효율적인 EJB 컴포넌트화를 위한 *Integrated DAO 패턴*

최성만° * 김정옥* 이정열** 유철중* 장옥배*
*전 북 대 학 교 컴 퓨 터 과 학 과
**정 인 대 학 사 무 자 동 화 과
sm3099@cs.chonbuk.ac.kr
{kjo3852, ljy8383}@hanmail.net
{cjyoo, okjang}@moak.chonbuk.ac.kr

Integrated DAO Pattern for Efficient EJB Componentization

Seong-Man Choi° * Jeong-Ok Kim* Jeong-Yeal Lee** Cheol-Jung Yoo* Ok-Bae Chang*
*Dept. of Computer Science, Chonbuk National University
**Dept. of Office Automation, Chongin College

요 약

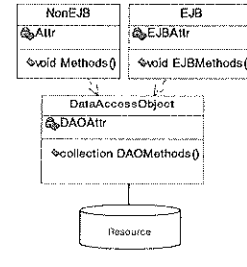
EJB는 표준 서버측 컴포넌트 표준 모델로서 객체지향 분산 애플리케이션의 개발 및 분산 배치를 위한 컴포넌트 아키텍처이다. EJB로 시스템을 구축시 프로그램의 개발을 쉽게 하고 단순화시키며 보안성, 영속성, 동시성, 트랜잭션 무결성, 보안 등의 처리를 자동으로 해주는 이점을 가진다. 또한 EJB 컴포넌트화 설계를 위해 디자인 패턴을 이용하면 설계 범위를 확장할 수 있고, 설계의 재사용성 효과를 높여주며, 설계 시간의 단축 및 의사소통에 대한 시간을 효과적으로 줄일 수 있다. 본 논문에서는 기존 시스템에서 데이터베이스 접근을 캡슐화하는데 이용하는 DAO의 트랜잭션 로직의 복잡성과 불필요한 DAO 생성 및 시스템 과부하의 문제점을 해결하고자 *Integrated DAO 패턴*을 제안한다. Integrated DAO 패턴은 컨테이너 관리 트랜잭션을 통해 트랜잭션 조작에 관한 복잡성을 줄여주기 때문에 시스템의 과부하 감소와 시스템 성능 향상에 효과가 있다.

1. 서론

컴포넌트 개발 방법은 비즈니스 환경 및 기술 변화를 수용할 수 없는 현 기술의 한계와 업무 통합 또는 분산 환경, 이식성, 경량화 요구, 개발자 개인의 능력과 비자동화된 개발 절차에 따른 문제점 등의 한계에 따라 필요성이 제기되었다. 컴포넌트 개발 방법은 설계와 구현 및 재사용을 통하여 생산성을 향상시키고 효율적으로 테스트된 코드를 사용함으로써 신뢰성을 증가시킬 수 있다[1]. 본 논문에서는 서버측 컴포넌트 모델인 EJB(Enterprise JavaBeans)를 이용한 기존 시스템에서 DAO(Data Access Object)에 적용된 디자인 패턴에 대해서 알아보고, 또한 기존 시스템에서 지적된 DAO의 트랜잭션 로직의 복잡성과 불필요한 DAO 생성 및 시스템의 과부하를 해결하고 개선하려는 Integrated DAO 패턴을 제안하고자 한다. 2장에서는 기존 시스템의 DAO를 분석해 보고, 3장에서는 DAO와 Integrated DAO 패턴에 적용된 디자인 패턴에 대해 설명하며, 4장에서는 제안한 Integrated DAO 패턴의 구조에 대해 설명한 후 이를 DAO 패턴과 비교 평가한다. 마지막 5장에서는 결론 및 향후 연구과제를 제시한다.

