자바 어플리케이션을 EJB로 래핑 하기 위한 기법

김동관 ° 정효택 양영종
한국전자통신연구원
{dgkim°, htjung, yangyj}@etri.re.kr

A Technique for wrapping Java applications in EJBs

Dong Kwan Kim°, Hyo Taeg Jung, Young Jong Yang
S/W.Contents Dept., Electronics and Telecommunications Research Institute

요 약

1990년대 초반에 등장한 자바 언어는 빠른 속도로 프로그래머들 사이에 보급되었으며 인터넷의 등장과 더불어 이는 더욱 가속화되었다. 또한 무선 플랫폼 등과 같은 새로운 컴퓨팅 환경에 빠르게 대처함으로써 자바 언어의 끝을 예측하기는 쉽지 않은 상태이다. 초기 자바 어플리케이션들은 단일 티어(single-tier)로 개발되었으며 환경의 변화로 인해 이런 어플리케이션들을 네트워크로 연결할 필요성이 대두되고 있다. 자바 언어는 분산 컴퓨팅 환경의 솔루션으로 Enterprise JavaBean(EJB)[1]을 제시하고 있다. 본 논문에서는 기존에 개발된 자바 어플리케이션을 EJB로 래핑하기 위한 기법들을 제공한다. 핵심 비즈니스 로직을 가진 클래스들을 수작업을 통해 EJB로 변환할 수도 있지만 본 논문에서는 반자동화된 방법을 통해 변환 상의 효율을 증대 시키고 변환 과정에서 발생할 수 있는 오류를 최소화하고자 한다. EJB 래핑 기법은 세션 빈(session bean)[1] 래핑과 엔터티 빈(entity bean)[1] 래핑으로 구성된다. 세션 빈 래핑은 자바 어플리케이션을 구성하는 클래스 가운데 질의문(query)을 가지지 않는 자바 클래스들을 래핑한다. 엔터티 빈은 질의문을 포함하는 자바 클래스를 래핑한다. EJB 래핑을 위해 리플렉션(reflection)[2]과 위임(delegation) 장치를 사용한다.

1. 서론

컴퓨팅 환경의 급속한 변화는 기존 시스템의 변화를 요구하게 되었으며 이것은 비교적 최근에 개발된 자바 어플리케이션들에 대해서도 예외는 아니다. 초기 자바 프로그램들은 애플릿이나 자바 어플리케이션 형태로 개발되었으며 인터넷 환경이 일반화됨으로써 이를 지원하는 환경으로 바뀌고 있다. 현재 운영 중인 어플리케이션이 별다른 오류가 없는 경우, 어플리케이션 개발자들은 기존 시스템을 그대로 유지하면서 새로운 컴퓨팅 환경을 수용하고자 할 것이다. 이를 위해 본 논문에서는 래핑 기법을 사용한다. 래핑 기법을 사용하면 내부적으로는 기존 시스템을 그대로 유지하지만 외부에서 시스템을 볼 때는 기존 시스템과 다른 인터페이스를 제공한다. 이러한 접근법은 개발 비용 및 기간 측면에서 기존 시스템을 완전히 재개발하는 것보다 효과적이며 재개발 시 발생할 수 있는 오류를 최소화할 수 있다는 장점이 있다. 본 논문은 기존의 자바 어플리케이션을 분산 환경에서 운영할 수 있도록 EJB 래핑 기법을 제시한다. EJB[1]는 썬 마이크로시스템즈 사에서 제시한 이질적이고 분산된 컴퓨팅 환경을 지원하는 컴포넌트 모델로 J2EE의 핵심 모듈이라고 할 수 있다. 기존의 단일 티어 자바 어플리케이션들은 EJB의 세션 빈이나 엔터티 빈으로 래핑됨으로써 분산환경에서 운영될 수 있다.

논문의 구성은 다음과 같다. 2장에서는 본 논문에서 사용한 리플렉션과 EJB에 대한 간단한 설명이 있고 3장에서는 EJB 래핑 개념도를 중심으로 EJB 래핑 과정을 설명한다. 4장과 5장에서는 자바 클래스를 엔터티 빈 또는 세션 빈으로 래핑 하는 기법을 설명한다. 6장은 결론 및 향후연구과제를 기술한다.

