Object Persistence 를 위한 EJB BMP 와 CMP 기법의 비교 평가

정광선 김수동
숭실대학교 컴퓨터학과
ksjung@otlab.ssu.ac.kr sdkim@comp.ssu.ac.kr

Comparison of EJB BMP and CMP Persistence

Kwang Sun Jung, Soo Dong Kim
Dept. of Computing, Soongsil University

요 약

Enterprise JavaBeans(EJB)에서 영속적인 데이터들을 표현하고 관리하는 엔티티빈의 두 가지 구현 방법은 Bean Managed Persistence(BMP)와 Container Managed Persistence(CMP)기법이 있다. 이렇게 엔티티빈이 데이터 베이스와 연결되어 사용되기 때문에 생기는 특성이 있는데, 이런 특성들이 BMP 와 CMP 를 사용하여 구현, 운용, 유지보수 및 재사용을 할 때 많은 차이점을 만든다. EJB 를 사용하는 소프트웨어 산업계에서, 이런 차이점에 대한 연구가 부분적으로는 이루어지고 있지만, 체계적인 비교 평가는 아직 미흡한 실정이다. 본 논문에서는 BMP 와 CMP 를 이용해서 빈을 구현할 때 나타나는 각각의 장점과 단점을 몇 가지 측면에서 살펴본다. 그리고 그 차이를 비교함으로써 BMP 와 CMP 기법을 평가한다.

1. 서론

빠르게 변하는 어플리케이션 요구 사항에 따라, 분산환경 기반의 클라이언트/서버 모델에서 서버측 어플리케이션을 더욱 빨리 개발하려는 노력이 계속되었다. 이런 노력은 비즈니스 로직에만 집중할 수 있도록 지원을 해주며 서버 플랫폼에 독립적인 EJB를 등장시키게 했다[1]. EJB는 J2EE에 기반해 있기 때문에 J2EE가 가지는 특성들 즉, 플랫폼 독립성, 이식성, 다중사용자, 보안 그리고 표준 엔터프라이즈 클래스 플랫폼 등의 개념을 지원한다.

이러한 기능은 데이터를 생성하고 분산된 데이터들을 관리하는데 있어서 안정성과 효율성을 제공한다[2]. 분산된 데이터 베이스의 영속성을 구현하고 관리하기 위해서 EJB에서 사용하는 엔티티빈은 두 가지 유형으로 구현된다. 그 한 가지인 BMP에서는 상황에 유연하게 작성될 수 있으며, CMP는 구현과 관리면에서 효율성을 제공한다. EJB의 초기 버전에서는 CMP에서 지원하는 기능들이 제한적이었지만 최근의 EJB 규격에서는 추상적 영속 스키마의 명세로 인해서 빈들간의 관계성을 컨테이너가 관리하게 됨으로써 더욱 폭넓은 상황에 CMP를 적용할 수 있다[3].

본 논문에서는 EJB의 BMP와 CMP를 여러 면에 걸쳐 비교함으로써 어떤 장단점이 있는지를 살펴보고, 그 사이에서 나타나는 차이점을 통해 BMP와 CMP 기법을 평가한다.

2. 관련 연구

2.1. EJB 의 아키텍처와 엔티티빈

EJB는 컴포넌트를 기반으로하는 분산 환경에서 서버측을 위한 아키텍처로써 썬 마이크로시스템에서 제공하고 있다. EJB는 새로운 프로그램 컴포넌트가 추가되거나 또는 변경될 때마다, 각 개별 클라이언트들을 갱신하지 않고서도 서버에서 변화를 통제할 수 있도록 하는 이점을 제공한다. 또한 EJB 컴포넌트들은 다중 응용프로그램들에서 재사용되는 장점을 가지고 있으며 EJB 빈이나 컴포넌트는 컨테이너라고 불리는 특정 응용프로그램내에 배치되어 사용되어야 한다.

EJB에서 구현하는 빈의 유형은 두 가지가 있는데 각각 세션빈과 엔티티 빈이라고 한다. 일반적으로 세션빈은 컴포넌트의 워크플로우를 담당하며 일시적인 성질을 지니고, 엔티티빈은 컴포넌트에서 사용하는 영속적인 데이터 모델을 담당하여 지속성을 가지고 있다[3]. 엔티티빈은 빈의 관리적인 측면에서 다시 BMP와 CMP로 나누어진다.

2.2. BMP

개발자가 빈의 속성과 관계들을 빈 내부에서 명시적으로 관리하는 것이다. 즉, 빈 내부에서 빈의 변수들을 선언하고 다른 빈들과의 연동 시 관계성을 고려하여야 하며 JDBC API 를 이용하여 특정 데이터 베이스에 접근하는 질의들을 작성해 주어야 한다. 개발자의 능력에 따라 코드를 효율적으로 작성하고 논리적 오류를 줄일 수 있다.

