EJB 기반의 효율적인 설계 패턴 및 엔터프라이즈 아키텍처 설계 기법

(Effective Design Pattern and Enterprise Architecture Design Techniques in EJB Environment)

민 현 기 † 김 수 동 ††
(Hyun Gi Min) (Soo Dong Kim)

요 약 산업계에서 개발비용과 기간을 단축시키기 위해 시스템을 점차 Enterprise JavaBeans(EJB)로 개발하는 추세이다. 그러므로 시스템 재사용성, 확장성과 이식성을 높이기 위해 EJB를 위한 아키텍처가 중요해졌다. 그러나, 상위 레벨 수준의 추상적인 아키텍처는 제공되지만, 현재 가능한 J2EE기술을 사용하여 실제화 시키는 구체적 방법은 제공되지 않아 실용적인 소프트웨어 아키텍처에 관한 연구가 부족하다. EJB 규약(Specification)은 EJB를 운용하기 위한 세션빈, 엔티티빈들의 특성과 소규모(Fine Grained)방식의 컴포넌트 아키텍처만을 제시하고 있다. 그러므로 EJB는 작은 재사용 단위이기 때문에 EJB, 미들웨어 기술을 사용해도 기대만큼 재사용되지 않는다. 본 논문에서는 EJB 기반의 시스템을 위한 엔터프라이즈 소프트웨어 아키텍처를 구체적인 구현 기술과 기법을 함께 제안한다. 또한 효율적인 EJB 아키텍처를 설계하기 위한 EJB 설계 패턴을 제안한다. 설계 패턴들의 장단점을 분석하여 엔터프라이즈 아키텍처의 각 계층에 적합한 EJB 디자인 패턴을 식별하고, 디자인 패턴을 적용한 컴포넌트를 통해 최적의 컴포넌트간의 상호관계를 지원하는 아키텍처가 되도록 한다. EJB 설계기법을 객체수준의 화이트박스 형식인 소규모 EJB 컴포넌트로 부터 대규모(Coarse Grained) 방식의 EJB 컴포넌트로 설계하는 기법을 5가지로 제시하고, EJB 기반의 트랜잭션, 조립기법을 포함한 엔터프라이즈 아키텍처 설계 기법을 구체적으로 제안한다.

키워드 : Enterprise JavaBeans(EJB), 컴포넌트, 소프트웨어 아키텍처, 디자인 패턴, 웹 서비스

Abstract *In industry, it is a current trend that systems are developed by using Enterprise JavaBeans(EJB) technology for reducing the cost and the time. Thus, the architecture of EJB is getting more essential to enhance reusability, extensibility and portability of system. However little has been studied in the realm of the practical software architectures for EJB. The architecture has just bean studied in abstract level, but not in concrete level providing the method to substantiate it using the practical J2EE techniques. Just using the EJB technology doesn't guarantee the reusability of the artifacts because EJB specification provides the characteristics and architecture for only fine grained components as session and entity bean. In this paper, we propose the enterprise software architecture for the systems based on EJB and the concrete techniques for implementing that. Also, design patterns of modeling efficient enterprise architecture are represented. By analyzing both the strengths and the weaknesses of suggested design patterns, EJB design patterns which are suitable for each layer of enterprise architecture will be identified. Through the component which design patterns are applied, the architecture can support the optimized relationship between the components. Five techniques for designing components from fine grained to coarse grained based on EJB technology, and architecture design techniques including transaction and assembling techniques are proposed.*

Key words : Enterprise JavaBeans(EJB), Component, Software Architecture, Design Pattern, Web Services

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 비 회 원 : 숭실대학교 컴퓨터학과
hgmin@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
sdkim@ssu.ac.kr
논문접수 : 2003년 3월 24일
심사완료 : 2003년 7월 22일

1. 서 론

복잡하고 다양한 소프트웨어를 개발하기 위하여 객체지향 방법으로 소프트웨어를 개발하는 추세는 대규모 형식으로 재사용할 수 있는 프레임워크 개발기법으로

발전되었다[1]. 컴포넌트 개발기법은 하드웨어의 "플러그 앤드 플레이"의 개념을 소프트웨어로 옮긴 것처럼 필요한 부분을 기존에 존재하는 다른 컴포넌트들의 조합으로 소프트웨어를 구성하는 개발 기법은 시스템을 구축하는데 적은 시간, 노력, 비용의 이득을 가진다[2].

그러므로 개발된 컴포넌트를 조합하기 위한 소프트웨어 아키텍처가 필요하다. 소프트웨어 아키텍처는 컴포넌트들의 인터페이스와 컴포넌트의 내부 모듈의 연결에 의해 운용되는 시스템의 전체 구조이다[1]. 소프트웨어 아키텍처는 시스템을 분할하고 구조화함으로써 시스템의 성능을 향상시킨다. 또한 시스템 전체의 가시성을 제공함으로써 시스템 아키텍처 설계자가 시스템의 목적, 범위, 기능 등을 명확히 한다[4]. 하지만 시스템이 대규모화되어 여러 가지 문제점들이 발생된다. 시스템 개발 후 사용되지 않는 문제, 새로운 시스템이 추가되는 문제 등을 방지할 수 있다[5].

기존 연구에서는 가용한 구현 기술을 이용하여 실제화 시킬 수 있는 EJB의 특성으로 고려한 구체적인 아키텍처의 제시 없이, 상위 레벨의 추상적인 아키텍처만을 4~5단계의 계층으로만 나누었다. EJB 규약에는 엔터프라이즈 빈 여러 개를 묶어서 대규모 형식으로 구현하는 것이 가능하다고 기술되지만, 그 방법은 제시되지 않는다. 그러므로 소프트웨어 아키텍처 사상 없이 어플리케이션을 만들기 때문에 많은 문제점들이 야기된다[6].

시스템을 컴포넌트화하여 개발을 하고는 있지만, 기존의 모듈화 시킨 프로그램처럼 컴포넌트화 시켰기 때문에 재사용성과 이식성의 효율이 좋지 않았다. 그러므로 아키텍처 설계 기반의 소프트웨어 재사용은 소프트웨어의 고품질, 고생산성을 위한 최상의 방법이며, 잘 정의된 인터페이스를 통해 다른 시스템과 쉽게 결합 연동시킴으로써 대형 어플리케이션의 개발에서의 문제인 복잡성을 극복하고, 개발과 운용의 다양성과 재사용성, 확장성을 증가시킬 수 있다[7].

본 논문의 구성은 2장에서 관련 연구로서 타 논문에서 제시한 소프트웨어 아키텍처를 정의하고, 3장에서는 EJB로 컴포넌트를 구성하는 기법에 관해 제안하고, 그 기법에 대한 장단점을 제시한다. 4장에서는 엔터프라이즈 소프트웨어 아키텍처를 소개하고 컴포넌트 기반의 소프트웨어 아키텍처를 검증하기 위한 방법으로서 기존의 MVC 아키텍처를 기반으로 추상화 소프트웨어 아키텍처를 확장한다. EJB 컴포넌트 기반의 소프트웨어 아키텍처를 설계하기 위해 구체화된 소프트웨어 아키텍처를 제안한다. 5장에서는 설계된 아키텍처의 각 계층간의 인터페이스를 도출하기 위해서 계층간의 상호관계와 메시지 흐름을 모델링 한다. 6장에서는 소프트웨어 아키텍처를 평가하며, 마지막으로 7장에서는 본 논문의 결론을 맺는다.

2. 기반 연구

2.1 소프트웨어 아키텍처

소프트웨어 아키텍처는 컴포넌트들의 인터페이스와 컴포넌트의 내부 모듈의 연결에 의해 운용되는 전체 시스템의 구조이며, 소프트웨어 공학 연구 단체에서는 소프트웨어 아키텍처의 정의에 관해 다양한 정의가 되어 있다[8]. 대부분의 정의들은 컴포넌트, 커넥터(Connector)와 제약사항, 원인 또는 속성과 관련된 약간의 어트리뷰트를 포함하여 정의한다. 또한, Rapide에서 제안한 아키텍처 정의 언어의 정형적 아키텍처의 구성 요소에는 인터페이스들간 연결로 표현한다[9]. 아키텍처는 시스템의 구성 요소들이 정의되고, 개발하는 방법과 그들이 상호작용하는 방법의 개념적인 모델이다. 아키텍처와 개념상의 모델의 구현인 하부구조(Infrastructure)와는 구별된다. 아키텍처는 개념상의 모델의 사상을 바탕으로 실현 가능해야 한다.

