J2EE 패턴기반 EJB 빈 클래스의 다중 DB 연동에 대한 설계 및 구현

이 돈 양† · 송 영 재††

요 약

최근에 객체지향 소프트웨어를 설계하거나 구현방법으로 EJB 기반의 소프트웨어 개발이 많이 이용되고 있다. 일반적으로 EJB 기반 어플리케이션에서는 데이터베이스를 이용한 영속적인 데이터를 사용하는 경우가 대부분이다. 본 논문에서는 서버 측 작성 프로그램 중 엔티티빈 클래스에서 담당하는 데이터베이스 엑세스에 관련된 부분을 J2EE의 DAO 패턴을 이용하여 클래스를 각각 분리하였다. 이는 기존의 패턴 방법과는 큰 차이는 없으나, 동일 패턴내의 공통의 클래스들을 합성이 가능하도록 설계하였다. 그리고 생성된 각각의 DBMS 클래스들은 다른 엔티티빈 클래스에서도 사용이 가능하게 할 뿐만 아니라 여러 DBMS 환경에서도 Data Source를 추가적인 프로그램의 변경이나 작성 없이 연동이 가능하도록 하고 있다.

Design and Implementation of Multiple DataBase Access using Choice Method for EJB Bean Class Based on J2EE Pattern

Don-Yang Lee† · Young-Jae Song††

ABSTRACT

Recently, software programming method based on EJB for object oriented software design and implement has been used frequently. Usually, case that use permanent data that use Database in EJB base application is most. Part connected with Database-Access that take charge in Entity Bean class of server side creation program in this paper using J2EE relationship DAO pattern class each separate. This is no much difference with existent pattern method, but in same pattern common classes are designed so that composition may be possible. And as well as use Entity Bean class that created each DBMS classes are different, is doing Data Source so that connection work is available without alteration or creation of additional program in several DBMS environments.

키워드 : EJB, J2EE, DAO, 디자인 패턴(Design Pattern), 엔티티빈 클래스(Entity Bean Class), 배치(Deploy)

1. 서 론

정보통신의 발달과 더불어 소프트웨어 산업의 발전에도 많은 변화가 일어나고 있다. 정보통신 산업은 경제적, 기술적 변화에 부응하여 발전해야한다. 이를 위해서는 시스템을 확장하거나 새롭게 재설계해야할 필요성을 갖고 있다. 그리고 재설계에 따른 비용을 절감하기위한 노력이 꾸준히 이루어지고 있다[1].

1990년대 초 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides의 Design Pattern(GoF Design Pattern) 기법이 정립되기 시작하여 객체지향 시스템을 설계할 때 이를 이용하여 설계정보를 추상화 할 수 있도록 하였다[2, 3]. 그리고 본 논문에서 설계방법에 적용하고 있는 J2EE(Java 2 Platform Enterprise Edition) 패턴역시 Gamma의 디자인패턴에 근거를 두고 있다.[4]

J2EE는 워크그룹 서버로부터 고급서버를 포함하는 비즈니스 어플리케이션개발에 필요한 환경으로 구상되었다. J2EE의 특징으로는 멀티티어(Multi-tier)로 구성되어 있으며, 특히 서버에 초점이 맞춰져있는 프로그램이고, J2EE 어플리케이션의 기본부분은 EJB 컴포넌트에 의해 구성되어있다. 그리고 또 다른 특징으로 J2EE는 기존에 사용되고 있는 시스템의 데이터베이스를 그대로 이용가능하게 하고 있다. 이런 특징 중에서 데이터베이스를 연동하여 정보시스템의 자원을 효율적으로 이용하기 위한 방법으로 본 논문에서는 패턴을 적용하여 설계하고 있다.

일반적으로 EJB 기반 어플리케이션 개발환경에서는 여러 개의 엔티티빈을 사용하고 있다. 각각의 엔티티빈들은 데이터베이스와 연동되어 운영되므로 비즈니스 객체의 코드 복잡도를 증가시킬 뿐만 아니라, 코드의 신뢰성과 개발환경성을 저하 시킬 우려가 있다.

본 논문에서는 이런 문제점들을 개선하기 위해 빈 클래

† 준 회 원 : 경희대학교 대학원 전자계산공학과
†† 종신회원 : 경희대학교 컴퓨터공학과 교수
논문접수 : 2003년 8월 11일, 심사완료 : 2003년 10월 7일

스에서 데이터베이스 연결 시 환경설정 및 연결부분을 DAO (Data Access Object) 방법을 적용하여 각 클래스를 Abstract 부분과 Concrete 부분으로 분리하고 인터페이스를 이용하여 조립하였다. 그 결과로 시스템의 환경이나 업무에 따라 다르게 작성되어 사용되는 데이터베이스를 프로그램의 재작성이나 각각의 빈(Bean) 클래스에서 데이터베이스의 연동에 필요한 프로그램을 추가하거나 변경할 필요 없이 재사용할 수 있도록 하였다.

2. 연구 배경

2.1 디자인 패턴

디자인 패턴은 객체지향 소프트웨어를 설계하는데 있어서 반복적으로 발생하는 설계상의 문제들을 패턴으로 등록하여, 설계자들이 어떠한 설계 문제에 대해서 다시 해결책을 모색하는 과정을 거치지 않고, 바로 디자인 패턴을 선택하여 설계에 적용할 수 있도록 한 것이다[8]. 1990년 초 Erich Gamma와 Richard Helm은 패턴을 여러 가지로 분류하여 그룹화하고, 약속된 일정한 형태로 표현하였다[9, 10]. 당신은 많은 객체 지향 시스템에서 클래스들과 의견 교환하는 객체들이 순환되는 패턴들을 찾을 것이다. 이들 패턴들은 특정한 설계문제들을 해결하며 보다 유연하고, 우아하고, 그리고 궁극적으로 재사용 가능한 객체지향 설계를 만들어준다[5]. 이들은 설계자가 이전의 경험에 근거해서 성공적인 설계를 재사용하는데 도움을 준다고 했다[1]. 일반적으로 자바 프로그램을 작성할 때 클래스들이 모여 있는 클래스 라이브러리를 이용한다. 그러나 디자인 패턴은 클래스의 라이브러리가 아니다. 즉, 라이브러리보다 일반적인 개념이다. 클래스 라이브러리는 부품이 되는 프로그램이지만, 디자인 패턴은 부품이 어떻게 조립되어있고, 각각의 부품이 어떻게 연관되어 커다란 기능을 완수하는지를 표현한다[14].

