2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

# 효과적인 EJB 디자인을 위한 패턴 적용 사례

이연숙[0] 허윤호 오기성 류성열
숭실대학교 컴퓨터학과
(ysbucket, hyh0408)@selab.ssu.ac.kr
ksoh@tongwon.ac.kr syrhew@computing.soongsil.ac.kr

# Case Study on Applying Pattern for Effective EJB Design

Yeon-Sook Lee[0] Yun-Ho Huh Ki-Sung Oh Sung-Yul Rhew
School of Computing, Soongsil University

요 약

패턴은 시스템 디자인 시에 자주 발생하는 문제들에 대한 재사용 가능한 해결책으로써, 증명된 이전의 경험으로부터 나온 공통적인 솔루션에 대한 표현이다. 소프트웨어 산업계에서는 컴포넌트 개발 기술에 대한 관심이 높아지면서 EJB 모델의 효율적인 디자인에 대한 많은 연구가 진행되고 있다. 본 논문에서는, EJB 모델의 설계 단계에서 발생되는 문제점들을 해결하여 보다 효율적인 EJB 디자인을 위해 패턴을 적용한 사례를 보인다. 또한 적용한 사례를 UML을 사용하여 구체화 시킨다.

## 1. 서 론

컴포넌트라는 소프트웨어 모듈의 재사용성과 독립성을 보장함으로써 소프트웨어의 복잡성과 생산성 문제를 해결하고자 하는 일종의 새로운 개발 패러다임이 제시되었고, 이에 소프트웨어 산업계에서는 이러한 컴포넌트 개발에 많은 관심을 보이고 있다.[4] 선(Sun) 사에서는 이러한 컴포넌트 개발을 위해 EJB(Enterprise Java Beans)라는 컴포넌트 아키텍처를 이용한 개발을 권장하고 있다. EJB 기반의 컴포넌트 개발 프로젝트를 진행하다 보면 많은 문제점이 발생할 수 있으나, 특히 EJB 디자인 단계에서 표준화된 디자인 기준이 없어 많은 설계자들이 고민하거나 좋지 않은 디자인을 하게 된다. 패턴의 등장은 설계자들의 이러한 고민을 어느 정도 해결해 주리라 생각한다. 본 논문에서는 EJB 기반의 컴포넌트 개발 프로젝트를 경험으로 EJB 모델 설계 단계에서 발생했던 문제점 또는 고민했던 부분들에 대해 각각에 적합한 패턴을 적용하여 보다 효과적인 EJB 디자인의 가능성을 고려해 본다. 본 논문에서 적용할 경험 사례로는, 인터넷 의료 처방전 시스템이라는 EJB 기반의 컴포넌트 개발 프로젝트이다. 인터넷 의료 처방전 시스템은 병원과 약국, 환자가 온라인으로 정보를 교환할 수 있는 원외 처방전 시스템이다. 인터넷을 통해서 의료기관과 약국간의 처방전 전달이 가능하며, 처방전을 환자가 지정한 약국으로 전송하여 약을 조제하도록 한다. 약국에서는 환자의 처방전에 대해 대체조제가 필요하면 그 대체조제 내역을 작성하여 병원으로부터 승인을 받아 처리하도록 한다. 위와 같은 시나리오를 갖는 인터넷 의료 처방전 시스템을 사례로 하여, 2장에서는 관련 연구로서 본 논문에서 적용한 패턴에 대한 소개와 용어 정의를 기술하고, 3장에서는 EJB 모델 설계 시 발생되었던 문제점과 그것을 해결하기 위해 각각에 적용한 패턴에 따라 새롭게 디자인 된 EJB 모델을 보인다. 4장에서는 패턴 적용 후에 개선된 점들에 대해 분석적인 평가를 하고, 마지막 5장에서는 결론 및 향후 연구 과제를 제시한다.