소프트웨어 아키텍처는 구현 가능하게 기술되어야 하며, 상호 운영을 위한 제품들이 있어야 한다. 최신 기술과 제품들만을 선택한다고 응집성이 있는 아키텍처를 보증하지 않는다. 현존하는 기술을 사용하여 어떻게 구현할 것인지에 관한 사상 없이 응집력 있는 아키텍처를 정의하는 것은 아키텍처의 개념 모델과 객체 모델 사이에 불일치를 야기할 수 있다. 이러한 불일치로 인해 개발주기가 길어졌다. 따라서 어플리케이션을 구성하기 전에 소프트웨어 아키텍처가 정의되어야 한다[8]. 표준 아키텍처를 통해 COTS 컴포넌트간의 재사용성과 상호작용 효율을 높일 수 있다[10].

2.2 아키텍처 계층[11]

뷰(View) 계층은 어플리케이션의 사용자 인터페이스를 모두 포함하고 있다. 또한 사용자(Human Actor)에게만 보여진다. 그리고 어플리케이션이 변경되면서 가장 많이 변경되는 부분인 뷰 계층은 어플리케이션 간에는 대부분이 재사용할 수 없는 계층이다. 사용자 인터페이스에서 사용하는 버튼 등의 툴들은 재사용을 하지만, 행위를 가지고 있지 않고, 기능이 있다고 하더라도, 그 행위 또한 제한적이다.

사용자에게 어플리케이션의 상태 및 결과를 보여주거나, 사용자의 입력값을 어플리케이션에 적용하도록 도와준다. 뷰 계층은 표현 계층과 강한 결합 관계가 있다. 윈도우나 다이알로그 창 등의 뷰 계층은 프리젠테이션 계층의 프리젠테이션 콘트롤러와 관계가 있다. 뷰 계층의 이벤트와 뷰 컨트롤러의 이벤트들을 서로 교환하고, 처리한다.

어플리케이션 모델(Application Model) 계층은 뷰

계층과 비즈니스 도메인 계층과의 연결관리를 한다. 뷰 컨트롤러(View Controllers), 유즈케이스 컨트롤러(Use Case Controller), 모델 어댑터(Model Adapter)의 3부분으로 구분된다. 이 클래스에서는 어플리케이션의 대부분의 행위를 구현한다. 뷰 컨트롤러는 표현해야 할 독립적인 윈도우 창이나 대화상자 등을 관리한다. 유즈케이스 컨트롤러는 각각의 유즈케이스에 관한 시스템 오퍼레이션을 제공한다. 모델 어댑터는 도메인 계층 인터페이스를 뷰 계층 인터페이스로 연결하는 역할을 담당한다.

도메인(Domain) 계층은 어플리케이션 도메인의 비즈니스 객체를 포함한다. 이 계층의 대부분의 클래스들은 재사용성이 용이하다. 또한 관련 있는 비즈니스 프로세스의 관계를 구현하고, 관리한다. 도메인 객체는 어플리케이션에서 관심 있는 비즈니스 정보를 유지, 관리의 책임을 가지고 있다. 또한 다른 저장 계층과 같은 다른 계층과는 낮은 결합도를 가져야 한다. 대부분의 도메인 클래스는 뷰 계층의 JavaBeans 등에서 직접 사용하지 않는다.

영속성(Persistence) 계층은 도메인 객체의 정보를 특정한 형식의 저장 매체에 저장하는 역할을 한다. 물리적 저장 장치와는 결합도가 높지만, 도메인 계층과는 낮은 결합도를 유지해야 한다. 영속성 계층은 어플리케이션에서 프레임워크처럼 재사용될 가능성이 매우 높지만, 특정 어플리케이션에 종속적으로 설계한다면, 재사용될 가능성이 매우 희박하게 된다.

2.3 J2EE와 웹 서비스 기반 컴포넌트 조립

J2EE 클라이언트는 서버측의 J2EE 컴포넌트를 JNDI를 통해 사용한다. 클라이언트 어플리케이션은 JNDI를 통해 EJB 레퍼런스를 검색하고 EJB 클라이언트 프락시(Proxy)를 반환 받아 EJB 컴포넌트를 사용하였다. 모든 J2EE는 일반적으로 RMI를 사용하여 통신을 한다.

웹 서비스를 사용한 클라이언트는 RMI가 아니라 Simple Object Access Protocol(SOAP)을 사용하여 서버측과 통신하여 JNDI 서비스를 받는다. 즉, 클라이언트는 JNDI 웹 서비스에 SOAP을 이용하여 요청을 전달한다. JNDI 웹 서비스는 어플리케이션 서버의 JNDI로 검색하여 J2EE 프락시를 얻는다. JNDI 웹 서비스는 검색한 J2EE 프락시를 SOAP 기반의 리모트 레퍼런스를 클라이언트로 전달한다. 클라이언트 어플리케이션은 이렇게 J2EE 자원을 이용하여 메소드를 처리할 수 있다. 각각의 메소드는 SOAP을 통해 J2EE 프락시로 전달된다.

3. EJB 컴포넌트 설계 패턴

소규모의 비즈니스 컴포넌트는 특정 비즈니스 도메인에서 비즈니스 정보와 그들 간의 관련된 행위로 구성된 명사와 같은 비즈니스 엔티티들이며, 대규모의 비즈니스 컴포넌트는 특정 비즈니스 도메인에서 기능과 비즈니스 프로세스들을 제공한다[12]. EJB 아키텍처는 기본적으로 소규모의 컴포넌트로 구성 된다. EJB 규약에는 여러 EJB들을 이용하여 대규모의 컴포넌트로 구현 가능하다고 되어 있지만, 어떻게 대규모의 컴포넌트로 구성할 것인가에 관한 방법은 제시되어 있지 않다[13]. 이 장에서는 EJB를 사용하여 컴포넌트 크기에 대한 여러 기법들을 제시하고, 장단점을 비교 평가한다.

3.1 소규모 세션빈 컴포넌트 기법(FGS)

소규모의 세션빈 컴포넌트를 이용한 기법(FGS : Fine Grained SessionBean)은 그림 1과 같이 가장 초기에 사용되어진 EJB 컴포넌트 아키텍처다. 초창기에는 EJB 서버들이 빈들 간의 규약을 충분히 지원해 주지 못한다. 컴포넌트의 특징인 모듈화된 블랙박스 형식의 API 제공보다는 화이트 박스 형식으로 세션빈을 직접 사용하는 방법이다. 이 기법을 사용하기 위해서는 클라이언트가 필요한 정보나 비즈니스 업무를 한번에 수행하는 것이 아니라, 직접 해당 세션빈의 컴포넌트 인터페이스를 직접 사용하여 모두 가져오게 된다. 그러므로 클라이언트에서는 세션빈의 모든 관련성들과 그 흐름을 모두 구현해주어야 한다.

클라이언트에서 각 세션빈에 개별적으로 접근하게 되므로 모든 세션빈과 직접 연관관계가 있어서, 클라이언트와 세션빈 사이에 결합도가 높아진다. 또한 세션빈을 소규모로 컴포넌트화함으로서 거의 빈 단위의 수준으로 패키지 해주어야 하며, 세션빈 간의 연관관계를 모두 개발자가 구현해주어야 하므로 엔티티빈처럼 EJB 2.0에 추가된 Container Managed Relationship(CMR)을 사용할 수 없어서, 확장성과 이식성이 떨어진다[15].

또한, 클라이언트 패키지에서는 비즈니스 도메인 조립에 필요한 모든 세션빈의 홈과 컴포넌트 인터페이스를 패키지 해야 하며, 해당 세션빈들의 컴포넌트 규약을 모두 인지해야 하는 부담을 구현자가 갖게 된다[15]. 이

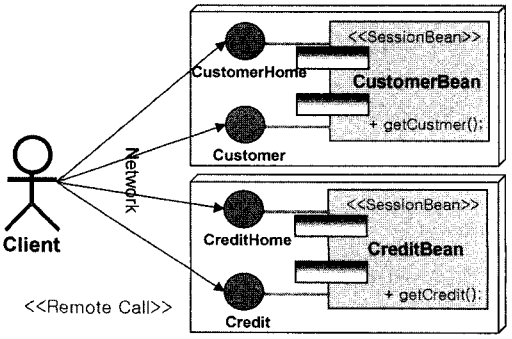

그림 1 세션빈을 이용한 소규모 컴포넌트 용례

FGS 기법은 작고 간단한 시스템 구성에 적합하며, 클라이언트 구현자가 세션빈의 내부 구현 및 테이블에 관한 정보를 모두 잘 아는 경우에 적합하며, 엔티티빈을 지원하지 않는 EJB 컨테이너에도 운영할 수 있다.

