 컴포넌트 플랫폼으로의 효율적인 모델변환 방법을 제시함으로서 EJB와 COM+ 컴포넌트간의 결합이 가능하다.

키워드 : EJB, COM+, 컴포넌트 결합, MDA, PIM, PSM, 모델변환

Abstract At present, new techniques based on different component reference models for the integration of component and system of different platforms, such as EJB and COM+, are introduced. The operation between the components in the identical component platform is realized by the composition at the source level. In case of the different component platform, however, it is impossible to use combined components in real condition although they are components of similar domain.

In this paper we proposed a solution for the composition problem by using component transformation methodology based on model between EJB and COM+ components which are different components. For the composition between EJB and COM+ components, we compared and analyzed each reference model, then proposed the Virtual Component Model which is implementation independent and the Implementation Table for the mutual conversion.

Reffering to the Virtual Component Model and the Implementation Table, we can generalize each Implementation model to the Virtual Component Model, make the Virtual Component Model which is implementation independent through the virtual component modeling, transform EJB and COM+ components selectively. Proposing the effective Model Transformation method to the different component platform, we can combine EJB and COM+ components.

Key words : EJB, COM+, Component Composition, Virtual Component Model, Implementation Table, MDA, PIM, PSM, Model Transformation

† 학생회원 : 숭실대학교 컴퓨터학과
lucifer@selab.ssu.ac.kr
†† 비 회 원 : 삼성 전자
kikiwa@selab.ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
syrhew@comp.ssu.ac.kr
논문접수 : 2003년 4월 4일
심사완료 : 2003년 8월 21일

1. 서 론

최근 CCM, COM+, EJB등 다양한 컴포넌트 표준들이 등장함에 따라, 서로 다른 컴포넌트 참조 모델을 기반으로 개발된 컴포넌트 간 호환성 문제가 제기되고, 나아가 EAI(Enterprise Application Integration)의 일환으로 컴포넌트 시스템 간 통합에 대한 연구가 필요시되고 있다[1].

기존의 시스템 통합에 관한 연구는 컴포넌트 레벨의 컴포넌트 간 결합(Component Composition)을 그다지 고려하지 않고 있다[2]. 실제로 COM+을 바탕으로 네트워크상의 COM+의 커뮤니케이션 문제해결을 위한 컨테이너 개발[3]이나 EJB와 CORBA의 결합[4]등에 대한 연구가 수행되고 있으나 COM+와 EJB의 연동 등과 같이 상이한 플랫폼을 기반으로 하는 컴포넌트 간 상호 운용에 대한 연구는 현재 미비하다.

지금까지 컴포넌트간의 운용은 동일한 컴포넌트 플랫폼하의 소스레벨 운용으로서, 상이한 컴포넌트 플랫폼에서 개발된 컴포넌트와 상호 운용은 하나의 참조 모델을 포기, 다른 하나의 참조 모델로 재개발을 수행해야만 했다. 그래서 상이한 컴포넌트 참조모델을 기반으로 개발된 컴포넌트들은 유사 도메인임에도 불구하고 상호간의 실질적인 재사용이 불가능하다[4]. 이러한 문제의 현실적 해결 방법은 소스 기반이 아닌 모델기반으로 컴포넌트 간 결합을 수행할 때 가능하다.

본 논문에서는 EJB와 COM+ 컴포넌트 간 결합 문제를 해결하기 위하여 모델기반 컴포넌트 변환 기법을 제시한다. 본 논문을 통하여 제시되는 "가상 컴포넌트 모델"은 EJB와 COM+의 구현에 종속적이지 않는 플랫폼에 독립적인 컴포넌트 모델인 동시에 EJB와 COM+의 구현 시 실용적으로 사용될 수 있는 모델이다. 가상 컴포넌트 모델의 정의를 위한 구조적 접근 방법으로 MDA(Model Driven Architecture)의 개념[5]을 확대 적용한다. 가상 컴포넌트 모델은 MDA의 PIM(Platform Independent Model)과 PSM(Platform Specific Model)의 중간단계에 위치하는 모델로 서로 다른 플랫폼 즉, EJB와 COM+간의 컴포넌트 명세(Specification)구조를 비교, 분석하여 일반화한 모델이다.

또한 구현에 종속적인 모델을 가상 컴포넌트 모델로 혹은 가상 컴포넌트 모델을 구현 종속적인 모델로 재구성하기 위한 모델 재구성 규칙(Restructuring Rule)으로 EJB와 COM+ 플랫폼에 상호 적용되는 제약 사항, 상호 연관성등과 같은 조건을 명시한 "구현 테이블"을 제시한다.

본 논문에서 제시하는 가상 컴포넌트 모델을 적용함으로서 EJB와 COM+간 모델기반의 컴포넌트 변환을 가능하게 하고 컴포넌트 개발 시 적용되는 플랫폼에 특성화 된 부분은 모델 재구성 규칙으로 구현 테이블을 제시함으로써 개발자에게 손쉬운 모델 변형을 제공할 수 있을 것으로 기대된다.

2. 관련 연구

2.1 MDA(Model Driven Architecture)

현재 컴포넌트 기술은 구현뿐만 아니라 컴포넌트 관련 기술 전반을 포함하는 표준구조를 정의하는데 그 초점을 맞추고 있다. 구현을 중심으로 컴포넌트 기술의 표준 구조를 정의하는 것은 시장성 측면에서 불가능할 뿐만 아니라 호환성 측면에서도 문제가 있기 때문에 컴포넌트 기술 요소들 간의 호환성을 유지하기 위해서는 컴포넌트와 관련된 요소들을 포함하는 표준 구조의 정의가 필수적이라 할 수 있다.

OMG의 MDA는 모든 컴포넌트 기술 요소의 표준 메타 모델을 정의하며, 정의된 표준 메타 모델을 기반으로 각 구성 요소를 정의함으로써 모든 컴포넌트 기술 요소들의 호환성 및 시스템 간 동작성을 보장하고 있다[6]. MDA는 시스템의 기능을 명세화 한 하나의 모델을 여러 플랫폼으로 실체화 할 수 있도록 하고, 상이한 어플리케이션들이 플랫폼 기술이 발전됨에 따라 시스템 간의 결합과 상호 운영을 가능하도록 한다.

IT(Information Technology) 시스템은 시스템이 제공하는 플랫폼에 독립적인 기능 부분과 특정 기술 플랫폼을 바탕으로 한 플랫폼 종속적인 기능에 대한 구현을 분리하여 명세화 할 수 있다. MDA에서는 IT 시스템을 플랫폼 독립적인 부분과 종속적 부분을 분리하여 명세화 하는 방법을 정의하고 구조화한다[7]. MDA에서 모델에 대한 구조는 그림 1과 같이 4계층으로 나뉜다. 최하위 계층으로 데이타를 나타내는 M0 계층, 모델 계층인 M1 계층, 메타 모델 계층인 M2 계층, 메타 메타 모델 계층인 M3계층으로 구분한다.

MDA의 핵심은 그림 2와 같이 메타 모델을 기반으로 구현 환경에 독립적 모델(Platform Independent Model

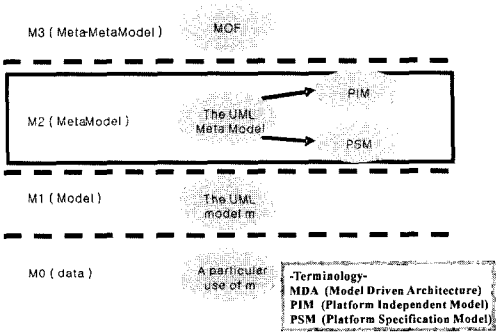


그림 1 MDA 모델의 계층구조

: PIM)을 자동으로 각 구현 환경에 적합한 구현 종속적 모델(Platform Specific Model : PSM)로 변환할 수 있는 구조를 정의하는 것이다. 즉 표준 메타 모델을 기반으로 구현 환경에 적합한 모델을 배치함으로써, 구현 단계에서의 생산성 향상뿐만 아니라 표준 메타 모델에 기반을 둔 시스템들의 일반적 호환성을 기대할 수 있게 되는 것이다[8].