## 2. 관련 연구

**Coarse-grained object와 Fine-grained object :** Coarse-grained 객체는 자기 자신의 라이프 사이클을 갖고, 다른 객체들에 대해서 자신의 relationship을 관리하는 객체이다. 반면, Fine-grained 객체는 coarse-grained 객체에 의존하는 객체로서 의존적인 객체(dependent object)라고도 한다. Fine-grained

객체는 coarse-grained 객체에 의해 라이프 사이클이 관리된다. 따라서 fine-grained 객체는 클라이언트에 직접적으로 노출되지 않는다. Fine-grained 객체는 그들의 존재를 정당화하기 위해서 항상 그들의 coarse-grained 객체를 가져야 한다. 트리 구조로서 두 객체를 설명하자면 coarse-grained 객체가 root node가 되며 fine-grained 객체가 leaf node가 된다.[1]

**2.1 Session Façade 패턴**

Session Façade 패턴은 워크플로우에 참여하는 비즈니스 오브젝트(엔티티 빈 또는 세션 빈)의 상호작용을 추상화하고 그들의 관계를 관리하며, 오직 요구된 인터페이스만을 노출시키는 서비스 레이어를 제공하는 Session Facade라는 일종의 세션 빈을 통해 처리가 이루어지도록 한다.[1]

**2.2 Value Object 패턴**

Value Object는 비즈니스 데이터를 캡슐화한 객체이다. 엔티티 빈의 모든 속성 값을 얻기 위해 여러 번의 리모트 메소드 호출을 하는 대신에 그러한 데이터를 캡슐화한 value object에 대해 한번의 리모트 메소드 호출을 통해 값을 얻을 수 있다. 또한 client copy로 인해 한번의 리모트 호출 이후부터는 로컬 메소드 호출이 이루어진다.[1]

**2.3 Composite Entity 패턴**

Composite Entity 패턴은 coarse-grained 객체와 그것과 관련된 모든 dependent 객체들을 표현하는데, 상호 관계있는 persistent 객체들을 하나의 엔티티 빈으로 묶는 개념이다.[1]

**3. 사례 적용**

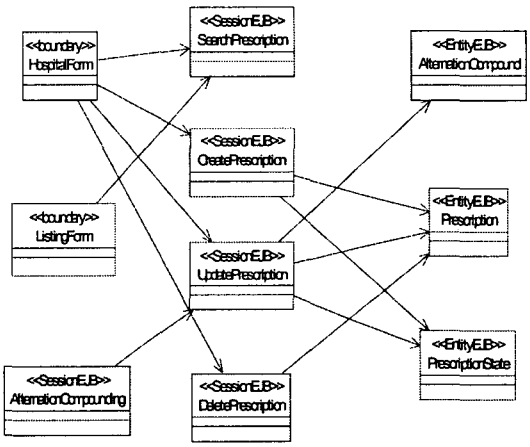
위에서 언급한 인터넷 의료 처방전 시스템 프로젝트를 진행하는 과정에서 많은 문제점이 발생했다. 그 중 EJB 모델 설계 시에 발생한 문제점을 다음의 세 가지 경우로 고려해보았다.

- **[경우 1] 세션 빈 디자인 시, 세부 비즈니스 로직 단위를 하나의 세션 빈으로 디자인 한 경우 및 그 방법**
- **[경우 2] 엔티티 빈 디자인 시, 각각의 엔티티 클래스를 각각의 엔티티 빈으로 디자인 한 경우 및 그 방법**
- **[경우 3] 엔티티 빈 디자인 시, 성능을 고려하지 않고 설계를 한 경우**

다음은, 인터넷 의료 처방전 시스템에서 위에서 고려한 세 경우에 해당하는 부분들에 대해 각각 적합한 패턴을 적용해 본다.