3.2 소규모 엔티티빈 컴포넌트 기법(FGE)

소규모의 엔티티빈을 이용한 컴포넌트 기법(FGE : Fine Grained EntityBean)은 FGS 기법에서와 같이 덜 진화된 초기의 컴포넌트 아키텍처다. 컴포넌트의 특징인 모듈화된 블랙박스 형식의 API 제공보다는 화이트박스 형식으로 엔티티빈을 직접 사용하는 방법이다. 그림 2와 같이 FGE 기법을 사용하기 위해서는 FGS 기법과 같이 비즈니스를 수행하기 위해 여러 번 필요한 정보를 얻어야 하고, 각 엔티티빈을 사용하기 위한 워크플로우도 구현하여야 한다. 그러므로 클라이언트 개발자는 엔티티빈의 모든 관련성들과 그 흐름을 모두 알아야 한다.

클라이언트에서 각 엔티티빈이 개별적으로 접근하게 되므로 엔티티빈과 직접 연관관계가 있어서, 클라이언트와 엔티티빈 사이에 결합도가 높아진다. 또한 엔티티빈을 소규모로 패키지해서 배치(Deploy)하므로, 현재 나와 있는 EJB 컨테이너에서는 다른 컴포넌트를 참조하기 위해, 다른 패키지의 추상 스키마 네임을 사용하여 빈을 식별할 수 없다. 따라서, 엔티티빈들 간의 연관관계를 EJB 2.0에 추가된 CMR을 사용할 수 없다.

이 FGE 기법은 작은 시스템에 적합하다. 하나의 트랜잭션을 처리할 때 테이블 간에 조인 등의 릴레이션이 많지 않고, 복잡하지 않은 시스템에 적합하다. 하지만, 빈번한 조회성 업무에서는 findXXX() 메소드를 사용할 때 불필요하게 엔티티빈이 컨테이너에 생성되므로, 서버에 부담을 가져올 수 있다. 클라이언트에서는 비즈니스 도메인을 수행하기 위해 필요한 여러 개의 모든 엔티티빈의 홈과 컴포넌트 인터페이스를 같이 포함하고 있어야 하며, 또한 Primary Key 클래스도 가지고 있어야 한다.

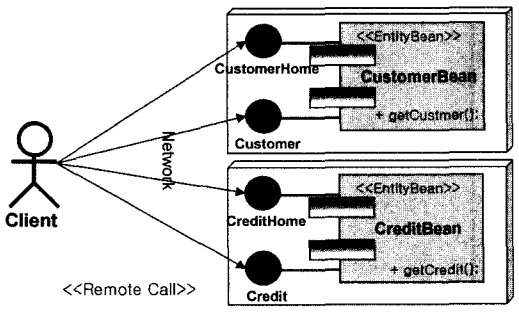

그림 2 엔티티빈을 이용한 소규모 컴포넌트 용례

3.3 대규모 세션빈 컴포넌트 기법(CGS)

세션빈을 사용한 대규모 컴포넌트 기법(CGS : Coarse-Grained SessionBean)은 엔티티빈을 사용하지 않고, 세션빈만을 사용한 기법이다. 세션빈에서 저장된 정보를 다른 세션빈을 이용하여 구현하는 것이다. 그림 3처럼 다른 컴포넌트들에게 메시지를 전달해주는 역할을 하는 컨트롤러는 퍼사드 패턴(Façade Pattern)을 사용한다. 세션 퍼사드 패턴은 클라이언트와 서버 사이의 의존성을 낮추게 된다.

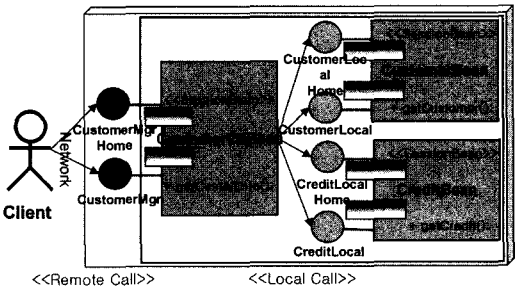

그림 3 세션빈을 이용한 대규모 컴포넌트 용례

시스템 내의 비즈니스 로직들을 적절하게 분리하여 구성할 수 있는지를 보이며, 하나의 유즈케이스에서 필요한 메시지 호출과 순서를 하나의 트랜잭션 단위로 묶어서 관리할 수 있도록 한다. 그러므로 클라이언트와 서버 사이의 메시지 교환 회수를 줄여, 네트워크를 통한 리모트 메시지가 줄여 효율적이고, 병목 현상을 줄일 수 있게 하는 패턴이다[17]. 비즈니스 워크플로우를 관리하는 컨트롤러 역할의 세션빈을 통해 관련 있는 세션빈들을 하나의 컴포넌트로 구성하여 컨트롤러 세션빈의 비즈니스 메소드를 컴포넌트의 API로 구성할 수 있다. 클라이언트에서는 이 컨트롤러 세션빈의 API를 통해 내부 세션빈을 사용할 수 있다.

CGS 기법은 내부 데이타베이스 관련 로직을 엔티티빈을 사용하지 않고, 세션빈만을 사용하는 것은 비효율적이다. EJB의 장점인 데이타베이스 필드를 추상 스키마를 사용하여, 소스 변경 없이 확장할 수 있는 장점을 사용 할 수 없다. 또한 CMR의 기능 또한 사용할 수 없다. 다른 어플리케이션에 적용된다면, 데이타베이스 관련 로직이 많아서, 재사용성이 떨어지게 된다. 그러므로 데이타베이스 구조 변경이 적은 컴포넌트를 조립하는 기법에만 유용하게 된다.

이 CGS 기법은 FGS 기법을 사용한 시스템을 확장하기에 적합하다. 클라이언트 구현자는 FGS 기법보다는 퍼사드 패턴을 사용하기 때문에 클라이언트와 각각의 세션빈과의 의존성은 적어지며, 클라이언트 개발자가 세

션빈의 내부 구조 및 테이블에 관한 정보를 모두 알아야하는 부담을 줄일 수 있다. FGS처럼 엔티티빈을 지원하지 않는 EJB 컨테이너에도 운영할 수 있으며, 테이블 변경이 거의 없는 조회성 업무에 유리하다.

컴포넌트에서는 하나의 API를 처리할 때 테이블 간에 조인 등의 복잡한 릴레이션이 많아 엔티티빈으로 구현에 적합하지 않을 때 사용하고, SQL 쿼리문으로 복잡한 릴레이션을 처리하는데 적합하다. 하지만, CGS 기법은 테이블이 안정화되어야 유리하며, 거대한 시스템에서는 유지보수하기가 복잡해진다.

3.4 대규모 엔티티빈 컴포넌트 기법(CGE)

엔티티빈을 이용하여 대규모로 구성된 컴포넌트 기법(CGE : Coarse Grained EntityBean)은 엔티티빈에서 데이타베이스 정보를 소규모로 구성된 엔티티빈을 이용하여 하나의 엔티티빈으로 구성하는 것이다. 대규모로 구성된 세션빈 컴포넌트 기법처럼 집합관계의 클래스들을 세션빈으로 매핑한 모양이다.

그림 4에서처럼 커멘드(Command) 역할의 엔티티빈을 사용하여 구성하게 된다. EJB 커멘드 패턴(Command Pattern)은 하나의 비즈니스 로직들을 일반 객체나 EJB를 사용하여 간단한 명령 단위로 묶는 것이다[17]. EJB 커멘드 패턴의 기법을 엔티티빈의 특정 메소드에 부분적으로 사용하여, 장점은 다른 엔티티빈의 명령 처리를 내부 엔티티빈을 클라이언트를 완벽하게 분리시킴으로서, 데이타베이스의 스키마가 변하더라도, 내부 엔티티빈만이 수정됨으로 클라이언트와 구현된 컴포넌트 로직의 결합도를 낮게 하며, 한 번의 네트워크 호출을 통한 명령이 트랜잭션 내에서 일련의 업무처리들을 실행하도록 지원할 수도 있다. 또한 이 오퍼레이션 명령의 단위로 성공 여부를 트랜잭션 로그로 남길 수 있다.

