컴포넌트의 성능향상과 재사용을 위한 EJB 2.1 컴포넌트에서 EJB 3.0로의 변환기법

(Techniques to Transform EJB 2.1 Components to EJB 3.0 for Performance Improvement and Component Reusability)

이 후 재 † 김 지 혁 †† 류 성 열 †††
(Hoo Jae Lee) (Ji Hyeok Kim) (Sung Yul Rhew)

요 약 최근 성능향상과 개발의 편리성을 주요 특징으로 보완한 EJB 3.0 명세가 발표되었다. 이에 따라 개발자들은 EJB 3.0 기반의 애플리케이션 환경에서 EJB 2.1 컴포넌트 전체를 EJB 3.0 컴포넌트로 완전 대체하기보다는 성능향상이 요구되는 컴포넌트부터 점진적 변환하는 방법을 주로 고려하게 되었다. 그러나 기존 연구는 애플리케이션의 서비스를 고려하지 않기 때문에 점진적 변환을 하는데 문제가 있고, 상이한 명세의 변환 방법으로 EJB 3.0 완전 대체 시 컴포넌트의 호환의 문제와 재사용을 하는데 어려움이 있다.

본 연구에서는 기존 애플리케이션에서 제공하였던 서비스를 고려하며, EJB 3.0 완전 대체 시에서도 컴포넌트의 호환과 재사용이 가능한 세가지 변환 기법을 제안한다. 제안하는 변환 기법은 직접 컴포넌트 연결을 하는 직접 변환 기법, EJB 커넥터를 사용하는 간접 변환 기법 그리고 간접 변환 기법에 템플릿 패턴을 적용하는 간접 템플릿 변환 기법으로 점진적 변환을 위한 기법들이다. 이에 제안하는 변환 기법을 재사용과 초당 처리량이라는 평가 기준으로 비교 검증하며, 본 연구를 통해 도출된 EJB 3.0 변환시의 특성들을 기반으로 기법 선택의 기준을 제공한다.

키워드 : Enterprise Java Bean(EJB), 변환 기법, 컴포넌트 성능, 컴포넌트 재사용, EJB 커넥터

Abstract The EJB 3.0 specifications, which were improved in terms of performance and ease of development, were recently announced. Accordingly, for the EJB 3.0 application environment, developers generally prefer the gradual transformation of components whose performance must be improved to the complete transformation of all the EJB 2.1 components into EJB 3.0 components. Previous studies, however, did not consider the service of the application and did not ensure the compatibility and reusability of the components in the full replacement of EJB 3.0 due to the transformation using different specifications.

This study proposed three transformation techniques that consider the service supported in the existing application, wherein the compatibility and reusability of the components are ensured in the case of the full replacement of EJB 3.0. The proposed transformation techniques are techniques for gradual transformation, such as direct transformation that directly connects components, indirect transformation that uses the EJB connector, and indirect template transformation wherein the template

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음

† 정 회 원 : 숭실대학교 컴퓨터학과
wjlee@daelim.ac.kr
†† 학생회원 : 숭실대학교 컴퓨터학과
jhkim78@ssu.ac.kr
††† 종신회원 : 숭실대학교 컴퓨터학과 교수
syrhew@ssu.ac.kr

논문접수 : 2008년 10월 21일
심사완료 : 2009년 2월 10일

정보과학회논문지 : 소프트웨어 및 응용 제36권 제4호(2009.4)

pattern is applied to the indirect transformation. The proposed transformation techniques were verified by comparing the reusability and processing capability of the components per second, and the standards for selecting a technique were provided based on the characteristics of the transformation into EJB 3.0 that were found in this study.

Key words : Enterprise Java Bean(EJB), Transformation Technique, Component Performance, Component Reuse, EJB Connector

1. 서 론

EJB(Enterprise Java Bean)는 Java 기반의 분산객체 애플리케이션을 쉽게 만들 수 있도록 제공하는 서버 측 컴포넌트 프레임워크다. EJB는 산업에서 이미 개발된 분산 Infrastructure를 가져와 서버 측 컴포넌트를 쉽고 빠르게 조립할 수 있다. EJB를 사용함으로써 확장성, 신뢰성을 얻는 애플리케이션을 개발할 수 있고 복잡한 분산 컴포넌트 프레임워크를 개발할 필요 없이 보안 애플리케이션을 개발 할 수 있다. 또한 EJB는 엔터프라이즈 미들웨어 서비스를 하는 기업을 위하여 이식이 가능하고 재사용이 가능한 애플리케이션을 지원하도록 설계되어 있다[1]. 위와 같은 특징으로 EJB는 많은 기업에서 개발하여 사용 중에 있지만, 성능 저하와 개발의 불편함, 생산성 저하 등의 문제로 EJB 사용을 망설이는 기업도 있다[2]. 이에 이런 문제점들은 EJB 3.0 명세서에 반영하여 개선이 되었다[3-5].

이에 기존의 EJB 2.1로 개발된 애플리케이션 시스템은 EJB 2.1이 가지고 있는 문제 해소를 위해 개선된 EJB 3.0으로의 변환이 필요하다. 그러나, EJB 2.1로부터 EJB 3.0으로의 변환은 명세의 차이로 인하여 모든 EJB 2.1 컴포넌트들을 EJB 3.0 명세서에 맞게 변경해야 하기 때문에 많은 비용과 시간이 필요하다. 그래서 EJB 3.0의 변환을 위해서는 애플리케이션 시스템내의 전체 컴포넌트를 한번에 대체하기보다는 점진적인 변경을 고려해야 하며, 기존 EJB 2.1 컴포넌트 중 성능 향상이 필요한 컴포넌트부터 부분적으로 변환해야 한다. 현재 EJB 3.0으로의 전환을 위한 주요한 방법은 EJB 3.0 기반의 애플리케이션 환경에서 애플리케이션 시스템을 EJB 2.1과 EJB 3.0이 공존하는 형태로 지속적으로 운영하면서 EJB 3.0으로의 완전 대체를 계획하는 것이다[6].

그러나 기존 연구들은 EJB 3.0 완전 대체 시 컴포넌트 호환 문제와 재사용을 하는데 어려움이 있고, 기존 애플리케이션의 서비스를 고려하지 않아 점진적 변환을 하는데 문제가 있다.

이에 본 연구에서는 EJB 3.0으로 완전 대체를 고려한 점진적 변환을 할 때, 기존 애플리케이션 서비스를 고려하고, 컴포넌트의 유연한 변경 및 관리가 용이하면서 성능이 향상된 컴포넌트의 재사용과 재사용을 통해 개발 노력을 절감할 수 있는 세가지 변환 기법을 기존 연구들과 비교하여 제안한다. 제안하는 변환 기법은 컴포넌트의 성능 향상이 요구되는 컴포넌트부터 EJB 3.0 컴포넌트로 점진적인 변환이 가능한 직접 변환(Direct Transformation) 기법, 간접 변환(Indirect Transformation) 기법, 간접 템플릿 변환(Indirect Template Transformation) 기법이다.

2. 관련연구

본 장에서는 점진적인 EJB 3.0 전환 연구 과정에서 발생될 수 있는 인터페이스와 요구사항의 불일치를 해결 할 수 있는 커넥터에 대해 살펴보고, EJB 2.1에서 EJB 3.0으로의 변환기법에 대한 기존 연구의 문제점을 도출한다, 또한, 제안 기법의 비교 평가를 위한 측정 지표 및 기준을 제시한다.