2. 관련 지식

본 논문과 관련된 지식으로 자바 언어의 리플렉션[2] 특성과 EJB에 대해 본 장에서 기술한다. 리플렉션은 실행 중인 자바 프로그램이 자기 자신의 내부 속성을 검사하고 조작할 수 있는 자바 언어의 특성이다. 내부 속성에는 클래스 이름, 패키지 이름, 상위 클래스 이름, 필드, 메소드에 대한 정보 등을 포함한다. 대표적인 예가 자바빈(JavaBeans)으로 동적으로 로드되는 자바빈의 속성 값을 얻을 수 있다. 좀 더 자세하게, 리플렉션 Application Programming Interface(API)가 제공하는 기능은 객체의 클래스 획득, 클래스의 메소드, 생성자, 상위 클래스에 관한 정보, 인터페이스의 상수나 메소드 선언부, 런타임까지 결정되지 않은 클래스의 인스턴스 생성, 메소드 호출 및 필드 값의 수정 등을 들 수 있다.

Enterprise JavaBeans(EJB)[1]는 분산환경의 다 계층 어플리케이션 아키텍쳐에 쓰일 수 있는 재사용이 가능한 비즈니스

로직 컴포넌트를 말하며 세션 빈과 엔터티 빈의 두 가지 타입을 가진다. 세션 빈은 특정 도메인의 비즈니스 로직을 메소드로 제공한다. 세션 빈은 클라이언트간에 공유되지 않으므로 하나의 클라이언트 당 하나의 세션 빈이 활성화된다. 또한, 세션 빈은 비영속적이므로 세션 빈의 데이터는 데이터베이스에 저장되지 않는다. 세션 빈은 일반적으로 빈이 공유될 필요가 없고 빈의 상태가 영속적이지 않는 경우에 사용된다.

엔터티 빈은 데이터베이스와 같은 영속적인 기억장치에 있은 비즈니스 개체를 나타낸다. 엔터티 빈은 영속성, 클라이언트간의 공유, 유일한 프라이머리 키 등을 가진다는 측면에서 세션 빈과 구별된다. 하나의 엔터티 빈은 다수의 클라이언트들에 의해 공유될 수 있다. 다시 말해, 다수의 클라이언트들이 하나의 엔터티 빈을 공유하며 해당 엔터티 빈의 데이터를 변경할 수 있음을 의미한다.

3. EJB 래핑 개념도

기존에 개발된 자바 어플리케이션을 엔터프라이즈 환경으로 변환하기 위한 기법으로 본 논문은 EJB 래핑(wrapping) 기법을 사용한다. 기존에 개발된 자바 클래스 가운데 핵심 비즈니스 로직을 가진 것들을 선택하여 소스코드에 대한 변화 없이 EJB로 래핑 하여 분산 환경에서 사용하고자 한다. 래핑 기법은 기존의 핵심 자바 클래스를 그대로 사용하므로 핵심 클래스를 EJB로 대체하는 기법에 비해 개발 기간의 단축과 오류 발생의 감소 측면에서 장점이 있다. 물론 주어진 시간 내에, 아무런 오류 없이, 핵심 클래스를 EJB로 교체하여 새로운 시스템을 개발한다면 래핑 기법보다 효과적이라 할 수 있다. 실제 환경에서는 이러한 래핑 기법을 재개발을 위한 중간 단계로 사용하기도 한다.