클라이언트에서는 명령 역할을 하는 엔티티빈을 통해 필요한 정보를 얻는다. 이 커멘드 속성의 엔티티빈 내부에서 다른 엔티티빈의 비즈니스 메소드나 각각의 엔티티빈에 getXXX(), setXXX() 메소드를 통해 필요한 정보를 검색 후, 한번에 명령을 처리한다. 또한 변경된 중

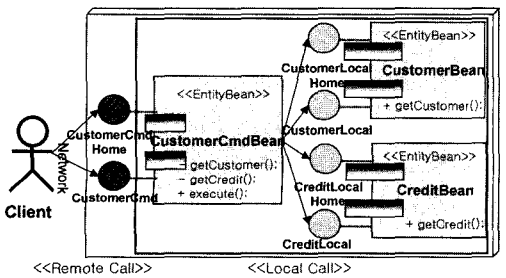

그림 4 엔티티빈을 이용한 대규모 컴포넌트 용례

요한 값은 커멘드에 값을 저장한 후 클라이언트에서 직접 참조를 하므로, 불필요한 엔티티빈의 호출의 회수를 줄일 수 있다. 그러므로 데이타베이스 접근이 필요한 메소드 호출시에만 ejbLoad(), ejbStore()를 호출하는 레이지 로딩(Lazy Loading)을 지원하지 않는 EJB 서버에서도 적용 가능하다.

커멘드 엔티티빈과 내부 엔티티빈 간에 추상 스키마를 사용한 CMR 및 EJB QL을 사용할 수 있으므로, 이식성과 유지보수성을 통해 재사용성을 향상시킬 수 있도록 설계할 수 있다. 클라이언트에서는 커멘드 엔티티빈과는 홈 인터페이스와 컴포넌트 인터페이스에 리모트로 접근해야 한다. 커멘드 엔티티빈은 내부 엔티티빈의 홈 인터페이스와 컴포넌트 인터페이스에 로컬로 접근할 수 있다.

CGE 기법은 추상 스키마 타입 사용이 가능하며, 이식성이 높아, 유지보수하기에 복잡한 대형 시스템에 적합하다. 하나의 트랜잭션을 처리할 때 테이블간에 조인 등의 릴레이션을 커멘드 엔티티빈에서 한번에 처리할 수 있으며, 엔티티빈을 사용하므로 CMR 기법을 사용하여, 유지보수성, 확장성 및 이식성이 좋으며, 클라이언트에서는 비즈니스 도메인을 수행하기 위한 모든 엔티티빈의 홈과 컴포넌트 인터페이스와 Primary Key 클래스를 같이 포함할 필요 없이, 커멘드 엔티티빈의 홈과 컴포넌트 인터페이스만 가지고 있으면 된다. 그러므로 GFE 기법보다 내부 엔티티빈의 API 수정시 클라이언트를 변경할 필요가 없어진다. 역시 FGE 기법처럼 게시판 등의 조회 업무일 경우에는 클라이언트가 검색을 원치 않는 엔티티빈까지 EJB 컨테이너에 로드되어 EJB 컨테이너의 부담이 커진다.

3.5 하이브리드 방식의 대규모 컴포넌트 기법(HySE)

세션빈을 통해 대규모의 엔티티빈을 이용한 컴포넌트 기법(HySE : Hybrid SessionBean and EntityBean)이다. 그림 5의 HySE는 세션빈을 이용한 CGS 기법과, 엔티티빈을 이용한 CGE 기법의 장점을 살린 기법이다.

CGS기법에서 처럼 컨트롤러 역할을 하는 EJB 뿐만 아니라, CGE기법의 커멘드 역할까지 포함하여 관리 EJB를 사용하여 관리하게 된다. 세션 퍼사드 패턴을 이용하여, 클라이언트에서 여러 번 호출할 메소드를 내부 워크 플로우를 한번으로 단일화시킴으로서 사용하기 쉽고, 네트워크 부하를 줄일 수 있으며, 커멘드 패턴을 사용하므로서 호출된 엔티티빈의 정보를 계속 사용할 수 있으므로, 엔티티빈의 호출을 줄일 수 있다. 또한 작업 단위 처리 결과를 트랜잭션 로그로 남길 수 있다.

그림 5의 엔티티빈의 비즈니스 메소드가 호출될 때 엔티티빈의 ejbLoad()가 수행이 되고, ejbStore()가 호출되기 때문에 데이타베이스의 변경이 아니더라도, 불필

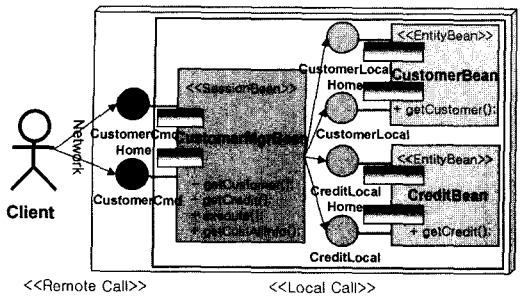

그림 5 세션빈을 통해 엔티티빈을 랩핑한 대규모 컴포넌트 용례

요한 호출과 저장이 반복되게 된다. 그러므로 엔티티빈의 목적인 데이타베이스 값을 캐쉬해 놓고 사용하므로서 데이타베이스 연결(*Connection*)을 오히려 더 자주 발생시키게 된다. 그러므로 세션빈에서 상태를 관리하여, 데이타베이스 검색 등의 필요한 로직 수행 때만 엔티티빈의 레이지 로드 패턴을 사용하여 비즈니스 메소드를 분석하여, 데이타베이스 쿼리가 필요할 때만 수행하도록 통제할 수 있다[17]. 클라이언트에서는 컨트롤러 역할을 하고, 공유해서 사용이 가능한 세션빈을 사용하여 정보를 얻는다.

이 컨트롤러 세션빈의 오퍼레이션은 컴포넌트 내부의 세션빈, 엔티티빈을 사용하여, 여러 메시지 흐름을 처리하고, 필요한 정보를 한번에 반환한다. 대부분은 이 메소드는 유즈케이스와 일치할 것이다. 그러므로 클라이언트와 내부의 세션빈, 엔티티빈과의 결합도를 낮출 수 있다.

이 HySE 기법은 CGS, CGE 기법의 장점을 모두 포함하고 있으므로, 유지보수하기에 복잡한 소형, 대형 시스템 모두에 적합하다. 하나의 트랜잭션을 처리할 때 업무의 성격에 따라 테이블간에 조인 등의 릴레이션을 컴포넌트의 인터페이스 역할을 하는 관리용 세션빈(Mgr-Bean)에서 BMP 방식으로 구현할 수 있으며, 엔티티빈을 사용한 CMR 기법을 사용하여, 유지보수성, 확장성 및 이식성이 좋으며, 클라이언트에서는 비즈니스 도메인을 수행하기 위한 모든 엔티티빈의 홈과 컴포넌트 인터페이스와 *Primary Key* 클래스를 같이 포함할 필요없이, 관리용 세션빈 의 홈과 컴포넌트 인터페이스만 가지고 있으면 된다.

CGE 기법에서는 엔티티빈들 간의 복잡한 쿼리를 수행하기 위해 EJB 컨테이너의 부담이 있었지만, HySE 기법의 관리용 세션빈은 CGS 기법에서 SQL을 이용한 단순 조회 업무나, 복잡한 조인 등을 구현 할 수 있다. 그러므로 FGE, FGS 기법보다 내부 엔티티빈의 API 수정시 클라이언트의 변경이 필요 없어지며, CGS 기법처럼 관리용 세션빈에서 불필요한 계층을 건너서 직접 데이타베이스에 접근하는 패스트 레인 패턴을 적용하여, 게시판 등의 조회 업무에도 적합하다.

하지만, 컴포넌트의 인터페이스 역할을 하는 관리용 세션빈에 워크플로우를 위한 로직이 아닌, 비즈니스 로직이 포함된다면, 엔티티빈과 세션빈 사이에 로직들이 나뉘어지므로 객체지향의 기본 사상인 캡슐화에 문제가 생길 수 있다. 데이타와 관련된 로직은 세션빈이 아닌 엔티티빈으로 캡슐화시킨다.