2.1 커넥터(Connector)

상호작용을 설계하기 위해 커넥터를 사용하는 것은 교체단위가 있는 컴포넌트 프레임워크 설계를 위해 유용한 기술이다. 커넥터는 인터페이스 및 요구사항을 불완전하게 만족하는 컴포넌트가 애플리케이션을 지원할 수 있도록 도와준다. 커넥터는 입력과 출력 포트를 가지고 있는 작은 소프트웨어 부품이다. 이러한 포트는 컴포넌트를 연결하기 위해 사용된다. 소스 컴포넌트(Source Component)는 커넥터를 통해 목표 컴포넌트(Target Component)에 메시지를 전달한다. 이러한 메시지는 소스 컴포넌트의 사용 인터페이스에서 커넥터의 입력 포트(Input Port)를 통해 전달된다. 그리고 커넥터는 컴포넌트 연결 및 컴포넌트간의 불일치를 해결하고, 출력 포트(Output Port)를 통해 목표 컴포넌트로 메시지를 전달한다. 이때 목표 컴포넌트의 Provide Interface가 호출된다[7].

이런 커넥터의 기능은 EJB 2.1 컴포넌트를 EJB 3.0 컴포넌트로의 점진적인 변환 과정에서 컴포넌트 연결 및 컴포넌트간 불일치의 문제를 해결할 수 있다.

2.2 자바 썬의 "EJB 2.x-EJB 3.0 호환성"

자바 썬의 "EJB 2.x-EJB 3.0 호환성" 기법[6]은 연관 EJB 2.x 컴포넌트와 호환되고 EJB 3.0의 기능을 사용 할 수 있게 컴포넌트를 만드는 기법이다. 자바 썬의 "EJB 2.x-EJB 3.0 호환성" 기법에 의한 컴포넌트는 EJB 2.x 명세를 고려한 인터페이스와 EJB 3.0 Session

Bean을 구현한다. "EJB 2.x-EJB 3.0 호환성"컴포넌트의 인터페이스는 EJB 2.x 명세를 고려하여 홈 인터페이스(Home Interface)와 리모트 인터페이스(Remote Interface) 클래스를 구현한다. 이는 EJB 3.0 컴포넌트가 비즈니스 인터페이스(Business Interface)만을 구현하는 것에 비교하면 상이한 구성이다. 이러한 "EJB 2.x-EJB 3.0 호환성" 컴포넌트의 구성은 EJB 2.x 컴포넌트와 유사한 인터페이스 구성으로 기존 EJB 2.x 클라이언트의 변경이 요구되지 않으며, 연관 EJB 2.x 컴포넌트와 호환이 된다.

하지만, 자바 썬의 "EJB 2.x-EJB 3.0 호환성" 기법은 변환 단계에서 기존 EJB 2.x 클라이언트 컴포넌트와 연관 EJB 2.x 컴포넌트의 호환만을 고려하고 완전 대체 단계를 고려하지 않았다. 그래서, 완전 대체 단계에서 자바 썬의 "EJB 2.x-EJB 3.0 호환성"컴포넌트는 다음과 같은 문제가 있다.

첫째, 다른 EJB 3.0 컴포넌트와 상이한 인터페이스 구성은 유지보수 시 혼란과 호환의 문제가 있다.

둘째, 이러한 컴포넌트간 인터페이스 불일치를 해결하기 위해서는 별도의 커넥터를 만들거나 기존 인터페이스를 비즈니스 인터페이스로 통합해야 하는 노력이 필요하다.

이와 같이 자바 썬의 "EJB 2.x-EJB 3.0 호환성"기법은 EJB 3.0으로 완전 대체 시 재사용을 위한 추가적인 노력이 필요하다.

2.3 Deepak의 마이그레이션

Deepark의 마이그레이션 기법[8]은 EJB 2.1 컴포넌트의 소스를 활용하여 EJB 3.0 컴포넌트로 마이그레이션하는 기법이다. Deepark의 마이그레이션 기법에 의한 컴포넌트 구현은 다음과 같다.

EJB 3.0 Session Bean의 비즈니스 인터페이스는 기존 EJB 2.1 Session Bean의 리모트 또는 로컬 인터페이스에 정의된 비즈니스 인터페이스를 복사한다. EJB 3.0 Session Bean의 bean 클래스에는 EJB 2.1 Session Bean의 bean에 구현된 비즈니스 로직을 복사한다. EJB 2.1 컴포넌트의 배포 디스크립터(Deployment Descriptor)내 설정사항은 EJB 3.0 컴포넌트의 소스에 Annotation을 사용하여 정의한다. 기존 EJB 2.1 클라이언트는 Session Bean의 홈 인터페이스 객체를 생성하여 리모트 또는 로컬 인터페이스에 정의된 비즈니스 메소드를 호출한다. EJB 2.1 클라이언트는 @Inject, @EJB, @Resource 등의 Annotation으로 EJB 3.0 컨테이너가 참조하고 있는 Session Bean의 비즈니스 로직을 호출하도록 수정한다.

이처럼, Deepark의 마이그레이션 기법은 기존 EJB 2.1 클라이언트의 수정을 요구하기 때문에 연관 애플리케이션의 수정도 필요하다. Deepark의 마이그레이션 기법은 기존 애플리케이션의 서비스를 고려하면서 점진적 변환을 위한 방법은 제시하지 않고, EJB 2.1 컴포넌트 소스를 활용하여 EJB 3.0 컴포넌트를 개발하는 방법만을 설명하고 있다.

2.4 컴포넌트의 성능 측정지표 / 재사용 측정 기준

본 절에서는 기존 연구와의 비교를 통해 제안 기법을 평가할 수 있는 측정 지표 및 기준을 제시한다.

컴포넌트 성능 측정 지표: EJB의 성능 측정 연구는 EJB 애플리케이션의 병목 위치를 찾는데 도움이 된다. 또한, 분석 정보는 EJB 애플리케이션의 성능 향상을 위한 유용한 자료가 된다[9]. EJB의 성능 측정을 위한 Lladó의 연구[10]에서는 EJB 1.1 명세로 만들어진 EJB 애플리케이션을 대상으로 단위 시간당 메소드 실행 수로 정의된 처리량과 응답시간으로 성능을 측정 하였고, Liu[11]는 Layered Queueing Model(LQM) 방법으로 EJB 기반의 분산 엔터프라이즈 애플리케이션의 성능을 서비스 요청 대비 부하량에 대한 응답시간으로 측정하였다. 나학청의 연구[9]에서는 애플리케이션의 서비스를 위한 워크플로우 동안 발생하는 생명주기에 관련된 빈의 상태 변화와 빈에서의 처리시간, 자원 사용률과 같은 성능 정보를 추출하였다. 오창남의 연구[12]에서는 처리 응답 시간(Response time), 효율성(Throughput), 예외 발생률(Exception), 메소드(Method)별 처리 응답 시간 외에 추가적인 컴포넌트 성능 측정방법으로 각 빈 처리 응답 시간, 트랜잭션 처리 응답 시간, 알고리즘 처리 응답 시간, 메모리 사용률, 풀(Pool)에 따른 시스템 자원 사용률로 측정 하였다.

이에 본 연구에서는 성능 향상이 요구되는 컴포넌트를 점진적인 방법을 통해 EJB 3.0으로 전환하는 것이 연구의 주요목적이기 때문에 컴포넌트 성능 측정 지표를 초당 처리량으로 한다.