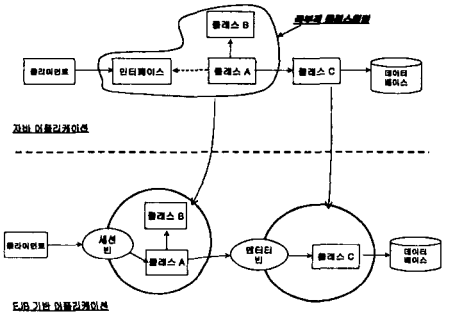

그림 1. 자바 어플리케이션을 EJB로 래핑 하기 위한 개념도

자바 어플리케이션을 EJB로 래핑 하기 위한 전체 개념도는 그림 1과 같다. 그림의 상단부는 자바 어플리케이션을 나타내고 하단부는 EJB 기반 어플리케이션을 나타낸다. 래핑 대상이 되는 자바 어플리케이션은 분산 환경이 아닌 독립형(standalone) 어플리케이션이다. 클라이언트 모듈과 서버 모듈로 구분되며 클라이언트 모듈은 자바 클래스이거나 서블릿(Servlet)[3]이 될 수 있다. 클라이언트의 호출은 중간 계층의 자바 클래스에서 처리되며 데이터베이스 접근을 위한 자바 클래스를 둔다. 래핑 과정을 통한 어플리케이션은 EJB기반으로 데이터베이스나 운영체제 차원에서 이질적인 플랫폼들이 연결된 분산환경에서 운영된다. 그림 1은 EJB로 래핑된 자바 클래스를 보여준다. 그림 1의 앞부분은 세션 빈으로 래핑된 것을 뒷부분은 엔터티 빈으로 래핑된 것을 보여준다.

자바 클래스를 EJB로 래핑 하는 과정은 크게 2단계로, 자바 클래스를 세션 빈으로 래핑 하는 단계(이하, 세션 빈 래핑 단계)와 엔터티 빈으로 래핑 하는 단계(이하, 엔터티 빈 래핑 단계)로 구분된다. 그림에서 보듯이 생성된 세션 빈은 클래스 A, B를 래핑하고, 엔터티 빈은 클래스 C를 래핑 한다. 클래스 A와 B는 직접적으로 데이터베이스를 접근하지 않으므로 세션 빈으로 래핑 되는 반면 클래스 C는 데이터베이스를 접근하므로 엔터티 빈에 의해 래핑 된다. 자바 어플리케이션에서 클래스 A는 클래스 C를 직접 호출하지만 EJB기반의 어플리케이션에서는 클래스 C가 엔터티 빈으로 래핑 됨으로 엔터티 빈을 통해서 클래스 C를 호출한다.

그림 1에서 보듯이 세션 빈 래핑 단계는 자바 클래스들간의 클러스터링 과정을 포함한다. 본 논문에서는 국부적 클러스터링이라는 용어를 사용한다. 국부적 클러스터링이란 래핑 대상이 되는 모든 클래스에 대한 클러스터링이 아니고 세션 빈으로 래핑 되는 일부 클래스만을 대상으로 클러스터링을 실시하는 것을 말한다. 엔터티 빈 래핑은 자바 클래스와 일대일로 발생하며 생성된 엔터티 빈은 데이터베이스 테이블의 하나의 행을 나타낸다.

4. 엔터티 빈으로 래핑

엔터티 빈으로 래핑 되는 자바 클래스는 질의문을 가진 클래스이다. 각각의 자바 클래스별도 별도의 엔터티 빈이 할당된다. 엔터티 빈 소스 코드의 자동 생성을 위해 세션 빈의 경우와 같이 템플레이트가 사용된다. 템플레이트에서 생성된 공통 코드와 메소드별로 위임되는 코드가 합쳐져서 완전한 엔터티 빈 코드가 생성된다.

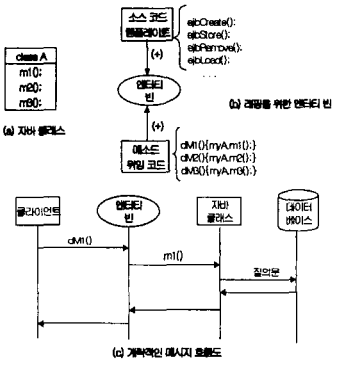

그림 2. 자바 클래스와 래핑을 위한 엔터티 빈

그림 2의 (a)는 자바 어플리케이션을 구성하는 자바 클래스를 나타낸다. 자바 클래스 class A는 메소드 m1, m2, m3를 가지며 해당 메소드는 질의문을 포함하고 있다. 그림 2의 (b)는 그림 (a)의 자바 클래스를 래핑 하기 위한 엔터티 빈이다. 그림에서 소스코드 템플레이트는 엔터티 빈을 위해 기본적인 패키지와 필수 메소드(ejbCreate(), ejbStore() 등) 등에 해당하는 기본 소스 코드를 가진다. 메소드 위임코드에서는 (a)의 자바 클래스의 메소드에 메시지 호출을 위임하는 소스 코드를 가진다. 예를 들어, 메소드 dM1()의 경우, class A에서 생성된 객체