2. 기존 시스템의 DAO 분석

객체로서 XML 데이터를 표현하고 데이터베이스 벤더의 독립적인 목적을 위해 기존 시스템에서는 DAO(Data Access Object)를 이용한다. DAO는 데이터베이스 접근을 캡슐화하는데 이용되며, 대규모 배치에 적합하고, 독립적인 리소스 벤더와 독립적인 리소스 구현 및 빈 관리 퍼시스턴스에서 컨테이너 관리 퍼시스턴스까지의 이동을 용이하게 해주는 이점을 가진다. [그림 1]은 이러한 DAO 구조를 보여주고 있다.



[그림 1] DAO 구조

[그림 1]을 통하여 알 수 있듯이 DAO의 참여자(participants)는 NonEJB, EJB, DataAccessObject, Resource 등이 있다. NonEJB 클래스는 EJB-required()를 제외한 비즈니스 메소드를 인스턴스화시켜 DAO를 이용하며, EJB 클래스는 EJB-required()를 담당하며 비즈니스 메소드가 호출될 수 있는 환경을 제공한다. DataAccessObject 클래스는 리소스에 대한 오퍼레이션 추상화와 특정한 타입의 데이터 처리와 접근하기 위한 표준 API를 제공한다. Resource는 임의적인 API 방법에 의한 검색과 영속적인 리소스 데이터를 ConcreteDataObject에 제공한다.

DAO의 예제 애플리케이션으로 Catalog 세션 빈의 구현은 CatalogImpl에서 상속받은 CatalogEJB에 의해 제공된다. 다음 코드는 Catalog 세션 빈에 대한 구현으로 CatalogDAO()에서는 요청된 데이터를 찾기 위해 데이터베이스에 실제적으로 접근하며, dao.getProducts()는 요청된 데이터에 만족한 결과를 반환하는 역할을 한다.

```
public Collection getProducts(String categoryId, int startIndex
int count) {
   Connection con = getDBConnection();
   try {
      CatalogDAO dao = new CatalogDAO(con);
         return dao.getProducts(categoryId, startIndex, co
   } catch ( SQLException se) {
      throw new GeneralFailureException(se);
   } finally {
      try {
         con.close();
      } catch (Exception ex) {
         ..... )
      }
   }
```

CatalogImpl.getProducts에 대한 코드와 대응되는 DAO 객체의 코드를 아래에서와 같이 보여주고 있다.

```
public Collection getProducts(String categoryId, int startInde
int count) throws SQLException {
   String qstr = "select itemid, listprice, unitcost, "+...+"orde
                 by name";
   ArrayList al = new ArrayList();
   Statement stmt = con.createStatement()
   ResultSet rs = stmt.executeQuery(qstr);
   HashMap table = new HashMAp();
   while (startIndex --> 0 && rs.next());
   while (count --> 0 && rs.next()) {
      int i = 1;
      .....
      Product product = null;
      if (table.get(productid) == null) {
         product = new Product(productid, name, descn);
         table.put(productid, product);
         al.add(product); }
   }
   rs.close();
   stmt.close();
   return al;
}
```

Catalog 세션 빈의 예제와 같이 모든 컴포넌트는 해당 컴포넌트마다 각각의 DAO를 생성한다. 이와 같이 여러 개로 나누어진 DAO는 데이터 조작시 서로 다른 트랜잭션에서 조작이 이루어지기 때문에 트랜잭션 로직의 복잡성으로 인하여 시스템에 과부하를 초래하게 된다. 또한, 각 빈과 연결된 DAO는 빈이 인스턴스화 될 때 동시에 인스턴스화 되어 불필요한 DAO가 생성된다. 따라서 많은 메모리를 차지하는 단점을 유발한다. 본 논문에서는 이러한 문제점을 해결하기 위해 Integrated DAO 패턴을 제안하고자 한다.

3. Integrated DAO 패턴에 적용된 디자인 패턴

기존 시스템의 DAO와 관련된 패턴으로는 다음과 같은 패턴들이 있다.