컴포넌트 재사용 측정 기준: 현재 소프트웨어 개발조직은 복잡한 소프트웨어를 빠른 시간에 고품질로 개발하여 사용자 요구사항을 만족시켜야 한다. 과거보다 더 복잡한 소프트웨어를 빠른 시간 내에 개발하기 위해서는 이미 검증된 소프트웨어 자산의 재사용을 통한 생산성 증대가 그 대안이 될 수 있다[13]. 소프트웨어 재사용은 소프트웨어 개발 과정에서 소프트웨어 개발 기간을 단축시키고 소프트웨어의 생산성과 품질 향상, 유지보수 비용과 테스트 비용을 절감할 수 있다[14]. 비용 절감 이외에 소프트웨어 재사용이 표방하고 있는 궁극적인 목적은 소프트웨어 애플리케이션을 기존 소프트웨어 컴포넌트들의 조립을 통해 개발하는 데 있다[15].

이처럼 컴포넌트 재사용에 대한 관심이 높아지면서 얼마나 컴포넌트를 재사용하였는지에 대한 측정연구가

요구되었다. 이에 Caldiera[16]는 소프트웨어의 재사용은 소프트웨어의 정확성(Correctness), 가독성(Readability), 테스트 가능성(Testability), 수정 용이성(Ease of Modification), 그리고 성능(Performance)에 의한 측정을 제시하였고, McClure[17]는 재사용 메트릭스를 위한 10종의 요소로서 컴포넌트의 공통성(Commonality), 컴포넌트의 재사용 임계값(컴포넌트 개발 비용의 회수를 위한 최소 재사용 횟수), 특정 컴포넌트의 타 영역/시스템 내에 사용 대비 재사용의 상대적 장점, 재사용 컴포넌트의 생성 비용, 재사용 컴포넌트의 사용 비용(수정 비용 포함), 컴포넌트 유지보수비용, 시스템의 공통성 정도, 시스템의 재사용성 정도, 시스템의 재사용 목표 수준, 시스템의 타 영역 내에 시스템 대비 재사용의 상대적 장점 등을 통해 측정해야 한다고 제시하였다. 또한, Prem의 연구[18]에서는 컴포넌트의 재사용 측정 기준을 구현 방식, 재사용되는 빈도, 검증된 컴포넌트 사용 여부, 기존 소스 수정 여부로 측정을 하였다.

이에 본 연구는 컴포넌트 변환기법이 점진적 EJB 3.0 전환 시 노력을 줄이는 방안임을 증명하기 위해 컴포넌트 재사용 측정 기준을 컴포넌트 개발과 재사용하기 위한 노력으로 한다.

3. EJB 2.1과 EJB 3.0의 차이 분석

본 장에서는 문헌을 기반으로 EJB 2.1과 EJB 3.0의 차이를 분석한다[1,4,19]. 이를 기반으로 변환 단계에서 EJB 2.1과 EJB 3.0의 혼재로 발생될 문제와 연결의 필요성을 도출 한다.

EJB 2.1과 EJB 3.0의 차이 분석의 결과는 다음과 같다.

Session Bean: EJB 2.1의 Session Bean은 ejbCreate, ejbActivate, ejbPassivate, ejbRemove 등의 콜백(Call Back) 메소드를 구현한다. 또한 비즈니스 메소드의 인터페이스를 로컬 인터페이스(Local Interface)나 리모트 인터페이스(Remote Interface)에 정의하고, 하나 이상의 create 메소드를 홈 인터페이스(Home Interface)에 구현한다. 하지만 EJB 3.0의 Session Bean은 POJI(Plain Old Java Interface) 기반으로 비즈니스 로직의 인터페이스를 비즈니스 인터페이스에 구현한다. EJB 3.0의 Session Bean은 트랜잭션 처리와 보안 속성, 다른 빈을 사용해야 할 경우 등의 다양한 서비스 정보들을 Java SE5.0의 메타데이터인 Annotation으로 정의한다. EJB 3.0 Session Bean은 홈 인터페이스가 사라지고, 리모트 인터페이스와 로컬 인터페이스가 비즈니스 인터페이스로 통합되었다. 이와 같은 차이로 EJB 2.1의 기능 향상을 위해 EJB 3.0으로 대체 시 기존 EJB 2.1로 개발된 연관 애플리케이션과의 호환성 문제를 해결할 수 있는 방안이 필요하다.

Entity Bean: EJB 3.0의 Entity Bean은 EJB 2.1 Entity Bean과는 달리 콜백 메소드를 필요로 하지 않고 POJO(Plain Old Java Object) 기반의 코딩과 Annotation으로 Bean 클래스만을 구성한다. EJB 3.0의 Entity Bean은 테이블의 1:M, M:1, M:M 등을 Annotation으로 지원한다. 특히 EJB 3.0의 Entity Bean은 오픈 소스인 Hibernate를 모델로 새롭게 정의하여 EJB 2.1 보다 월등한 성능과 개발의 편리성을 제공한다. 이와 같이 EJB 3.0의 Entity Bean에서는 모든 인터페이스 클래스가 사라지고 POJO기반의 Bean 클래스만으로 구성이 된다. EJB 2.1 Session Bean은 Entity Bean의 홈 인터페이스(로컬 또는 리모트)와 컴포넌트 인터페이스(홈 및 리모트)의 Bean 인스턴스(Instance)를 생성하여 사용한다. 하지만 EJB 3.0에서의 Session Bean은 @EntityManager Annotation을 적용하여 간단하게 Entity Bean을 사용할 수 있다. EJB 3.0의 Entity Bean 구성은 기존의 EJB 2.1 Entity Bean과 완전히 다르고, EJB 3.0 Session Bean에서의 Entity Bean 사용 방법도 EJB 2.1과 다르다. 그러므로 Entity Bean의 상이함은 Session Bean에서 상이함을 처리할 수 있는 방안이 요구된다.

Deployment Descriptor: EJB 2.1은 트랜잭션 처리와 보안속성, 미들웨어에서의 Bean에 대한 환경설정 사항과 같은 다양한 서비스 정보가 xml 형태의 배포 디스크립터에 반드시 작성되어야 한다. 그러나 EJB 3.0은 배포 디스크립터를 소스 코드 내에 Annotation으로 정의하여 대신할 수 있다. 이처럼, 배포 디스크립터와 Annotation의 상이함을 해결할 수 있는 방안이 필요하다.

Client View: EJB 3.0 Session Bean의 인터페이스가 비즈니스 인터페이스로 통합되었기 때문에 기존 EJB 2.1 클라이언트에서 EJB 3.0 컴포넌트를 사용하기 위해서는 기존 EJB 2.1 클라이언트에 많은 변경이 요구된다. 그러므로 EJB 2.1 클라이언트의 변경 없이 EJB 3.0을 사용할 수 있는 방안이 필요하다.

표 1은 EJB 2.1과 EJB 3.0의 차이를 분석 정리한 것이다. 차이 분석의 결과는 다음과 같이 몇 가지로 요약된다. 첫째, EJB 3.0 Session Bean의 인터페이스가 비즈니스 인터페이스로 통합되어 EJB 2.1 Session Bean의 인터페이스 구조와 상이하다. 이에 따라 클라이언트 뷰도 상이하다. 둘째, EJB 3.0 Entity Bean은 모든 인터페이스가 사라지고 POJO기반의 bean 클래스만으로 구성되어 EJB 2.1 Entity Bean과 상이하다. 셋째, EJB 3.0은 Annotation으로 배포 디스크립터를 대신하여 사용할 수 있다는 점이 EJB 2.1과 상이하다.