2.2 J2EE 디자인 패턴

EJB(Enterprise JavaBeans) 기반 시스템을 설계할 때 중요한 것 중 하나는 퍼포먼스나 유지보수성, 또는 이식성과 같은 사항들을 충족시키기 위한 정확한 아키텍처를 선택하거나 로직을 분할하는 것이다[5]. 잘 만들어진 EJB 프로젝트들은 대부분을 최적의 디자인 패턴을 적용하고 있다[1]. 일반적으로 EJB 레이어 아키텍처 패턴으로는 Session Facade와 Message facade 패턴을 사용한다. 그리고 퍼포먼스의 향상을 위한 방법에서 네트워크를 통한 적은 단위의 메소드 호출을 피하기 위해서 Value Object 패턴을 사용하고 있으며, 한번의 클라이언트 요청에 대한 다수의 네트워크 호출을 피하기 위해서는 Value Object Factory/Value Object Assembler 패턴을 사용하고 있다[11]. 또한 J2EE 어플리케이션에서는 영속적인 데이터를 사용하는 경우가 대부분이다. 여기서 영속적인 데이터를 나타내기 위한 엔티티빈과 같이 영속적이고 공유되고 분산된 컴포넌트를 사용할 수 있다[7]. 엔티티빈이 영속적인 기억장치를 엑세스할 때 어플리케이션의 엔티티빈에서 빈 관리 영속성을 쓰도록 고려해야 한다. 이를 관리하기 위한 기법으로는 DAO Factory 패턴이 사용되기도 한다.

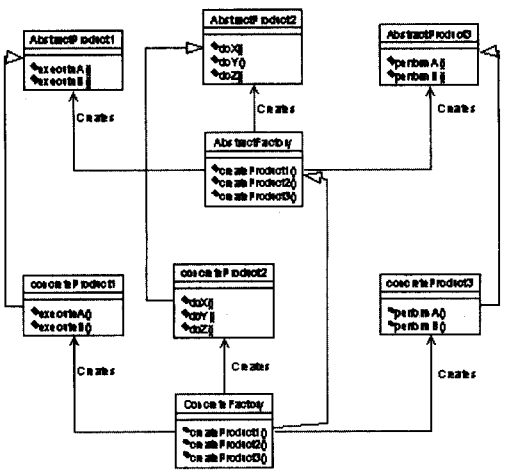
3. 디자인 패턴 선정 및 적용

본 논문에서는 EJB기반으로 구현하고자 하는 빈 생성은 엔티티빈 클래스의 데이터베이스 연결 시 환경설정 및 연결부분을 Abstract Factory를 이용하여 각각의 클래스로 분리하고 조립하였다.

여기서 Abstract Factory 패턴을 적용하게 된 것은 빈 클래스와 데이터베이스의 관계를 추상적인 설계를 통해 인터페이스(PI)에 주목하여 그 인터페이스만을 사용해서 부품을 조립하여 완성하기 위한 방법으로 채택하기 위함이며, 이는 처음부터 구체적인 구현을 위한 목적에 중점을 두는 것보다 추상적인 방법을 통해서 클래스를 생성하여 다른 클래스에서 상속될 수 있도록 하고 있다.

그리고 Abstract Factory 패턴은 Factory 패턴을 추상화시킨 것으로 SuperClass로부터 상속된 여러 개의 SubClass 중 하나를 선택할 수 있도록 해줄 수 있다. 또한 본 논문에서는 이 패턴의 기본 형태를 이용하여 어플리케이션 시스템을 개발할 시 여러개의 DBMS를 정의하고, 공통의 부분을 추상적인 SuperClass로 정의하고 구체적인 Concrete에 해당하는 DBMS의 선택부분을 SubClass로 두었다. 만약 이 패턴을 적용하지 않는다면 SuperClass에 해당하는 공통의 정의부분을 각각의 SubClass에서 동일한 코드부분을 정의해 주어야하는 문제가 발생된다.

비슷한 생성적 패턴으로 Builder 패턴이 있으나 추상클래스를 생성하는 것은 비슷하나 복잡한 인스턴스를 조립하기 위한 방법으로 이용된다[12].

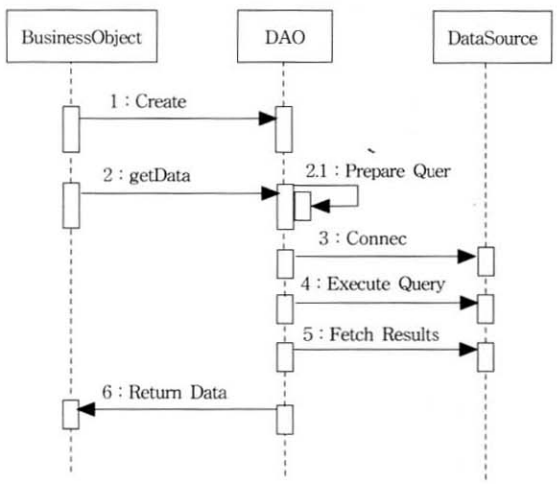


(그림 1) Abstract Factory 패턴

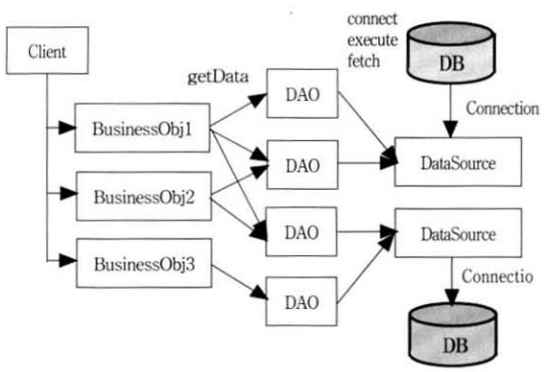
일반적으로 EJB 어플리케이션에서 일반적으로 여러 개의 엔티티빈들이 사용되고 있다. 이들에게 공통적으로 적용되고 있는 데이터베이스 엑세스에 대한 코드부분을 관련된 몇 개의 클래스로 분리함으로써 이를 다른 엔티티빈에서 공용으로 사용할 수 있도록 인터페이스를 생성하여 코드의 신뢰성과 개발의 생산성을 향상시킬 수 있도록 할 수 있다[6].