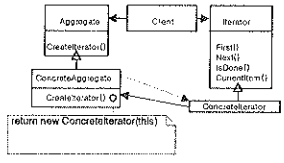
- **Adapter pattern** : Adapter 패턴은 클래스의 인터페이스를 그 클래스가 사용하는 클라이언트에 적합한 다른 인터페이스로 바꾸어야 할 때 사용한다. Adapter는 클라이언트 클래스에 기능을 제공하기 위한 중간 클래스의 역할을 한다. 어댑터 패턴을 이용하면 코드의 재사용성을 높일 수 있다[2, 3].
- **Facade pattern** : Facade 패턴은 클라이언트 입장에서 접근해야 되는 클래스가 많은 경우 Facade 클래스를 하나 만들어 이 클래스를 통해 모든 접근을 하게 된다. 하부구조의 인터페이스 집합에 대한 통합된 인터페이스를 제공해야 할 때 이용하며, Facade 패턴을 랩퍼(Wrapper) 또는 헬퍼(Helper)라고도 한다. 클라이언트와 내부 구현 클래스간의 결합도를 최소화시켜주기 때문에 클라이언트에 영향을 주지 않고도 자유롭게 내부 구현 클래스를 바꿀 수가 있다는 이점이 있다[2, 3].
- **Proxy pattern** : Proxy 패턴은 다른 많은 패턴들 중에서 가장 일반적인 패턴으로 단순한 객체에 의해 복잡한 객체를 표현하기를 원할 때 클라이언트가 직접 구현 클래스를 접근하지 않고, 프록시라는 대리자를 통해 접근한다. 큰 이미지와 같은 객체를 호출하는데 유용하다[2, 3].
- **Flyweight pattern** : Flyweight 패턴은 많은 수의 매우 작은 객체들을 효과적으로 공유할 필요성이 있을 때 이용한다. Flyweight 패턴은 객체의 수를 최소화하는 것이 목적이다[2, 3].

DAO에서는 트랜잭션 로직의 복잡성과 빈이 인스턴스화 될 때 동시에 인스턴스화 되어 DAO가 생성므로 불필요한 DAO로 인해 메모리 낭비를 초래하게 되며, 기능성이 저하된다. 또한 다수의 사용자에 의해 호출될 경우 시스템 과부하가 발생한다.

이러한 문제점을 해결하고 개선하기 위한 방법으로 본 논문에서 제안할 Integrated DAO 패턴에 적용될 또 다른 디자인 패턴은 *Iterator* 패턴으로서 그 내용은 다음과 같다.

- **Iterator pattern** : Iterator 패턴은 디자인 패턴

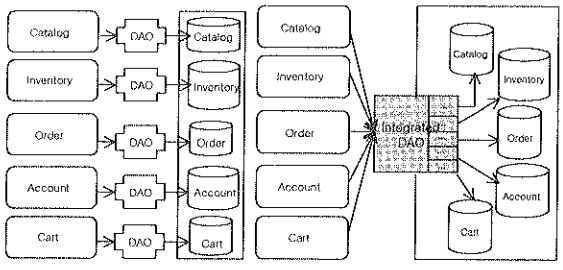
중에서 가장 많이 이용되며, 가장 간단한 패턴중의 하나로서 콜렉션에서 객체를 순차적으로 접근하기 위해 메소드를 선언한 인터페이스를 정의한다. 그러한 인터페이스를 통해 단지 콜렉션을 접근하는 클래스는 그 인터페이스를 구현하는 클래스와 독립적으로 존재한다. Iterator 패턴은 데이터 집합에 대해 수행하는 방법을 나타내지 않고 데이터 요소의 집합을 통해서 정의된 방법을 제공하는 이점을 가지고 있다[2, 3]. [그림 2]는 Iterator 패턴의 구조를 보여주고 있다.



[그림 2] Iterator 구조

4. Integrated DAO 패턴의 구조 및 비교 평가

컴포넌트화를 개발하려고 하는 것은 복잡한 시스템을 개발할 때 매우 고도화된 코드와 디자인 패턴을 효과적으로 재사용 할 수 있도록 하고자 함이다[4]. [그림 3]은 기존의 DAO 시스템으로 각각의 컴포넌트에 해당하는 DAO를 생성하여 데이터베이스에 접근하는 방식이기 때문에 트랜잭션 로직의 복잡성을 증가시키고 각 빈과 연결된 DAO는 빈이 인스턴스화 될 때 동시에 인스턴스화 되어 생성되기 때문에 불필요한 DAO를 생성하게 된다. 결과적으로 시스템의 성능을 저하시키며, 메모리를 낭비하는 원인이 된다.



