EJB 기반의 시스템 개발 절차에 대한 연구

이종국*, 김수동**
*숭실대 컴퓨터 학과
e-mail : jklee690@selab.soongsil.ac.kr
sdkim@computing.soongsil.ac.kr

A Study of Development Process on EJB

*Jong-Kook Lee, SooDong Kim
*Dept. of Computing, Soongsil University

요 약

EJB 기반의 시스템 개발은 RPC 기반의 시스템 개발이나 MTS, CORBA 기반의 시스템 개발에 비해 훨씬 효율적이다. 효율적인 시스템 개발을 위해서는 반드시 containter-managed Entity Bean 을 사용해야 하며 트랜잭션에 대한 처리를 분석 단계부터 반영해야 한다. 그러나 자료 사전와 클래스 다이아그램, 소스 사이의 일관성 유지가 개발시 많은 부담이 된다. 이를 해결하기 위해 소스 자동 생성 도구와 CASE tool 이 필요하다.

1. 서론

최근 EJB 기반의 IT 시스템 개발이 각광을 받고 있다. 웹 기반 서버 개발이 붐을 이룸에 따라 EJB 는 웹 어플리케이션 개발을 위한 최상의 기능을 가지고 있는 것으로 평가받고 있다. 우선 웹 어플리케이션 구현시 가장 큰 문제인 트랜잭션 처리와 보안, 부하분산 등의 문제를 EJB containter 자체에서 해결해 줌으로써 개발자들은 비즈니스 로직 개발에만 집중하면 된다. 또한 객체지향, 컴포넌트 소프트웨어를 지향함으로 개발 후의 유지 보수를 위해 적합하다.[4]

그러나 EJB 기반의 시스템 개발을 위해서는 정확한 방법론과 개발 절차가 준수되어야 한다. 본 논문에서는 EJB 기반의 시스템 개발 절차가 기존의 2-tier 나 3-tier 의 개발 절차와는 어떻게 다른지를 설명하고자 한다. 또한 EJB 와 공통적인 개념을 가진 MTS, CORBA 기반의 시스템 개발에 비해 어떤 장점을 가지고 있는지를 설명한다. 마지막으로 EJB 기반의 개발을 위해 반드시 필요한 절차와 문제점, 요구되어 지는 CASE 도구에 대해 설명한다.

2. 2-tier 에서의 개발 절차

2-tier 로 시스템 개발 시는 정확한 데이터베이스 설계가 개발 성공의 열쇠가 된다. 클라이언트는 데이터 서버와 밀접하게 연결되어 있기 때문에 데이터 베이스의 변경은 클라이언트에도 영향을 끼치게 된다. 클라이언트는 서버의 특정 데이터 베이스에 종속되기 때문에 유지 보수나 모듈화가 매우 어렵다.

3. 3-tier 에서의 개발 절차

3-tier 아키텍처는 2-tier 에서 발생하는 특정 데이터베이스 종속적인 문제와 유지 보수의 문제를 해결하기 위해 제안되었다.[1] 3-tier 에서는 데이터베이스와 별도로 비즈니스 로직을 다른 tier 로 분리함으로써 client 와 로직, 데이터가 분리되고 모듈화가 가능해 진다. 그러나 현실적으로는 비즈니스 로직에서 특정 데이터 베이스에 종속적인 SQL 문을 사용하기 때문에 모듈화는 완전하게 이루어지지는 않는다. 또한 2-tier 에서는 발생하지 않는 데이터 베이스와 비즈니스 로직, 클라이언트에 대한 많은 문서와 설계도를 관리해야 한다. 만일 비즈니스 로직에 대한 설계와 관리가 제대로 이루어지지 않으면 2-tier 보다 시스템이 더 복잡해 지고 유지보수가 어렵게 된다. 따라서 3-tier 개발시는 개발 절차를 준수하고 방법론을 정확히 지켜야 한다. 또한 데이터베이스나 구현 코드의 표준을 준수해야 변경시 유지 보수가 용이하게 된다. 일반적으로 3-tier 개발시는 자료 사전 작성, 데이터 베이스 설계, 업무 로직 개발, 클라이언트 개발로 진행된다. 이

때 단계마다의 표준을 준수하고 많은 문서를 관리하기 위해 CASE 툴을 사용하는 경우도 있다. 이 때는 자료 사전를 데이터베이스에 저장하고 이를 기반으로 한 데이터베이스 스키마도 데이터베이스에 저장하고 DDL 문과 비즈니스 로직을 구현하기 위한 기본적인 코드는 자동 생성한다. 필요한 문서 자체도 데이터베이스에 저장하여 필요할 때마다 출력하게 한다. 이런 CASE 도구는 3-tier 개발에 매우 효율적으로 사용된다.