4. 패턴 기반의 엔터프라이즈 아키텍처 설계 기법

3장에서 제시한 컴포넌트 구성 기법을 이용하여 이미 검증된 MVC 모델을 확장하여, EJB 아키텍처 설계 기법을 최적화하여, 효율적인 분산 엔터프라이즈 아키텍처를 설계하고, 금융시스템을 예로 보인다.

4.1 엔터프라이즈 추상화 아키텍처 설계 기법

그림 6(a)처럼 MVC 아키텍처를 엔터프라이즈 환경에 맞게 세분화하면, 뷰는 사용자 계층, 접근 계층으로 나누어지고, 컨트롤러는 작업흐름 계층과 비즈니스 도메인 계층으로 나눌 수 있다. 마지막으로 모델은 데이타 접근 계층과 데이타베이스 계층으로 나눌 수 있다[16].

사용자 계층은 모델의 데이타를 사용자에게 표현하고, 액터인 사용자의 이벤트를 받는 장치 역할을 한다. 이 장치는 인트라넷, 인터넷 기반 등의 터미널 시스템, 유선 시스템, 무선 시스템 등의 다양한 여러 기반으로 사용된다. 이 계층은 데이타의 정보를 사용자에게 보여주고, 사용자의 데이타 변경의 의도를 시스템에 전달한다. 예를 들어, 은행 자동화 기기, 판매 및 마케팅, 상담 서비스 등을 위한 사용자 화면을 지원하는 장치이다.

접근 계층은 사용자 계층과 시스템과의 연결 접점이다. 모델로부터 데이타를 받고, 데이타를 어떻게 표시할지를 정의한다. 예를 들어, Point of Sale(POS) 단말기, Kiosk, 자동 현금 입출금기, 현금 인출기, 유/무선 인터넷 시스템, 인트라넷이나 인터넷을 통한 전자 금융 서비스, 모바일 컴퓨팅 서비스 등에 의해서 데이타를 받을 수 있도록 네트워크를 연결한다.

작업 흐름 관리 계층은 사용자에게 지원할 작업의 단위이며, 즉 트랜잭션의 단위이다. 사용자의 요구 사항을 뷰를 통해 전달받아 요구 사항을 분석하고, 수행해야 할 기능들을 정의한다. 이러한 기능들의 처리 순서를 제어하는 역할을 한다.

비즈니스 도메인 계층은 비즈니스 로직을 가지고 있으며, 객체, 컴포넌트 단위로 구성된다. 각종 비즈니스 시스템을 묶어 비즈니스 지원 어플리케이션을 만들며, 뷰로부터 사용자의 요구 사항을 받아 규칙에 의해 모델을 변경시킨다. 이것은 모델의 품질 및 일관성 유지를

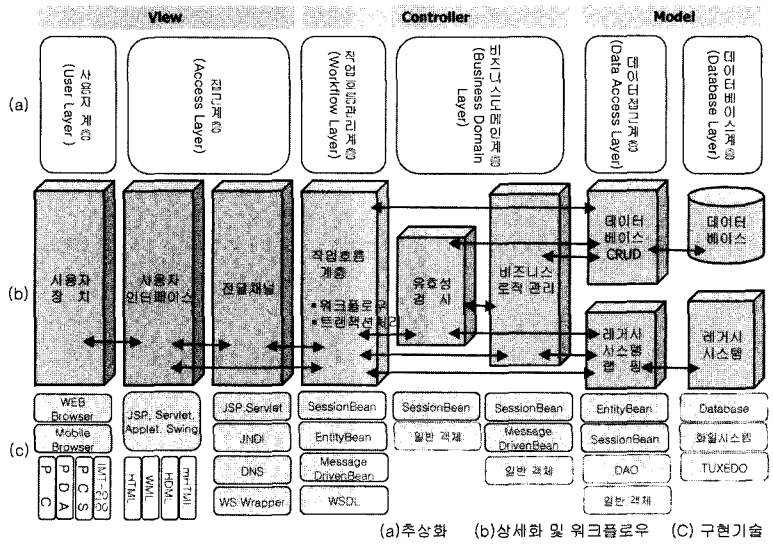

그림 6 엔터프라이즈 소프트웨어 아키텍처

위한 일련의 규칙을 갖는다. 뿐만 아니라, 모든 접근 채널에서 재사용할 수 있다.

데이타 접근 계층은 어플리케이션 데이타와 비즈니스 규칙을 지원한다. 물리적 데이타베이스에 객체지향적으로 관련된 데이타를 캡슐화하여 지원하며, 데이타를 수정하고, 데이타 처리를 수행하고 변경되는 데이타를 감시한다. 또한 뷰를 위해 모델의 상태를 조회할 수 있는 기능을 제공한다. 데이타베이스 계층은 물리적 데이타가 있는 계층이다. 어플리케이션의 상태 및 정보를 저장하고 레거시 시스템(Legacy System)을 지원 가능하도록 한다.

4.2 엔터프라이즈 상세화 아키텍처 설계

사용자 장치는 그림 6(b)처럼 사용자와 시스템 간의 대화를 위한 매체이다. 이 계층은 유무선 인터넷을 이용한 PDA에도 서비스 지원하기 위해 웹 브라우저를 지원하고 씬 클라이언트(Thin Client)와 확장성을 지향한다. 예를 들어, 무선 인터넷에서는 Mobile Phone, IMT 2000 Phone, Smart Phone, PDA를 모두 지원한다.

접근계층은 접근 채널(Access Channel)과 전달 채널(Delivery Channel)로 세분화된다. 접근계층은 사용자 인터페이스의 종류에 따라 HTML, Applet, Swing, WML, WML Script, HDML, mHTML기반을 사용하여 구현한다. 전달 채널은 IP 주소를 이용하여 작업 흐름 관리 계층으로 전달하거나, 네이밍/디렉토리(Naming & Directory) 서비스 등을 사용하여 네트워크 상에 존재하는 모든 리소스들에 대해서 물리적인 위치를 몰라도 디렉토리와 자원 이름을 통하여 작업 흐름 관리 계층으로 접근 가능하게 한다[15].

작업 흐름 관리 계층에서 컨트롤러는 JNDI 서비스를 이용하여 해당 컨트롤러를 지원하고, 사용자가 요청한 작업을 일련의 순서대로 작업을 지원한다. 업무의 성격에 따라 중요한 작업은 트랜잭션 상태에서 처리할 수 있다. 그러나, 데이타의 참조 등의 중요도가 낮은 작업들은 이 계층에서 트랜잭션을 처리하지 않을 수 있도록 조절 가능하다.

표 1과 같이 작업흐름 계층에서는 일반적으로 HySE 기법을 사용하여 비즈니스 도메인 계층의 컴포넌트와 조립된다. 업무 흐름을 제어하는 계층이므로 데이타베이스를 사용하는 HySE기법은 필요 없을 것 같지만, 이 업무의 단위가 트랜잭션 단위이므로, 트랜잭션 상태를 관리하거나 예외처리 발생시 처리 결과를 저장하기 위한 업무가 필요하게 된다.

표 1 계층별 제안된 기법

선호 : ○, 가능 : △

기법 / 계층	FGS	FGE	CGS	CGE	HySE
작업흐름관리 계층	△		△		○
유효성검사 계층	○		○		
비즈니스 로직관리 계층	△		△		○
DB CRUD 계층		○		○	
레거시 랩핑 계층	○		○		○

작업흐름 관리 계층에 작업 단위군으로 비즈니스를 처리 할 수 있도록 하여, 네트워크의 부하를 줄일 수 있다. 간단한 작업흐름 구조를 가지고 있는 경우에는 FGS

나 CGS로도 가능하다. 또한 단순 조회성 업무등은 HySE 기법의 패스트 레인 패턴을 이용하여 데이타베이스 접근 계층 컴포넌트나 데이타베이스와 직접 통신 할 수 있다.

비즈니스 도메인 계층은 사용자 요청을 해결하기 위해 협력해야 하는 일련의 상하구조의 클래스들을 구성한다. 유효성 검사 계층은 FGS기법을 사용하여 입력받은 데이타의 유효성을 체크한다. FGS기법을 사용하는 이유는 대부분의 유효성 검사는 여러 컴포넌트간의 통신을 통한 작업이 아닌, 하나의 컴포넌트에서 처리가 가능하다. 그러므로 유효성 검증에서 실패할 경우 비즈니스 로직까지 프로세스를 넘기지 않아서 효율적이고, 예상치 못한 입력에 대한 에러를 예방할 수 있다.