차이 분석을 통해 나타난 EJB 2.1을 EJB 3.0으로의 점진적 변환 시 문제점은 EJB 3.0으로 변환된 컴포넌트를

표 1 EJB 2.1과 EJB 3.0의 차이 분석

구 분	EJB 2.1	EJB 3.0	차 이
Runtime Environment	JDK1.3 또는 그 이상	JDK 5.0 또는 그 이상	EJB 3.0은 JDK5.0 또는 그 이상에서만 실행이 가능함
Session Bean	로컬 홈 인터페이스 리모트 홈 인터페이스 로컬 인터페이스 리모트 인터페이스 Bean 클래스	비즈니스 인터페이스 Bean 클래스(POJO)	EJB 3.0 Session Bean의 홈 인터페이스가 사라지고 리모트 인터페이스와 로컬 인터페이스가 POJI 기반의 비즈니스 인터페이스로 통합됨
Entity Bean	로컬 홈 인터페이스 리모트 홈 인터페이스 로컬 인터페이스 리모트 인터페이스 Bean 클래스	Bean 클래스(POJO)	EJB 3.0 Entity Bean에서는 모든 인터페이스 클래스가 사라지고 POJO 기반의 bean 클래스만으로 구성됨
Deployment Descriptor	xml형태의 배포 디스크립터를 반드시 작성	xml형태의 배포 디스크립터를 상황에 따라 선택하여 작성	EJB 3.0은 xml형태의 배포 디스크립터를 Annotation으로 대신할 수 있음
Client View	홈 인터페이스(로컬 또는 리모트)와 컴포넌트 인터페이스(홈 및 리모트)의 Bean 인스턴스를 생성하여 사용	@EJB등의 Annotation을 정의하여 EJB 컨테이너가 참조하고 있는 EJB를 검색하여 사용	EJB 3.0은 홈 인터페이스가 사라짐에 따라 Annotation을 사용하여 비즈니스 메소드 호출함

호출하는 연관된 EJB 2.1 컴포넌트를 찾아 수정해야 하는 것이다. 그러므로 기존 애플리케이션의 서비스를 고려하면서, 연관된 EJB 2.1 컴포넌트를 수정하지 않고 EJB 3.0컴포넌트를 호출할 수 있는 변환기법이 필요하다.

4. EJB 2.1에서 EJB 3.0 변환 기법

본 장에서는 3장의 EJB 2.1와 EJB 3.0의 차이 분석을 통해 도출된 문제점을 해결하고, 기존 애플리케이션의 서비스는 고려하였지만 EJB 3.0으로의 점진적인 변환 기법을 제시하지 않은 Deepark의 마이그레이션 기법과 EJB 3.0으로 완전 대체 시 재사용의 어려움이 있는 자바 썬의 "EJB 2.x-EJB 3.0 호환성" 기법의 문제점을 해결할 수 있는 변환 기법을 제안한다. 본 연구에서 제안하는 세가지 변환 기법은 직접 변환 기법, 간접 변환 기법, 간접 템플릿 변환 기법이다.

4.1 방안 1 : 직접 변환 기법

직접 변환 기법은 기존 EJB 2.1 컴포넌트의 Required Interface를 변경하여 EJB 3.0 컴포넌트의 Provided Interface를 호출하는 형태로 변환하는 기법이다. 직접 변환 기법은 EJB 3.0으로 완전 대체 시 EJB 3.0 컴포넌트의 재사용이 가능하지만, 변환 단계에서 기존 EJB 2.1 컴포넌트에 EJB 3.0 컴포넌트를 사용하기 위한 노력이 요구된다.

그림 1은 제안하는 직접 변환 기법의 구성도이다. 직접 변환 기법은 EJB 2.1 클라이언트에서 EJB 2.1 Session Bean의 홈 인터페이스를 생성하여 EJB 2.1 Session Bean의 리모트 인터페이스에 정의된 비즈니스 메소드를 호출한다. EJB 2.1 Session Bean의 리모트 인터페이스에 정의된 비즈니스 메소드는 EJB 2.1

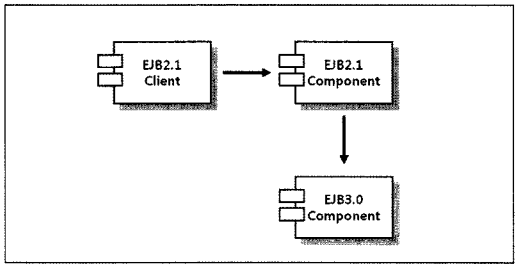

그림 1 직접 변환 기법의 구성도

Session Bean의 bean 클래스에 정의된 비즈니스 메소드를 호출한다. 이때, 기존의 비즈니스 로직을 수행하지 않고 EJB 3.0 컴포넌트의 비즈니스 인터페이스를 생성하여 EJB 3.0 컴포넌트의 비즈니스 메소드를 호출한다.

그림 2는 제안하는 직접 변환 기법에 대한 구현 소스이다. 직접 변환 기법은 EJB 2.1 클라이언트의 변경 없이 EJB 2.1 컴포넌트에서 EJB 3.0 컴포넌트의 비즈니스 메소드를 호출한다. EJB 3.0의 비즈니스 인터페이스의 구현은 @Remote Annotation으로 리모트 인터페이스를 정의하고 EJB 2.1 클라이언트의 요청사항 대한 비즈니스 메소드를 정의한다. EJB 3.0 Session Bean의 bean 클래스의 구현은 @Stateless Annotation으로 Session Bean을 Stateless Session Bean으로 정의한다. 그리고 @Resource Annotation을 정의하여 SessionContext와 @PersistenceContext Annotation을 정의하여 Container Managed Entity Manager를 각각 참조하도록 하고 EJB 2.1 클라이언트의 요청사항에 대한 비즈니스 로직을 구현한다. 기존 EJB 2.1에서는 Session Bean에서 Entity Bean을 호출하기 위해서 Entity Bean 검

EJB3.0 Session Bean의 비즈니스 인터페이스

```
@Remote
public interface JBEJB3_S_E {
    void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year,String sCuri_num);
}
```

EJB3.0 Session Bean의 bean 클래스

```
@Stateless
public class JBEJB3_S_E_Bean implements JBEJB3_S_E {
  @Resource
   SessionContext ctx;
  @PersistenceContext (unitName="manager")
   EntityManager em;
  public void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year, String sCuri_num) {
       JBEJB3_E jb = new JBEJB3_E(); //Serializable Class
       jb.setYear(sYear);
        ...
       em.persist(jb);
  }
}
```

EJB2.1 Session Bean의 bean 클래스 : EJB3.0 호출

```
       Properties props = new Properties();
       props.put(InitialContext.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
       props.put(InitialContext.PROVIDER_URL, "jnp://localhost:1099");
       props.put(InitialContext.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces");

       Context ctx = new InitialContext(props);
       JBEJB3_S_E look = (JBEJB3_S_E) ctx.lookup("JBEJB3_S_E_Bean/remote");
       look.insertEJB2_3( sYear, sSmt, sStudent_cd, sDept_cd, sCom_year, sCuri_num );
```

EJB2.1 클라이언트 : 변경사항 없음

```
  try {
      Context ctx = new InitialContext(env);
      Object obj = ctx.lookup( "JBEJB2_S_JNDI" );
      JBEJB2_S_Home home = (JBEJB2_S_Home)PortableRemoteObject.narrow(obj, JBEJB2_S_Home.class);
      JBEJB2_S look = home.create();
      look.insertEJB2_3(sYear,sSmt,sStudent_cd,sDept_cd,sCom_year,sCuri_num);
  } catch(Exception e) { ... }
```

그림 2 직접 변환 기법의 애플리케이션 소스

색 과정이 필요하였으나 EJB 3.0에서는 @Persistence-Context Annotation을 정의하면 Entity Bean 검색 과정 없이 EntityManager 객체 생성으로 Entity Bean을 사용할 수 있다.