3.1 DAO(Data Access Object) 패턴

DAO 패턴은 Data Source로부터 데이터를 조회하거나 사용할 수 있도록 하는 추상적인 방법으로 사용된다. DAO 패턴의 사용자는 데이터가 Oracle database, Microsoft SQL Server, LDAP Server 또는 Web Server를 어디서 조회되는지를 알 수 없다. 어플리케이션이 Data Source에 직접적으로 접근하지 않는 사실에 근거해 DAO 패턴은 유동적으로 제공되고 있다. 대신 중간정도 부분에서 Data Source에 접근하는데 있어서 DAO를 생성하고 사용할 수 있다. 만약 조회하고자 하는 데이터가 있으면 조회데이터를 가지고 있는 Value Object를 사용할 수 있다. 여기서 Value Object 패턴은 데이터베이스에 저장되어 있는 데이터가 어떻게 저장되어 있는가에 대한 물리적인 구조에 대해 추상적으로 처리하고 있다.



(그림 2) DAO Pattern Sequence Diagram



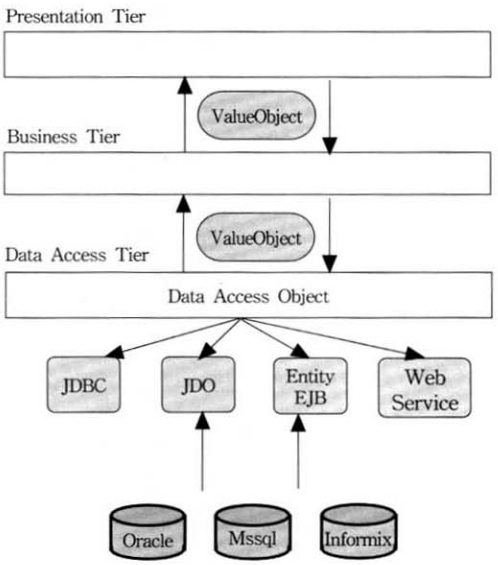
(그림 3) DAO Pattern 구조

DAO 패턴이 가지고 있는 강점으로는 Data Source로부터 조회되는 데이터를 추상적인 방법으로 처리한다는 점과, 작업을 수행하는 데 있어서 API(JDBC API, SQLJ, Entity EJB, JDO)를 통해 데이터를 접근할 때 java을 이용하고 있는 점이다[13].

그리고 business와 data logic의 합성으로 어플리케이션의 개발에 대한 최소한의 시도를 통한 business와 data logic을 명확히 분리하는 방법을 제공하고 있다.

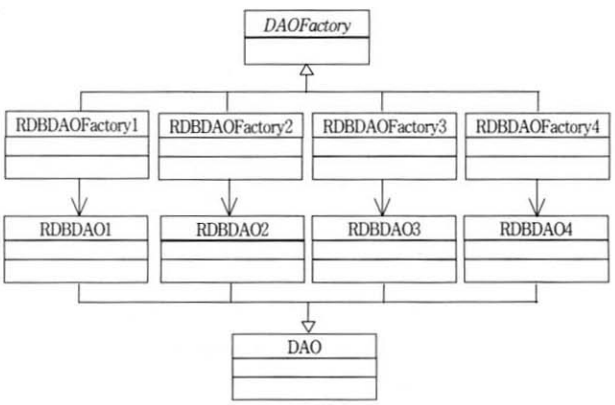
3.2 영속적 프레임워크

Data Source를 사용하고 다루는 것에 대한 어플리케이션을 적용하는 소프트웨어 서비스에서 영속적 프레임워크(Persistence Framework)는 데이터자원의 조직과 API(JDBC, JDO, entity EJB)을 이용한 자원의 접근에 대한 방법으로 (그림 4)과 같이 적용되고 있다.



(그림 4) 영속적 프레임워크 구조

3.3 DAO 패턴의 적용방법



(그림 5) Abstract Factroy Pattern의 Class Diagram

본 논문에서는 엔티티빈 클래스에서 Data Source에 관련된 DBMS를 (그림 5)에서와 같이 Abstract Factory와 Concrete Factory 부분으로 분류를 하고 있다[15, 16]. 이는 각기 다른 DBMS 환경에서 Data Source에 접근하고자 할 때 매번 변경된 프로그램을 작성할 필요없이 어플리케이션 구축의 환경을 선택적으로 설정이 가능하도록 하고 있다.

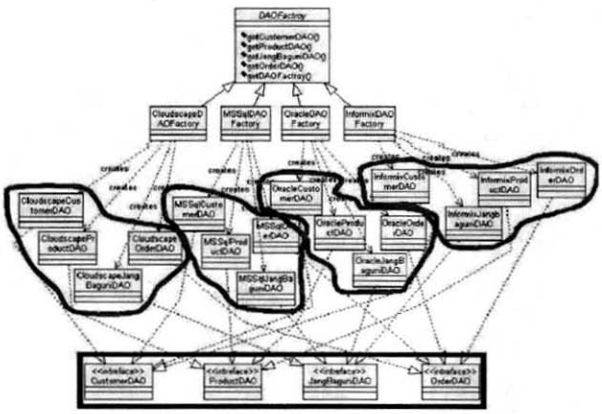
4. 패턴설계 및 구현

이장에서는 본 논문에서 제안한 엔티티빈에서 Data Source와 연결되는 부분을 DAO 패턴의 방법을 이용하여 Abstract 부분과 Concrete 부분으로 분리하였다. 그리고 현재 사용되어지고 있는 다양한 Data Source 중 Oracle, Mssql, Informix, Cloudscape 등 4가지를 가지고 선택하여 연결하여 사용할 수 있도록 하였다. 여기서 적용하고 있는 시스템의 환경은 <표 1>과 같다.

