도메인객체의 일관성 있는 참조를 위한 연관정보 관리 서비스☆

A Relationship Management Service for Consistent References Between Domain Objects

최 윤 석*
Yun-seok Choi

요 약

엔터프라이즈 아키텍처 패턴 중 하나인 도메인모델은 재사용성 및 확장성, 그리고 유지보수에 많은 장점을 갖고 있다. 도메인모델을 적용하기 위해서는 지속성계층과의 매핑 및 트랜잭션 관리를 위한 다양한 서비스가 필요하다. 본 논문에서는 도메인모델을 적용하여 개발한 시스템에서 도메인객체의 삭제 트랜잭션 처리 시 발생할 수 있는 참조의 일관성 오류 문제를 해결하기 위해 도메인객체의 연관정보를 관리하는 서비스를 제안한다. 제안한 연관정보 관리 서비스는 다대다 연관을 맺고 있는 도메인객체를 모니터링 하고 수행흐름을 가로채어 삭제 시 발생하는 객체의 참조 오류를 해결한다. 도메인객체 및 기존 서비스의 수정 없이 서비스를 제공하기 위해 AOP(Aspect-Oriented Programming)를 적용하였다.

Abstract

The domain model pattern which belongs to enterprise architecture patterns has reusability, scalability and maintainability. To use the domain model pattern, mapping with persistency layer, transaction management and various services are needed. This paper proposes that relationship management service to solve a consistency error which arises in case of removing domain objects. The proposed relationship management service monitors methods of domain objects and intercepts the flow of control to solve the reference errors. This service is implemented by using AOP(Aspect-Oriented Programming), so it can provide the service without modifications of domain objects and other services.

☞ Keyword : 아키텍처 패턴, 도메인모델, 참조 일관성 오류, AOP(Aspect-Oriented Programming)

1. 서론

기업의 응용시스템 구축에는 다양한 엔터프라이즈 아키텍처 패턴을 사용하고 있다. 업무로직 구성에는 트랜잭션 스크립트(transaction script), 도메인모델(domain model), 그리고 테이블모듈(table module)등과 같은 아키텍처 패턴을 사용할 수 있다[1]. 이 중 객체지향의 개념을 도입하여 업무로직을 표현하는 도메인모델[2][3]은 시스템의 유지보수, 확장, 그리고 업무로직의 재사용 및 복잡성의 해결 측면에서 많은 장점을 갖고 있다[4]. 도메인모델을 효과적으로 개발에 적용하기 위해서는 개발자의 객체지향에 대한 깊은 이해와 관계형 데이터베이스 등으로 구성한 지속성계층(persistence layer)과의 매핑[5], 그리고 트랜잭션 상에서 발생할 수 있는 도메인객체의 삽입, 검색, 수정, 그리고 삭제 등을 일관성 있게 수행할 수 있는 다양한 서비스 계층[6]을 구성하는 추가적인 노력이 필요하다.

자바 언어 기반으로 도메인모델을 적용하여 시스템 개발 시 발생하는 추가적인 노력의 한 가지 예는 업무로직의 트랜잭션 상에서 도메인객체 삭

* 정 회 원 : 동덕여자대학교 정보학부 전임강사
cooling@dongduk.ac.kr
[2007/07/16 투고 - 2007/07/24 심사 - 2007/08/01 심사완료]
☆ 이 논문은 2005년도 동덕여자대학교 학술연구비 지원에 의하여 수행된 것임

제 시 발생하는 도메인객체의 참조 일관성 문제를 해결하는 것이다. 자바의 경우 객체를 생성한 후 삭제한다는 것은 메모리에서 물리적으로 삭제하는 것이 아니라 우선 대상 객체에 대한 참조를 제거하는 것을 의미한다. 모든 참조가 제거된 객체는 가비지컬렉터가 메모리에서 삭제한다[7]. 이러한 특성을 반영하여 도메인객체의 삭제는 객체를 저장하고 있는 객체 풀(pool)이 자신이 저장하고 있는 삭제 대상 객체에 대한 참조를 제거하는 형태로 구현한다. 만일 삭제하고자 하는 도메인객체에 대해 여러 다른 객체들이 연관을 갖고 있다면 업무의 트랜잭션 상에서 객체 삭제를 수행하여 객체 풀이 참조를 제거하였다 하더라도 연관을 갖고 있는 다른 객체들은 여전히 참조를 유지하고 있으므로 삭제한 객체를 참조하는 참조 일관성 오류가 발생할 수 있다[8]. 이를 해결하기 위해서는 트랜잭션 상에서 객체를 삭제할 경우 삭제한 객체와 연관을 맺고 있는 모든 객체들은 해당 객체에 대한 참조를 제거해야 한다. 서비스 계층은 이와 같은 참조의 일관성 유지 서비스를 제공하여야 한다.