**3.1 [경우 1] 해결을 위한 Session Façade 패턴 적용**

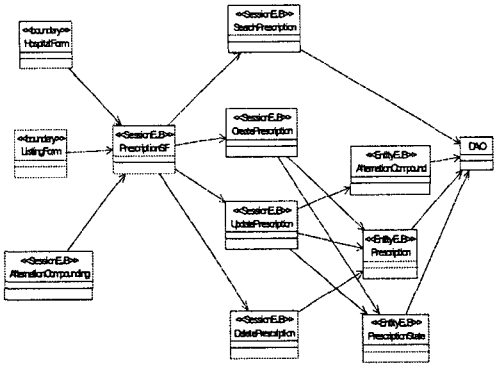
그림 1은 기존에 디자인 된 인터넷 의료 처방전 시스템의 한 부분인 처방전 트랜잭션 처리에 대한 클래스 다이어그램이다.




**<그림1 : 처방전 트랜잭션 처리의 클래스다이어그램>**

하나의 use case를 설계하기 위해 그 내부에서 서로 관련있는 로직 및 엔티티 빈들에 대해 클라이언트가 각각을 호출하도록 설계하는 것은 너무 많은 리모트 호출을 야기한다. 또한, 세부 비즈니스 로직 단위를 하나의 세션 빈으로 매핑하는 방법도 오직 한 종류의 상호작용만을 서비스하는 fine-grained 세션 빈을 너무 많이 생성하게 된다. 이렇게 생성된 많은 세션 빈은 클라이언트에서 해당 서비스를 요구할 때마다 각각의 세션 빈에 대해 호출이 이루어진다. 이때 호출은 항상 네트웍을 통한 리모트 호출이다. 따라서 리모트 호출 수가 증가하게 되고 이에 따라 네트웍 부하가 증가하게 된다. 또, 비즈니스 오브젝트들과 클라이언트 간에 결합도(coupling) 또한 증가하게 된다. 이러한 문제를 해결하기 위해 Session Façade 패턴을 적용한다. 다음은 위의 도메인에 대해 Session Façade 패턴을 적용한 모델이다.

그림 1에서 보면, 클라이언트 측에서는 트랜잭션 처리를 위해 처방전 검색(SearchPrescription), 처방전 생성(CreatePrescription), 처방전 수정(UpdatePrescription), 처방전 삭제(DeletePresciption) 라는 네 개의 빈에 접근을 하고자 한다. 하지만 이들 각각에 대한 리모트 호출 대신 PrescriptionSF라는 세션 빈으로 한번의 리모트 호출을 통

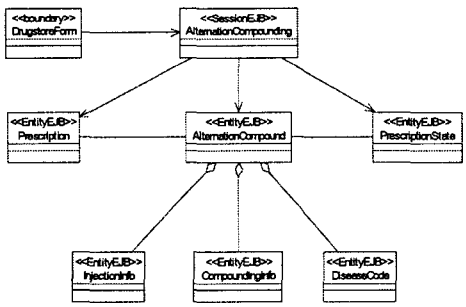



<그림 2 : 그림 1에 Session Façade 패턴을 적용한 모델>

해 네 개의 빈을 접근할 수 있다. 이때, 네 개의 빈들 간의 관리는 PrescriptionSF 세션 빈이 책임진다(그림2).

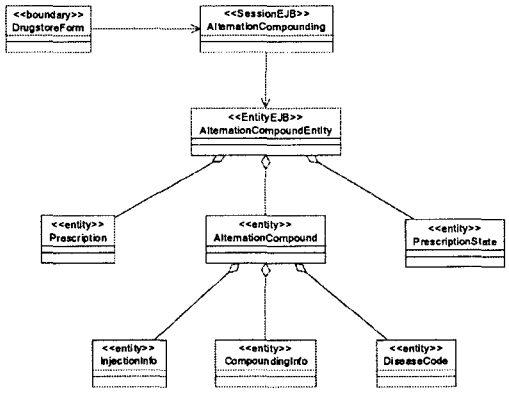
### 3.2 [경우 2] 해결을 위한 Composite Entity 패턴 적용

그림 3은 기존에 디자인 된 인터넷 의료 처방전 시스템의 한 부분인 대체조제에 대한 클래스 다이어그램이다.