그러나, 현재 MDA에서는 PIM과 PSM을 위한 구조만 정의되었을 뿐 각 플랫폼의 특성들 때문에 PIM과 PSM간의 변환 방법 및 PSM에서 PIM으로의 역공학 방법등에 구체적인 연구가 필요하다. 또한 PIM과 PSM의 사이에 특정 플랫폼간의 변환을 손쉽게 지원하는 자동화 도구가 지원되어야 한다. MDA를 지원하는 자동화 도구로는 IO사의 ArcStyler, PLASTIC S/W의 PLASTIC2003 등을 들 수 있는데 ArcStyler의 경우 CBD방법론을 개발 지원도구와 함께 제공하며, PLASTIC의 경우 UML Profile을 지원, Approach라는 개념을 도입하여 여타 개발 프로세스와의 연계를 지원한다. 이러한 도구들은 각각 PIM기반의 모델에 Stereotype, TagDefinition, DataType등을 이용한 UML의 확장과 메타프로그래밍을 수행하여 PSM 혹은 소스코드로의 자동 변환을 유도한다.

그러나 이러한 방법은 확장성이 뛰어난 반면 변환 표준이 존재하지 않고 복잡하므로 본 논문에서는 이러한 문제의 일부를 EJB와 COM+ 모델간의 가상 컴포넌트 모델과 구현테이블을 이용하여 손쉬운 모델간의 변환을 수행하였다.

2.2 EJB와 COM+의 구조

2.2.1 EJB의 구조

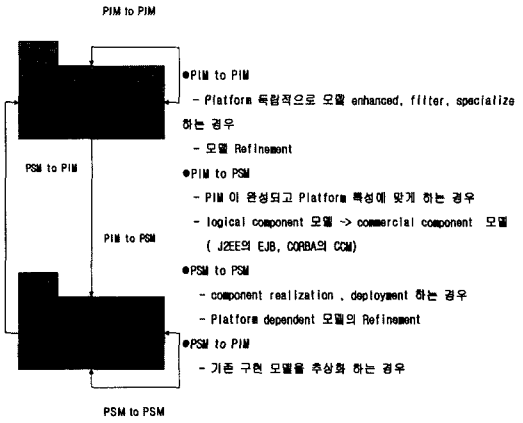


그림 2 MDA의 PIM과 PSM

EJB의 구조는 표 1, 그림 3과 같이 구성된다[9,10].

그림 3은 EJB 구조와 EJB 컴포넌트 시스템 개발을 위해 인터페이스와 클래스를 모델링 한 클래스 다이어그램의 예이다.

2.2.2 COM+의 구조

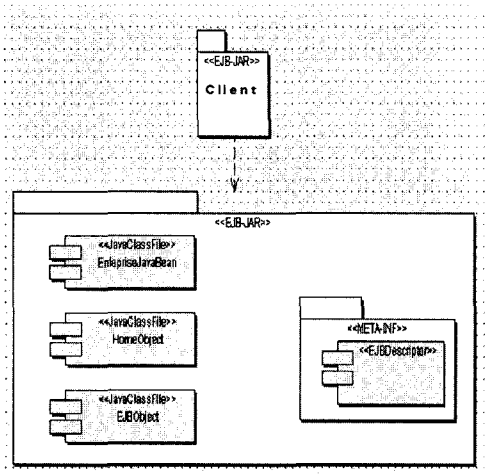
COM+의 구조는 표 2, 그림 4와 같이 구성된다[11,12].

그림 4는 COM+구조와 COM+ 컴포넌트 시스템을 개발할 때, 컴포넌트를 구성하는 인터페이스와 클래스들을 일반화하여 클래스 다이어그램으로 표현한 것이다.

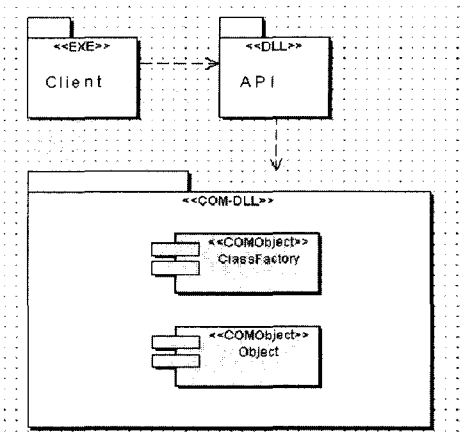
표 1, 2의 EJB와 COM+의 구조를 살펴보면, 관점에 따라 비슷한 역할을 하는 인터페이스 및 유사 개념들이 존재한다[14]. 이러한 기능적 유사성을 이용하여 EJB, COM+, CCM 컴포넌트를 엔티티(Entity), 세션(Ses-

표 1 EJB 구성요소

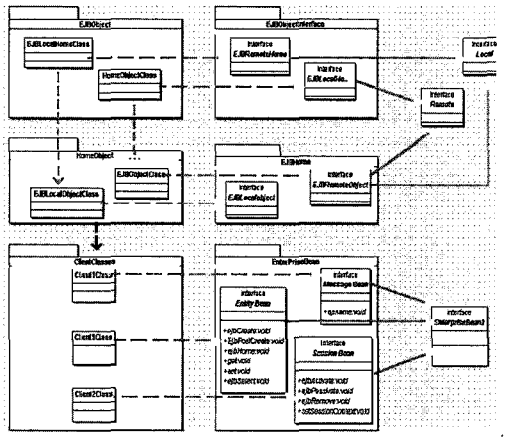
구성요소 및 EJB 인터페이스		
이름		설명
EnterpriseBean :EnterpriseBean은 클라이언트에게 서비스를 제공하는 메소드를 가지고 있는 부분으로 엔티티 빈, 세션 빈, 메시지 드리븐 빈의 세 종류의 빈이 있다.	EntityBean	데이타 저장 및 검색하기 위한 인터페이스. EnterpriseBean을 구현함
	SessionBean	계산이나 작업(Process)을 처리.
	MessageDrivenBean	컨테이너는 클라이언트로부터 JMS 메세지를 받는 경우 실행.
EJBObject :Object Interface인 EJBObject는 클라이언트가 서비스 요청을 할 때 HomeObject로부터 생성된다. EJBObject는 클라이언트와 EJBBean 사이의 중간 매개자로서 클라이언트가 EJBBean의 메소드를 직접 접근하지 않고 EJBBean 서비스를 받을 수 있도록 하며, EJBRemoteObject와 EJBLocalObject 2가지 종류가 있다.	Remote Local	클라이언트가 호출할 수 있는 비즈니스 메소드를 선언하기위한 인터페이스.
HomeObject :Home Interface의 Object인 HomeObject는 클라이언트가 Bean을 사용하기 위해 처음으로 EJB 서버에 서비스를 요청하는 부분으로 EJBHome과 EJBLocalHome 두 종류가 있다.	EjbLocalHome EjbHome	클라이언트를 위해EJB 서버에 설치된 엔터프라이즈빈을 생성, 관리, 삭제하기위한 인터페이스.