2.3. CMP

개발자가 데이터 베이스에 관련된 코드들을 빈 내부에 작

성해 주지 않고, Deployment Descriptor에 요구되는 몇 가지 정보들을 작성함으로써, 컨테이너가 데이터 베이스에 접근하는 코드들을 자동으로 생성하게 된다. 따라서 개발자는 데이터 베이스등의 하부 층에 독립적으로 비즈니스 로직에만 신경쓰면 된다. 빈 내부에 빈에 관한 스키마와 관계를 나타내는 코드가 없는 CMP는 더욱 여러 환경에서 재사용하기 좋다[3]. 이를 위해서, 개발자는 빈을 추상 클래스로 만들어 주어야 하고 Deployment Descriptor에 CMP 필드와 Containmer Managed Relationship(CMR) 필드를 작성해야 한다[3].

3. BMP 와 CMP 의 Criteria

3.1. 재사용성

EJB는 컴포넌트 기반의 아키텍처이기 때문에 EJB로 개발된 빈들은 빈의 인터페이스를 만족하기만 하면 새로운 어플리케이션을 개발할 때 재사용된다. 새로운 환경에서 엔티티 빈을 재사용되려면 어플리케이션 서버와 데이터 베이스에 대해 독립성이 보장되어야 한다.

BMP의 경우 특정 데이터 베이스에 종속된 질의문들을 빈 클래스 내에 직접 작성한다. 따라서 다른 데이터 베이스 환경에서 재사용할 때, 빈 내부의 메소드들에 포함된 질의문들을 모두 수정해 주어야 한다. 또한, 배치될 기존의 시스템에서 사용하는 데이터 베이스 스키마에 따라 빈 내부에서 사용하는 모든 변수들을 수정해 주어야 한다.

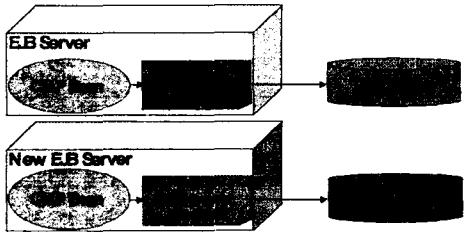

그림 1. 다른 환경에서 재사용되는 CMP Bean

어플리케이션을 개발할 때, CMP에 의해 작성된 엔티티빈을 재사용할 경우에는, 기존의 빈을 수정할 필요 없이 EJB서버와 데이터 베이스에 관한 정보 및 스키마에 관련된 정보를 빈의 Deployment Descriptor 내의 요소(Element)에 명시해 주기만 하면 된다[3]. 그림 1에서처럼 기존에 쓰이던 하나의 동일한 빈은 수정된 Deployment Descriptor를 통해 새로운 서버와 데이터 베이스에서 재사용된다.

빈 내의 여러 메소드들에 포함된 질의들과 속성들을 수정하는 것은 많은 추가 비용과 시간을 요구하여 재사용성이 떨어진다. 따라서 환경에 관한 정보만을 수정함으로써 재사용할 수 있는 CMP가 더욱 재사용성이 높다.

3.2. 개발 효율성

BMP는 개발자가 JDBC의 API의 코드와 질의를 프로그램상에서 직접 작성해 주어야 한다.

```
public Collection ejbFindBigAccounts(double balanceGreaterThan) {
  try{
    con = getConnection();
    ps = con.prepareStatement("select id from ejbAccounts where bal > ?");
    ps.setDouble(1, balanceGreaterThan);
    ps.executeQuery();
    ResultSet rs = ps.getResultSet();
    while (rs.next()) {
      pk = rs.getString(1);
      v.addElement(pk);
    }
    return v;
    } catch () {
    } finally {
      cleanup(con, ps);
    }
}
```

그림 2. BMP에서 ejbFind의 구현

그림 2에서, BMP기법은 데이터 베이스에 접근이 필요할 때 접근과 질의에 대한 절차적인 코드가 필수적으로 요구된다. 이런 코드가 데이터 베이스를 사용하는 모든 메소드에 중복되어 존재하고 있다. 빈의 개발 시에 소요되는 비용과 기간을 고려할 때, 이런 중복 코드는 비효율적 이다.

```
public void ejbFindBigAccounts() {
}
```