〈표 1〉 Bean 클래스생성 환경요소

Platform	Application Server	DBMS	Programming Language
Windows 2000 Server, Java2(J2EE)	EJB, WebLogic6.1	Mssql7.0	Java, XML

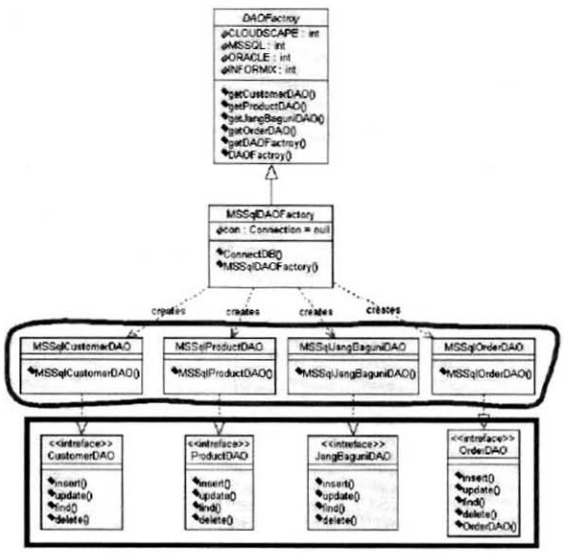
또한 본 논문에서는 J2EE 디자인패턴에서 분류된 DAO 패턴의 기본적인 설계유형과 방법을 적용시키고 있으나 DAO 인터페이스부분과 Concrete의 Create 클래스에 해당되는 공통적 메소드를 많이 가지고 있는 클래스들을 합성하고 있다. 그 결과 전체적인 클래스의 개수의 감소와 프로그램 작성 시 프로그램의 라인수를 줄일 수 있었다. (그림 6)은 쇼핑몰에 대한 사례연구를 통해서 J2EE 디자인 패턴에서 제시하고 있는 유형의 모델로 전체시스템을 설계한 것이며, (그림 7)은 4개의 DBMS를 선택에 대한 4개의 SubClass중 MSSql을 연동하는 다이어그램을 설계하였다. 그리고 (그림 8), (그림 9)에서는 본 논문에서 제시한 공통클래스의 합성에 관한 적용을 보여주고 있다.



(그림 6) 쇼핑몰 Class Diagram

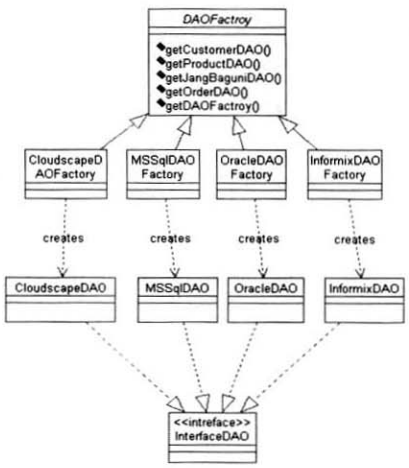
4.1, 4.2에서는 현재 어플리케이션시스템개발에서 주류를 이루고 있는 컴포넌트생성에서 분산객체 시스템을 효과적으로 구현하도록 하는 아키텍처인 EJB 기반의 빈 생성에 여기서 제안한 방법을 적용하고 있다. 또한 EJB는 Java의 객체지향개념을 가지고 있어 컴포넌트 생성 및 재사용에 아주 적합한 기반을 내포하고 있다.

(그림 6)에서 짙은 직사각형으로 영역이 표시되어 있는 부분은 4개의 인터페이스영역으로 하나의 인터페이스로 합성을 하는 부분이며, 곡선으로 4개의 영역으로 분리되어있는 부분은 4개의 DBMS Create 클래스에 대한 합성클래스 영역을 표시하고 있다.



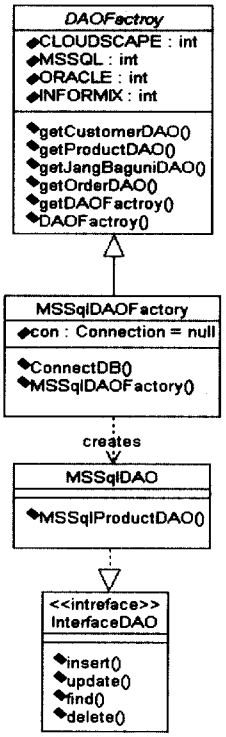
(그림 7) MSSqlDAOFactory Class Diagram

그리고 (그림 7)에서는 (그림 6)에서 표본으로 분리한 MSSql의 클래스 다이어그램으로 J2EE의 Abstract Factory 패턴을 기준으로 설계한 것이다.



(그림 8) 클래스합성 쇼핑몰 Class Diagram

(그림 8), (그림 9)는 본 논문에서 제시한 방법을 적용하여 각각의 공통클래스를 합성한 클래스 다이어그램이며 하나의 인터페이스를 통하여 DBMS의 Create 클래스를 생성할 수 있도록 하고 있다.