(a) EJB의 구조

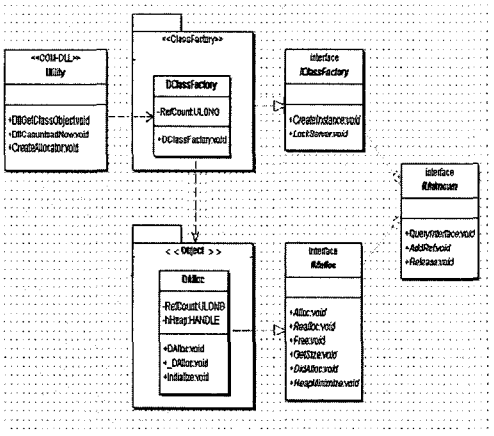


(a) COM+의 구조



(b) EJB 시스템 설계의 예

그림 3



(b) COM+시스템 설계의 예

그림 4

표 2 COM+ 구성요소

구성요소	COM인터페이스	
	이름	설명
DLL API <<DLL>>API는 현재 Microsoft의 OS를 통해 COM+ 라이브러리에서 기본적으로 제공되며, 클라이언트가 COM+ 서버에 서비스를 요청할 때 처음으로 호출되는 부분이다. 이 영역은 개발자가 실제 COM+을 개발할 때 구현할 필요가 없음으로 가상 컴포넌트 모델을 정의할 때 고려대상에서 제외한다.	IUnKnown	모든 오브젝트 뿐 아니라 모든 인터페이스가 보유해야 하는 기본적인 오퍼레이션을 가지고 있는 COM에서의 기본 인터페이스.
COM DLL <<COM-DLL>>패키지는 클라이언트가 서비스를 요청할 때, 호출된 API 내부 함수로부터 <<COM-DLL>>의 Utility 클래스가 호출되는 부분이다.		
COMObject ClassFactory <<COMObject>> ClassFactory는 <<COM-DLL>>의 Utility로부터 호출되며, 클라이언트에게 서비스를 제공하는 <<COMObject>> Object를 생성하는 역할을 한다.	IClassFactory	부여된 CLSID에 해당하는 클래스의 오브젝트를 조작하기 위한 오퍼레이션을 정의한 인터페이스.
COMObject Object <<COMObject>> Object부분은 클라이언트가 원하는 서비스를 제공하는 메소드를 포함하고 있다.		

sion), 서비스(Service)의 공통 기능군으로 분류하고 이 구조에 맞게 컴포넌트를 구분[13]하는 등의 시도들이 있으나, 아직까지 EJB와 COM+의 기능적 구조 및 관계에 대한 비교를 통하여 컴포넌트 간 결합을 수행한 사례는 미비하다.

3. EJB와 COM+ 참조 모델의 구조적 결합

3.1 가상 컴포넌트 모델과 구현 테이블

본 논문에서 제시하는 가상 컴포넌트 모델과 구현 테이블의 정의는 다음과 같다.

[정의 1] 가상 컴포넌트 모델(Virtual Component Model)
: 특정 컴포넌트 모델을 상이한 컴포넌트 모델과 구조적으로 결합하기 위해 기존 컴포넌트 표준 구조들을 결합, 일반화하여 플랫폼에 독립적으로 정의한 모델. 개념 모델과 상세 설계 모델의 중간 단계에 위치하는 모델이다.

[정의 2] 구현 테이블(Implementation Table)
: 가상 컴포넌트 모델의 요소와 특정 플랫폼 기반의 컴포넌트 모델 요소와의 관계 및 역할(Role)을 정의한 테이블.

본 논문에서 제시하는 가상 컴포넌트 모델은 도메인의 비즈니스 프로세스 모델(Business Process Model)이나 개념적 설계 모델(Conceptual Design Model)과는 차이를 갖는다. 가상 컴포넌트 모델은 상세 설계 모델이기는 하나 특정 컴포넌트 스펙이 반영되지 않은 상태로 존재하고 특정 컴포넌트 스펙의 특징을 반영하기 위한 방법으로 구현 테이블을 사용한다. 본 논문에서 고려하는 컴포넌트 모델은 EJB와 COM+로 한정된다.

그림 5는 컴포넌트 시스템 개발에 있어서 가상 컴포넌트 모델과 구현 테이블의 관계를 설명한다. 가상 컴포넌트 모델은 EJB와 COM+ 표준 구조를 일반화하여 정의한다. 가상 컴포넌트 모델을 적용함으로 EJB와 COM+ 플랫폼 상의 컴포넌트들을 모델기반으로 결합

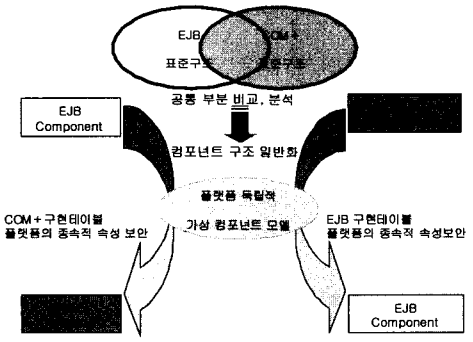


그림 5 컴포넌트 개발에 있어 가상 컴포넌트 모델과 구현테이블간의 관계

가능하고, 플랫폼에 따라 달라지는 컴포넌트의 구현 구조는 구현 테이블을 적용하여 EJB 컴포넌트 구조나 COM+ 컴포넌트 구조로 매핑 할 수 있다. 즉, 이전에는 서로 다른 플랫폼상의 컴포넌트가 기능적으로 결합 가능성이 있다 하더라도 실질적인 컴포넌트간의 결합이 불가능 하였지만, 가상 컴포넌트 모델과 구현 테이블에 적용함으로써 컴포넌트 기반의 플랫폼에 상관없이 선택에 따라 기능적으로 결합된 컴포넌트를 특정 플랫폼 기반의 컴포넌트 모델로 매핑, 새로운 컴포넌트 혹은 시스템으로 개발 할 수 있게 되므로 컴포넌트의 구조적 결합 문제가 해결 가능하다.

EJB와 COM+ 컴포넌트 기반 시스템의 구조적 결합을 위한 가상 컴포넌트 모델은 MDA에서 제시하는 모델구조를 차용한다. 그림1에서 MDA의 M2 계층은 플랫폼에 독립적 모델인 PIM과 플랫폼에 따른 명세화 모델인 PSM으로 세분화되고 PIM을 PSM으로 변환하기 위한 규칙을 필요로 한다. 본 논문에서는 MDA의 PIM, PSM의 개념을 적용, 가상 컴포넌트 모델은 PIM과 PSM의 사이에 위치하는 구조적 측면의 컴포넌트 결합을 위한 플랫폼에 독립적인 모델로 정의, EJB와 COM+ 컴포넌트의 표준 구조를 비교, 분석 하고 각 컴포넌트에서 제공하는 인터페이스, 오퍼레이션들의 공통점과 가변적인 부분을 추출한다.

[정의 1]의 가상 컴포넌트 모델은 OMG의 MDA모델에서 제시하는 PIM과 PSM의 중간적 특성을 가지고 있다. 가상 컴포넌트 모델은 EJB와 COM+의 컴포넌트 표준 구조를 일반화한 플랫폼에 독립적인 모델이며 UML을 사용 표기한다.

[정의 2]의 구현 테이블은 플랫폼에 독립적인 가상 컴포넌트 모델을 구축 후, 플랫폼의 특성을 따르는 새로운 컴포넌트 모델로 변환하기 위해 각 요소들의 관계 및 역할을 정의, 변환을 위해 사용하는 테이블이다.