myA의 메소드 m1()에게 메소드 호출을 위임한다. 이와 같이 소스코드 템플레이트에서 얻어진 코드와 메소드 위임 코드를 통해 엔터티 빈을 구성하는 빈 클래스[1], 홈 인터페이스[1], 리모트 인터페이스[1], 프라이머리 키 클래스[1]를 생성할 수 있다. 물리적으로 그림 2 (a)의 자바 클래스와 (b)의 엔터티 빈은 같은 jar 파일로 묶여 지며 EJB서버에 전개된다. 본 논문에서 제시한 래핑을 위한 엔터티 빈의 메소드들은 빈에서 관리(bean-managed) 한다. 그림 2의 (c)는 자바 클래스가 엔터티 빈으로 래핑 되었을 때의 메시지 흐름 관계를 보여준다. 클라이언트는 엔터티 빈의 메소드 dM1()을 호출하면 위임에 의해 자바 클래스의 메소드 m1()이 호출된다. 메소드 m1()은 데이터베이스에 질의문을 수행시켜 클라이언트의 요청을 처리한다. 처리된 결과는 호출의 역순으로 클라이언트에게 전달 된다. 엔터티 빈 dM1()의 기능성은 메시지의 수신 객체를 자바 클래스로 변경하는 것이다. 이것은 5장에서 기술할 세션 빈에 의한 래핑에서도 동일하다.

5. 세션 빈으로 래핑

자바 클래스를 EJB로 래핑 하는 첫 번째 단계는 세션 빈을 통한 래핑이다. 이전에 설명한 것처럼 세션 빈으로 래핑 되는 자바 클래스는 데이터베이스를 직접 접근하지 않는 클래스들이 대상이 된다. 먼저 래핑 대상이 되는 클래스들에 대한 국부적 클러스터링이 실시된다. 자바언어에서 지원하는 구체적 및 추상 클래스 상속(extends)과 인터페이스 상속(implements) 관계에 있는 클래스들은 클러스터링을 한다. 또한 기존 자바 어플리케이션에서 동일한 패키지에 속하는 클래스들은 클러스터링 대상이 된다. 패키지에 속하는 클래스들이 많은 경우는 여러 개의 클러스터가 생성되지만 그렇지 않은 경우는 패키지 당 하나의 클러스터가 생성된다. 패키지에 속하는 클래스가 많은 경우는 클래스들간의 메시지 전달을 고려하여 클러스터링을 한다. 2개 이상의 클러스터에 공유되는 자바 클래스는 복사하여 각각의 클러스터에 포함시킨다.

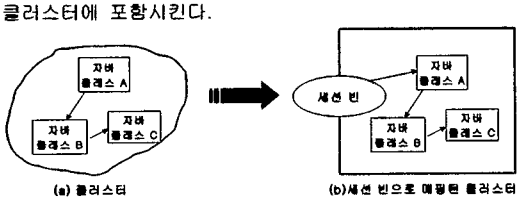

그림 3. 자바 클래스를 세션 빈으로 래핑 하기

그림 3은 국부적 클러스터링 과정을 통해 생성된 클러스터에 대한 세션 빈 래핑을 보여준다. 그림 3의 (a)의 클러스터는 래핑 대상이 되는 자바 어플리케이션을 구성하는 클러스터를 보여준다. 클러스터에는 자바 클래스 A, B, C가 포함되어 있다. 그림 (b)는 세션 빈으로 래핑 된 클러스터이다. 클러스터에 속하는 클래스 A, B, C와 그 들간의 관계성은 그대로 유지되며 세션 빈이 새롭게 추가된다. Gamma의 설계 패턴 중 하나인 facade pattern[4]의 facade처럼 세션 빈은 클러스터를 구성하는 클래스들에 대한 단일 통로를 제공한다. 그림 (a)의 클러스터의 경우는 클라이언트의 호출이 각각의 클래스 즉, A, B, C에 대해 발생하는 반면 그림 (b)의 경우는 래핑 하고 있는 세션 빈을 통해 모든 클라이언트의 접근이 이루어진다. 클라이언트 관점에서는 세션 빈을 보고 메시지를 전달할 뿐이지 세션 빈 뒤에 어떤 클래스가 있는 지는 모르는 상태이다.