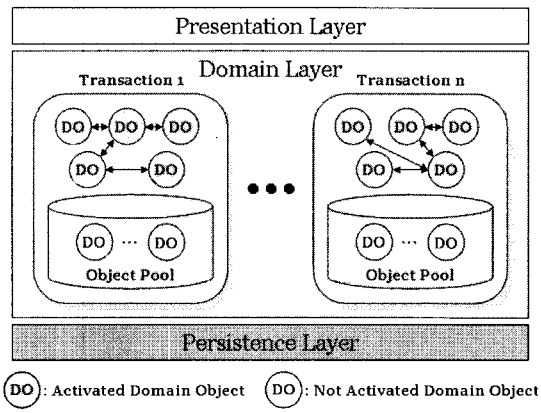
본 논문에서는 자바 기반으로 도메인모델을 적용하여 개발한 시스템에서 도메인객체의 삭제 트랜잭션 처리 시 발생할 수 있는 참조의 일관성 문제를 해결하기 위해 도메인객체의 연관정보를 관리하는 서비스를 제안한다. 제안한 서비스는 도메인객체의 연관정보를 모니터링하며, 도메인객체 삭제 시 참조의 일관성 오류가 발생하지 않도록 삭제를 처리한다.

본 논문의 구성은 다음과 같다. 2장은 도메인모델을 적용한 시스템의 트랜잭션 관리를 위한 구성요소와 도메인객체 삭제 시의 참조의 일관성 유지 방안에 대하여 알아본다. 3장은 제안한 연관정보 관리 서비스의 구성 및 구현방법에 대해 기술하며, 4장은 제안한 연관정보 관리 서비스의 실용성을 알아보기 위해 사례시스템을 제시한다. 5장은 결론을 제시한다.

2. 도메인모델을 적용한 트랜잭션 관리

2.1 도메인객체 풀의 구성

업무로직에 도메인모델 패턴을 적용하여 개발한 응용 시스템은 트랜잭션 관리를 위해 객체 풀을 구성하고 지속성계층에서 로딩 한 객체 및 트랜잭션 수행 중에 생성한 객체를 저장하고 관리한다[8]. 객체 풀은 전체 시스템에서 모든 트랜잭션들이 공유할 수 있도록 하나의 객체 풀만을 두어 사용하거나 업무 트랜잭션 별로 자신만의 객체 풀을 생성하고 사용하는 방식으로 구성할 수 있다. 업무 로직을 처리하기 위해 필요한 모든 도메인객체들은 사용하기 전에 지속성계층에서 객체 풀로의 로딩을 수행하며, 객체 풀 상에서 유일하게 존재하도록 관리한다. 도메인객체의 유일성 식별은 일반적으로 회원 아이디, 부서코드와 같이 해당 도메인의 업무상에서 유일하게 식별할 수 있는 정보를 사용하여야 한다[1]. 각 트랜잭션 별로 객체 풀을 갖는 시스템 구성의 개요는 그림 1과 같다.