가상 컴포넌트 모델과 구현 테이블을 정의함으로서 컴포넌트 개발시 플랫폼에 독립적인 가상 컴포넌트 모델을 구축하면 구현테이블을 이용 EJB나 COM+ 컴포넌트로 변환이 가능하다. 또한 역공학을 통하여 기존 레거시 컴포넌트의 상호 변환이 가능하다. 예로 가상 컴포넌트 모델을 구축한 후 선택적으로 EJB나 COM+ 컴포넌트를 구축하거나 기존 레거시 COM+ 컴포넌트 모델을 플랫폼에 독립적인 가상 컴포넌트 모델로 변환, 구현 테이블을 사용하여 EJB 컴포넌트 구조로 변환될 수 있다.

3.2 EJB와 COM+의 공통 모델 추출

EJB와 COM+의 스펙 구조를 바탕으로 실제 구현된 컴포넌트들의 클래스 다이어그램을 비교, 다음과 같은 방법에 따라 공통점과 차이점을 분석한다.

• Rule 1 : EJB와 COM+ 표준 스펙 구조를 비교, 기능

구조상의 공통부분 추출.

각 컴포넌트 스펙의 기능 구조상의 공통부분은 다음과 같다.

Rule 1.1: EJB의 HomeObejct와 COM+의 ClassFactory는 클라이언트가 컴포넌트의 서비스를 요청하는 객체를 생성하는 기능을 수행한다.

Rule 1.2: EJB의 EnterpriseBean과 COM+의 Object는 클라이언트가 원하는 서비스를 요청하고 서비스를 제공한다.

- Rule 2 : 기능 구조상 공통부분으로 추출된 EJB의 HomeObejct, COM+의 ClassFactory와 EJB의 EnterpriseBean, COM+의 Object를 실 구현된 구현모델의 클래스 다이어그램을 통해 비교, EJB와 COM+에서 관계 구조상의 공통부분 추출.

Rule 2.1: EJB의 HomeObejct와 COM+의 ClassFactory에 대한 관계를 비교한다.

그림 6의 a)에서 EJB의 HomeObject에서 Remote인터페이스는 EJBRemoteHome인터페이스를 Generalize하고, EJBRemoteHome은 EJBRemoteHomeClass로 Realize된다. b)의 COM+의 IUnknown 인터페이스는 IClassFactory 인터페이스를 Generalize하고 IClassFactory는 DClassFactory로 Realize된다.

Rule 2.2: EJB의 EnterpriseBean와 COM+의 Object에 대한 관계를 비교한다.

그림 7의 a)와 같이 EnterpriseBean2, 인터페이스는 EntityBean, SessionBean, MessageBean 인터페이스를 Generalize하고, EntityBean, SessionBean, MessageBean은 Client1Class, Client2Class, Client3Class로 각각 Realize 된다. b)는 IUnknown 인터페이스는 IMalloc 인터페이스를 Generalize하고 IMalloc는 DAlloc으로 Realize된다.

- Rule 3 : EJB구조와 COM+의 구조를 비교, 서로 다른 기능을 수행하는 기능상의 가변성 부분 추출.

Rule 3.1: 그림 3의 EJB 구조에서 <<JavaClassFile>> EJBObject는 중간 매개자로서 역할을 수행하는 부분이 독립적으로 존재하지만, 그림 4의 COM+ 구조에서 이에 대한 기능을 <<COMObject>> Object에서 수행한다.

Rule 3.2: 그림 3의 a)의 EJB 구조에서는 클라이언트가<<JavaClassFile>> EJBObject로 서비스 요청을 하기 위해 <<EJB-JAR>>중간 클래스가 없지만, b)의 COM+ 구조에서는 클라이언트가 서비스를 요청할 때, 호출된 API 내부 함수로부터 <<COM-DLL>>의 Utility 클래스가 호출되고 <<COM-DLL>>의 Utility로부터 <<COMObject>> ClassFactory는 호출되는 기능을 <<COM-DLL>>DLL에서 제공한다.

- Rule 4 : EJB구조와 COM+구조에서 관계상의 가변성을 찾기 위해 기능 구조상 공통 부분으로 추출된 그림 3 a)의 <<JavaClassFile>> HomeObejct, <<JavaClassFile>> EnterpriseBean와 그림 4 a)의 <<COMObject>> ClassFactory, <<COMObject>> Object를 실제 구현된 클래스다이어그램을 통해 비교하고 EJB와 COM+구조 관계상의 가변성 추출.

Rule 4.1: 기능 구조상 가변성 부분으로 추출된 <<JavaClassFile>> EJBObject, <<COMObject>> Object와 <<EJB-JAR>>, <<COM-DLL>>의 Utility는 이미 가변적인 영역이므로 이 단계에서 제외한다.

Rule 4.2: 그림 6 a)의 EJB에서 RemoteHome인터페

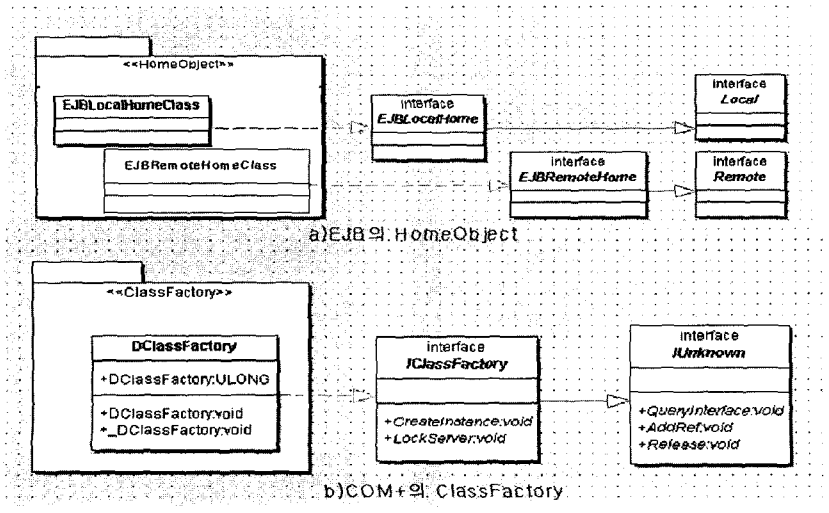


그림 6 a) EJB의 HomeObject b) COM+의 ClassFactory의 구조 비교

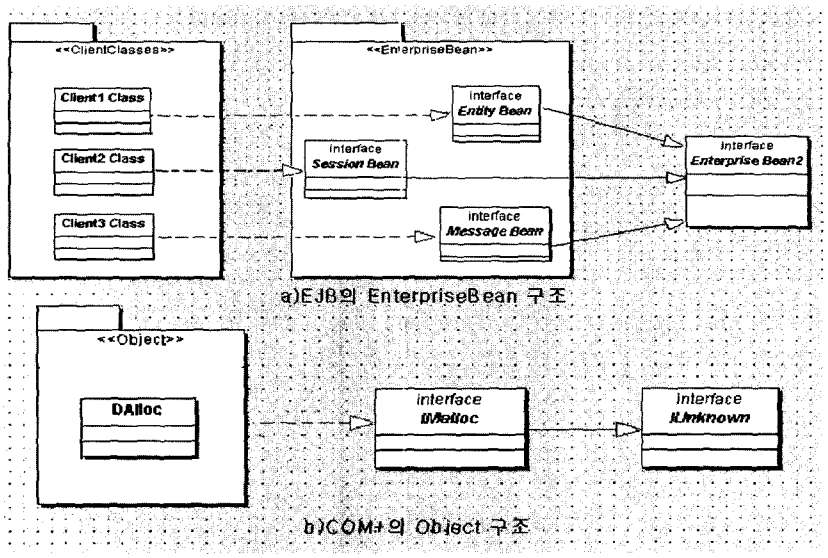


그림 7 a) EJB의 EnterpriseBean b) COM+의 Object의 구조 비교

이스는 실체화된 클래스가 직접적으로 표현되지 않지만, 그림 6 b)의 COM+에서 ClassFactory의 IClassFactory는 실체화된 DClassFactory 클래스를 가진다.