4.2 방안 2 : 간접 변환 기법

간접 변환 기법은 기존 EJB 2.1 컴포넌트의 Required Interface에서 커넥터를 사용하도록 변경하여 EJB 3.0 컴포넌트의 Provided Interface를 호출하는 형태로 변환하는 기법이다.

복잡한 서브시스템에 대해서 단순한 인터페이스를 제공하고 클라이언트와 다른 서브 시스템간의 결합도를 줄이며 일관된 하나의 인터페이스를 제공하는 커넥터를 제안한다. 이러한 커넥터는 복잡성, 연결성, 종속성, 빈번한 메소드의 호출을 감소시키며 서브시스템의 변경에 독립적으로 자유롭고 복잡한 내부를 숨길 수 있기 때문에 Façade 패턴과 유사하다.

그림 3은 간접 변환 기법의 구성도이다. 간접 변환 기법은 EJB 2.1 클라이언트에서 EJB 2.1 Session Bean의 홈 인터페이스를 생성하여 EJB 2.1 Session Bean의

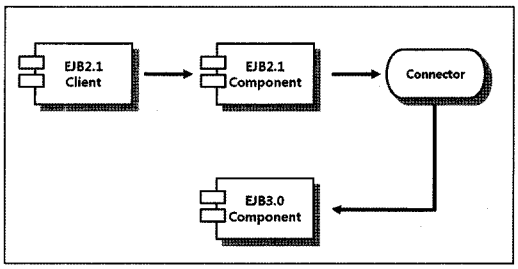

그림 3 간접 변환 기법의 구성도

리모트 인터페이스에 정의된 비즈니스 메소드를 호출한다. 호출된 리모트 인터페이스의 비즈니스 메소드는 EJB 2.1 Session Bean의 bean 클래스에 정의된 비즈니스 메소드를 호출한다. 이때, 기존의 비즈니스 로직을 수행하지 않고 커넥터의 객체를 생성하여 커넥터에 정의된 메소드를 호출한다. 커넥터의 메소드는 EJB 3.0 컴포넌트의 비즈니스 인터페이스를 생성하여 EJB 3.0 컴포넌트의 비즈니스 로직을 호출한다.

그림 4는 간접 변환 기법에 대한 구현 소스이다. 간접

EJB3.0 Session Bean의 비즈니스 인터페이스

```
@Remote
public interface JBEJB3_S_E {
    void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year, String sCuri_num);
}
```

EJB3.0 Session Bean의 bean 클래스

```
@Stateless
public class JBEJB3_S_E_Bean implements JBEJB3_S_E {
  @Resource
   SessionContext ctx;
  @PersistenceContext (unitName="manager")
   EntityManager em;
  public void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year,
      String sCuri_num) {
        JBEJB3_E jb = new JBEJB3_E(); //Serializable Class
        jb.setYear(sYear);
        ...
        em.persist(jb);
  }
}
```

커넥터 클래스

```
public class EJB23Connector {
  public JBEJB23Connector(){}
  public void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year, String sCuri_num) {
          ...
          Properties props = new Properties();
          props.put(InitialContext.INITIAL_CONTEXT_FACTORY, "org.jnp.interfaces.NamingContextFactory");
          props.put(InitialContext.PROVIDER_URL, "jnp://localhost:1099");
          props.put(InitialContext.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces");
          Context ctx = new InitialContext(props);
          JBEJB3_S_E look = (JBEJB3_S_E) ctx.lookup("JBEJB3_S_E_Bean/remote");
          look.insertEJB2_3(sYear,sSmt,sStudent_cd,sDept_cd,sCom_year,sCuri_num);
          ...
  }
}
```

EJB2.1 Session Bean의 bean 클래스 : 커넥터를 호출

```
  try{
      EJB23Connector connector = new EJB23Connector();
      connector. insertEJB2_3( sYear, sSmt, sStudent_cd, sDept_cd, sCom_year, sCuri_num );
  }catch (Exception e) {...}
```

EJB2.1 클라이언트 : 변경사항 없음

```
  try {
      Context ctx = new InitialContext(env);
      Object obj = ctx.lookup( "JBEJB2_S_JNDI" );
      JBEJB2_S_Home home = (JBEJB2_S_Home)PortableRemoteObject.narrow(obj, JBEJB2_S_Home.class);
      JBEJB2_S look = home.create();
      look.insertEJB2_3(sYear,sSmt,sStudent_cd,sDept_cd,sCom_year,sCuri_num);
  } catch(Exception e) { ... }
```

그림 4 간접 변환 기법의 애플리케이션 소스

변환 기법에 대한 EJB 2.1 클라이언트 구현 소스는 직접 변환 기법과 마찬가지로 변경사항이 없다. 하지만, 간접 변환 기법의 EJB 2.1 Session Bean의 bean은 EJB 3.0 컴포넌트를 직접 호출하는 직접 변환 기법과는 달리 커넥터를 호출한다. 커넥터는 EJB 3.0 컴포넌트의 비즈니스 인터페이스를 생성하여 EJB 3.0 컴포넌트의 비즈니스 메소드를 호출한다.

간접 변환 기법은 EJB 2.1 컴포넌트의 변경 시 연관 애플리케이션과 호환이 가능하고, EJB 3.0으로 완전 대체 시 EJB 3.0 컴포넌트의 재사용이 가능하다. 간접 변환 기법을 직접 변환 기법과 비교해 보면, 커넥터의 사용으로 유연한 변경 및 컴포넌트 관리의 편리성을 제공한다.

4.3 방안 3 : 간접 템플릿 변환 기법

간접 템플릿 변환 기법은 간접 변환 기법에 템플릿(Template) 패턴을 적용하여 EJB 2.1 컴포넌트를 EJB 3.0 컴포넌트로 변환하는 기법이다. 간접 변환 기법은 다수의 EJB 2.1 컴포넌트를 EJB 3.0 컴포넌트로 변환 시 커넥터에서 EJB 3.0 컴포넌트를 참조하기 위해 반복적인 개발 노력이 필요하다. 이러한 반복적인 개발 노력을 절감하고 타입에 대한 투명성 지원을 위해 템플릿 패턴 클래스를 제안한다.

그림 5는 간접 템플릿 변환 기법의 구성도이다. 간접 템플릿 변환 기법은 EJB 2.1 클라이언트에서 EJB 2.1 Session Bean의 홈 인터페이스를 생성하여 EJB 2.1

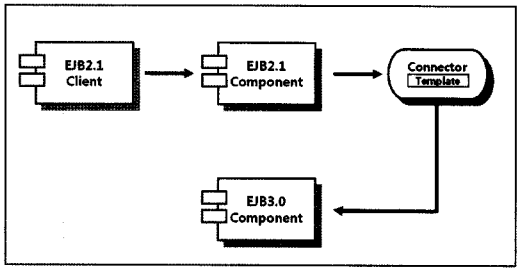

그림 5 간접 템플릿 변환 기법의 구성도

Session Bean의 리모트 인터페이스에 정의된 비즈니스 메소드를 호출한다. 호출된 리모트 인터페이스의 비즈니스 메소드는 EJB 2.1 Session Bean의 bean 클래스에 정의된 비즈니스 메소드를 호출한다. 이때, 기존의 비즈니스 로직을 수행하지 않고 템플릿 커넥터의 객체를 생성하여 템플릿 커넥터에 정의된 메소드 호출을 한다. 템플릿 커넥터의 메소드는 템플릿 패턴을 통해 EJB 3.0 컴포넌트의 비즈니스 인터페이스를 생성하고 EJB 3.0 컴포넌트의 비즈니스 로직을 호출한다.