비즈니스 로직 관리 계층은 다른 비즈니스 로직 계층 및 데이타 처리 계층에 접근하는 컴포넌트이다. HySE 기법을 사용하여 각종 비즈니스 시스템을 묶어 비즈니스 지원 어플리케이션을 만들고, 비즈니스 수행 중 필요한 정보는 데이타베이스나 레거시 시스템으로부터 정보를 얻어 업무를 수행 후 데이타베이스에 모델을 저장한다. 간단한 비즈니스 업무의 경우에는 FGS 또는 CGS 기법을 사용하여 비즈니스 로직을 처리하기 위해, 하나의 EJB 또는 여러 개의 EJB를 통해 처리할 수 있다.

데이타접근 계층은 데이타베이스 및 레거시 시스템에 저장된 데이타를 접근 가능하게 한다. DB CRUD (Create, Read, Update, Delete) 계층은 FGE, CGE기법을 사용하여 데이타베이스에 데이타를 생성, 수정, 삭제 및 검색을 수행하고, 어플리케이션의 데이타를 보호한다. 레거시 랩핑(Legacy Wrapping)은 기존 시스템 어플리케이션의 변경이 없이 이루어지는 것으로서, 기존의 시스템의 데이타를 이용하기 위하여 사용한다. 곧, 레거시 시스템의 API를 이용하여 처리하는 계층이다. Jolt API등을 사용하여 TUXEDO등과 연동하여 사용할 수 있다. 데이타베이스 계층은 시스템에서 관리하여야 물리적 데이타베이스이다. 레거시 시스템은 기존의 데이타를 가지고 있고, 그 데이타를 관리하기 위해 호스트상에 캡슐화되어 있다.

4.3 엔터프라이즈 아키텍처 구현 기법

그림 6(c)처럼 사용자 장치는 웹 브라우저나 무선 인터넷 기반의 장치를 이용하여 시스템과 대화를 할 수 있다. 무선 인터넷에서는 Mobile Phone, IMT 2000 Phone, Smart Phone, PDA등을 지원한다. 사용자 인터페이스 기술에서 사용자 인터페이스에 따라 HTML, Applet, Swing, WML, WML Script, HDML, mHTML기반으로 JSP, Servlet, Applet, Swing을 이용하여 구현한다.

전달 채널 기술에서는 JNDI를 이용하여 구현한다. Naming & Directory 서비스는 네트워크 상에 존재하는 모든 리소스들에 대해서 물리적인 위치를 몰라도 디렉토리와 이름 구조로 접근할 수 있다. 각 Naming & Directory 서비스 벤더마다 JNDI인터페이스를 위한 드라이버를 제공해야 한다.

또한 웹 서비스를 사용하기 위하여 JAXM메시지와 JAX RPC 서비스를 사용하는데 SOAP기반으로 웹 서비스를 호출 할 수 있도록 한다. 이러한 웹 서비스를 사용하기 위한 랩퍼로 사용할 수 있다. 이 계층에서 웹 서비스 정의 언어(WSDL: Web Services Description Language)인 WSDL화일의 위치 정보가 필요하게 된다. WSDL을 이용하여 인터넷 상의 다양한 언어로 개발한 컴포넌트들의 조립이 가능하게 되었다. 인터페이스를 WSDL을 이용하여 배포한다면, 클라이언트는 구현상의 세부내역을 알 필요 없이 호출이 가능하다[14].

작업흐름 관리 계층을 기술하기 위해서는 EJB에서 세션 빈을 이용하여 구현한다. 세션 빈에서 다른 빈의 메소드나 일반 객체의 메소드를 사용할 수 있다. 이 작업흐름 관리계층의 주요 역할은 작업흐름과 트랜잭션 관리를 담당한다.

작업흐름 관리 계층은 비즈니스 수행의 순서를 정의하고, 트랜잭션은 컨테이너 관리에 의한 트랜잭션(CMT :Container Managed Transaction), 빈 관리에 의한 트랜잭션(BMT:Bean Managed Transaction), 트랙잭션 처리가 없는 경우와 트랜잭션 클래스(Transaction Class)를 별도로 구현하는 방법으로 나누어질 수 있다. 작업흐름 관리 계층은 웹 서비스를 사용하여 서비스 제공이 가능하다. 웹 서비스로 사용될 계층을 사용하는 클라이언트에서는 웹 서비스로 사용될 EJB 객체를 사용하기 위해서는 WSDL파일 정보를 사용하여 작업흐름 계층의 EJB 컴포넌트 인터페이스를 사용할 수 있다.

컨테이너 관리 트랜잭션의 경우에는 NotSupport, Required, Supports, RequiresNew, Mandatory, Never 등의 속성을 설정할 수 있으며, 트랜잭션 독립성 속성인 READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE을 디플로이먼트 디스크립터에 설정할 수 있다. 그러므로 작업흐름 계층의 구현과, 다음의 계층의 구현에 영향을 주지 않고 변경, 유지보수할 수 있다. 배치성 업무 등의 사용자의 요청에 바로 응답하는 동기식이 아닌 사용자의 요청에 의해, 결과값을 받을 때까지 기다리지 않고, Java Message Service(JMS)를 이용하여, 비동기적인 수행을 할 수 있다[13].

비즈니스 로직 계층을 구현하기 위해 비즈니스 로직 및 데이타 접근 로직을 구현한다. 비즈니스 로직을 수행하면서 특정한 속성 값을 보관하며 작업 수행을 해야하

는 경우에는 상태유지 세션 빈 속성으로 구현하고, 이러한 속성 값 보관이 필요 없는 비즈니스 로직인 경우에는 비상태 세션 빈으로 구현한다. 비상태 세션 빈인 경우에는 클라이언트의 상태 정보를 유지하지 않고, 하나의 인스턴스는 여러 명의 사용자에 의해서 호출될 수 있다. 한 클라이언트와의 세션유지는 하나의 메소드 동안에만 유지되며, 수동(Passivate) 상태를 갖지 않는다.

상태유지 세션빈인 경우에는 클라이언트의 상태 정보를 계속 유지할 수 있고, 하나의 인스턴스는 하나의 클라이언트에 매핑 된다. 각각의 클라이언트는 각각의 빈 인스턴스와 매핑 되며, 수동적인 상태를 갖는다. 이 비즈니스 로직은 세션빈에서 엔티티빈이나 레거시 랩핑을 이용하여 데이타를 이용할 수 있다. 데이타베이스 CRUD 계층 구현은 데이타베이스와 1:1 매핑이 되므로 엔티티빈으로 구현한다. 엔티티빈은 컨테이너 관리에 의한 저장과 빈 관리에 의한 저장으로 구별된다.

컨테이너 관리에 의한 저장은 빈이 가지는 속성과 영구적인 저장소 간의 동기화를 컨테이너가 직접 처리한다. 빈 개발자는 JDBC를 이용한 데이타베이스 처리 로직과 같은 영구적인 저장소와의 인터페이스 로직을 작성하지 않아도 된다. 빈 관리에 의한 저장은 빈이 가지는 속성과 영구적인 저장소 간의 동기화를 위한 코드를 개발자가 직접 작성해야 한다. 빈 개발자는 JDBC를 이용한 데이타베이스 처리 로직과 같은 영구적인 저장소와의 인터페이스 로직을 작성해야 한다. 빈 구현 클래스 내에 ejbStore(), ejbLoad() 등의 메소드에서 영구적인 저장소와의 처리 로직을 구현한다.

레거시 랩핑(Legacy Wrapping)하기 위해서는 기존 시스템의 데이타를 이용하기 위하여 세션 빈이나 일반 객체로 구현이 가능하다. 레거시 시스템의 API를 이용하여 처리하는 구현을 한다. 서로 동일 언어가 아닐 경우에는 레거시 랩핑 계층에서 JNI나 소켓으로 바이트 스트림을 통해 인터페이스 할 수 있다. 또는 Jolt API를 사용하여, TUXEDO등과 연동시킬 수 있다. 그러므로 레거시 시스템의 수정에 의해, 레거시 랩핑 계층에만 영향을 받으므로 전체 시스템에는 영향을 주지 않는다.