그림 3. CMP에서 ejbFind의 구현

반면 그림 3처럼 CMP기법에서는, 개발자가 빈의 코드 내에 데이터 베이스에 관한 어떠한 코드도 작성해 주지 않고 그림 4처럼 Deployment Descriptor에서 메소드명과 질의, 그리고 그 질의에 포함될 매개변수만 정의해 주면 된다.

```
<query>
    <query-method>
      <method-name>findBigAccounts</method-name>
      <method-params>
        <method-param>double</method-param>
      </method-params>
    </query-method>
    <ejb-ql>
      <![CDATA[SELECT OBJECT(a) FROM AccountBean AS a
                                WHERE a.balance > ?1]]>
    </ejb-ql>
</query>
```

그림 4. CMP Deployment Descriptor 에서의 구현

이러한 차이로 CMP 개발자는 코드의 작성보다는 비즈니스 로직에 집중할 수 있으며 그에 따라 더 빠른 개발이 이루어지고 비용과 기간을 줄일 수 있다. 또한 BMP에서 이루어질 수 있는 기술적 위험 즉, 코딩상의 오류나 작성 실수 등의 위험 요소도 줄어든다. 기술적 위험이 작아짐으로써 개발 비용과 기간은 더욱 단축된다. 따라서, BMP와 CMP를 개발 비용과 기간 및 기술적 위험 면에서 비교해 볼 때 CMP로의 작성이 매우 효율적임을 볼 수 있다.

3.3. 성능

빈 인스턴스를 생성하고 관리하기 위해서는 데이터 베이스를 호출하여 조건에 맞는 레코드 정보를 가져와야 한다. 이때 데이터 베이스에 대한 호출은 많은 시스템 자원을 요구하며 어플리케이션의 성능에 중요한 영향을 미친다.

빈이 데이터 베이스에 접근하여 엔티티빈의 인스턴스를 생성할 때, 우선 Find 메소드를 통해서 프라이머리 키(PK)들을 찾아낸다. 그리고 반환된 PK들을 이용해서 데이터 베이스로 다시 접근한다. 이렇게 얻어낸 레코드로 인스턴스의 필드 값들을 설정함으로써 인스턴스를 생성한다.

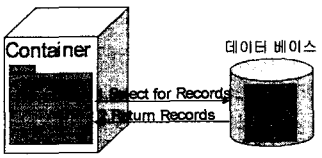

그림 5. BMP와 CMP의 인스턴스 생성을 위한 DB 호출

CMP에서는 비트 단위의 컨테이너 식별 플래그를 이용해서 다수의 레코드를 읽어오는 한번의 SELECT로 다수의 인스턴스를 적재할 수 있다. 그림 5처럼 ejbFind() 메소드의 경우, 인스턴스를 생성하기 위한 데이터 베이스로의 호출은 단 한번뿐이다. 생성할 인스턴스의 수를 N개라 할 때 일반적인 접근으로는 1+N번의 데이터 베이스 접근을 필요로 하지만, CMP의 최적화를 통해 단 한번의 접근으로 동일한 인스턴스를 생성할 수 있다[5]. 또한 CMP에서는 CallBack메소드와 비즈니스 메소드의 그룹핑을 통해서 단위 기능 수행에 필요한 일부 속성들만을 적재하는 Lazy Loading를 지원한다. 이를 통해서 DB 스키마를 효과적으로 적재한다.

CMP가 위의 두 가지 기법 외에 여러 장치들을 제공함으

로써 엔티티빈을 간편하게 최적화 시켜주지만, BMP기법에서 개발자가 코딩상에서 직접 최적화를 해주는 것처럼 융통성 있지 못하다. BMP기법에서는 데이터 베이스에 접근하고 질의를 구현하는데 사용되는 여러 뛰어난 코딩 기법을 통해 CMP에서 제공하는 여러 기법들에 의한 최적화보다 더 나은 성능을 가지도록 구현할 수 있다.

3.4. 엔티티빈 행동 모니터링

엔티티빈을 모니터링 하기 위해서는 빈이 수행하는 메소드들을 단위로 하며 그 메소드들 간이나 하나의 단위 메소드를 모니터링하게 된다. BMP에서는 개발자가 빈이 수행하는 모든 메소드들을 직접 작성하기 때문에 모니터링 하고자 하는 메소드의 어느 곳에서나 원하는 모니터링을 수행할 수 있다. 예를 들면한가지 방법으로 모니터링 데몬을 띄워놓고 빈 내의 임의의 위치에서 데몬을 호출하여 빈의 상태를 모니터링하는 방법이 있다.