(그림 1) 트랜잭션 별 도메인객체 풀의 구성

2.2 도메인객체 삭제 시 참조의 일관성 유지 방안

객체 풀에 존재하는 도메인객체들이 일대다 또

는 다대다 연관관계를 맺고 있을 경우 앞서 언급한 객체의 삭제에 따른 참조의 일관성 오류 문제가 발생할 수 있다. 이를 해결하기 위해 첫 번째로 도메인객체가 삭제되었을 경우 자신과 연관을 맺고 있는 모든 객체에 삭제 메시지를 직접 전달하도록 관찰자패턴(observer pattern)[8] 또는 객체 등록 및 확인 패턴(object registration and validation pattern)[10]을 적용하는 방법을 생각해볼 수 있다. 이 방법은 참조에 의한 오류를 막을 수는 있으나 도메인객체는 업무 로직과 관련한 기능에 집중해야 한다는 도메인객체 설계 개념에 어긋난다. 두 번째로 서비스계층이 도메인객체의 연관정보를 관리하는 방법을 생각해볼 수 있다. 서비스계층이 도메인객체 삭제 시 삭제한 도메인객체와 연관을 맺고 있는 모든 도메인객체들의 연관정보를 추적하여 참조의 일관성을 유지한다. 그러나 이 방법 역시 도메인객체가 연관과 관련된 정보를 서비스계층에 전달하는 업무로직 이외의 기능을 갖게 되며, 이로 인해 서비스 계층과의 결합도가 높아지는 문제가 생긴다. 또한 연관정보 관리 서비스를 추가하기 위해 기존 서비스 계층의 내용을 수정해야 하는 추가적인 작업이 발생한다.

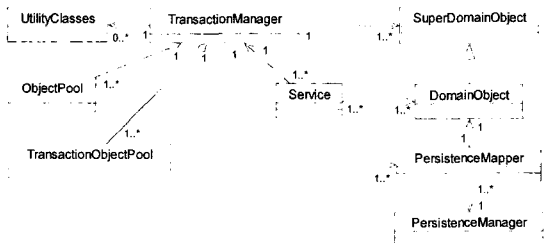
이에 본 논문에서는 도메인객체에 업무기능 이외의 기능 추가를 피하고, 서비스계층에 연관정보 관리 서비스 추가 시 서비스 계층의 수정을 최소화하며 서비스를 구성할 수 있도록 AOP(Aspect-Oriented Programming)[11]를 적용한 연관정보 관리 서비스의 구성을 제안한다.

3. 연관정보 관리 서비스

연관정보 관리 서비스는 도메인모델 패턴을 적용하여 구성한 트랜잭션 관리 서비스와 결합하여 도메인객체들 사이의 연관정보를 관리한다. 본 논문에서는 트랜잭션 별로 객체 풀을 보유하는 형태로 트랜잭션 관리 서비스를 구성하고 이에 적용 가능한 연관정보 관리 서비스를 제안한다.

3.1 트랜잭션 관리 서비스의 구성

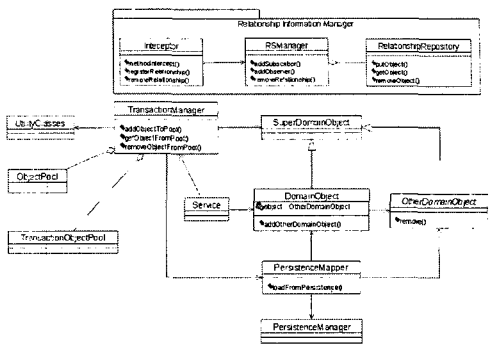
트랜잭션 관리 서비스는 트랜잭션 별로 객체 풀을 보유하며, 업무로직을 구성하는 도메인객체, 그리고 지속성계층으로부터 도메인객체를 로딩하는 매퍼[1] 등으로 구성한다. 다음 그림 2는 트랜잭션 관리 서비스의 구성을 나타낸다.



(그림 2) 트랜잭션 관리 서비스의 구성

최상위 도메인객체(SuperDomainObject)는 모든 도메인객체(DomainObject)가 가져야 할 공통 인터페이스를 정의한 클래스로서 도메인객체의 유일성 및 동일성 처리와 같은 식별정보 관리, 트랜잭션의 확인 및 객체 생성에 따른 정보관리에 관한 인터페이스를 정의한다. 도메인객체는 업무로직을 구성하는 객체로서 업무에 필요한 정보를 보유하고 있으며, 최상위 도메인객체를 상속하여 생성한다. 객체매퍼(PersistenceMapper)는 도메인객체 별로 객체의 속성 정보를 지속성계층으로부터 로딩한 후 설정을 수행하여 객체를 생성하는 역할을 담당하며, 대상 객체와 연관을 갖는 객체들의 로딩도 함께 처리한다. 트랜잭션 관리자(TransactionManager)는 트랜잭션의 시작, 확인 그리고 종료 등을 담당하며, 객체 풀(ObjectPool)과 트랜잭션 풀(TransactionObjectPool)을 관리한다. 객체 풀은 트랜잭션에 참여하기 위해 지속성계층으로부터 로딩한 모든 객체들을 보관하며 트랜잭션 풀은 트랜잭션에 참여하여 변경이 발생한 도메인객체들의 정보를 보관한다. 업무로직에서 사용하고자 하는 도메인객체가 있을 경우 트랜잭션 관