그림 6은 간접 템플릿 변환 기법에 대한 구현 소스이다. 간접 변환 기법과 비교되는 간접 템플릿 변환 기법의 소스는 커넥터에서 EJB 3.0 컴포넌트 참조하는 로직

EJB3.0 Session Bean의 비즈니스 인터페이스

```
@Remote
public interface JBEJB3_S_E {
    void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year, String sCuri_num);
}
```

EJB3.0 Session Bean의 bean 클래스

```
@Stateless
public class JBEJB3_S_E_Bean implements JBEJB3_S_E {
  @Resource
   SessionContext ctx;
  @PersistenceContext (unitName="manager")
   EntityManager em;
  public void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year,
      String sCuri_num) {
       JBEJB3_E jb = new JBEJB3_E(); //Serializable Class
       jb.setYear(sYear);
       ...
       em.persist(jb);
  }
}
```

템플릿 패턴 클래스

```
public class TemplateConnector<T> {
   private String m_JNDI_Name;
   public TemplateConnector (String JNDI_Name){ this.m_JNDI_Name=JNDI_Name;}
   public T getBean(){
      Context ctx = JNDIHelper.getJNDIContext();
      T Ret= null;
      try{  Ret = (T)ctx.lookup(this.m_JNDI_Name);  }catch(Exception e){ e.printStackTrace();   }
      return Ret;
   }
}
```

커넥터 클래스

```
public class EJB23Connector {
  public JBEJB23Connector(){}
  public void insertEJB2_3(String sYear,String sSmt,String sStudent_cd,String sDept_cd,String sCom_year, String sCuri_num) {
     TemplateConnector<JBEJB3_S> connector = new TemplateConnector("JBEJB3_S_Bean/remote");
     connector.getBean(). insertEJB2_3(sYear,sSmt,sStudent_cd,sDept_cd,sCom_year,sCuri_num); ...
  }
}
```

EJB2.1 Session Bean의 bean 클래스 : 커넥터를 호출

```
try{
    ConnectorTemplate<JBEJB3_S_E> connector = new ConnectorTemplate("JBEJB3_S_E_Bean/remote");
    connector.getBean().insertEJB2_3( sYear, sSmt, sStudent_cd, sDept_cd, sCom_year, sCuri_num );
}catch (Exception e) {...}
```

EJB2.1 클라이언트 : 변경사항 없음

```
try {
    Context ctx = new InitialContext(env);
    Object obj = ctx.lookup( "JBEJB2_S_JNDI" );
    JBEJB2_S_Home home = (JBEJB2_S_Home)PortableRemoteObject.narrow(obj, JBEJB2_S_Home.class);
    JBEJB2_S look = home.create();
    look.insertEJB2_3(sYear,sSmt,sStudent_cd,sDept_cd,sCom_year,sCuri_num);
} catch(Exception e) { ... }
```

그림 6 간접 템플릿 변환 기법의 애플리케이션 소스

을 분리하여 JNDI(Java Naming and Directory Interface) Helper 클래스와 템플릿 패턴 클래스를 구현한다. 템플릿 패턴 클래스는 EJB 3.0 인터페이스의 객체 생성 로직을 공통으로 사용할 수 있다. 간접 템플릿 변환 기법은 EJB 3.0 인터페이스의 객체 생성 로직을 반복적으로 개발하지 않기 때문에 다수의 EJB 2.1 컴포넌트를 변환 시 개발 생산성을 높여 준다.

5. 사례연구

본 연구에서 제시한 EJB 변환 기법을 성능 향상이 요구되는 D대학의 행정 ERP 시스템에 적용해보았다. D대학의 행정 ERP 시스템은 Unix 운영체제와 JBOSS (버전 4.0.4) 애플리케이션 서버 기반으로써 급여, 회계 등의 일반행정 업무와 등록, 수강 신청 등의 학사행정 업무를 EJB 2.1로 개발되어 운영중인 시스템이다. 그러나 D대학의 행정 ERP 시스템은 수강신청 기간 동안 시스템의 성능저하 현상이 발생되는 문제점이 있었다. 이를 확인하기 위해 수강신청 기간 동안 J사의 애플리케이션 성능 관리 툴을 사용하여 성능 진단하였다. 성능 진단 결과, D대학의 행정 ERP 시스템은 하루 평균 방문자 수 4000여명, 최고 방문자 수 6,000여명이었고 수강 신청 관련 프로그램은 시스템 사용 대비율이 91%로 분석되었다. 또한, 수강 신청 프로그램의 평균 처리시간은 6초로 분석되었다. 이에 성능 관리 툴에서 제공하는 SQL 쿼리 성능 분석 등의 결과를 가지고 SQL 쿼리 튜닝과 데이터베이스 성능 튜닝의 방법으로 조치하였으나 수강신청 프로그램의 성능저하 현상은 해결 되지 않았다. 이러한 이유로 수강신청관련 컴포넌트들에 대한 성능향상이 요구되었다. 그래서, EJB 3.0 기반의 애플리케이션 환경을 지원하는 JBOSS 애플리케이션 서버에서 12개의 수강신청관련 컴포넌트를 EJB 3.0으로 변환하기 위해 자바 썬의 "EJB 2.x-EJB 3.0 호환성" 기법, Deepak의 마이그레이션 기법과 본 연구에서 제시한 직접 변환 기법과 간접 변환 기법, 간접 템플릿 변환 기법을 적용하였다.

EJB 3.0으로의 변환은 성능 향상을 하기 위한 것이지만, 변환 단계를 거치면서 구현된 EJB 3.0 컴포넌트의 재사용을 위한 노력을 고려해야 한다. 따라서, 본 연구에서는 성능에 대한 품질향상요소를 측정하기 위해 초당 처리량을 측정하였고, 컴포넌트 재사용을 위한 개발 노력을 측정하여 비교 분석 하였다.

표 2는 Apache JMeter를 사용하여 각 기법의 초당 처리량을 비교한 것이다. 각 기법의 초당 처리량 비교는 샘플 데이터 1000건을 동시 입력하는 방법으로 10회 실시하여 나온 초당 처리량의 평균값으로 하였다. 모든 기법들은 기존 EJB 2.1보다 초당 처리량이 향상이 되었다. 성능이 우수한 기법 순은 Deepak의 마이그레이션 기법, 자바썬의 "EJB 2.x-EJB 3.0 호환성" 기법, 간접 템플릿 변환 기법, 간접 변환 기법, 직접 변환 기법으로 Deepak의 마이그레이션 기법이 가장 우수하였다. 하지만, 성능 측정 과정에서 1KB이내의 오차는 자주 발생이 되었다. 오차를 고려하면 모든 기법은 성능이 동일하다고 간주할 수 있다.