CMP에서는 메소드 단위의 모니터링은 가능하지만 BMP에서 처럼 메소드내의 어느곳에서나 유연하게 모니터링을 수행할 수는 없다. 개발자가 추상 메소드에 대해서 임의의 위치에 코드를 삽입할 수는 없기 때문이다. 이것은 EJB 서버 공급자가 컨테이너가 빈을 모니터링 할 수 있도록 하는 기능을 지원하기 전까지는 CMP에서 불가능하다.

3.5. DB 스키마 변화에 대한 적응성

개발된 빈은 환경 변화나 고객의 요구에 따라 수정된다. 데이터의 스키마가 변할 경우, BMP의 경우는 수정한 필드에 관련된 빈 내부의 모든 로직을 찾아서 일일이 수정해야 한다. 그러나, 그림 6처럼 CMP에서는 Deployment Descriptor의 CMP 필드 수정으로 모든 작업을 완료할 수 있다.

```
<cmp-field>
        <field-name>accountId</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>balance</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>accountType</field-name>
      </cmp-field>
      <primkey-field>accountId</primkey-field>
```

그림 6. CMP의 Deployment Descriptor 내의 CMP 필드

또한, 배치되어 사용하는 중에 다른 DBMS나 다른 버전으로의 환경 변화가 있을 때, BMP에서는 특정 스키마와 특정 데이터 베이스의 API에 종속된 질의를 사용하기 때문에 빈 내의 모든 질의들을 수정해 주어야 한다. 그러나 CMP는 Deployment Descriptor 상에서 데이터 베이스에 관한 명세를 담당하는 요소(Element)만 바꾸어 주면 된다.

따라서, BMP와 CMP를 유지 보수 측면에서 비교할 때, 어플리케이션 서버와 데이터 베이스에 독립적인 CMP가 BMP보다 유지 보수 비용이 적게 든다. 기업에서 사용하는 서버급 어플리케이션들의 규모가 크다는 경향을 고려할 때 이런 차이는 더욱 커진다.

4. 결론

본 논문에서는, 영속적인 데이터들을 표현하기 위해 EJB에서 제공하고 있는 엔티티빈을 구현하기 위한 두 가지 기법인 BMP와 CMP를 여러 측면에서 비교했다.

표 1. BMP 와 CMP 의 비교 평가 결과

	BMP	CMP
재사용성		우수
개발 효율성		우수
성능	우수	
엔티티빈 행동 모니터링	우수	
스키마 변화에 대한 적응성		우수

표 1처럼 3장에서 비교한 5가지 항목중 엔티티빈 행동 모니터링과 성능을 제외한 모든 면에서 CMP가 우수한 것을 확인할 수 있다. BMP는 개발자가 직접 모든 코드를 작성하기 때문에 매우 융통성 있는 개발이 가능하지만 여러 프로그램 기법에 능숙하지 못하면 효율적인 어플리케이션을 만들기 어렵고 유지보수성과 재사용성이 떨어진다. 이것은 환경 변화에 대한 형상관리와 어플리케이션의 개발에 있어서 신속성을 떨어뜨리고 비용을 증가 시킨다.

이와 같이 5가지 항목에 대한 비교 결과를 고려할 때, 컴포넌트 기반의 어플리케이션 개발시 EJB의 엔티티빈을 구현함에 있어서 CMP기법을 우선적으로 고려 하는 것이 좋다. BMP기법은 BackEnd 시스템의 지원 기능을 현재의 CMP기법으로는 사용하기 힘든 상황이거나, 뛰어난 개발자들을 보유한 상태에서 재사용 및 유지보수를 염두에 두지 않고 성능에만 초점을 두는 상황에 융통성 있는 개발 방법으로써 사용해야 한다. 그러나 대부분의 어플리케이션들이 대형화 되고 시스템 통합에 의한 유지보수와 재사용 측면이 강조되는 현재의 상황에서 성능에만 치우친 어플리케이션 개발은 현실성이 떨어진다. 따라서, CMP기법의 사용을 적극 권장한다.

5. 참고문헌

[1] Adatia R., Arni F., Gabhart K., Griffin J., Juric M., Lott J, McAllister T., Mulder A., Nagarajan N., Connor D., Osborne T., Sarang P., Tost A., Young D., Berry C., Professional EJB, Wrox Press, 2001.

[2] Alur D., Crupi J., Malks D., Core J2EE Patterns, Prentice Hall PTR, 2001.

[3] *Enterprise JavaBeans™ Specification, Version 2.0*, Sun Microsystems, 2001.

[4] Roman E., Ambler S., Jewell T., *Mastering Enterprise JavaBeans Second Edition*, Wiley & Sons, Inc., 2002.

[5] Girdley M., Woollen R., Sandra L., Emerson S., *J2EE™ Application and BEA™ WebLogic Server*, Prentice Hall PTR. 2002.