객체지향 어플리케이션을 위한 퍼시스턴스 레이어의 객체-관계 매핑 아키텍처 평가

이재성*, 신성욱**
*고려대학교 컴퓨터정보통신대학원 소프트웨어공학과
**고려대학교 컴퓨터공학과
e-mail : jaysonree@2e.co.kr*, stnoble@korea.ac.kr**

Object-Relation mapping System Architecture Evaluation as Persistence layer for Object-Oriented Applications

JaeSung Lee* SungOog Shin**
*Computer Engineering of Computer Information and Technology, Korea University,
**Computer Science and Engineering, Korea University

요 약

객체 중심의 개발방법은 오브젝트(Object)를 중심으로 프로세스를 표현하고 있으며, 이에 반해 관계형 DBMS 로 표현 되는 데이터 영역에서는 테이블(Table)을 기반으로 데이터를 집합체로 제시하고 있다. 이러한 차이로 인해 상호 영역이 맺어지는 영구 저장 영역(Persistence Layer)의 구현과 유지보수에 어려움을 겪고 있다. 최근 대두된 Object-Relation Mapping(ORM) 기술은 이러한 영구 저장 영역의 구현을 상대적으로 쉽고 빠르게 해 준다고 하나 이 또한 기술 적용의 어려움과 테이블의 조합으로 만들어지는 조회와 같은 주요 아키텍처 품질 요소를 만족시키기가 어렵다. 이에 본 논문에서는 영구 저장 영역을 ORM 으로 설계하는 데 있어 고려해야 할 요소들을 정리하여 시스템 별로 적절한 아키텍처 선정에 참조할 수 있는 기준을 제시하였다. 또한 이동통신사의 실제 비즈니스 사례를 들어 ORM 이 적용되지 않은 시스템을 대상으로 제안한 기준에 의해 해당 아키텍처를 선정하고 구현하였으며, 적용 전과 적용 후를 비교하여 주요 요구사항을 만족시키는 지 평가하였다.

1. 서론

최근 기업의 어플리케이션 개발 환경은 객체지향 기술과 관계 형 DBMS 를 기반으로 구축되고 있다. 객체 지향의 관점은 비즈니스를 모델링 하는 데 있어 데이터와 행위를 가진 객체를 통해 실제 세계를 모델링을 통해 구현하며 이에 반해 관계 형 DBMS 는 영속적으로 저장되어야 할 데이터를 수학적인 집합 개념을 통해 관리하고 조작하는 기반을 제공한다.

어플리케이션 개발 시 이러한 오브젝트(Object)와 관계 형 DBMS 의 테이블은 쉽게는 상호 일치하는 것처럼 보일 수 있으나 객체에서 가공, 처리된 정보를 영구적인 영역에 저장하고 다시 해당 영역에서 꺼내는 Data Access Layer 에서의 표현은 서로 다른 기술의 개념과 기술의 차이로 많은 노력이 필요하다.

ORM(Object-Relation Mapping)은 이러한 객체와 테이블간의 상호 불일치를 연결 해주는 아키텍처이다. 이상적으로는 이러한 기술만으로 자동으로 문제가 해결될 것으로 보이나 다양한 기업 환경과 주요 아키텍처 품질 속성을 모든 상황에 만족시키는 경우를 찾기는 힘들다. 결국 이러한 아키텍처의 선정을 위해서는 해당 기업이나 프로젝트의 환경이나 기본적인 요구사항 등을 바탕으로 비즈니스 요구사항에 따라 취사 선택되어야 할 것이며, 실제 비즈니스를 통한 주요 속성의 검증을 통한 평가가 필요하다.[1]

본 논문에서는 이러한 상호 불일치 문제점을 간략히 정리하고 Persistence layer 에서의 ORM 아키텍처를 설명하였다. 또한 아키텍처 선택을 위해서 고려되어야 할 주요 요소들을 수집하여 아키텍처 선택에 참조할 수 있는 기준을 작성하였다. 또한 검증을 위해 실제 통신사의 Data Mart 시스템에 해당 조건을 참고하여 적절한 아키텍처를 선택하고 실제 구현 후 기존 시스템과 ORM 이 적용된 시스템을 비교하여 적절한 아키텍처로 확산이 가능한 지 확인하였다. 결론에서는 향후 추가로 고려되거나 연구되어야 할 영역들을 기입하였다. 이는 기존 ORM 을 적용하지 않은 시스템에서 적용 시 참고할 수 있는 기본 자료로 사용할 수 있을 것이다.