리자는 목표 도메인객체가 객체 풀에 존재하는지를 먼저 판단하여 존재하면 풀에서 객체를 로딩하여 전달하고, 존재하지 않으면 지속성계층에서 목표 도메인객체의 정보를 읽어온 후 객체를 생성하여 반환한다. 트랜잭션 처리 도중 특정 도메인객체의 삭제를 수행하면 도메인객체가 갖고 있는 삭제 메소드를 호출하여 트랜잭션 관리자에게 삭제 사실을 전달하고, 트랜잭션 관리자는 객체 풀에서 해당 객체를 제거하고 삭제정보를 트랜잭션 풀에 기록한다.



(그림 3) 연관정보 관리 서비스 구성의 개요

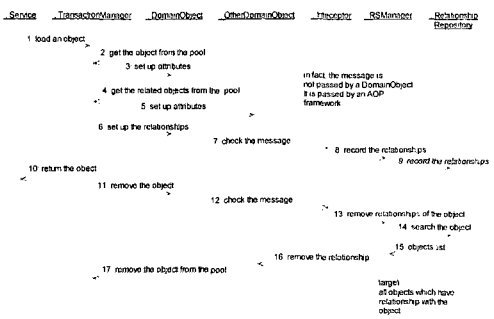
3.2 연관정보 관리 서비스의 구성

도메인객체 삭제 시 참조의 일관성을 유지하기 위해 연관정보 관리 서비스를 트랜잭션을 관리하는 서비스 계층 및 도메인객체들과 결합한다. 연관정보 관리 서비스는 트랜잭션에 참여하는 모든 도메인객체들의 연관정보를 관리한다. 객체 풀에 도메인객체를 로딩 할 경우 대상 도메인객체와 연관을 맺고 있는 모든 객체들에 대한 정보를 관리한다. 트랜잭션 처리 시 특정 도메인객체의 삭제가 발생하면 도메인객체를 모니터링하고 있는 연관정보 관리 서비스는 해당 객체와 연관을 맺고 있는 모든 객체들에게 객체의 삭제 사실을 알린다. 그림 3은 트랜잭션 서비스와 도메인객체를 모니터링 하고 서비스를 제공하는 연관정보 관리 서비스 구성의 개요를 나타내며, 음영 부분은 연관정보 관리 서비스가 직접적으로 작용하는 부분을 나타낸다.

연관정보 관리 서비스는 인터셉터(Interceptor), 연관관리자(RSManager), 그리고 연관정보저장소(RelationshipRepository)로 구성한다. 인터셉터는 도메인객체의 메소드 호출을 모니터링 하는 감시자 역할 및 필요에 따라 수행흐름을 가로채는 역할을 담당한다. 연관관리자는 도메인객체의 연관정보를 기록하고 연관정보의 변경이 있을 경우 참조의 일관성을 유지하기 위한 관리를 수행한다. 연관정보저장소는 도메인객체 사이에 존재하는 모든 연관관계를 맵 형태로 관리한다.