4. 객체 기반의 3-tier 개발 절차

ENTERA 나 TEXIDO, TOPEND 와 같은 RPC 기반의 3-tier 개발은 CASE 도구와 방법론을 적용함으로써 효율적으로 개발될 수 있다. 그러나 **RPC** 기반의 3-tier 개발은 구조적 방법론을 적용할 수 밖에 없는 한계를 지닌다. 객체 기반의 3-tier 개발은 객체 중심으로 시스템을 개발함으로써 효율적으로 시스템을 모듈화할 수 있다. 더 나아가 비즈니스 로직을 컴포넌트화 하여 업무 노하우가 들어 있는 비즈니스 로직을 상품화 할 수도 있다.

객체기반의 3-tier 개발을 위한 기본 아키텍쳐로는 MTS 와 CORBA, EJB 등이 있다. 객체 중심이므로 방법론은 객체 지향 방법론을 적용해야 한다. 본 논문에서는 구현된 사례를 중심으로 EJB 기반의 개발 절차를 설명하고자 한다.

5. EJB 기반의 개발 절차

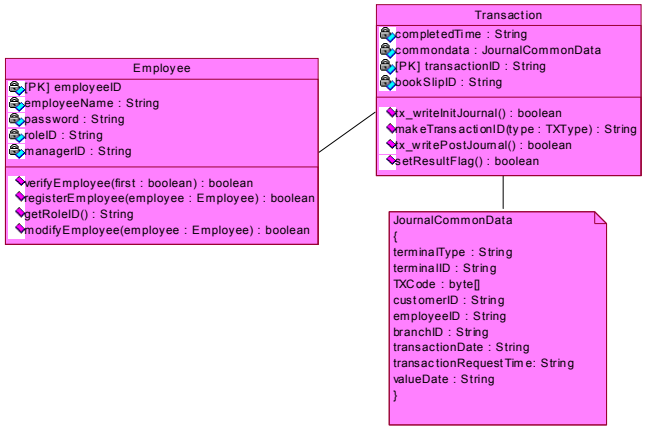
우선 3-tier 개발시와 동일하게 업무를 파악하고 분석, 설계를 표준화 하기 위해 자료 사전를 작성한다. 자료 사전는 inception 단계에서 작성되지만 설계가 끝날때까지 계속 관리되야 한다. 작성된 자료 사전를 기준으로 유즈 케이스 다이아그램, 클래스 다이아그램, s 시퀀스 다이아그램을 작성한다. 여기까지는 일반적인 객체지향 분석 단계와 동일하다. 설계시에는 EJB 의 특성을 고려해야 한다. 두가지 특성이 설계시에 고려되어야 한다.

1.Bean 의 종류를 결정함

EJB 의 객체는 stateless session Bean, stateful session Bean, Bean-managed Entity Bean, container managed Entity Bean 이 있다. 먼저 분석된 클래스를 session Bean 과 Entity Bean 으로 분리해야 한다. 분석된 클래스에서 메소드는 session Bean 으로, attribute 는 Entity Bean 으로 분리한다. 그 다음에는 Entity Bean 의 종류를 결정해야 한다. Bean-managed Entity Bean 을 사용하는 경우는 SQL 문을 데이터베이스에 직접 전달해야 한다. 이렇게 하면 RPC 기반의 3-tier 개발시 발생했던 특정 데이터베이스 종속적인 문제가 다시 발생하게 된다. 비즈니스 로직을 데이터 베이스 로직에서 완전히 분리하기 위해서 container-managed Entity Bean 을 사용한다. 이 경우 SQL 문으로 처리할 수 있는 로직을 비즈니스 로직에서 프로그램으로 구현해야 하기 때문에 performance 의 문제가 발생할 수 있다. 그러나 container-managed Entity Bean 을 사용하는 것은 객체지향적인 분석단계를 설계시 변경하지 않기 때문에 일관성이 있다. 즉 container-managed Entity Bean 을 사용함으로서 객체지향적인 분석과 관계형 데이터베이스 사이의 갭을 줄일 수 있다. 또한 객체지향 데이터베이스를 사용하지 않고 관계형 데이터베이스를 사용해도 분석, 설계의 일관성이 유지됨으로 객체지향 방법론을 적용할 때는 객체지향 데이터베이스를 사용해야 한다는 부담에서 벗어날 수 있다. 시스템을 개발해 본 결과 SQL 문을 직접 사용할때보다 퍼포먼스 문제가 크게 발생하지 않았다.