2. 관련 연구

객체 지향 개발 방법론(OOAD)은 오늘날 소프트웨어 개발방법 중 가장 많이 쓰이는 방법론 중 하나이

다.

실제 세계를 모델링 하는 데 있어 추상화를 통해 오브젝트를 도출하고 오브젝트의 관계를 통해 비즈니스 요구사항을 IT 요건으로 도출해낸다. 오브젝트는 크게 속성과 오퍼레이션으로 구성되며 특히 오브젝트 상호간의 연관을 통한 참조방법에 있어 직접적인 오브젝트의 연산 사용은 불가하며 상호 메시지를 주고받거나 노출된 기능 등을 통해 사용한다. 즉 객체는 상태와 행위를 포함하고 다른 객체들과의 관계를 다양한 각도로 입체적으로 표현해 내는 데 중점을 두고 있다.

이에 반해 관계 형 모델은 데이터를 테이블의 단위로 관리하며 테이블은 상호 참조 키를 통해 다양한 데이터의 집합으로 가공이 가능하다. 결국 객체에서 가공, 처리된 정보는 일시적이며 DBMS 를 통해 영구적인 저장과 조회가 가능하다. 시스템 아키텍처에서 이러한 영역을 표현하는 Persistence layer 의 성숙도 모델을 <표 1>에 정리하였다.[2][3]

<표 1 단계별 Persistence Layer 성숙도>

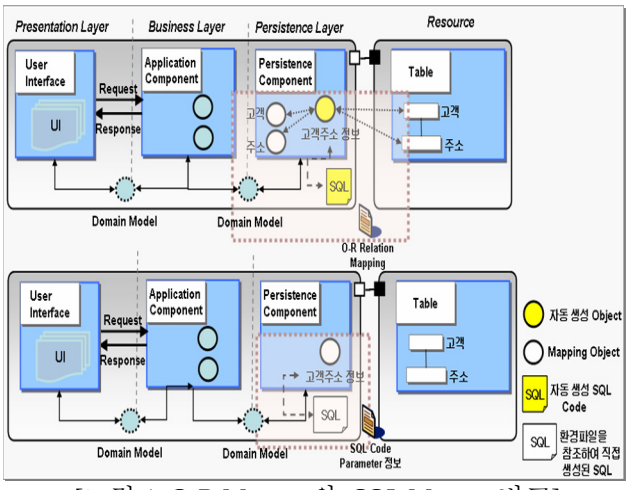
구현 형태	장점	단점
오브젝트 내 SQL 직접 사용 (Embedded SQL)	단순한 접근법,빠른 개발 RDBMS 디자인과 오브젝트 디자인의 차이 관계 없이 적용 가능	오브젝트와 테이블 간 커플링 발생하며 오브젝트와 SQL 둘 다 알아야 함 RDBMS 를 Access 하는 오브젝트 재 활용이 힘듦
Data Base Access Object (DAO)	RDBMS Access Code 와 Business 오브젝트가 분리 DBMS Access Object 재 활용 가능	오브젝트는 DAO 를 통해 여전히 커플 링 발생 SQL 을 알아야 하며 RDBMS 플랫폼 의존적임
Object-Relation Mapping (ORM)	객체지향적인 환경에서 유리하며 재활용성 증대 상황에 따라 코드(SQL 포함)의 변경 없이 다양한 DBMS 지원 가능	개발자의 신 기술 습득 시간 발생 아키텍처의 복잡성 증대 주요 품질 속성 Trade off 발생(성능 등)
Service	플랫폼 독립적 Web Service 의 경우 시장 표준 적용 어플리케이션 간 재활용 가능	서비스는 여전히 발전 중인 단계 메시지 기반이기에 성능 및 대량 데이터 전송 등에 취약

이 중 세 번째 단계에 해당하는 Persistence layer 의 구성은 객체와 테이블 간의 연결(Mapping)이 필요하게 되며 이를 자동으로 처리하는 전략으로 [그림 1]의 Object-Relation Mapping(ORM) 아키텍처의 사용을 고려해 볼 수 있다.[5]

O-R Mapping 을 세분화 해 보면 첫 번째는 그 개념에 충실하게 오브젝트와 테이블을 연결하고, 관계(Relation)를 별도로 정의함으로써 표현하는 방법으로 가장 이상적인 방법이라 할 수 있다.