인터셉터는 도메인객체를 지속성계층에서 로딩할 때부터 모니터링하며 도메인객체가 멤버메소드를 호출하면 메소드를 수행하기 전에 수행흐름을 가로챈다. 인터셉터는 가로챈 메소드 정보(메소드를 소유하고 있는 도메인객체, 메소드 명 등)를 분석하여 호출한 메소드가 관심사항(도메인객체들 사이의 연관정보 설정 및 삭제)에 해당하는 메소드인지 판별한다. 비관심사항 메소드로 판별하면 원래의 수행흐름으로 복귀하며, 도메인객체는 본래의 업무를 수행한다. 관심사항 메소드로 판별하면 메소드의 종류를 식별하고 식별결과에 따라 연관관리자에게 메소드 정보를 전달한다. 연관관리자는 인터셉터가 전달한 정보를 판별하여 다른 도메인객체와 연관을 설정하는 메소드 호출일 경우, 연관정보를 구성하는 연관객체와 피연관객체의 정보를 구분하고 관찰자패턴을 적용하여 각각 게시자와 관찰자로 나누어 연관정보저장소에 기록한다. 도메인객체의 삭제메소드 호출일 경우, 연관정보저장소를 조사하여 삭제한 도메인객체와 연관을 맺고 있는 모든 도메인객체들을 검색하고 삭제 도메인객체의 참조제거를 수행한다. 이와 같은 절차에 따라 도메인객체의 삭제 수행 시 해당 객체와 연관을 갖고 있는 모든 도메인객체들이 갖고 있는 참조의 삭제도 함께 수행하므로 삭제에 의한 참조의 오류 없이 연관정보의 일관성을 유지할 수 있다. 그림 4는 연관정보 관리

서비스 수행절차를 순차도로 표현한 것이다.



(그림 4) 연관정보 관리 서비스의 수행 절차

3.3 연관정보 관리 서비스의 구현

연관정보 관리 서비스는 도메인객체의 수정 없이 도메인객체의 메소드 호출을 모니터링하며 메소드의 종류에 따라 호출 흐름을 가로챌 수 있어야 한다. 이와 더불어 서비스계층의 변경을 최소화하기 위해서 트랜잭션 서비스에서 직접 연관정보 관리 서비스를 수행하는 것이 아니라 트랜잭션 서비스와는 독립적으로 수행할 수 있어야 한다. 본 논문에서는 도메인객체의 변경 없이 메소드 호출을 모니터링하고 이에 따른 기능을 수행하며, 연관정보 관리 서비스의 독립적인 수행을 위해 AOP를 적용한다. AOP는 특정 관심사에만 관련 있는 기능이 아닌 어플리케이션 전체에 걸쳐 수행하여야 하는 기능을 모듈화 하여 개발하는 방법이다. 연관정보 관리 서비스의 경우 모든 도메인객체와 관련을 맺어야 하므로 AOP를 적용하여 개발하는 것이 적합하다. AOP를 지원하는 다양한 프레임워크들이 존재하고 있으며 본 논문에서는 최근 실무현장에서 많은 관심을 받고 있는 스프링(Spring)프레임워크[12] 기반으로 연관정보 관리 서비스를 구성하였다. 스프링은 자바 기반의 오픈소스 어플리케이션 프레임워크로서 의존관계 주입(dependency injection)과 AOP 프로그래밍 모델을 제공한다. 다음 표 1은 AOP의 용어에 따라 연관정보 관리 서비스의 구성요소를 분류한 것이다.

(표 1) AOP 용어별 연관정보 관리 서비스 구성의 분류

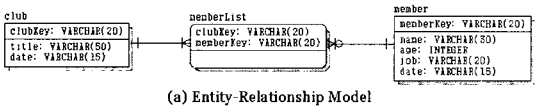
AOP 용어	연관정보 관리 서비스의 구성
관점(aspect)	도메인객체의 삭제 시 객체 참조의 일관성 유지
결합점(joinpoint)	도메인객체의 모든 멤버메소드
충고(advice)	도메인객체들 사이의 연관정보를 관리(RSManager)
포인트컷(pointcut)	다른 도메인객체와 연관을 설정하는 메소드 명 도메인객체가 갖고 있는 삭제 메소드 명
도입(introduction)	도메인객체에 연관정보 관리 서비스 제공 기능을 추가
목표(target)	연관을 맺고 있는 모든 도메인객체들
프록시(proxy)	도메인객체의 메소드의 모니터링 기능을 구현(Interceptor)
엮기(weaving)	실행시간 동안 도메인객체와 연관정보 관리 서비스를 결합