(그림 9) 클래스합성 MSSqlDAOFactory Class Diagram

그리고 (그림 10)은 (그림 9)의 합성된 인터페이스 클래스를 구현하고 있으며, (그림 11)은 (그림 9)의 합성된 Create 클래스를 구현하고 있다.

```
interface CustomerDAO {
        public int insertCustomer() ;
        public int updateCustomer() ;
        public int findCustomer() ;
        public int deleteCustomer() ;
}
interface ProductDAO {
        public int insertProduct() ;
        public int updateProduct() ;
        public int findProduct() ;
        public int deleteProduct() ;
}
interface JangbaguniDAO {
        public int insertJangbaguni() ;
        public int updateJangbaguni() ;
        public int findJangbaguni() ;
        public int deleteJangbaguni() ;
}
interface OrderDAO {
        public int insertOrder() ;
        public int updateOrder() ;
        public int findOrder() ;
        public int deleteOrder() ;
}
public interface InterfaceDAO extends CustomerDAO,
ProductDAO, JangbaguniDAO, OrderDAO {

//.....
}
```

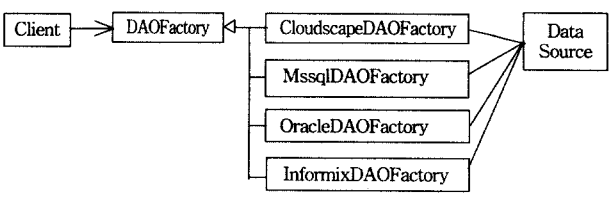
(그림 10) InterfacceDAO 클래스

```
public class MSSqlDAO implements InterfaceDAO
{
        public MSSqlCustomerDAO()
        {
                .....
        }
        public MSSqlProductDAO()
        {
                .....
        }
        public MSSqlJangbaguniDAO()
        {
                .....
        }
        public MSSqlOredrDAO()
        {
                .....
        }
  .....
}
```

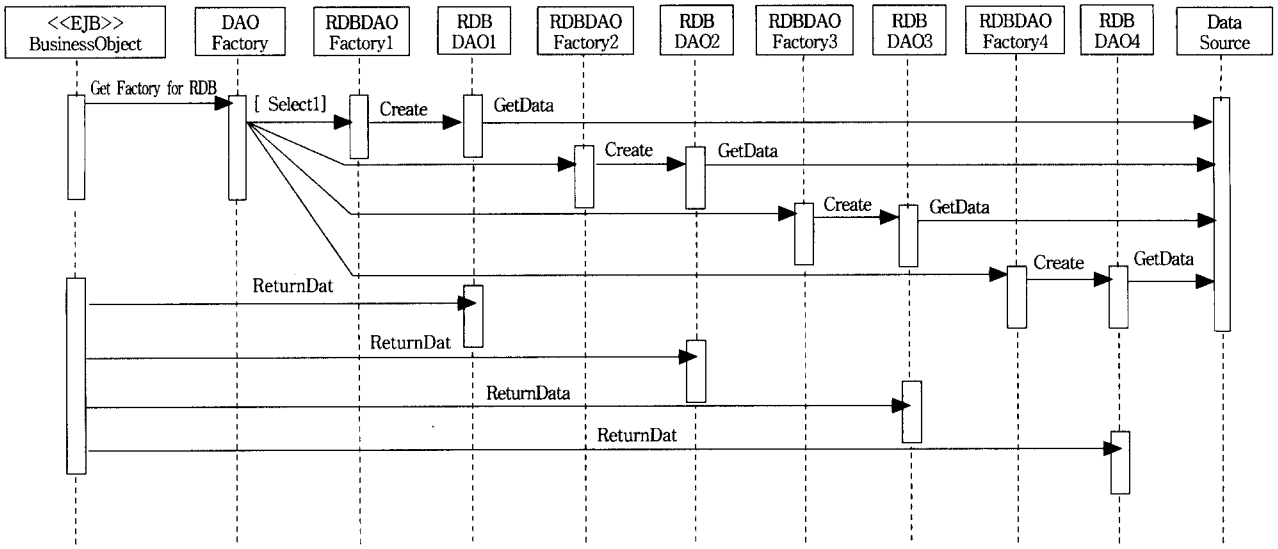
(그림 11) MSSqlDAO 클래스

4.1 Abstract Factory 형태로 엔티티 빈 설계

EJB에서는 어플리케이션 시스템을 개발하기 위해서는 서버 측에서 구현하는 프로그램과 클라이언트 측에서 구현하는 프로그램으로 나뉘어진다. 서버 측 프로그램으로는 세션빈(Session Bean)과 엔티티빈(Entity Bean)으로 구분할 수 있는데, 본 사례연구에서 적용한 엔티티빈에는 컨테이너 관리 퍼시스턴스(CMP)와 빈 관리 퍼시스턴스(BMP)로 나누어진다. 여기서는 BMP를 이용한 엔티티 빈을 구현하였으며 이는 리모트 인터페이스(Remote Interface), 홈 인터페이스(Home Interface), 엔티티빈 클래스(Entity Bean Class), 그리고 프라이머리 키 클래스(Primary Key Class)로 구성된다. 그리고 BMP에서는 개발자가 직접 퍼시스턴스(persistence)와 관련된 Logic을 명시적으로 코딩해야한다. 서버 측 작성 프로그램 중 엔티티빈 클래스에서 데이터베이스와 연결하는 부분을 만들 수가 있다.



(그림 12) DAO를 이용한 선택적 DBMS 구조



(그림 13) DAO를 이용한 선택적 DBMS Sequence Diagram

여기서 DAO 패턴을 적용한 방법은 (그림 12)과 같이 DAOFactory를 이용하여 Cloudscape, Mssql, Oracle, Informix 등 4가지의 일반적으로 많이 사용되고 있는 DBMS를 연결하는 방법을 보여 주고 있다.