[그림 3] DAO [그림 4] Integrated DAO

[그림 4]는 제안한 Integrated DAO 패턴으로서 DAO의 복잡한 트랜잭션 로직과 불필요한 DAO 생성 및 시스템의 과부하를 해결하기 위해 개선한 패턴이다. 각각의 컴포넌트에서 Integrated DAO 패턴에 접근하면 각 데이터베이스의 특징을 가진 Integrated DAO에서는 해당 데이터베이스에 실시간(real-time)으로 접근한다. 또한, Integrated DAO 패턴은 다수의 DAO가 작업하는 내용을 트랜잭션으로 관리하는 것보다 통합된 Integrated DAO 패턴을 이용하는 것이 컨테이너 관리 트랜잭션에서 트랜잭션 조작에 관한 복잡성을 줄여주기 때문에 결과적으로 시스템의 과부하를 줄여주는 효과가 있다.

또한, 시스템의 성능과 확장성 및 효율성을 증대하는 효과를 기대할 수 있다. 기존의 DAO와 본 논문에서 제안한 Integrated DAO의 예상되는 효과를 분석해보면 [표 1]과 같다.

[표 1] DAO와 Integrated DAO 비교

패턴 / 내용	DAO	Integrated DAO
시스템 과부하	크다	작다
트랜잭션 로직	복잡하다	단순하다
호출시간	많다	적다
메모리 효율성	나쁘다	좋다
불필요한 DAO	생성한다	생성하지 않는다
기능향상 및 확장성	나쁘다	좋다

5. 결론 및 향후 연구과제

디자인 패턴을 이용하여 컴포넌트화를 하면 알고리즘이 더욱 간결하고 접근하기 쉬운 개념으로 생성된다[5]. 본 논문에서는 기존 시스템의 DAO에서 지적된 DAO의 트랜잭션 로직의 복잡성과 불필요한 DAO 생성 및 시스템의 과부하를 해결하기 위해 DAO와 관련된 패턴 외에 Iterator 패턴을 추가한 Itegration DAO 패턴을 제안하였다. Itegration DAO 패턴을 이용하면 트랜잭션 로직을 단순화시키며, 불필요한 DAO 생성을 하지 않으므로 시스템의 성능과 메모리의 효율성을 증대시키는 장점을 가진다.

향후 연구과제로는 생성된 디자인 패턴과 애플리케이션에 종속된 제약사항을 만족하는지를 검증하고, 제안한 Integrated DAO 패턴의 예상되는 효과에 대한 구체적인 측정 연구가 필요하다. 또한, 제안한 패턴을 시스템에 적용해보며 SI 업계에서 EJB 컴포넌트화 개발에 더욱 적합한 디자인 패턴을 제안하도록 지속적인 연구가 필요할 것이다.

6. 참고 문헌

[1] 서동수, 홍기형, "컴포넌트 기반 소프트웨어 개발 프로세서", 정보과학회 소프트웨어공학회지, 제12권 3호, pp.49-56, 1999.

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns : Elements of Reusable Object-Oriented Software", Addison-Wesley Longman, Inc., 1995.

[3] Mark Grand, "Patterns in Java, Volume1 : A Catalog of Reusable Design Patterns Illustrated with UML", John Wiley & Sons, Inc., 1998.

[4] Jun Ginbayashi, Rieko Yamamoto, Keiji Hashimoto, "Business Component Framework and Modeling Method for Component-based Application Architecture", in Proceedings of EDOC, pp.184-193, 2000.

[5] Stephen S. Yau, Ning Dong, "Integration in Component-Based Software Development Using Design Patterns", in Proceedings of COMPSAC, pp.369-374, 2000.