두 번째는 오브젝트와 테이블 사이의 관계를 SQL 과 SQL 의 집합 오브젝트의 매핑으로 표현하는 방법으로서 관계형 DBMS 의 물리 모델링의 결과와 오브젝트 간의 직접적인 연결이 힘든 경우에 해당된다.

이와 같은 아키텍처 선정을 위해서는 해당 시스템의 특성을 고려하고 아키텍처를 선정 후 검증을 거쳐 평가하는 절차가 필요하다.[4][5][6]



[그림 1. O-R Mapper 와 SQL Mapper 비교]

3. Data Mart 시스템 대상 ORM 아키텍처 적용 사례

적용 대상 시스템은 이동 통신사의 Data Mart 시스템이다. 기간 계 시스템에서 실시간으로 생성된 정보 중 주요 데이터를 매일 주기로 이행하여 Mart 를 구축하며 주요 정보는 고객 정보로서 가입 계약 기준 200 만 건에서 300 만 건 정도이다.

아키텍처 적용을 위해 첫 번째로 기존 어플리케이션의 유형 및 세부 아키텍처를 분류하였다. 기존 개발된 어플리케이션은 대부분 Embedded SQL Application(DAO 형태)으로 구현되어 있으며 세부 유형은 다음과 같이 구분된다.

1) 단순 코드 조회 및 수정: 1~2 개 테이블로 구성된 단순 조회 및 변경 / 전체 프로그램의 20% 정도
2) 복수 개 테이블 조회 : 4 ~ 7 개 테이블로 구성된 복합조회. Legacy SQL 일부 사용존재 / 전체 프로그램의 40%
3) 통계 성 대량 정보 조회: 비즈니스 로직이 DBMS 영역에서 실행되며 화면에 대량 데이터를 출력하는 프로그램 / 전체 프로그램의 40%

우선 세 번째의 대량 데이터를 출력하는 유형은 ORM 아키텍처 적용을 배제하였다. 이는 우선 대량 데이터의 가공 및 처리가 대부분 관계형 DBMS 영역에서 이루어 지고 있다는 점이었다. 또한 화면에 적게는 수천 건에서 많게는 수만 건까지 엑셀이나 리포팅 툴 등을 활용한 다양한 출력형태 또한 ORM 적용이 힘든 사항으로 파악되었다.

두 번째로 시스템의 주요 환경적인 요소 및 주요 요구사항을 파악하였다. 월 1 회 수준의 주요 요구사항 접수 및 유지보수와 데이터 디자인의 물리 모델의 변경에 따라 SQL 문이 바뀌는 경우가 자주 발생했다. 이럴 경우 쉽게 변경 사항이 반영되어야 했었다. 또한 성능도 중요한 품질 속성으로서 기존 시스템은 복잡한 SQL 및 해당 관계형 DBMS 에 특화된 기능 등을 사용하였다. 개발 방법은 어플리케이션 개발자와

RDBMS 개발자가 구분되어 있어 RDBMS 개발자가 분석, 설계를 진행하고 일부 복잡한 SQL 은 직접 구현하여 전달하는 구조였다.

이러한 시스템 고려 요소들을 고려하여 어플리케이션 별 로 선정된 ORM 아키텍처는 <표 2>과 같다.