<그림 3 : 대체조제 기능의 클래스 다이어그램>

각각의 엔티티 클래스를 각각의 엔티티 빈으로 매핑하는 것 또한 너무 많은 fine-grained 엔티티 빈이 생성된다. 이러한 매핑은 어플리케이션을 복잡하게 할 뿐만 아니라, 엔티티 클래스 간의 relationship이 그대로 엔티티 빈 간에도 존재하게 된다. 이러한 relationship은 엔티티 빈들 간의 호출을 야기하고 그러한 호출 또한 리모트 호출로서 네트웍 부하를 증가시킨다. 또, 클라이언트가 각각의 fine-grained 엔티티 빈을 관리함으로써 데이터 스키마의 변화에 의존하게 된다. 이러한 문제를 해결하기 위해 Composite Entity 패턴을 적용한다. 그림 4는 대체조제 기능에 대해 Composite Entity 패턴을 적용시킨 모델이다. 대체조제에 필요한 기존 여섯 개의 엔티티 빈인 처방전(Prescription), 대체조제(AlternationCompound), 처방전상태(PrescriptionState), 주사제정보(InjectionInfo),

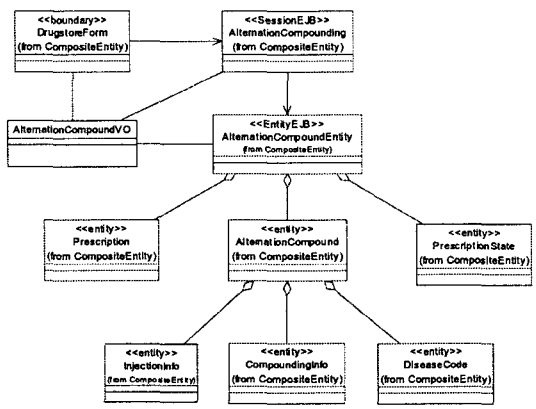



<그림4:그림3에 CompositeEntity 패턴을 적용한 모델>

조제 정보(CompoundingInfo), 질병코드(DiseaseCode)를 객체 화하고 그들을 대표하는 복합 엔티티 빈인 대체조제엔티티 (AlternationCompoundingEntity)빈을 생성한다. 클라이언트 측은 이 복합 엔티티 빈을 통해 여섯 개의 객체를 접근, 상호 작용한다.

### 3.3 [경우 3] 해결을 위한 Value Object 패턴 적용

엔티티 빈의 경우 모든 속성에 대해 get/set 메소드를 사용하여 데이터를 처리하게 된다. 만약 데이터의 일부가 변경되어 해당 데이터를 읽어와서 변경해야 한다면 그때마다 각 데이터의 get/set 메소드를 통해 호출하게 될 것이다. 하지만 이러한 설계는 매우 많은 리모트 호출을 야기한다. 따라서 이러한 데이터를 캡슐화 시킨 하나의 객체로서 처 리한다면 리모트 호출의 수를 줄일 수 있을 것이다. 이를




<그림5 : 그림3에 Value Object 패턴을 적용한 모델>

해결하기 위해 Value Object 패턴을 적용한다. Value Object는 처음 한번의 리모트 호출로서 캡슐화된 하나의 객체, 즉 Value Object를 리턴한 후, 그 이후부터는 그 객체를 클라이언트 측에 복사 하여 로컬 호출로서 접근할 수 있게 한다. 그림 5는 그림 3에 대해 Value Object 패턴을 적용한 모델을 보인다. 3.2에서 생성한 복합 엔티티 빈에 대해 AlternationCompoundVO라는 value object를 생성한다. 이후부터는 클라이언트 측과 세션 빈에서 이 value object를 가지고 처리를 하게된다.