표 3은 변환 단계에서 개발한 컴포넌트를 EJB 3.0 완전 대체 시 컴포넌트 재사용하기 위한 시도의 결과이다. EJB 3.0 완전 대체 시 자바 썬의 "EJB 2.x-EJB 3.0 호환성"기법은 EJB 2.1을 위한 추가적인 클래스가 구현(홈 또는 리모트 인터페이스)되어 있어 다른 EJB 3.0 컴포넌트와 함께 사용이 가능하도록 신규 커넥터 개발 등의 노력이 필요하였다. Deepak의 마이그레이션 기법과 제안한 3가지 기법 모두는 재사용이 가능하였다. 하지만, 변환 단계에서 Deepak의 마이그레이션 기법은 EJB 3.0 컴포넌트 사용이 가능하기 위해서 기존 EJB 2.1 클라이언트의 소스 코드를 수정하는 노력과 연관 애플리케이션 전부를 찾아 소스 코드를 수정해야 하는 노력, 기존 애플리케이션의 서비스에 문제가 없도록 하기 위한 노력이 필요하였다. 변환 단계에서 제안 기법 중 직접 변환 기법은 기존 EJB 2.1 컴포넌트 내에서 EJB 3.0 컴포넌트를 사용하기 위한 구현에 노력이 요구되었다.

각각의 기법은 동일한 성능 향상을 보였으나, 자바 썬

표 2 초당 처리량 비교

구 분	EJB 2.1 (제안적용전)	변 환 기 법				
		자바 썬의 EJB 2.x-EJB 3.0 호환성	Deepak의 마이그레이션	**직접변환**	**간접변환**	**간접템플릿변환**
초당 처리량	44.89 KB	59.58 KB	59.63 KB	58.35 KB	59.13 KB	59.47 KB

* 초당 처리량 (KB/sec) : 1초에 처리할 수 있는 KB

표 3 EJB 3.0 완전 대체시 컴포넌트 재사용

구 분	자바 썬의 EJB 2.x-EJB 3.0 호환성	Deepak의 마이그레이션	**직접변환**	**간접변환**	**간접템플릿변환**
컴포넌트 재사용	다소 어려움	가능	가능	가능	가능

표 4 단계별 개발 노력

구 분	자바 썬의 EJB 2.x-EJB 3.0 호환성	Deepak의 마이그레이션	직접변환	간접변환	간접템플릿변환
변환단계	4 M/M	4 M/M	3.5 M/M	3 M/M	3 M/M
완전대체단계	1.5 M/M	1 M/M	1 M/M	1 M/M	1 M/M
합 계	5.5 M/M	5 M/M	4.5 M/M	4 M/M	4 M/M

* M/M: Man Month

표 5 EJB 3.0 변환 기법 선택 고려 사항

기 법 / 고려 사항	자바 썬의 EJB 2.x-EJB 3.0 호환성	Deepak의 마이그레이션	직접변환	간접변환	간접템플릿변환
성능 향상	O	O	O	O	O
기존 애플리케이션 서비스 고려	O	X	O	O	O
컴포넌트 재사용	△	O	O	O	O
컴포넌트 재사용을 위한 개발 노력 절감	X	O	O	O	O
컴포넌트의 유연한 변경 및 관리	X	X	X	O	O
다수의 EJB 2.1 변환시 편리성 제공	X	X	X	X	O

* 만족도 : O - 상, △ - 중, X - 하

의 "EJB 2.x-EJB 3.0 호환성" 기법이 EJB 3.0의 일반적인 클래스 구성을 벗어나고 추가적인 인터페이스를 구현하는데 어려움이 있었고 EJB 3.0 완전 대체 시 연관 애플리케이션과의 호환문제로 재사용을 하기 위해서는 추가적인 노력이 들었다. Deepak의 마이그레이션 기법은 EJB 3.0 명세서를 지원하고 있으나 EJB 3.0 변환 단계에서 기존 연관 애플리케이션과의 호환을 고려하지 않아 서비스의 중단과 같은 심각한 상황이 발생되었다. 그러나 제안된 기법 모두는 기존 연관 애플리케이션과의 호환 문제가 없었다. 제안된 기법 모두는 EJB 3.0 명세서를 준수하고 있기 때문에 EJB 3.0으로 완전 대체 시에 EJB 3.0 컴포넌트의 재사용이 가능하였다. 특히, 간접 변환 기법과 간접 템플릿 변환 기법은 개발 노력이 적게 소요되었다. 표 4는 5년 이상의 경험을 가진 중급기술자 5명이 각각 수강 신청과 관련된 12개의 컴포넌트를 개발하는데 소요된 시간을 M/M로 측정하여 단계별 개발 노력으로 비교한 것이다.

표 5의 6가지 고려사항은 2.4절에서 언급한 컴포넌트의 성능 측정 지표 및 재사용 측정 기준과 본 연구를 통해 나타난 EJB 3.0 변환 시 특성을 기반으로 정의한 것이다.

성능향상: 변환된 EJB 3.0 컴포넌트가 성능향상 되었음을 평가할 수 있는 평가 요소가 필요하였고, 이에 기존 연구인 컴포넌트의 성능 측정 지표로 중 초당 처리량을 평가 요소로 도출하였다.

기존 어플리케이션 서비스 고려: 변환된 EJB 3.0 컴포넌트가 운영중인 다른 어플리케이션과 호환이 됨을 평가 할 수 있는 평가 요소가 필요하였다. 이에 다른 컴포넌트와의 인터페이스 호환 여부를 평가요소로 도출하였다.

컴포넌트 재사용: 변환된 EJB 3.0 컴포넌트가 완전 대체단계에서 재사용되었음을 평가할 수 있는 평가 요소가 필요하였고, 이에 기존 연구인 컴포넌트 재사용 측정 기준 중 컴포넌트 호환을 위한 기존 소스 수정 여부를 평가 요소로 도출하였다.

컴포넌트 재사용을 위한 개발 노력 절감: 컴포넌트를 재사용 하기 위한 노력을 평가할 수 있는 평가 요소가 필요하였고, 이에 기존 연구인 컴포넌트 재사용 측정 기준 중 재사용 컴포넌트의 생성 비용과 재사용 컴포넌트의 수정 비용을 평가 요소로 도출하였다.

컴포넌트의 유연한 변경 및 관리: EJB 2.1 컴포넌트와 EJB 3.0 컴포넌트가 혼재한 변환단계에서 요구사항으로 인한 컴포넌트의 변경 및 관리에 대해 평가할 수 있는 평가 요소가 필요하였다. 이에 요구사항에 대한 중재 여부를 평가요소로 도출하였다.

다수의 EJB 2.1 변환 시 편리성 제공: 다수의 EJB 2.1 컴포넌트를 변환 시 반복적 개발 노력을 줄일 수 있는 방안이 필요하였다. 이에 반복적 개발 문제 해결 여부를 평가 요소로 도출하였다.

자바 썬의 "EJB 2.x-EJB 3.0 호환성"기법은 컴포넌트 재사용을 하기에는 다소 노력이 필요하였고, Deepak의 마이그레이션 기법은 컴포넌트 재사용은 가능하나 기존 연관 애플리케이션과의 호환에 문제가 있었다. 하지만 제안 기법들은 기존 연구들이 가진 문제점들을 모

두 해결 하였다. 특히 간접 템플릿 변환 기법은 성능 향상, 기존 애플리케이션의 서비스, 컴포넌트의 재사용, 개발 노력 절감, 컴포넌트의 유연한 변경 및 관리의 용이함, 다수의 EJB 2.1을 변환 시 편리성 제공 여부 등의 기준을 모두 만족하는 기법이다.