지금까지 EJB와 COM+구조의 기능상, 관계상의 공통점과 차이점을 정리하면 표 3과 같다. 표 3에서 EJB와 COM+의 공통부분은 (1)~(6)부분이다. (1)~(3)과 (4)~(6)로 구분하여 구조와 관계를 살펴보면, EJB의Remote, Local과 COM+의 IUnknow, IClassFactory은 구조적, 관계상 일치하여 (1)과 (2)가 각각 하나의 항목이 되고, (3)은 COM+에서 ClassFactory로 실체화하기 위한 항목으로 나누어진다. (4)~(6)은 EJB와 COM+의 구조, 관계가 모두 일치하여 공통의 3개의 항목을 이룬다. 한편 (7), (8)은 EJB의 EJBObject에 대한 항목이고, (9)는 COM+에서 DLL에 대한 항목이다.

3.3 가상 컴포넌트 모델의 구조

가상 컴포넌트 모델은 표 3을 바탕으로 그림8과 같이 정의한다.

그림 8의 가상 컴포넌트 모델은 11개의 인터페이스 클래스, 3개의 클래스 그리고 이들의 관계를 정의한 14개의 연관 관계로 구성된다. 가상 컴포넌트 모델을 구성하고 있는 항목의 이름은 인터페이스는 I로 클래스는 C로 시작 된다. 가상 컴포넌트 모델은 EJB와 COM+의 공통적인 특성을 모두 포함하고 있는 플랫폼에 독립적인 모델이므로 EJB와 COM+의 구조가 모두 가상 컴포넌트 모델에 적용될 수 있다. 가상 컴포넌트 모델의 각 클래스의 역할은 표 4의 구현테이블에 설명되어 있다.

3.4 구현 테이블의 구조

구현 테이블은 가상 컴포넌트 모델 구현을 위해 특정 플랫폼에 따른 컴포넌트 모델을 만들 때 생성되는 컴포넌트의 인터페이스, 클래스, 오퍼레이션과 인터페이스와

표 3 EJB와 COM+간의 비교

분류	EJB				COM+				비교
	기능상 구조	인터페이스	관계		기능상 구조	인터페이스	관계		
(1)	EJBHome	Remote,Local	EJBHome(1)은 EJBRemoteHome(2)를 Generalization		ClassFactory	IUnknown	IUnknown(1)은 IClassFacory(2)를 Generalization		공통점
(2)		EJBRemoteHome, EJBLocalHome				IClassFacory		ClassFactory(3)은 IClassFacory(2)를 Realization	
(3)						ClassFactory			
(4)	EJBBean	Enterprise Bean	Enterprise Bean (4)는 (5)를 Generalization		Object	IUnknown	IUnknown(4)은 Interface(5)를 Generalization		
(5)		Entity Bean, Session Bean, Message Bean		Class(6)은 (5)를 Realization		Interface		Class(6)은Interface (5)를 Realization	
(6)		Class				Class			
(7)	EJBObject	Remote,Local	EJBObject(7)은 EJBRemoteObject(8)를 Generalization						차이점
(8)		EJBObject, EJBLocalObject							
(9)					DLL	DLL(Utility)			

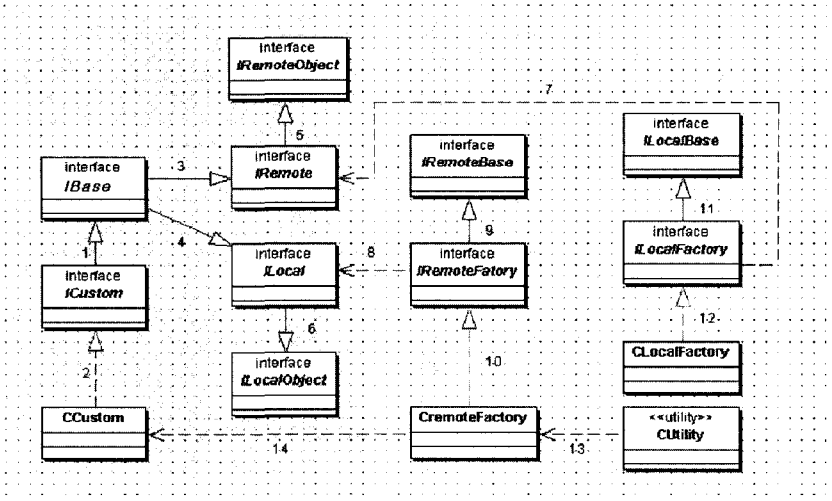


그림 8 가상 컴포넌트 모델의 구조(번호는 클래스간의 관계를 나타냄)

표 4 EJB와 COM+간 가상 컴포넌트 모델의 관계를 설명한 구현테이블

	가상모델명	EJB 구현 매핑		COM 구현 매핑	
		인터페이스	오퍼레이션	인터페이스	오퍼레이션
인터페이스 및 클래스	IBase	EnterpriseBean		IUnknown	QueryInterface AddRef Release
	ICustom	EntityBean	ejbCreate ejbPostCreate ejbHome getEntityContext ejbStore ejbRemove ejbSelect	사용자 정의 인터페이스	
		SessionBean	ejbActivate ejbPassivate ejbRemove setSessionContext		
		MessageBean	ejbRemove setMessageDrivenContext		
	CCustom	ICustom에 대한 실체화 클래스		사용자 정의 클래스	
	IRemoteObject	EJB Local Object		해당사항 없음	
	IRemote	Local		해당사항 없음	
	ILocal	Remote		해당사항 없음	
	ILocalObject	EJBObject		해당사항 없음	
	IRemoteBase	EJBHome		IUnknown(IBase와 동일)	
	IRemoteFactory	RemoteHome	create finder home findByPrimeryKey	IClassFactory	CreateInstance LockServer
	CRemoteFactory	해당사항 없음		IClassFactory 실체화클래스	DClassFactory ~DClassFactory
	ILocalBase	EJBLocalHome		해당사항 없음	
	ILocalFactory	LocalHome	create finder home findByPrimeryKey	해당사항 없음	
	CLocalFacory	해당사항 없음		해당사항 없음	
	CUtility	해당사항 없음		Dll에서 Export된 함수	DllGetClassObject (REFCLSID clsid, REFIID iid,void **ppv) DllCanUnloadNow(void)
관계	1	유효		유효	
	2	유효		유효	
	3	유효		삭제	
	4	유효		삭제	
	5	유효		삭제	
	6	유효		삭제	
	7	유효		삭제	
	8	유효		삭제	
	9	유효		유효	
	10	삭제		유효	
	11	유효		삭제	
	12	삭제		삭제	
	13	삭제		유효	
	14	삭제		유효	

클래스의 관계를 정의한 것이다. 표 4는 그림 8에서 정의한 모델의 각 항목이 EJB나 COM+ 컴포넌트로 구현할 때 어떻게 실체화되는지를 나타낸 것이다.

표 4의 구현테이블의 EJB, COM+ 구현 매핑 항목은 각 스펙의 구성요소가 가상 컴포넌트 모델의 어떤 구성요소와 매핑 되는가를 나타내고 매핑 되지 않은 클래스는 일반적인 사용자 정의 클래스로 존재하게 된다. 관계 부분 또한 가상 컴포넌트 모델이 EJB, COM+으로 매핑될 때 유지되거나 삭제되어지는 클래스간의 관계를 나타낸다.