'관점'은 어플리케이션 전체에 걸쳐 수행하고자 하는 기능을 의미하며, 연관정보 관리 서비스의 도메인객체 삭제 시 참조의 일관성 유지 기능이 관점이 된다. '결합점'은 '관점'을 적용할 시스템의 특정부분을 의미한다. 연관정보 관리 서비스의 경우 도메인객체의 모든 멤버메소드가 결합점이다. '충고'는 '관점'을 실제 구현한 것으로서 구현한 연관정보 관리 서비스이다. '포인트컷'은 '결합점'과 '충고'를 결합하기 위한 규칙을 의미하며, 도메인객체의 모든 멤버메소드 중 다른 도메인객체와의 연관정보를 설정하는 메소드 명과 도메인객체를 삭제하는 메소드 명이 된다. '도입'은 기존 클래스에 추가적인 메소드 및 속성을 부가하는 것으로서 도메인객체에 연관정보 관리 서비스가 제공하는 기능을 추가하는 것을 의미한다. '목표'는 '관점'이 적용되는 대상을 의미하며 모든

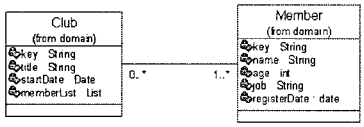
도메인객체가 '목표'이다. '프록시'는 AOP 프레임워크가 '목표'와 '충고'를 결합하여 생성하는 객체를 의미하며 AOP 프레임워크는 도메인객체와 연관정보 관리 서비스의 기능을 결합하여 프록시를 생성한다. '엮기'는 '목표'와 '충고'를 결합하여 프록시를 생성하는 시점을 의미하며 연관정보 관리 서비스는 실행시간에 도메인객체와 엮기를 수행한다.

4. 적용사례

제안한 연관정보 관리 서비스를 다대다 연관관계를 갖는 도메인모델에 적용하였을 때 정상적으로 참조의 일관성을 유지하는지 확인하기 위해 사례시스템을 개발하였다. 다대다 관계를 표현하기 위해 포털사이트의 클럽과 클럽의 멤버를 도메인모델로 구성하였다. 클럽은 여러 명의 멤버를 가질 수 있으며, 멤버 역시 여러 클럽에 가입할 수 있다. 이를 표현한 E-R 모델과 도메인모델은 그림 5와 같다.



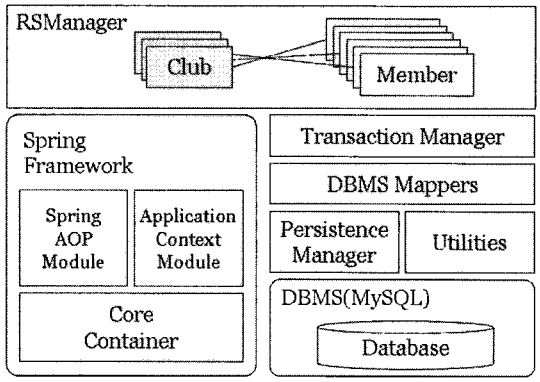
(a) Entity-Relationship Model



(b) Domain Model

(그림 5) 사례시스템의 E-R 모델과 도메인모델

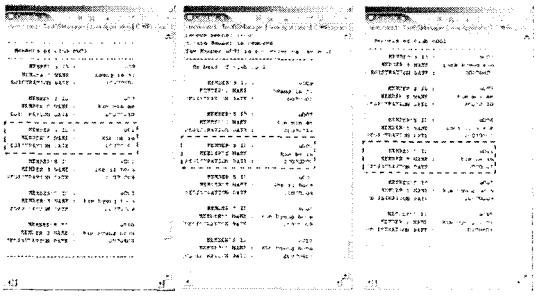
관계형 데이터베이스는 다대다 관계의 직접적인 표현이 불가능하므로 그림 5의 (a)와 같이 클럽의 멤버목록을 관리하는 memberList 연관테이블을 추가하였다. E-R 모델 상의 다대다 관계는 도메인모델에서 자바 표준 클래스 라이브러리의 List 계열 인터페이스 및 클래스를 사용하여 구현하였다. 도메인객체와 테이블 사이의 매핑은 본 논문의 관심사항이 아니므로 가장 간단한 매핑 형태를 선택하여 테이블 대 클래스 일대일 매핑을 적용하였다[5]. 도메인객체인 Club과 Member는 해당 도메인의 업무와 관련한 속성 및 기능만을 갖도록 구현하였으며, 도메인 객체의 검색 기능 및 삭제를 위한 서비스, 트랜잭션 관리자, 그리고 지속성계층과의 매퍼 등을 구현하였다. AOP를 적용하기 위해 스프링 프레임워크를 사용하였으며, 지속성계층은 DBMS인 MySQL을 사용하여 구축하였다. 사례시스템의 구성은 그림 6과 같다.