4.2 엔티티 빈 클래스 분리

일반적으로 EJB 엔티티빈에서 Data Souce를 연동하기 위한 방법으로 Client와 DAOFactory 그리고 DBMS와 Data Source의 관계를 Abstract Factory 패턴을 이용하여 설계하고 있지만 실질적으로는 (그림 21)의 EntityBeanClass.class에서 정의하여 사용하고 있다. (그림 14)는 엔티티빈 클래스에서 데이터베이스와 연동하는 일반적인 형태를 보여주고 있다. 그러나 본 논문에서는 DAO 패턴형태로 엔티티빈 클래스(Entity Bean Class)의 데이터베이스 연결부분을 원래의 프로그램인 (그림 14)의 ②에 해당되는 ConnectDB() 부분을 (그림 16)과 같이 Abstract Factory 부분으로 분리하였다. 그리고 Concrete Factory 부분을 각각의 DBMS의 사용 환경에 따라 (그림 17), (그림 18), (그림 19), (그림 20)으로 분리하고 있다.

```
import javax.ejb.* ;
import java.sql.* ;
import java.util.* ;
public class ProductsBeanClass1 implements EntityBean
{
    ......
    private Connection con = null ;
    private PreparedStatement ps = null ;
    private ResultSet rs = null ;
    public void setEntityContext(EntityContext ctx)
    {
      this.ctx = ctx ;
      try
      {
        ConnectDB() ; //데이터베이스와 연결설정   ①
      }
      catch (Exception ex)
      {
        throw new EJBException("setEntityContext : "
        + ex.getMessage()) ;
      }
    }
    ...
public void ConnectDB()   ②
{
    String strURL = "jdbc : oracle : thin : @127.0.0.1 : 1521 : ORCL" ;
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver") ;
    }
    catch (java.lang.ClassNotFoundException e)
    {
      System.out.print("에러입니다.") ;
    }
    try
    {
      con = DriverManager.getConnection(strURL, "lee", "1111") ;
    }
    catch (SQLException se)
    {
      System.out.println("연결할 수가 없습니다.") ;
    }
}
......
```

(그림 14) 엔티티빈 클래스

(그림 14)에서는 데이터베이스를 연결하기 위해서 ConnectDB() 함수를 같은 프로그램에서 호출하고 있다. JDBC 드라이버를 이용하여 Data Source와 연결하기 위한 방법으로 두 가지의 인터페이스인 getPooledConnection과 getConnection이 사용되고 있다.

```
mport javax.ejb.* ;
import java.sql.* ;                                          ④
import java.util.* ;
public class ProductsBeanClass1 extends MssqlDAOFactory
implements EntityBean
{
    ...
   private Connection con = null ;
   private PreparedStatement ps = null ;
   private ResultSet rs = null ;
   public void setEntityContext(EntityContext ctx)
   {
     this.ctx = ctx ;
     try
     {                                                       ③
       ConnectDB() ;  // 데이터베이스와 연결설정
     }
     catch (Exception ex)
     {
       throw new EJBException("setEntityContext : "
       + ex.getMessage()) ;
     }
   }
...
```

(그림 15) 패턴적용 엔티티빈 클래스 생성

(그림 15)의 ③에서는 (그림 14)의 데이터베이스 연결부분을 같은 프로그램이 아닌 다른 MsssqlDAOFactory.class 파일을 (그림 17)에서 상속을 받아 Data Source와 연결하고 있다.

```
/* DAOFactory.java */
//public abstract class DAOFactory extends DBSelect {
public abstract class DAOFactory {
 public abstract CustomerDAO getCustomerDAO() ;
 public abstract ProductDAO getProductrDAO() ;
 public abstract JangbaguniDAO getJangbaguniDAO() ;
 public abstract OrderDAO getOrderDAO() ;

 public static final int whichFactory = 1 ;
 public static DAOFactory getDAOFactory (int whichFactory) {
        switch (whichFactory) {
                case CLOUDSCAPE :
                        return new CloudscapeDAOFactory() ;
                case ORACLE :
                        return new OracleDAOFactory() ;
                case INFORMIX :
                        return new InformixDAOFactory() ;
                case MSSQL :
                        return new MssqlDAOFactory() ;
                default :
                        return null ;
        }
 }
}
```

(그림 16) Abstract Factory 생성

(그림 16)에서는 각종 Data Source를 추상적인 형태로 선언한 부분으로서 선택을 통하여 임의의 DBMS 객체를 생성하고 있다. 여기서는 일반적으로 많이 사용하고 있는 DBMS를 선정하여 작성하였으며 필요에 따라서 추가가 가능할 수 있도록 하고 있다. 그리고 (그림 17), (그림 18), (그림 19), (그림 20)은 (그림 16)에서 생성되는 부분이다. 추후 본 논문에서는 이 부분을 자동화 가능한 프로그램으로 작성하고자 한다.

```
/* MssqlDAOFactory.java */
import javax.ejb.* ;
import java.sql.* ;
import java.util.* ;
public class MssqlDAOFactory extends DAOFactory{
  public Connection con = null ;
  public void ConnectDB()
  {
    String strURL = "jdbc : weblogic : mssqlserver4
                    : pubs@localhost" ;
    try
    {
      Class.forName("weblogic.jdbc.mssqlserver4.Driver") ;
    }
    catch (java.lang.ClassNotFoundException e)
    {
      System.out.print("에러입니다.") ;
    }
    try
    {
      con = DriverManager.getConnection(strURL, "lee", "1111") ;
    }
    catch (SQLException se)
    {
      System.out.println("연결할 수가 없습니다.") ;
    }
  }
}
```

(그림 17) Mssql Concrete Factory 생성

(그림 17)에서는 MSSql을 연결하는 클래스이며 JDBC(Java Database Connectivity)를 이용한 데이터베이스 연동을 하고 있다. 그리고 웹로직의 ConnectionPool을 이용하여 연결을 설정하였다. 이는 URL을 "String strURL = "jdbc : weblogic : mssqlserver4 : pubs@localhost" ; "로 작성하면 된다. 또한 이런 방법으로 나머지 Cloudescape, Informix, Oracle등을 연동시키면 된다.

```
/* CloudscapeDAOFactory.java */
  ......
public class CloudscapeDAOFactory extends DAOFactory{
  public Connection con = null ;
  public void ConnectDB()
  {
    String strURL = "jdbc : cloudscape : weblogic@localhost
                    : 7001" ;
    try
    {
      Class.forName("COM.cloudscape.core.JDBCDriver") ;
    }
    ......
  }
}
```

(그림 18) Cloudscape Concrete Factory 생성

```
/* InformixDAOFactory.java */
  ......
public class InformixDAOFactory extends DAOFactory{
  public Connection con = null ;
  public void ConnectDB()
  {
    String strURL = "jdbc : weblogic : informix4
                          : pubs@localhost" ;
    try
    {
      Class.forName("Weblogic.jdbc.informix4.Driver") ;
    }
    ......
  }
}
```