5. 계층간 워크플로우 설계 기법

5.1 계층간 상호관계 설계 기법

사용자 장치와 사용자 인터페이스와의 상호 관계에서는 웹 브라우저에서 사용자의 입력 사항이나 서비스 사항을 요청에 의해 접근 채널로 전달한다. 접근 채널인 JSP에서는 사용자에게 질의나, 사용자가 요구한 사항을 응답하기 위해서 HTML, WML 등의 형식으로 만들어서 사용자 인터페이스 쪽으로 전달한다. 상호관계는 그림 6(b)처럼 화살표로 표현했다.

사용자 인터페이스와 전달 채널 상호 관계에서는 접근 채널인 JSP에서는 전달 채널을 이용하여 컨트롤러를 사용하게 된다. JSP에서는 Naming & Directory 서비스의 JNDI 인터페이스를 이용하여 컨트롤러에 접근한다. 전달 채널과 컨트롤러 상호 관계에서는 전달 채널에서는 JNDI 인터페이스를 이용하고, 접근 채널에서 요구한 컨트롤러 계층의 컨트롤러 홈 인터페이스 객체를 반환한다.

접근 채널과 작업흐름관리 상호 관계에서는 접근 채널은 UI에서 요청한 서비스를 지원하기 위해서 입력 사항을 컨트롤러에게 전달한다. 컨트롤러는 접근 채널에서 요청한 서비스를 수행하고, 그 결과를 반환한다. 작업흐름 관리와 유효성 사이의 흐름 상호 관계에서는 서비스하기 위해 접근 채널로부터 받은 서비스 및 입력 값들이 유효한지를 검증한 후 작업흐름을 계속 진행한다. 작업흐름 관리와 비즈니스 로직 상호 관계에서는 컨트롤러에서 할 작업흐름을 비즈니스 로직을 이용하여 수행한다. 비즈니스 로직은 컨트롤러에서 요청한 서비스를 수행하고 그 결과 값을 컨트롤러에 반환한다.

작업흐름 관리와 데이타베이스 CRUD 상호 관계에서는 컨트롤러에서 시스템 성능향상을 위해서 단순 데이타베이스 조회 등의 단순 로직은 데이타베이스 CRUD 기능을 이용해서 직접 데이타베이스 정보를 얻을 수 있다. 데이타베이스 CRUD에서는 컨트롤러에서 요청한 데이타베이스 검색 결과 값을 반환한다. 작업흐름 관리와 레거시 랩핑 상호 관계에서는 컨트롤러에서 시스템 성능향상을 위해서 단순 레거시 시스템 조회 등의 단순 로직은 레거시 랩핑의 API를 이용해서 직접 레거시 데이타베이스 정보를 얻을 수 있다. 레거시 랩핑에서는 컨트롤러에서 요청한 레거시 데이타베이스 검색 결과 값을 반환한다.

비즈니스 로직과 데이타베이스 CRUD 상호 관계에서는 비즈니스 로직을 수행하면서 필요한 데이타를 데이타베이스 CRUD에게 요청한다. 비즈니스 로직과 레거시 랩핑 상호 관계에서는 비즈니스 로직을 수행하면서 필요한 레거시 데이타를 레거시 랩핑에게 요청한다. 그러므로 비즈니스 로직에서는 레거시 시스템의 구조를 알지 못해도 레거시 랩핑에 의해 레거시 시스템 제어가 가능하다.

데이타베이스 CRUD와 데이타베이스 상호 관계에서는 데이타베이스 CRUD에서 JDBC를 이용하여 물리적 데이타베이스에 값을 생성, 수정 및 삭제를 수행한다. 레거시 랩핑과 레거시 시스템 상호 관계에서는 레거시 랩핑에서 레거시 시스템의 API를 이용하여 레거시 데이타의 값을 생성, 수정, 삭제 및 조회를 수행한다.

5.2 트랜잭션의 제어 흐름 설계 기법

트랜잭션 처리를 위한 계층간 메시지 흐름인 그림 7을 보면, ④는 사용자로부터 받은 요구를 처리할 작업흐름을 호출한다. ⑤는 검색을 하기 전에 유효성 검사를 할 요건이 발생할 경우 유효성을 요청한다. 업무요건에 따라 ⑤~⑥은 반복적으로 수행된다. 이때 비즈니스 로직은 업무요건에 따라 다른 비즈니스 로직을 호출 가능하다.

⑥은 컨트롤러가 요청한 유효성 검증의 결과를 넘겨준다. 업무요건에 따라 ⑤~⑥은 반복적으로 수행된다. 만약 유효성 검사가 실패한다면 그 결과를 즉시 반환한다. ⑦은 업무의 단위당 트랜잭션의 특징인 ACID를 관리할 수 있다. 그러므로 그림 7의 사각형을 보면 트랜잭션 관리객체에게 트랜잭션 시작을 알리고, ⑪은 비즈니스 로직 처리 결과를 컨트롤러에게 넘겨준다. 업무요건에 따라 ⑧~⑪은 반복적으로 수행된다. ⑫,⑭는 트랜잭션 관리자에게 트랜잭션이 정상적인지 아닌지를 알려준다. 그러므로 트랜잭션의 작업 단위 처리 결과가 성공일 때만 ⑬과 같이 데이타베이스에 저장을 하게 된다.

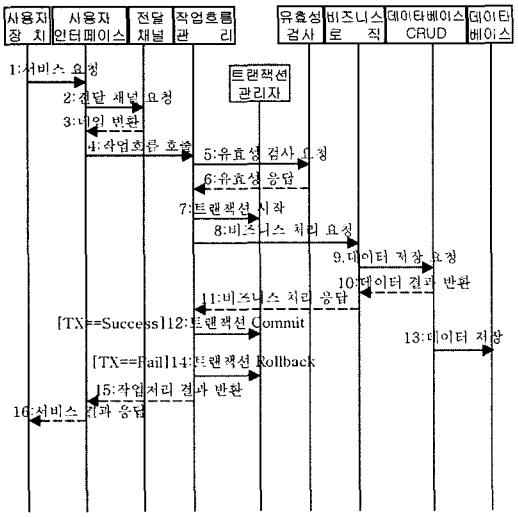

그림 7 트랜잭션의 제어 흐름

6. 평 가

본 논문에서 제시한 아키텍처를 기반으로 한 시중은행의 여신 업무 시스템에 적용하였다. 적용한 아키텍처는 확장성, 성능, 가용성, 유지보수성, 재사용성 등의 아키텍처의 품질 요소를 만족한다. 레거시 시스템을 이용하여 서비스하는 컴포넌트의 수정이 다른 컴포넌트에 미치는 영향을 최소화한다. 그러므로 개발 도메인의 외부 시스템이 변경되어도, 그에 따른 컴포넌트 변경이 줄어든다. 은행에서 다른 시스템과의 연동시, 효율적인 웹 서비스를 적용하여 연동이 수월하며, JMS 서비스 기반의 메시지 드리븐 빈을 사용할 수 있다. 본 논문에서 제안한 EJB기반의 컴포넌트 소프트웨어 아키텍처를 평가한다.

제시된 아키텍처의 모델은 이미 검증된 MVC모델을 확장하였으며, 각각의 컴포넌트의 역할을 계층화시켰다. 또한 관련연구에서 제시한 소프트웨어 아키텍처의 정의에 현재의 기술이 적용가능 해야 하는 부분을 만족시키기 위해 각 계층에 구현할 기반 기술과 설계 패턴의 제공 및 구현 기법까지 제공한다.

분산 컨트롤(Distribution Control)을 지원하는 제시된 아키텍처는 하나의 컴포넌트로 이루어지지 않고, 컴포넌트 간에 상호작용을 통해 운영되고 통제된다. 각각의 컴포넌트는 그들간의 조립에 따라 업무가 수행되며, 이들 요소들은 분산 시스템에서 동작이 가능하고, 분산 배치가 가능하여 시스템에 문제가 있을 때 다른 시스템으로의 우회 및 로드 밸런싱이 가능하다.

위치 투명성(Location Transaction)을 지원하는 제시된 아키텍처의 컴포넌트들은 JNDI의 이름과 서버의 위치만 알면, 물리적인 서버 위치와 컴포넌트에 상관하지 않고, 다른 EJB 컴포넌트 및 CORBA 시스템과 상호 운용이 가능하며, 웹 서비스 지원이 가능하다.