2. 트랜잭션의 종류를 결정함

CORBA 나 MTS 를 사용할 때는 SQL 문을 사용하여 특정 데이터베이스의 트랜잭션 처리를 직접 사용하기 때문에 분석, 설계 자체에서 트랜잭션의 특성을 반영할 수 없었다. 그러나 EJB 에서는 객체 단위로 트랜잭션을 처리하기 때문에 분석단계부터트랜잭션을 고려해야 한다. 트랜잭션의 시작과 끝을 분석 단계에서 명확하게 나타내기 위해 본인이 참여한 프로젝트에서는 트랜잭션 자체를 한 클래스로 두고 트랜잭션이 처리되는 메소드는 tx 를 이름에 붙여서 일관성을 유지하고자 했다. 또한 트랜잭션을 클라이언트에서 시작되게 할 것인지 서버에서 시작되게 할 것인지도 결정해야 한다.



[그림 1] 트랜잭션이 반영된 클래스 diagram

설계가 마무리 된 후에는 EJB 구현에 들어간다. 이 때 Entity Bean 은 데이터베이스 schema 를 가지고 있는 스크립트 파일에서 자동 생성되게 했다. Entity Bean 은 테이블과의 매핑이기 때문에 자동 소스 코드 생성 도구를 사용하면 코딩 작업을 줄일 수 있다. Session Bean 의 경우는 시스템적인 기능은 container 에서 처리하기 때문에 소스에는 비즈니스 로직만 나타나게 된다. Session Bean 은 Entity Bean 에서 데이터를 가져오고 가공하여 저장하는 기능이 대부분이므로 코딩 자체를 표준화하여 구현하면 작업량을 줄일 수 있다. 예를 들어 다음과 같이 코딩을 표준화 했다.

1. Entity Bean 의 home interface 를 얻는다.

2. Entity Bean 을 생성하거나 조건에 맞는 Entity Bean 을 가져온다.
3. 클라이언트에 전달할 데이터를 생성하기 위해 Entity Bean 의 attribute 의 값을 얻는다.
4. 전달할 데이터를 serializable 한 클래스에 담아서 전달한다.
5. Entity Bean 의 값에 새로운 값을 assign 하여 변경한다.
6. 1-6 까지의 절차에 에러가 발생할 수 있으므로 exception 처리한다.

본인이 진행했던 프로젝트에서는 거의 1-6 의 절차를 벗어나는 코딩은 없었다. 코딩이 효율적으로 이루어졌기 때문에 구현시에는 기술적인 문제에 부딪친 적은 없었고 남은 시간은 클라이언트를 구현하는데 집중할 수 있었다.

6. MTS, CORBA 과 개발 효율성 비교

이상의 개발 절차를 MTS 와 CORBA 기반의 시스템 개발 절차와 비교해 보자. EJB 가 MTS, CORBA 와 다른 점은 특정 데이터베이스에 종속적인 SQL 문을 사용하지 않는다는 것이다.