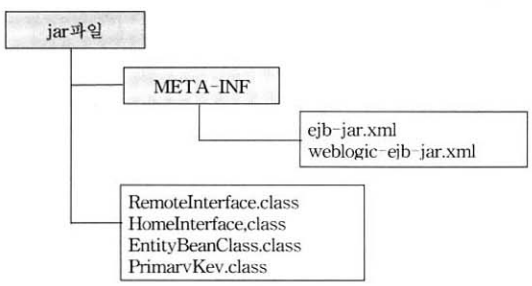
(그림 19) Informix Concrete Factory 생성

```
/* OracleDAOFactory.java */
  ......
public class OracleDAOFactory extends DAOFactory{
  public Connection con = null ;
  public void ConnectDB()
  {
    String strURL = "jdbc : oracle : thin : @127.0.0.1 : 1521
                          : ORCL" ;
    try
    {
      Class.forName("oracle.jdbc.driver.OracleDriver") ;
    }
    ........
  }
}
```

(그림 20) Oracle Concrete Factor 생성

4.3 엔티티 빈의 배치 디스크립터(Deployment Descripter) 구조

EJB 빈을 생성하기 위해서는 일반적으로 .jar 파일 만들어야 한다. 이것은 (그림 21)과 같은 구조로 되어있으며 서버에서 요구되어지는 클래스 파일과 2개의 xml 파일로 구성되고 있다.

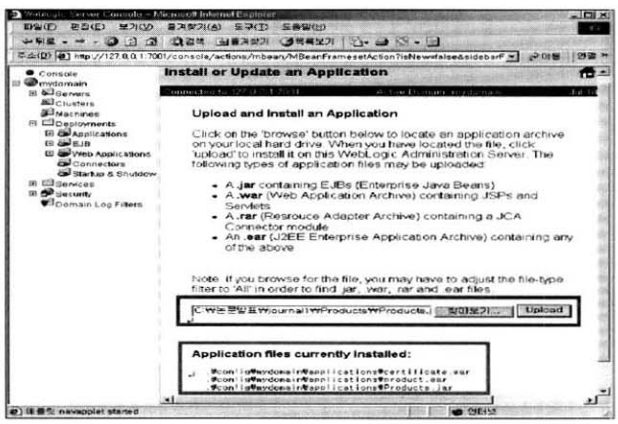


(그림 21) jar 파일 구성요소

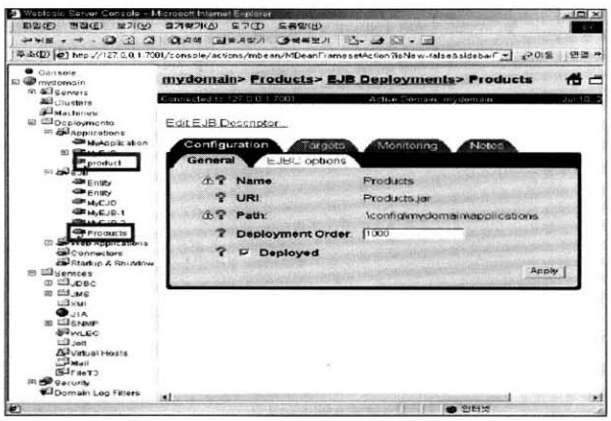
여기서 리모트 인터페이스(RemotrInterface.class)는 엔티티빈 클래스에서 구현할 비즈니스 메소드를 선언하는 인터페이스이고, 홈 인터페이스(HomeInterface.class)는 엔터프라이즈 빈을 컨테이너에서 생성하고 사용할 수 있게 해주는 create() 메소드를 선언하는 클래스이다. 엔티티빈 클래스(EntityBeanClass.class)는 데이터베이스와 연동하여 빈을 생성하는 클래스이다. 그리고 엔티티빈에서 프라이머리키를 구성하는 프라이머리키클래스(PrimaryKey.class)가 있다.

4.4 EJB 빈 배치

EJB 기반의 빈 생성을 위한 서버로 일반적으로 J2EE 서버가 널리 이용되고 있다. 그리고 Deploy Tool을 이용하여 자동으로 엔터프라이즈 빈에 대한 배치 스크립터인 xml 파일을 생성한다. 그러나 본 논문에서 제안한 다중 DBMS의 선택적 사용을 위해서는 배치 ejb-jar.xml 파일을 프로그램 작성자가 빈번히 수정을 해야 하는 번거로움을 가져야 한다. 그래서 배치 디스크립터의 수정이 용이하고 널리 사용되며 빠른 J2EE 스펙을 구현할 수 있는 BEA사의 웹로직(WebLogic) 서버를 이용하였다. 빈의 생성을 위해서는 서버 측 프로그램으로 작성되어 생성된 클래스 파일들은 배치 디스크립터 파일과 함께 .jar 파일로 묶어서 압축을 해야 한다. 그리고 .jar 파일은 .ear 파일에 포함이 되어 빈으로 생성되어진다. 그 후 Web Logic Server의 Deployments의 Applications에서 생성된 Products.jar 파일을 찾아서 불러오고 'Upload' 버튼을 누르면 Deploy가 된다.



(그림 22) Bean 배치



(그림 23) 생성된 Bean 추가

(그림 23)에서는 Deployment의 Application에 'product'와 EJB 노드에 'products'가 생성되어 있음을 확인 할 수 있다. 여기서 생성된 빈은 클라이언트에서 인터페이스 하여 사용이 가능하다.

4.5 평 가

EJB 기반 빈 생성을 위한 Data Source의 연동에서 다양한 DBMS를 선택하여 프로그램을 개발할 수 있다. 또한 이미 생성된 컴포넌트들을 현재의 시스템 환경에 맞게 수정해야만 하는 경우가 빈번히 발생할 수 있어서 본 논문에서는 기존의 DAO 패턴을 개선하여 공통의 클래스와 인터페이스를 합성시키고 있다. 그리고 <표 2>에서와 같이 사례 연구를 통한 패턴의 비교에서 그 결과를 볼 수 있다. 특히 엔티티빈 클래스의 DBMS의 연동부분을 Abstract 클래스와 Concrete 클래스로 분리하여 전자에는 템플릿 메소드에서 사용하는 추상메소드를 선언하고, 후자는 전자에서 선언한 메소드들을 구현하였다. 이 제안은 클래스의 재사용에 많은 이점과 함께 다른 DBMS 환경에서도 쉽게 적응할 수 있는 방법으로 적용이 가능하다.