<표 2. 고려 요소 적용 결과 선정된 ORM 아키텍처>

고려 요소	1.단순 코드 성 데이터 (O-R Mapper)	2.복수 개 테이블 조회성 (SQL Mapper)
오브젝트와 테이블 간 형태	오브젝트와 테이블간의 매핑이 비교적 쉬운 편이다. 코드 정보는 일반 어플리케이션 정보에 비해 상대적으로 변경이 적다.	여러 테이블 Join 을 통한 데이터의 집합으로 표현하며 Inline View 나 Procedure (Oracle 지원)등이 사용된다. 주기적인 요구사항에 따라 테이블이 변경되고 대부분 SQL 로 표현한다.
시스템 환경 요소	당장 다양한 DBMS 를 지원할 필요는 없으나 향후 대형 시스템에 통합될 가능성이 있다. SQL 표현이 다른 어플리케이션에 비해 쉽다. 어플리케이션의 신기술 적용에 적극적인 편은 아니다.	특정 DBMS 에 의존적인 기능을 사용함으로써 다양한 DBMS 를 SQL 의 변경 없이 적용하기 힘들다. 복잡한 SQL 로 구성된다. DBMS 설계자 및 개발자의 영향력이 더 높다.
아키텍처 요소	비즈니스 로직은 단순하며 직접적인 테이블의 CRUD 로 표현된다.	원하는 데이터를 다양한 형태로 Join 하여 출력하는 형태로 SQL 을 구성하는 실력이 필요하다. 대부분의 로직이 SQL 로 처리되고 해당 처리 속도가 중요한 고려요소이다.

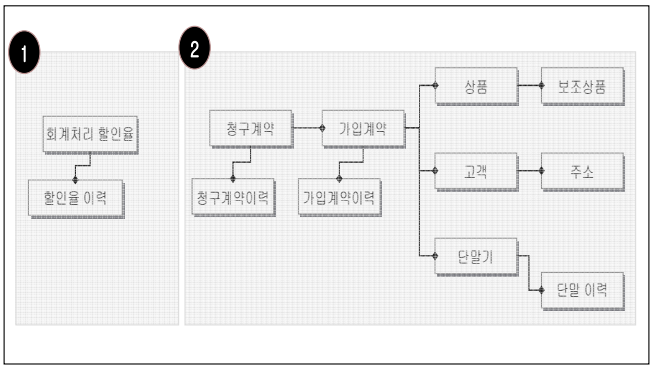
4. 검증 및 결과

이렇게 선정된 아키텍처를 실제 전체 시스템에 확산 적용하기 전 대표적인 어플리케이션을 선정 및 Pilot 을 통해 해당 아키텍처가 적절한 지 평가해 보기로 하였다.

우선 기존 O-R Mapping Framework 으로 사용할 제품을 선별한 결과 순수 O-R Mapper 로는 Hibernate 를, SQL Mapper 로는 iBatis 를 선택하였다. 이유는 이미 각 솔루션이 각각의 기능적 요구사항을 충족하고 다양한 시스템에 기 적용되어 사용되어 있으며 최근 가장 많이 사용되는 제품들로 알려져 있기 때문이다. [7][8]

두 번째로는 유형별 어플리케이션을 [그림 3]과 같이 선정하였다. 선정 기준은 해당 유형의 성격을 가지며 사용 빈도수가 높은 순서로 선택하였다. 해당 결과 코드 성 데이터로는 '회계처리 할인율 관리'가 선정되었으며 두 번째 유형으로는 '청구계약 별 가입고객 현황'을 적용 대상으로 선정하였다.

첫 번째 프로그램인 회계처리 할인율의 경우 실제 화면에서 조회되고 처리되는 테이블과 어플리케이션의 오브젝트와 1:1 로 매핑이 가능한 경우이며 프로그램의 잦은 변경이 발생하지 않는 성격의 어플리케이션이다. 이에 반해 '청구 계약 별 가입고객 현황' 은 요금을 청구하는 고객을 중심으로 해당 고객에 할당되어 있는 가입 고객별로 해당 가입고객의 상품과 고객 정보, 단말기 정보를 조회하는 고객 통합 정보 조회 프로그램이다. 이미 구현된 프로그램은 복잡한 SQL 구문(Inline View, Stored Procedure 등)이 사용 중에 있으며 DBMS 설계자에 의해 결과 테이블이 수시로 변경되고 추가되는 상황이었다.



[그림 3 선정 대상 업무 E-R Diagram]

해당 어플리케이션의 구현 및 Test 를 위해 다음과 같은 개발환경을 구성하여 진행하였다.

- ✓ 운영환경: Window XP / Intel Centrino Duo / 2048 MB
- ✓ S/W: Weblogic 8.1 / Oracle 0g / Hibernate / iBatis
- ✓ Hibernate 적용 프로그램: 회계처리 할인율 관리
- ✓ iBatis 적용프로그램: 청구계약 별 가입고객 현황

실제 Test 할 범위에서 화면(UI)에서 출력되는 구간을 최소화 하기 위해 별도의 인증 체계 없이 데이터를 출력하는 별도 화면으로 구성하였으며 성능 테스트는 Thread 를 이용, 가상 사용자를 발생시켜 입력값으로 다른 Data 를 받아 처리, 응답 시간을 확인하는 것으로 하였다. 또한 성능의 경우 운영시스템에서 실행되는 것이 아니므로 기존 ORM 미 적용 프로그램을 대상으로 응답 시간을 추출하여 적용 프로그램 응답시간과 비교해 보는 방식으로 진행하였다.