(그림 6) 사례시스템의 구성

연관정보 관리 서비스가 정상적으로 서비스를 제공하는지 확인하기 위해 우선 연관정보 관리 서비스 비활성 상태에서 여러 클럽들과 연관을 맺고 있는 멤버를 선정하여 멤버의 삭제업무를 수행하였다. 삭제 대상 도메인객체는 각각 클럽아이디가 'c001'과 'c002'인 클럽에 속한 아이디 'u001'인 멤버객체를 선정하였다. 그림 7의 (a)와 같이 클럽 'c001'의 멤버로 'u001'이 있음을 확인한 후 'u001'의 삭제를 수행하였다. 'u001'을 삭제하였으므로 연관을 맺고 있는 클럽 'c001'과 'c002'는 더 이상 'u001'을 참조할 수 없어야 함에도 불구하고 각각의 멤버목록을 검색하면 그림 7의 (b), (c)에서 보는 바와 같이 'u001'이 나타나고 있음을 확인할 수 있다. 이는 도메인객체의 삭제가 실제로 삭제되는 것이 아닌 객체 풀에서 해당 객체와 맺고 있는 참조를 제거하는 것이고 연

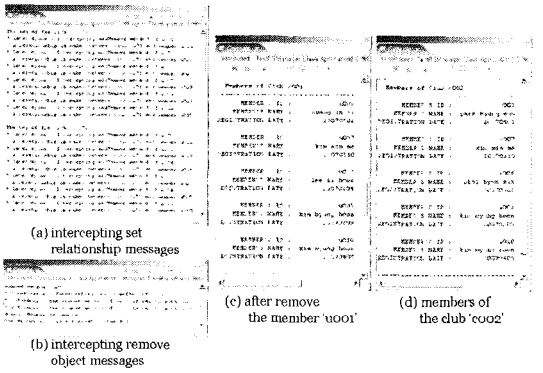
관을 맺고 있는 다른 도메인객체들은 여전히 참조를 유지하고 있기 때문에 발생하는 문제이다.



(그림 7) 연관정보 관리 서비스 비활성화 상태의 도메인객체 삭제 수행

그림 8은 연관정보 관리 서비스를 활성화시킨 후 동일한 작업을 수행했을 때의 결과를 나타낸다. 그림 8의 (a)는 도메인객체 로딩 시 연관정보를 설정하는 메소드 호출을 연관정보 관리 서비스가 가로채어 연관정보를 저장하고 있음을 보여주며, (b)는 도메인객체 삭제 시 삭제메소드를 가로채어 연관정보에 따라 'c001'과 'c002' 각각에서 삭제한 도메인객체의 연관을 제거하는 것을 보여준다. 클럽 'c001'의 멤버로 속한 'u001'을 삭제한 후 'c001'의 멤버목록을 살펴보면 그림 8의 (c)와 같이 'u001'이 클럽의 멤버목록에서 삭제되어 있음을 확인할 수 있다. 또한 'u001'이 속해있었던 다른 클럽 'c002'의 멤버목록에서도 삭제되어 있음을 그림 8의 (d)를 통해 확인할 수 있다. 이는 연관정보 관리 서비스가 도메인객체의 모니터링을 수행하는 중 삭제메소드가 호출되었을 경우 삭제한 도메인객체와 연관을 맺고 있는 모든 다른 도메인객체의 연관삭제메소드를 호출하고 삭제한 도메인객체와의 참조를 제거하였기 때문이다.