예로 가상컴포넌트 모델링을 통한 가상 컴포넌트 모델이 EJB로 매핑 될 때는 IBase, ICustom, CCustom, IRemoteObject, IRemote, ILocal, ILocalObject, IRemoteBase, IRemoteFactory, ILocalBase, ILocalFactory클래스를 EJB의 각 클래스들로 매핑하게 된다. 또한 관계 중 1, 2, 3, 4, 5, 6, 7, 8, 9, 11 번의 관계는 유지가 되고 10, 12, 13, 14번의 관계는 없어지게 된다. 이 구현테이블을 템플릿으로 하여 가상 컴포넌트 모델의 구조를 EJB, COM+로, 혹은 EJB, COM+를 가상 컴포넌트 모델로 선택적으로 변환 가능하다.

4. 사례 연구 및 평가

상이한 플랫폼간의 컴포넌트 결합을 위해 제시한 모델기반의 컴포넌트 결합 기법에 대한 검증을 위해 사례연구를 수행한다. 사례연구를 통하여 수행된 개발 환경은 다음 표 5와 같다.

우선 각 시스템의 도메인 분석을 수행한다. 도메인 분석은 유즈케이스 작성과 비즈니스 타입 모델을 바탕으로 수행 한다. 그림 9는 개발할 시스템B의 유즈케이스 다이어그램과 비즈니스 타입 모델의 한 부분이다.

다음으로 유스케이스와 비즈니스 타입 모델을 바탕으로 컴포넌트를 추출, 컴포넌트 인터페이스와 명세를 수행 한다. 다음 그림 10은 시스템B의 컴포넌트 중 CompanySavings 컴포넌트의 명세이고, 그림 11은 CompanySavings에서 제공하는 ICompanySavings 인터페이스 명세이다.

새로 개발되어질 시스템 B의 유즈케이스 모델과 비지니스 타입모델이 설계되어지고 컴포넌트의 인터페이스가 선별되어 구현할 컴포넌트가 결정되어지면 기존의 레거시 시스템 A와 공통기능을 수행하는 컴포넌트 선정 작업을 수행한다.

레거시 시스템A의 EJB 컴포넌트와 시스템 B의 COM+ 컴포넌트 중 공통 기능 부분을 추출하기 위해

표 5 사례연구 개발환경

	레거시 시스템A	시스템B
개발언어 / 컴포넌트 플랫폼	JAVA / EJB	VisualBasic / COM+
개발 도메인	은행 업무 / 개인 거래	은행 업무 / 개인, 기업 거래
사례연구의 목적	기 개발된 시스템 A는 EJB 기반의 은행 업무시스템으로 개인 고객을 대상으로 하며, 개발되어질 시스템 B는 COM+기반의 은행 업무 시스템으로 기업 고객을 대상으로 한다. 시스템B의 구현을 위해 시스템A의 컴포넌트와 가상 컴포넌트 모델 기반의 컴포넌트 결합을 통하여 상이한 컴포넌트 플랫폼의 컴포넌트 재사용을 통한 시스템 구현.	

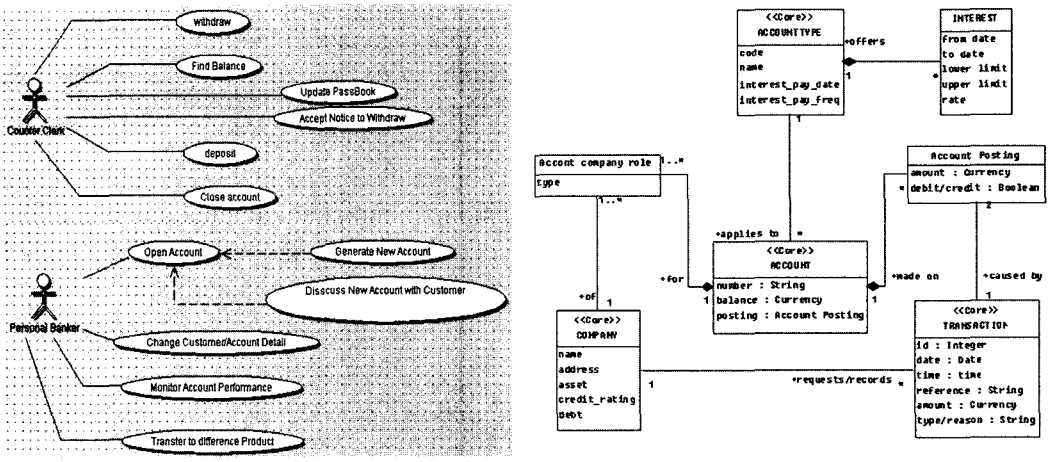


그림 9 시스템B의 유스케이스 모델과 비즈니스 타입 모델의 예

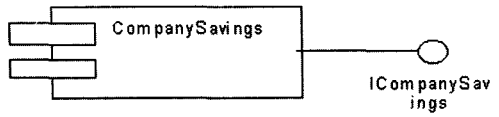


그림 10 CompanySavings 컴포넌트의 명세

레거시 시스템의 개발 산출물인 비즈니스 타입 다이어그램, 컴포넌트 명세, 인터페이스 명세들을 새로 작성된 시스템B의 다이어그램들과 비교하여 공통의 비지니스 컴포넌트 및 인프라스트럭처 비즈니스 컴포넌트를 추출한다.

작성된 컴포넌트 명세, 인터페이스 명세, 비즈니스타입 명세의 비교 사항은 표 6과 같다.

본 사례연구에서는 EJB기반 레거시 시스템A의 ISavings 비즈니스 컴포넌트를 재사용 컴포넌트로 추출, 새로 개발할 COM+ 기반 시스템B의 ICompanySavings