그림 (b)의 클러스터는 jar파일로 묶이는 대상이다. 세션 빈 하나에 나머지 3개의 클래스는 도움 클래스(helper class)로 하나의 jar파일에 포함된다. jar파일은 EJB 전개의 기본 단위가 된다. 그림 (b)에 생성된 세션 빈은 수작업으로 개발 될 수도 있지만 본 논문에서는 자바에서 제공하는 리플렉션과 위임 장치[5, 6]를 이용한 자동 생성을 사용한다. 리플렉션 장치를 통해 자바 클래스의 클래스 파일(.class file)로부터 기본적인 정보를 얻을 수 있다. 클래스 이름, 패키지 이름, 필드 정보, 메소드 정보 등을 추출할 수 있다. 세션 빈은 클래스의 메소드들에 대한 위임 코드 만을 가지고 있는다. 클라이언트 호출이 발생하면 이를 받아 실질적인 구현 코드를 가지고 있는 자바 클래스로 위임한다. 세션 빈의 빈 클래스의 메소드는 EJB 스펙에서 요구하는 필수 메소드(ejbCreate(), ejbStore() 등)와 위임 메소드로 구분 할 수 있다. 세션 빈의 코드 생성은 공통적으로 사용되는 소스코드를 가지는 템플레이트를 기반으로 이루어진다.

6. 결론 및 향후 연구과제

본 논문에서는 기존에 개발된 자바 어플케이션을 EJB 기반의 엔터프라이즈 어플리케이션으로 변환하기 위한 방법으로 EJB 래핑 기법을 제안한다. EJB 래핑은 세션 빈 래핑과 엔터티 빈 래핑으로 구분된다. 자바 클래스 중 Standard Query Language(SQL) 코드를 포함하지 않는 클래스는 세션 빈으로 래핑이 되며 SQL 코드를 가진 자바 클래스는 엔터티 빈으로 래핑 된다. 세션 빈으로 래핑 되는 자바 클래스는 먼저 클러스터링 작업을 수행한다. 클러스터링을 통해 얻어진 클러스터들은 하나의 세션 빈에 의해 래핑 되며 하나의 jar파일로 패키징 된다. 세션 빈의 코드는` 자바의 리플렉션과 위임 장치를 통해 생성된다. SQL 코드를 가진 자바 클래스는 엔터티 빈으로 일대일 래핑 되므로 클러스터링 과정이 필요하지 않다. 생성되는 엔터티 빈의 메소드는 빈에서 관리 된다.

향후 연구과제로 SQL 소소코드를 가진 자바 클래스를 컨테이너 관리(container-managed)기반의 엔터티 빈으로 변환하는 연구가 필요하다. 본 논문에서 생성된 엔터티 빈은 영속성 관리가 빈에서 이루어지고 있다. 추가로 핵심 비즈니스 로직을 가진 자바 클래스에 대한 자동 추출 기능이 요구된다. 본 논문에서 제시한 기법은 개발자에 의해 자바 클래스가 입력되지만 좀 더 자동화율을 높이기 위해서는 핵심 비즈니스 클래스를 식별하여 개발자에게 제시해주는 기능이 요구된다.

참 고 문 헌

[1] Sun Microsystems, EJB specification 1.1 http://java.sun.com/products/ejb/index.html
[2] Sun Microsystems, Reflection API, http://java.sun.com/docs/books/tutorial/reflect/index.html/
[3] Sun Microsystems, Java Servlet Specification, http://java.sun.com/products/servlet
[4] Gamma 외 3, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[5] Tony Loton, Make an EJB from any Java class with Java Reflection, Java World, Dec. 2000. http://www.javaworld.com/javaworld/topicalindex.
[6] Gorsen Huang, Utilize the EjbProxy, Java World, Oct. 2001. http://www.javaworld.com/javatips/jw-javatip118_p.html#resources