사례시스템에 연관정보 관리 서비스의 도입 시 시스템을 수정해야 할 부분은 극히 일부분이었다. 연관정보 관리 서비스는 스프링 프레임워크의 AOP를 적용하여 구현하였으므로 도메인객체는 수정할 필요가 없다. 트랜잭션 관리자의 경우 신규로 도메인객체를 생성하는 부분만 스프링 프레임워크의 BeanFactory 클래스를 사용하여 객체를 생성하도록 수정하여 연관정보 관리 서비스의 사용이 가능하였다.



(그림 8) 연관정보 관리 서비스 활성화 상태의 도메인객체 삭제 수행

5. 결론

도메인모델은 복잡도 높은 업무를 효율적으로 표현하고, 시스템의 유지보수 및 확장이 용이한 장점이 있으나 지속성계층과의 매핑, 그리고 트랜잭션 상의 도메인객체 관리를 위한 추가적인 노력이 필요하다. 본 논문에서는 도메인객체의 연관정보를 일관성 있게 관리하는 연관정보 관리 서비스를 제안하였다. 제안한 서비스는 도메인객체들 사이의 연관정보를 관리하여 객체의 삭제에 의해 발생할 수 있는 참조의 오류 문제를 해결하였다. 특정 도메인객체를 삭제할 경우 이를 참조하는 다른 모든 도메인객체들이 참조의 일관성을 유지할 수 있도록 도메인객체들 사이의 연관정보를 모니터링 및 관리할 수 있도록 서비스를 구성하였다. 서비스 구현에 있어서 AOP를 적용하여 연관정보 관리 서비스를 사용하기 위한 서비스 계층의 변경을 최소화 하였으며, 특히 도메인객체

에 업무로직 외의 기능 추가 없이 서비스를 제공할 수 있었다.

향후에는 제안한 서비스의 효과적인 적용을 위해 서비스 및 설정 정보에 대한 자동 생성 연구가 필요하다. 도메인객체의 정보에 의존적인 연관정보 관리 서비스 구성요소와 AOP 프레임워크 설정 정보를 분석하여 이를 자동 생성하는 도구의 개발이 유용할 것이다. 이외에 도메인객체의 연관정보를 기반으로 트랜잭션 상에서 새로이 생성한 도메인객체의 정보를 지속성계층의 트랜잭션과 연계하여 어떻게 일관성 있게 유지할 것인지에 대한 연구가 필요하다.

참 고 문 헌

[1] Martin Fowler, "Patterns of Enterprise Application Architecture", Addison-Wesley, 2003

[2] Neil Iscoe, Gerald B, Williams and Guillermo Arango, "Domain Modeling for Software Engineering", Software Engineering, 1991. Proceedings., 13th International Conference on, 13-16 May 1991 pp.340-343

[3] Craig Larman, "Applying UML and Patterns 3rd", Prentice Hall, 2005

[4] Paul Oldfield, "Domain Modeling", white paper of Appropriate Process Group, 2002

[5] Scott W. Ambler, "Agile Database Technique", John Wiley&Sons, 2003

[6] Martin Fowler, "Analysis Patterns" Addison-Wesley, 1997

[7] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, "The Java Language Specification 3rd", Addison-Wesley, 2005

[8] 김형준, "도메인모델을 이용한 J2EE 컴포넌트 만들기", 컴스페이스, 2006

[9] Gamma, Helm, Johnson, and Vlissides, "Design Patterns", Addison-Wesley, 1995

[10] Object registration and validation pattern, http://www.ddj.com/cpp/199905993

[11] Kiczales, Gregor, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming", Proceedings of the European Conference on Object-Oriented Programming, vol.1241, pp.220-242

[12] Spring, http://www.springframework.org

◐ 저 자 소 개 ◑

최 윤 석(Yun-seok Choi)

1997년 숭실대학교 소프트웨어공학과 졸업(학사)
1999년 숭실대학교 대학원 컴퓨터학과 졸업(석사)
2001년 숭실대학교 대학원 컴퓨터학과 박사과정 수료
2005~현재 동덕여자대학교 정보학부 전임강사
관심분야 : 소프트웨어 아키텍처, 프레임워크, 소프트웨어 개발방법론.
E-mail : cooling@dongduk.ac.kr