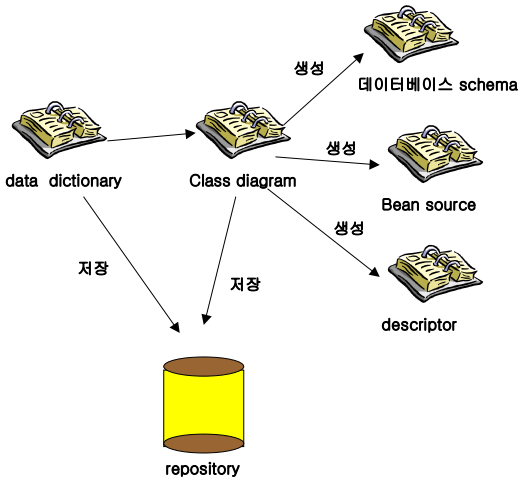
MTS 를 기반으로 개발할 때는 자료 사전, 설계 자료, 소스 코드를 관리해야 하며, 추가적으로 컴포넌트마다 ID 를 관리해야 한다.[3] MTS 의 경우는 CORBA 나 EJB 의 경우처럼 naming service 가 없기 때문에 이름으로 컴포넌트를 관리할 수 없다. 또한 설치 후에는 컴포넌트가 설치된 OS 의 registry 를 관리해야 한다. MTS 는 EJB 처럼 객체를 packaging 하여 사용하기 때문에 실행 프로그램 관리는 편하다. 그러나 선언적으로 트랜잭션이나 컴포넌트의 구성요소를 관리하지 않기 때문에 구현후 변경 사항 발생시 개발에 대한 부담감이 증가된다. MTS 도 테이블 변경을 위해서는 SQL 문을 사용해야 하기 때문에 특정 데이터베이스에 종속적이 된다. 트랜잭션은 시작 시점과 완료 시점을 코드상에 명확하게 표시할 수 있으므로 분석시에 트랜잭션 처리를 반영할 수 있다.

CORBA 의 경우는 IDL 을 기초로 소스 코드를 생성하기 때문에 인터페이스 설계가 매우 중요하다. 구현후 인터페이스가 수정되면 코드를 변경하기가 매우 어렵다. 또한 skeleton, stub 소스가 많이 생성되고 packaing 이 되지 않기 때문에 소스 관리가 어렵다. CORBA 는 트랜잭션 서비스에 대한 spec 를 가지고 있지만 구현된 제품이 드물기 때문에 , 보통 개발시에는 SQL 문과 특정 데이터베이스에 종속적인 트랜잭션을 사용하며 설계시에 트랜잭션 처리를 반영하기 어렵다.

MTS, CORBA 와 비교해 볼 때, EJB 는 선언적인 트랜잭션, 보안 관리, packaing 을 통해 실행 프로그램을 관리 하기 편하다는 점, 분석시 트랜잭션 처리를 반영할 수 있다는 점 때문에 개발이 효율적이다.

7. 개선 사항

보다 효율적인 EJB 기반 시스템 개발을 위해서는 자동화 도구가 필수적이다. EJB 로 개발할 때도 자료 사전와 데이터베이스, 분석 자료 사이의 일관성 유지는 프로젝트 관리자에게 큰 부담이 된다. RPC 기반의 시스템 구축시에는 자료 사전와 데이터베이스, ERD 사이의 일관성 유지가 문제였지만 EJB 기반의 시스템 구축에서는 자료 사전와 클래스 다이아그램, 데이터베이스, Entity Bean 의 소스와 descriptor 까지 일관성을 유지해야 한다. 대규모 프로젝트를 진행할 때는 관리해야 하는 문서의 종류가 늘어남으로서 효율성이 떨어질 수 있다. 따라서 자료 사전와 클래스 다이아그램의 일관성을 유지하고 클래스 다이아그램에서 DDL, Entity Bean 소스와 descriptor 를 자동 생성하는 도구가 필요하다.[그림 1] 또한 session Bean 의 경우는 로직을 표준화 할 수 있으므로 Visual C++의 wizard 와 유사한 기능을 사용하여 소스 코드를 생성할 수 있다.



[그림 2] 개선된 EJB 기반의 시스템 개발 절차

8. 결론

이상에서 EJB 기반의 시스템 개발 절차를 살펴보았다. EJB 기반의 시스템 개발은 다른 아키텍처에 비해 업무 개발 절차가 빠르고 효율적이다. 그러나 효율적인 개발을 위해서는 반드시 container-managed Entity Bean 을 사용하고 분석시에 트랜잭션 처리를 반영하며 CASE 도구와 소스 자동 생성 도구가 필요하다. 본인은 다음 연구에서 EJB 기반의 시스템 개발을 위한 CASE 도구의 기능과 아키텍처를 제시하겠다.

참고문헌

[1] Geri Schneider, Jason P. Winters “Applying Use Cases”, Addison Wesley, 1997

[2] Object Management Group, “The Common Object

Request Broker 아키텍처 and Specification", Revision 2.2 at URL: http://www.omg.org/corba/corbaiiop.html, 1998

[3] Microsoft Corp. "The Component Object Model Specification", Microsoft Press, 1995

[4] Sun MicroSystems Inc. "Enterprise JavaBeans Specifications" at URL: http://www.javasoft.com