전체적으로 기존 연구와 제안기법을 비교해 볼 때, 컴포넌트의 성능 향상됨은 동일하였다. 하지만, 제안기법은 기존 연구의 문제점인 점진적 변환단계에서 기존 애플리케이션의 서비스를 고려하지 못하는 것과 EJB 3.0 완전 대체 시 컴포넌트 재사용의 문제를 해결하였고, 단계별 개발 노력도 약 1M/M가 절감되었다. 특히, 제안기법 중 간접 템플릿 변환 기법은 유연한 변경 및 컴포넌트의 관리와 다수의 EJB 2.1 변환 시 편리성까지 제공하여 기존 연구들 보다는 향상된 기능을 제공하였다.

6. 결 론

본 연구에서는 EJB 3.0 기반의 애플리케이션 환경에서 EJB 2.1을 사용하다가 성능 향상이 요구되는 컴포넌트부터 EJB 3.0으로의 점진적 변환을 하기 위해 변환 단계와 완전 대체 단계로 구분을 하였다. 변환 단계에서는 기존 애플리케이션의 서비스를 고려하며 EJB 3.0의 향상된 기능을 사용하고 변환 단계와 완전 대체 단계에 소요되는 개발 노력을 최소화할 수 있는 변환 기법을 제시하였다. 기존 연구의 기법과 제안된 변환 기법을 D 대학의 행정ERP시스템에 적용하여 컴포넌트의 재사용/개발 노력, 초당 처리량, 변환 기법 선택 시 고려사항으로 평가를 한 결과 간접 템플릿 변환 기법이 우수함을 알 수 있었다. 그러나 기존 애플리케이션의 서비스와 성능 향상만을 절대적으로 고려해야 하는 환경에서는 자바 썬의 "EJB 2.x-EJB 3.0 호환성" 기법의 사용도 권장한다.

본 연구는 EJB 2.1 컴포넌트에서 EJB 3.0 컴포넌트로의 점진적 변환에 대한 지침을 제공하며, 제안된 변환 기법의 사용으로 컴포넌트 재사용과 성능향상을 할 수 있고 개발 노력을 최소화 할 수 있다.

향후 연구로는 본 연구에서 제안한 점진적 변환을 위한 EJB 커넥터 자동 생성에 대한 연구와 EJB 컴포넌트의 완전 대체를 위한 자동화 툴 개발에 대한 연구가 필요하다.

참 고 문 헌

[1] Rima Patel Sriganesh, Gerald Brose, Micah Silverman, Mastering Enterprise JavaBeans™ 3.0, 4th ED., Wiley, 2006.

[2] Rod Johnson, Expert One-on-One™ J2EE™ Development without EJB™, Wiley, 2004.

[3] Rahul Biswas, Scott Oak, Eileen Loh, "Writing Performance EJB Beans in the Java EE Platform 5 (EJB 3.0) Using Annotations," http://developers.sun.com/learning/javaoneonline/j1sessn.jsp?sessn=TS-1624&yr=2006&track=coreenterprise, 2006.

[4] Raghu R. Kodali, Jonathan R. Wetherbee and Peter Zadrozny, Beginning EJB 3 Application Development from Novice to Professional, Apress, 2006.

[5] LIU Yao, "Support for DSL in Eclipse," Technical University of Hamburg-Harburg, 2006.

[6] Ken Saks, "EJB 3.0 COMPATIBILITY AND MIGRATION," http://java.sun.com/mailers/techtips/enterprise/2007/TechTips_Feb07_static.html#1, 2007.

[7] Hyun Gi Min, Si Won Choi, and Soo Dong Kim, "Using Connectors to Resolve Partial Matching Problems in COTS Component Acquisition," Proceedings of 7th International Symposium on Component-based Software Engineering, Edinburgh, Scotland, pp. 40-47, 2004.

[8] Deepak Vohra, "Migrating EJB 2.1 Entity and Session Beans to EJB 3.0," http://www.regdeveloper.co.uk/2006/04/25/EJB 3_migration/, 2006.

[9] 나학청, 김수동, "EJB 어플리케이션의 성능 모니터링 기법", 정보과학회논문지 : 소프트웨어 및 응용 제30권, 제5·6호, pp. 529-539, 2003. 6.

[10] Catalina M. Lladó, Peter G. Harrison, "Performance Evaluation of an Enterprise JavaBean Server Implementation," Proceedings of the 2nd international workshop on Software and performance(WOSP 2000, Ontario, Canada), pp. 180-188, 2000.

[11] Te-Kai Liu, Santhosh Kumaran, Zongwei Luo, "Layered Queueing Models for Enterprise JavaBean Applications," Fifth IEEE International Enterprise Distributed Object Computing Conference, Sept. 2001.

[12] 오창남, "J2EE 컴포넌트의 자동 성능 측정 도구 시스템 설계 및 구현", 한국항공대, 2000.

[13] B.W. Boehm, M. H. Pendo, A. B. Pyster, E.D. stuckle and R. D. William, "An Environment for Improving Software Productivity," IEEE Computer, June 1984.

[14] C. W. Krueger, "Software Reuse," ACM Computing Surveys, Vol.24, No.2, pp. 131-184, Jun 1992.

[15] 박수진, 박수용," 컴포넌트의 다면성과 서비스를 기반으로 하는 재사용 모델", 정보과학회논문지 : 소프트웨어 및 응용, 제34권, 제4호, pp. 303-316, 2007. 4.

[16] G. Caldiera & R. Basili, "Identifying and Qualifying reusable Software components," IEEE Computer, Volume 24, Issue 2, pp. 61-70, Feb 1991.

[17] C. McClure, The Three R's of Software Automation: Re-Engineering Repository Reusability, Prentice Hall, Inc., 1992.

[18] Prem Debanbu, Sakke Karstu, Walcelio Melo and

William Thomas, "Analytical and Empirical Evaluation of Software Reuse Metrics," In the 18th International Conference on Software Engineering, pp. 189-199, 1996.

[19] Ed Roman, Rima Patel Sriganesh, Gerald Brose, Mastering Enterprise JavaBeansTM, 3rd ED., Wiley, 2004.

이 후 재

1994년 경기대학교 물리학과 이학사. 2004년 숭실대학교 정보과학대학원 공학석사 2005년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 컴포넌트 기반 개발(CBD), 소프트웨어 재사용, 소프트웨어 아키텍처, 소프트웨어 프로세스, 오픈 소스 소프트웨어

김 지 혁

2003년 숭실대학교 컴퓨터학부 공학사 2005년 숭실대학교 컴퓨터학과 공학석사 2005년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 소프트웨어 유지보수, 소프트웨어 재사용, 서비스 지향 아키텍처, 오픈 소스 소프트웨어, 소프트웨어 품질 보증

류 성 열

1981년~현재 숭실대학교 교수. 1982년~1995년 숭실대학교 전자계산연구소 및 중앙전자계산소 소장. 1997년~1998년 George Mason University 객원 교수. 1998년~2001년 숭실대학교 정보과학대학원 원장. 2004년~현재 한국품질재단 운영위원회 위원장. 2006년~현재 공정거래위원회 성과관리위원회위원. 2008년~현재 정보통신연구진흥원 비상임이사. 관심분야는 소프트웨어 유지보수, 소프트웨어 재사용, 오픈 소스 소프트웨어, 소프트웨어 재공학/역공학