우선 시스템의 전체 적용 전 시스템 고려 요소를 조사한 후 해당 내용을 바탕으로 적용 고려 요소를 검토한 후 선정된 아키텍처를 평가하는 형태로 진행하였다.[9]

주요 평가 기준에 따라 검증 대상 모델 별 기본 결과는 다음과 같다.

1) 오브젝트와 테이블간 형태: 둘 다 기존 어플리케이션구성에서 Persistence Layer 의 수정 및 환경설정 등으로 구성 가능하였다. 또한 기존 SQL 의 변경 없이 적용하였다.

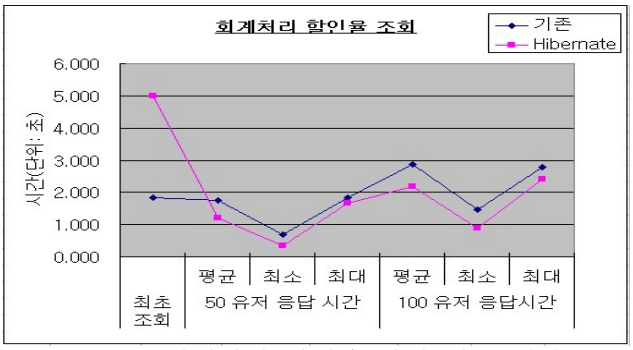
2) 시스템 환경 요소: iBatis 로 표현한 SQL 의 경우 Hint 나 Inline View 등이 포함된 다소 복잡한 구문이었으나 적용 결과 기존 어플리케이션과 동일한 결과값을 가져왔다. Hibernate 가 적용된 회계 처리 할인율 프로그램 또한 명시적인 SQL 의 사용 없이 XML 환경 설정 만으로 기존 SQL 구현과 동일한 표현을 할 수 있었다.

3) 아키텍처 요소: 오브젝트의 생성 및 변경에 따라 Hibernate 로 연결된 테이블 또한 동일하게 발생하

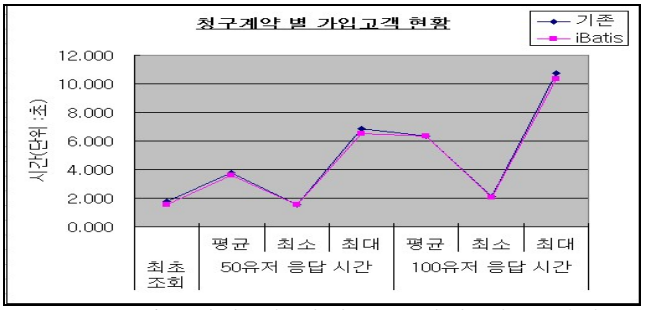
였다. iBatis 또한 기존 프로그램과 동일하게 표현 가능하였다.

4) 개발자 간 의사소통: Hibernate 가 적용된 프로그램의 경우 비교적 단순한 SQL 로 구성된 경우이기에 DBMS 개발자와 직접적인 의사소통은 필요하지 않았다. 그러나 보다 복잡한 SQL 의 경우 로그나 Trace 를 통한 확인이 이루어 지고 SQL Tuning 과 같은 미세한 조절 등이 힘들 것으로 추정된다.

5) 성능 비교: 기존 프로그램과 동일한 환경으로 추출한 결과는 [그림 4], [그림 5]에 표현하였다.



[그림 4. 회계처리 할인율 관리 성능 결과]



[그림 5. 청구계약 별 가입고객 현황 성능 결과]

성능 시험 결과 추정되는 결과는 다음과 같다.