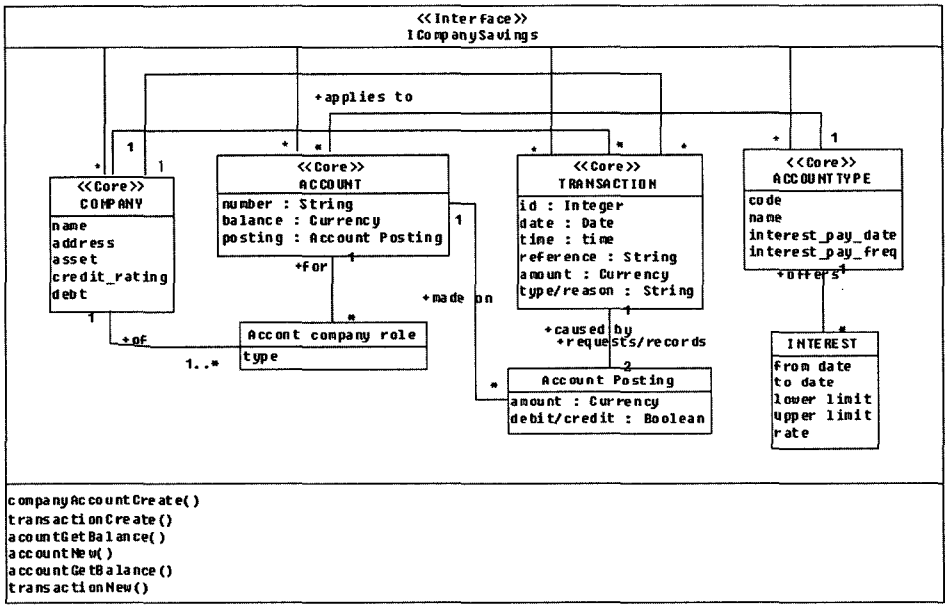


그림 11 CA사의 Cool:Joe를 이용한 ICompanySavings 인터페이스 명세

표 6 시스템A와 시스템B 공통부분 추출을 위한 명세의 비교

	CORE	DETAIL	CORE	DETAIL	공통성 여부
비즈니스 타입	ACCOUNTTYPE	INTEREST	ACCOUNTTYPE	INTEREST	공통
	ACCLUNT	ACCOUNT ROLE	ACCOUNT	ACCOUNT ROLE	공통
		ACCOUNT POSTING		ACCOUNT POSTING	공통
	TRANSACTION	None	TRANSACTION	None	공통
	PARTY	None	COMPANY	None	가변
컴포넌트	NAME	INTERFACE	NAME	INTERFACE	
	Savings	ISavings	ComSavings	IComSavings	
인터페이스	TYPE	ACTION Reference	TYPE	ACTION Reference	컴포넌트 분류
	ACCOUNTTYPE	accountCreate() transactionCreate() accountGetBalance() accountNew() accountGetBalance() transactionNew()	ACCOUNTTYPE	companyAccountCreate() transactionCreate() accountGetBalance() accountNew() accountGetBalance() transactionNew	비즈니스 컴포넌트
	ACCOUNT	accountNew() accountGetBalance() transactionNew()	ACCOUNT	accountNew() accountGetBalance() transactionNew()	비즈니스 인프라스트럭쳐 컴포넌트
	TRANSACTION		TRANSACTION		
	PARTY	accountCreate() transactionCreate() accountGetBalance() accountNew() accountGetBalance() transactionNew()	COMPANY	companyAccountCreate() transactionCreate() accountGetBalance() accountNew() accountGetBalance() transactionNew	비즈니스 컴포넌트

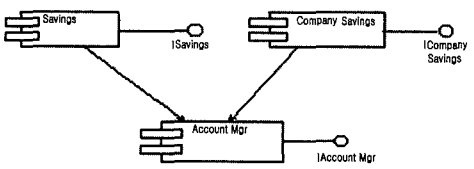


그림 12 컴포넌트 통합후의 컴포넌트 명세

비즈니스 컴포넌트와 결합하여 컴포넌트를 구축한다. 표 6에서는 기 개발된 EJB 컴포넌트 ISavings을 새로 개발할 COM+의 ICompanySavings컴포넌트의 일부분으로 결합하기 위하여 비교를 수행하였다. 이러한 모델기반의 결합을 위하여 시스템A의 컴포넌트 명세와 설계모델을 구현테이블을 통하여 가상 컴포넌트 모델로 재구성한다. 새로운 시스템B는 그림 12와 같이 시스템A로부터 재사용되는 비즈니스 컴포넌트 Savings과 새로 구성되는 비즈니스 컴포넌트 CompanySavings, 그리고 비즈니스 인프라스트럭처 컴포넌트 AccountMgr로 총 3개의 컴포넌트가 생성이 된다.

새로운 Savings, CompanySavings, AccountMgr 컴포넌트 명세와 컴포넌트에서 제공하는 각각의 인터페이스 명세를 가상 컴포넌트 모델을 기반으로 모델링을 수행한다.

그림 13에서는 비즈니스 인프라스트럭처 컴포넌트 AccountMgr 컴포넌트 모델을 보여주고 있다. 그림 14은 비즈니스 컴포넌트 Savings의 컴포넌트 모델이다. EJB기반의 레거시 시스템 설계 모델을 가상 컴포넌트 모델로 정제 후 다시 구현테이블을 이용하여 COM+기반의 컴포넌트로 구현하였다.

그림 13은 설계시 가상 컴포넌트 모델을 기반으로 모델링이 수행되어 COM+ 컴포넌트 형태로 구현 모델이 작성되었으며 그림 14의 경우 레거시 EJB 모델을 구현테이블을 통하여 IBase, ICustom, CCustom, IRemoteObject, IRemote, ILocal, ILocalObject, IRemoteBase, IRemoteFactory, ILocalBase, ILocalFactory의 가상컴포넌트 모델 요소로 매핑, 가상 컴포넌트 모델로 정제 후 다시 IUnknown, IClassFactory, 사용자 정의 인터페이스와 클래스로 변환, COM+ 모델로 변환하였다.

4.2 평 가

본 논문에서는 가상 컴포넌트 모델과 구현테이블을 적용하여 상이한 플랫폼 컴포넌트간의 모델기반 결합 가능성을 제시하였다. 또한 사례연구를 통하여 EJB 컴포넌트와 COM+컴포넌트간의 모델기반 결합 사례를 보여 새로운 컴포넌트 시스템을 구축하는데 상이한 플랫폼의 레거시 컴포넌트가 모델기반으로 변환, 재사용 가능함을 검증하였다.

표 7 관련 기술 비교

	MDA (Model Driven Architecture)	모델기반 컴포넌트 변환기법을 이용한 결합
상이한 플랫폼 컴포넌트간 연동	O	X
		모델의 변환이 필요
미들웨어	필요	필요 없음
모델링 방법	제약 없음	가상 컴포넌트 모델의 제약을 따름
변환기법	제시되어있지 않음	절차 제시
재사용 지원	PIM	단위 컴포넌트
지원 컴포넌트	모든 컴포넌트	EJB, COM+