〈표 2〉 사례 연구를 통한 패턴 비교

적용패턴	Abstract Class 수	Concrete Class 수	Create Class 수
DAO패턴	1	4	16
제안패턴 (DAO패턴개선)	1	4	4
적용패턴	인터페이스 수	DBMS연결	빈 배치
DAO패턴	4	엔티티빈에 종속	수정 어려움
제안패턴 (DAO패턴개선)	1	엔티티빈에 독립	수정 용이

* 참조 : (그림 6)을 사례 연구 비교에 적용

5. 결론 및 향후연구

J2EE의 EJB 기반 어플리케이션 개발은 빈의 생성에 있어서 상당한 지식과 프로그램 작성의 능력뿐만 아니라 인내가 요구되어진다. 그러나 소프트웨어의 발전과 함께 컴포넌트에 대한 새로운 인식 및 중요성이 대두되고 있다. 특히 소프트웨어 재사용 측면에서는 컴포넌트의 개발에 대한 중요성이 무엇보다 우선시 되어진다. 컴포넌트에 사용되는 언어 및 도구들이 매우 다양하지만 java 기반의 EJB는 무엇보다도 플랫폼에 독립적으로 운용이 가능하기 때문에 어플리케이션 시스템을 구축하는데 매우 원활하게 사용될 수 있다.

그리고 빈의 생성에서는 일반적으로 엔티티빈에 의한 영속적인 데이터의 사용이 많이 이용되고 있다. 이는 대부분 동일한 DBMS를 사용한 데이터베이스 연동이어서 각각의 엔티티빈에서 DB에 Access하고 있다. 그러나 Data Source의 환경이 DBMS의 사용 종류에 따라 매번 바꿔야만 하는 문제점들이 발생되고 있어 시스템에 대한 퍼포먼스 및 유지보수 그리고 이식성 등에서 문제점으로 작용되어 시스템의 성능저하로 이어 질 수 있다. 이를 보완하기 위해서 본 논문에서는 일반적으로 사용되고 있는 4개의 DBMS를 선정하여 엔티티빈 클래스에서 연동이 가능하도록 각각의 연결 프로그램을 DAO 패턴기법을 이용하여 분리시켜 작성하고 있다. 그 결과 사용되는 데이터베이스에 따라 프로그램을 재작성해야 하는 불편함 뿐만 아니라 개발자로 하여금 어렵거나 또 다른 문제점의 발생에 대한 예방책과 소프트웨어 재사용 측면에서도 매우 유용하게 이용될 수 있도록 하였다.

마지막으로 향후 추가적인 연구는 본 논문에서 BEA사의 웹로직(WebLogic)을 이용하여 제시하고 구현한 부분을 좀 더 발전시켜, 다른 서버의 환경에서도 적용할 수 있고 상용화할 수 있는 방법으로 사용되도록 하고, 동일패턴 내에서 공통의 클래스를 비교하여 추출하고 합성 및 조립에 필요한 다양한 연구가 이루어져야 할 것이다.

참 고 문 헌

[1] Software Engineering : A practitioner's approch, McGraw-Hill International Edition, 1997.

[2] Fichman, R. G. and C. F. Kemerer, "Object-Oriented and Conventional Analysis and Design methodologies," Computer, Vol.25, No.10, pp.22-39, Oct., 1992.

[3] Gamma, E. et al., Design Patterns, Addison- wesley, 1995.

[4] Gamma, E., et al., Design Patterns : Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[5] Floyd Marinescu, EJB Design Pattern, Wiley, 2002.

[6] Krishnan Subramanian, EJB Design Patterns : Designing EJBs for maximum re-useablity, compactness and flexiblity.

[7] 이돈양, 이창수, 송영재, "EJB Persistence Pattern을 이용한 효과적인 엔티티빈 설계", 한국정보과학회, 2002 가을 학술발표논문집(II), 2002.

[8] Dirk Riehle, "Composite Design patterns," OOPSLA '97, 11, pp.218-228, 1997.

[9] Mark Grand, "Patterns in Java," WILEY, 1998.

[10] E. Gamma, R. Helm, R. Johbson and J. Vlissides, "Design Pattern : Elements of Reuseable Object-Oriented Software," Addison-Weley, 1995.

[11] http://java..sun.com/blueprints/Corej 2eepatterns/Patterns/TransferObject.html.

[12] http://java..sun.com/blueprints/Corej 2eepatterns.

[13] http://www.devx.com/java/wrox/5288-chap03.pdf.

[14] http://www.hyuki.com/dp/.

[15] Riehle, Dirk, "Composite Design Patterns," In Proceedings of the 1997. Conference on Object-Oriented Programming System, Language and Application(OOPSLA '97) ACM Press, pp.218-228, 1997.

[16] Yoder, Joseph, Ralph Johnson and Quence Wilson. "Connecting Business Objects to Relational Databases," Pattern Languages of Programmning Conference Procedding, 1998. http://jerry.cs.uiuc.edu/~plop/plop98/final-submissions/p51.pdf.

이 돈 양

e-mail : dylee6211@hanmail.net

1987년 대구대학교 통계학과(이학사)

1993년 경희대학교 전자계산학과(공학석사)

2002년 경희대학교 전자계산공학과(박사수료)

1995년~2002년 대한상공회의소

2003년~현재 경인여대 전산정보 겸임교수

관심분야 : EJB, 디자인패턴, XML, OOD, 소프트웨어 재사용

송 영 재

e-mail : yjsong@khu.ac.kr

1969년 인하대학교 전자공학과(공학사)

1976년 일본 keio 대학교 전산학과(공학석사)

1980년 명지대학교 전산학과(공학박사)

1976년~현재 경희대학교 컴퓨터공학과 교수

관심분야 : 소프트웨어재사용, CASE 도구