브라우져 기반의 사용자 인터페이스(Browser based User Interface)는 컴포넌트 시스템을 사용하는 사용자가 GUI를 통해 사용할 수 있도록 제공하는 플랫폼이다. 따라서 제시된 아키텍처는 PC 뿐만 아니라, 웹 브라우져가 내장된 PDA, Mobile Phone 등이 사용 가능하다.

제안된 아키텍처의 모델로 구축된 시스템을 관리 및 모니터링하기 위한 기법[19]을 적용하여 새로운 툴을 만들 수 있지만, 새롭게 제작하지 않고, 컴포넌트의 트랜잭션 수나 접근자의 수, 롤백 회수 등은 EJB 서버의 관리자 기능을 사용하여 작업흐름 계층에 사용된 EJB들을 모니터링하여 모니터링 툴 없이 관리 감독할 수 있다.

제안한 아키텍처는 낮은 결합도를 만족한다. 아키텍처의 계층은 EJB 시스템 성능상 조회 부분은 데이타베이스에 직접 접근하는 부분을 제외하고는 계층간의 결합도를 낮추어 다른 계층의 컴포넌트가 교체되더라도 그 영향을 줄여 유지보수 및 교체 기간을 줄일 수 있는 구조로 되어 있다. 또한 계층간에 다른 서버에서 운용되더라도, 서버간의 동기적 메시지에 대한 부담을 덜기 위해 메시지 이벤트를 사용하여 비동기적 메시지 처리도 가능하다.

본 논문에서 제안한 컴포넌트 기반의 소프트웨어 아키텍처를 아키텍처 스타일에 따른 변화 적응 지원 기법[19], Architecture Tradeoff Analysis Method (ATAM) 기법[21], 아키텍처에 소프트웨어 통합시의 문제[6]등에

표 2 타 아키텍처와 제안한 아키텍처 비교 평가

항 목	Meyer Tanuan	Chip Wilson	Paul D.Manuel	IsraelBen Shaul	제안한아키텍처
아키텍처 제안 동기	4+1 뷰, UML 표현법	Application 아키텍처	데이타, 프로세스	이밴트, 통합	패턴, 구체적
제시 플랫폼		EJB	J2EE,.Net	CORBA	EJB
추상 아키텍처 레이어	○(4개)	○(4개)	○(3개)	△(3개)	○(6개)
구체적 아키텍처 레이어	○				○
재사용성 및 대체성		○	○		○
계층간 낮은 결합도		○	○		○
네트워크 부하 및 성능			○	○	△
트랜잭션 관리					○
레거시 시스템 연동			△	○	△
비동기 메시지 제안			○	○	○
컴포넌트 조립 방법					○
컴포넌트 Granularity			○		○
데이타 추상 스키마					○
구현 패턴				○	○
현존 기술과 매핑		○	△		○
시스템 모니터링 기법					△

서 추출한 평가항목을 사용하고, 다른 논문에서 제시된 4개의 아키텍처[8,11,18,22]와 비교 평가한 결과는 표 2와 같다.

7. 결 론

본 논문에서는 개념적 수준의 아키텍처 계층뿐만 아니라, 제안된 아키텍처를 현재 제공되는 기술을 이용하여 구체적으로 제시하기 위해, 각 계층에 적용되는 EJB 구현 기법을 제시하였으며, EJB 규약에 제시되고 있지 않는 대규모의 컴포넌트 조립 기법을 제안하였다. 기본적으로 소규모로 구성된 EJB에서부터 대규모의 컴포넌트로의 장단점을 분석하였다. 이들 기법을 최적화하여 엔티티빈과 세션빈과의 상호작용, 네트워크 부하 성능, 배포 등에 효율적인 HySE기법을 제안하였다.

또한 기존에는 엔터프라이즈급 EJB기반에 적합한 아키텍처 제시가 없었으며, 이에 관한 상세한 지침이 없었다. 본 논문에서는 MVC 아키텍처 모델을 발전된 소프트웨어 아키텍처로 발전시키기 위해서 추상화시키고 구체화시키는 절차적 방법에 의해서 대규모 시스템 환경에 적합한 소프트웨어 아키텍처를 설계하는 기법을 보였다.

이를 바탕으로 여섯 계층으로 추상화한 아키텍처를 설계하였다. 이를 좀더 구체화 시켜 EJB 컴포넌트 환경에 적합한 아키텍처로 설계 가능하였으며, 대규모 시스템에 적합한 EJB 환경과 레거시 시스템과 랩핑하는 절차적 과정을 소개하였다. 검증을 위한 또 다른 기법으로서 효율적인 트랜잭션 관리 및 조립을 이용하여 컴포넌트 기반의 소프트웨어 아키텍처의 적합성 여부를 검증하였다. 제안된 아키텍처와 해당 계층에 적절한 EJB 컴포넌트 설계 기법으로 컴포넌트를 만들면, 웹 서비스와도 적합하며, 재사용성, 확장성, 이식성등의 컴포넌트의 특징에 도움이 될 것으로 기대된다.

참 고 문 헌

[1] Desmond F. D'Souza & Alan Cameron Wills, Objects, Components, and Frameworks with UML, The Catalysis Approach, Addison Wesley, 1999.

[2] Clemens Syzperski, Component Software, Addison-Wesley, 1999.

[3] D. Garlan and M.Shaw, "An Introduction to Software Architecture, Advances in Software Engineering and Knowledge Engineering," Vol. 1, World Scientific Publishing Company, 1993.

[4] Kazman, Software Architecture in Practice, Addison-Wesley, 1996.

[5] Mary Shaw & David Garlan, Software Architecture, Perspectives on an Emerging Discipline, Prentic Hall, 1996.

[6] D Garlan, R Allen and J Ockerbloom, "Architectural Mismatch : Why Reuse Is So Hard," 0740-7459 IEEE Software, 1994.

[7] Microsoft Corporation, "Microsoft Reference Architecture for Commerce," msdn, 2001. http://msdn.microsoft.com/library/default.asp?url=/library/en us/dnrac/html/mracv2_ch01.asp

[8] Israel Ben Shaul, "An Integrated Network Component Architecture," IEEE Software, September/October, 1988.

[9] D.C. Luckham and J.Vera, "An Event Based Architecture Definition Language," IEEE Trans. Software Eng., Vol 21, 1995, pp.717-734.

[10] Kim, S. D., "Lessons Learned from a Nationwide CBD Promotion Project," Journal of CACM,

October, 2002.
[11] Chip Wilson, "Application Architectures with Enterprise JavaBeans," Component Strategies, 1999.
[12] Heineman, Councill, Component Based Software Engineering, Addison-Wesley, 2001.
[13] Linda G. Demichiel, Enterprise JavaBeans™ Specification, Version 2.1, Sun Microsystems, 2002.
[14] Patrick Cauldwell, Rajesh Chawla, "Professional XML WEB Service," Wrox,, 2001.
[15] Ed Roman, Mastering Enterprise JavaBeans™ and the JavaTM2 Platform, Enterprise Edition, WILEY, 1999.
[16] Kruchten, P.B., "The 4+1 Views Model of Architecture," IEEE Software, Nov. 1995.
[17] Marinescu, F., EJB Design Patterns, John Wiley and Sons, Inc., 2002.
[18] Meyer Tanuan, "Software Architecture in the Business Software Domain," ACM, 1998.
[19] 나학청, 김수동, "Performance Monitoring Techniques for EJB Applications," 정보과학회논문지, 제30권, 제5,6호, 2003.
[20] Nancy & Debra, "What Makes One Software Architecture More Testable Than Another?," SIGSOFT 96 Workshop, 1996.
[21] P. Clements, Evaluating Software Architectures, Addison-Wesley, 2002.
[22] Paul D., Jarallah, "A data-centric design for n-tier architecture," Information Sciences, 2002.

민 현 기
1999년 건양대학교 전자계산학과 공학사 2001년 숭실대학교 컴퓨터학과 공학석사 2001년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 소프트웨어 아키텍처, 컴포넌트 개발 방법론(CBD), 제품-라인 공학(PLE), 모델 기반 아키텍처(MDA)

김 수 동
정보과학회논문지 : 소프트웨어 및 응용 제 30 권 제 1 호 참조