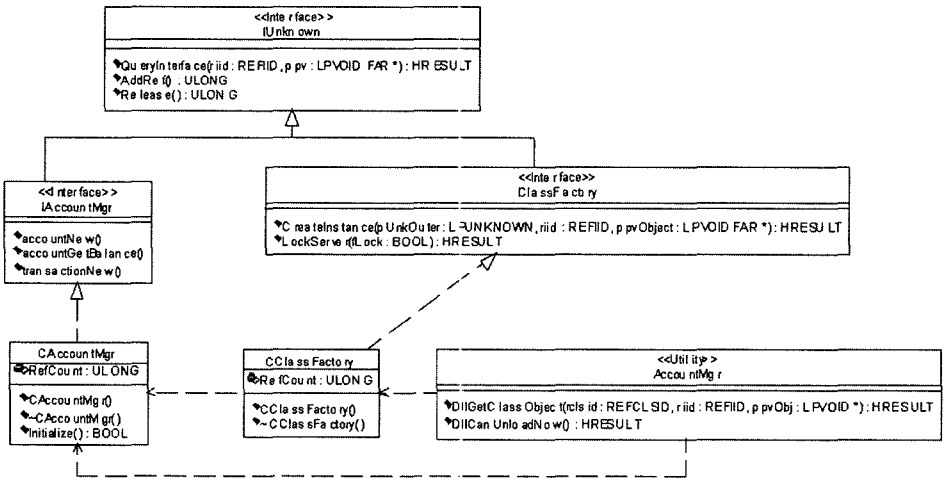


그림 13 가상 컴포넌트 모델링을 통한 AccountMgr 컴포넌트의 COM+ 모델

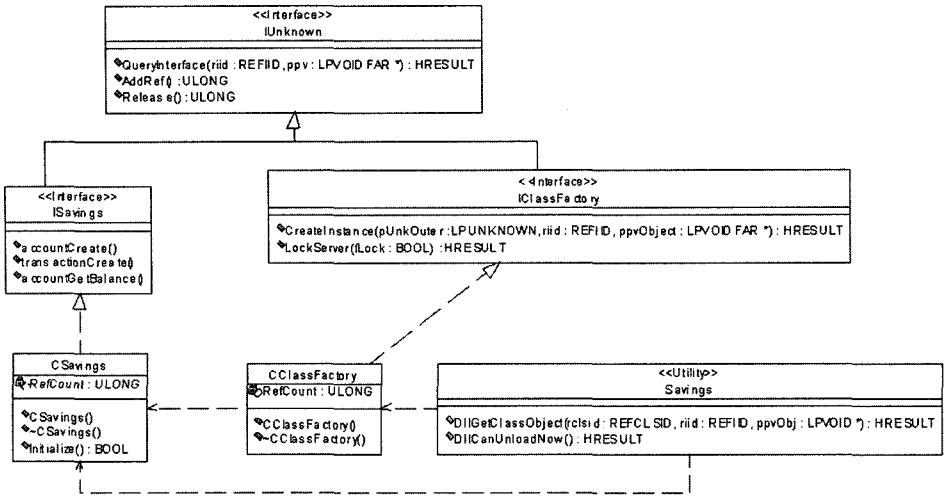


그림 14 EJB 레거시 컴포넌트에서 변환된 Saving 컴포넌트의 COM+ 모델

이러한 가상 컴포넌트 모델을 이용한 컴포넌트 변환 기법은 컴포넌트 모델의 단순한 변환을 통한 손쉬운 컴포넌트의 변환이 가능한 반면 현 MDA 지원 도구들이 제공하는 다양한 확장 기법을 수용하지 못한다. 또한 스펙간의 매핑이 어려운 사용자 정의 클래스 부분과 가상 컴포넌트 모델의 제약사항 때문에 그림 13, 14에서 보는 것과 같이 가상 컴포넌트 모델의 특정 형태를 따르게 되는 단점이 있고 클래스 다이어그램을 위주로 시스템의 정적인 부분만을 다루고 있으므로 다양한 관점의 다이어그램들과 연계되어 정제가 수행 되어야만 한다.

5. 결 론

일반적으로 기존의 컴포넌트 결합은 동일한 플랫폼 기반의 컴포넌트 결합[3,4]으로 소스 레벨에서 컴포넌트의 기능적 결합을 바탕으로 수행 해왔다. 반면 OMG에서 제시한 MDA는 플랫폼이 다른 시스템 개발 문제를 기저로 제기, 소스 기반의 인프라를 모델기반의 인프라로의 변화를 목적으로 한다. 그러나 MDA를 지원하는 개발툴이 미약한 상태이고 PIM과 PSM간, PSM과 PIM간의 상호 변환 전략도 아직 문제로 남아 있다.

본 논문에서 제시한 가상 컴포넌트 모델을 적용한 상이한 플랫폼간의 컴포넌트 변환 기법은 모델기반의 구조적 표준 결합을 기반으로 한다.

EJB와 COM+의 참조 모델을 기반으로 컴포넌트의 구조를 비교 분석, 각각의 일반화된 모델을 통해 플랫폼에 독립적인 가상 컴포넌트 모델을 정의하였다. 가상 컴포넌트 모델은 EJB와 COM+이 제공하는 인터페이스와 클래스를 모두 포함하며, EJB와 COM+의 인터페이스, 클래스와 관계들을 일반화 한 모델로 플랫폼에 독립적이다. 또한 가상 컴포넌트 모델을 EJB나 COM+로 구현 시 필요로 하는 오퍼레이션과 같은 플랫폼에 특성화된 요소는 구현 테이블을 이용하여 정의하여 각 컴포넌트간의 모델기반 변환 기법을 제시하였다.

가상 컴포넌트 모델은 EJB와 COM+를 결합한 컴포넌트로 개발할 때 적용할 수 있을 뿐만 아니라, 가상 컴포넌트 모델과 구현 테이블을 이용하여 개발자가 선택적으로 기존 EJB 컴포넌트를 COM+컴포넌트로 손쉽게 변환, 재개발 할 수 있으며, 같은 방법으로 COM+ 컴포넌트를 가상 컴포넌트 모델을 적용하여 EJB 컴포넌트로 변환을 수행할 수 있다.

본 논문에서 제시하는 컴포넌트 변환기법을 통하여 상이한 플랫폼 기반의 컴포넌트 결합 문제를 해결하며, 가상 컴포넌트 모델을 통해 차후 개발자가 선택적으로 다른 플랫폼으로 시스템을 개발 할 수 있을 뿐만 아니라, EJB와 COM+컴포넌트 기반의 레거시 시스템 결합을 수행 하도록 재공학을 지원하므로 컴포넌트의 재사용성을 더욱 높일 수 있다.

참 고 문 헌

[1] Santiago Comella-Donda, Kurt Wallnau, Robert C. Seacord, Jhon Robert, *"Survey of Legacy System Modernization Approaches,"* Technical Node, CMU/SEI-2000-TN-003, April. 2000.

[2] Frederic Doucet, Sandeep Shukla, Rajesh Gupta *"An Environment for Dynamic Component Composition for Efficient Co-Design,"* DATE, pp736-743, 2002.

[3] YanXia, Anthony Tung Shuen Ho, *"CIMO Component Integration Model,"* 1530-1362/00, APSEC, pp344-348, 2000.
[4] Rahim Adatia, Faiz Arni, *"Professional EJB,"* Wrox Press Ltd, 2001.
[5] Architecture Board ORMSC, *"Model Driven Architecture(MDA),"* Document number ormsc/ 2001-07-01, pp 17-23, July 9, 2001.
[6] Tolbert,D., *"CWM:A Model Based Architecture for Data Wherehouse Interchange,"* Workshop on Evaluating Software Architecture Solution 2000, University of California at Irvine, May, 2000. http://www.cwmforum.org/uciwesas2000.htm
[7] OMG Model-Driven Architecture Home Page : http:://www.omg.org /mda /index.html
[8] OMG Architecture Board MDA Drafting Team, "Model-Driven Architecture :A Technical Perspective," ftp://ftp.omg.org/pub/docs/ab/01-02-01.pdf
[9] Linda G. DeMichiel, L.Umit Yalcinalp, Sanjeev Krishnan, *"Enterprise Java Beans Specification, Version 2.0,"* Sun Microsystems, August 14, pp 17-23, 2001.
[10] Ed Roman, *"Mastering EJB and the Java 2 Platform, Enterprise Edition,"* WILEY, pp 140-146, 1999.
[11] Microsoft Corporation, *"The COM Specification,"* 1995.
[12] *"Common Warehouse Metamodel Specification version 1.0,"* OMG, 2001.
[13] Markus Volter, Alexander Schmid, Eberhard Wolff, *"Server Component Patterns,"* WILEY, 2002.
[14] Gopalan Suresh Raj, http://my.execpc.com/~gopalan/, "A Detailed comparison of the EJB and MTS models".

최 일 우
1995년 숭실대학교 전자계산학과 학사(공학사). 1997년 숭실대학교 컴퓨터학과 석사(공학석사). 2002년 숭실대학교 컴퓨터학과 박사 수료. 관심분야는 소프트웨어 개발 프로세스, 소프트웨어 재공학/역공학, 재사용, CBD, CBSE

신 정 은
2001년 숭실대학교 컴퓨터학과 학사(공학사). 2003년 숭실대학교 컴퓨터학과 석사(공학석사). 2003년 현재 삼성 전자 근무중. 관심분야는 소프트웨어 개발 프로세스, EJB

류 성 열
1997년 아주대학교 컴퓨터학부(공학박사) 1997년~1998년 George Mason University 교환교수. 1998년~2001년 숭실대학교 정보과학대학원 원장. 1981년~현재 숭실대학교 정보과학대학 컴퓨터학부 교수. 1998년~현재 숭실대학교 전자계산원 원장. 관심분야는 소프트웨어 유지보수, 소프트웨어 재사용, 소프트웨어 재공학/역공학