**4. 평가**

본 논문에서 고려한 세 가지 경우에 대해 각각 패턴을 적용한 결과를 리모트 호출 수, 네트웍 부하정도, 관리 용이성, 결합도, 복잡도 측면에서 패턴을 적용하기 전과 비교해 보았다. 본 논문에서 적용한 세 가지 패턴의 궁극적인 목적은 리모트 호출 수를 감소시켜 그에 따른 네트웍의 부하를 줄이는 것이다. 따라서 패턴 적용 전과 후에 대한 각각의 리모트 호출 수를 조사하고 그에 따라 네트웍 부하 정도를 나타낸다. 이때, 리모트 호출 수는 클라이언트와 서버 사이 그리고 서버 내에서 빈들 사이의 리모트 호출로 분리하여 평가한다. 또, coarse-grained 객체나 대표 객체를 통해 내부의 많은 비즈니스 오브젝트들을 통제하도록 하므로써 클라이언트와 비즈니스 오브젝트들 간의 결합도와 관리 용이성에 대해 평가한다. 여기서 관리 용이성은 기존에 클라이언트가 여러 개의 비즈니스 오브젝트들을 직접 관리할 때와 대표 객체를 통해 그들을 관리함으로 인해 관리가 쉬워진 정도를 말한다. 결합도 또한 이러한 측면에서 클라이언트가 오브젝트들과 직접 관계되지 않도록 처리하여 얻어지는 요인을 말한다. 이러한 결과를 바탕으로 빈들과 커 뮤니케이션 하게 되는 어플리케이션의 복잡도에 대해서도 평가를 한다. 표 1은 인터넷 의료 처방전 시스템의 일부에 대한 샘플 시나리오를 구성하고 위에서 설명한 평가 요인들에 대해 패턴을 적용하기 전에 비해 적용 후에 변화된 정도를 분석적인 평가 방법으로서 비교한 결과이다.

| 평가요인 | SF 적용 후 | CE 적용 후 | VO 적용 후 |
|---|---|---|---|
| 클라이언트와 서버 간의 리모트 호출 수 | 5%감소 | 변화없음 | 15%감소 |
| 서버 내부의 빈들 간의 리모트 호출 수 | 변화없음 | 25%감소 | 10%감소 |
| 네트웍 부하정도 | 약간감소 | 감소 | 감소 |
| 관리 용이성 | 용이함 | 용이함 | 약간용이 |
| 결합도 | 감소 | 감소 | 약감감소 |
| 어플리케이션 복잡도 | 감소 | 감소 | 약간감소 |

**[표 1: 패턴 적용 전에 대한 적용 후의 변화]**

SF : Session Façade, CE : Composite Entity, VO : Value Object

**5. 결론 및 향후 연구**

EJB 기반의 컴포넌트 개발 프로젝트인 인터넷 의료 처방전 시스템 개발을 진행해 나가는 동안에 발생했던 EJB 디자인 시의 문제점들을 살펴보았고, 각각에 적합한 패턴을 적용하여 새로운 EJB 모델을 디자인하였다. 자주 발생하는 문제들에 대해 이전의 증명된 경험으로부터 나오는 공통적인 솔루션인 패턴을 적용하여 디자인함으로써 보다 효과적인 EJB 디자인을 할 수 있었다. 하지만 패턴을 적용 한 후에 향상된 성능 및 예상 효과에 대한 구체적인 측정 연구가 더 필요할 것이다. 또한, 패턴의 적용으로 인해 나타날 수 있는 단점들에 대해서도 연구해 볼 필요가 있을 것이다.

**6. 참고 문헌**

[1] Deepak Alur, John Crupi, and Dan Malks, core J2EE PATTERNS, Sun Microsytems, 2001

[2] CT Arrington, Enterprise Java with UML, John Wiley & Sons, Inc., 2001

[3] Sun Microsystems, Inc., Enterprise JavaBeans Specification, version 2.0, 2000

[4] oonewsletter, 디자인 패턴, http://sslab.icu.ac.kr/oonews

[5] 최성만, 김정옥, 이정열, 유철중, 장옥배, "효율적인 EJB 컴포넌트화를 위한 Integrated DAO 패턴", 정보과학회 소프트웨어공학회지, 제28권 1호, 2001