(1) 동일하거나 유사한 성능 제공: Hibernate 나 iBatis 모두 기존 프로그램과 동일하거나 유사한 성능을 제공하였다. Hibernate 의 경우 최초 SQL 실행 시 기존 프로그램 보다 느린 응답 시간을 보였으나 2 회 실행부터는 동일하거나 오히려 더 빠른 시간을 보였다. 이는 최초 실행 시 오브젝트간의 Mapping 정보를 이용 해 사용될 오브젝트를 동적으로 생성하고 해당 RDBMS 의 SQL 을 생성하는 데 걸리는 시간으로 보인다. iBatis 의 경우 기존 프로그램과 거의 동일한 결과값을 보였다. 그러나 성능 요소는 Caching 전략이나 시스템 리소스의 확보 등으로 충분히 보완될 수 있는 요소일 것이다.

(2) O-R Mapper 인 Hibernate 의 경우 SQL 을 직접 쓰지 않고 환경 파일의 mapping 만으로 테이블간의 관계(relation)을 표현 가능하였으며 RDBMS 디자인의 변경에도 속성 값의 변경으로 쉽게 적용이 가능하다. 그러나 해당 RDBMS 에 종속적인 구문은 표현하기 힘들 것으로 보이며 일부 모델링과 기존 코드에 영향이 필요할 수 있다. 그러나 정형화되어 있고 객체 중심적인 모델링으로 RDBMS 까지 구현된 구조에서는 Hibernate 사용을 고려해 볼 수 있을 것으로 추정된다. 또한 다양한 RDBMS 를 기존 어플리케이션 수정 없이 적용 가능하다는 것은 대단한 장점이라 할 수 있다.

(3) iBatis 는 SQL 을 직접 써야 하며 DBMS 디자인 변경에 따라 SQL 을 예전과 동일하게 수정, 반영해야 한다. 또한 평가 기준에는 제외되었으나 시스템 가동 시간(startup time)도 해당 mapping 객체정보를 메모리에 옮기는 과정이 동반 됨으로써 더 많은 시간을 요구하는 것으로 추정된다. 그러나 해당 DBMS 만의 특수한 SQL 을 그대로 사용 가능하며 속도는 기존 시스템과 큰 차이 없이 처리하는 결과를 예측할 수 있었다. 또한 Hibernate 에 비해 단순한 아키텍처와 기존 embedded-SQL 형태의 어플리케이션의 최소화된 변경만으로 적용 가능할 것이라는 결과를 예측할 수 있었다.

5. 결론

본 논문은 ORM 을 미 적용한 시스템에서 적용 시 고려될 다양한 요소들을 바탕으로 적합한 아키텍처를 선정하고 아키텍처의 적합성을 구현 및 검증으로 평가하였다. 단순히 ORM Framework 의 기능을 상호 비교하는 것 보다는 해당 기업 환경에 적합한 아키텍처를 선정하고 평가하는 사례로서 프로젝트 환경에 따라 알맞은 아키텍처 선정 및 진행 방법론으로 참조될 수 있을 것이다.

향후 연구 과제로서는 시스템의 리소스 사용률이나 서버의 기동 속도 등을 고려한 보다 다양한 고려 요소의 보완이 필요하며 다양한 비즈니스 영역별 적용 및 평가가 필요할 것이다. 또한 ORM 적용에 적합한 방법론이나 가이드 또한 고려될 요소이다.

참고문헌

[1]Sabu M. Thampi, Ashwin A.K, Performance Comparison of Persistence Frameworks, Department of CSE L.B.S College of Engineering Kasaragod-671542 Kerala, India, 2007

[2]Scott W. Ambler, Agile Database Technique, Wiley, 2003

[3]Richard Sperko, java persistence for relational database, Apress, 2003

[4]Jeffrey M. Barnes, Object-Relational Mapping as a Persistence Mechanism for Object-Oriented Applications, the Department of Mathematics and Computer Science at Macalester College in Saint Paul, Minnesota, 2007

[5]Melek Oktay 외, Architectural, Technological and Performance Issues in Enterprise Applications, International Journal of Computer and Information Science and Engineering Vol 1 number2, 2007

[6]박현준, "오픈 소스 자바 퍼시스턴스 프레임워크 비교 분석", 국민대학교 석사논문, 2005

[7]http://www.dev2dev.co.kr/pub/a/2003/07/Java_Persistence.jsp

[8]http://unspun.amazon.com/Best-ORM-Layer/list/show/994

[9] Bass,Clements,Kazman, software architecture in practice 2nd, Addison Wesley, 2003