EJB 환경에서 객체지향 상속 관계 설계 패턴

최 시 원† · 김 수 동††

요 약

소프트웨어 개발 생산성 향상과 유지 보수성 향상을 위한 객체지향 분석 및 설계는 학계와 산업계의 기본적인 패러다임으로 자리 잡고 있다. Enterprise Java Beans(EJB)는 높은 이식성을 제공하고 신속한 어플리케이션 개발 환경을 제공하기 때문에, 널리 사용되는 플랫폼중 하나이다. EJB가 객체지향 패러다임을 지원하지만, 객체지향 모델을 EJB 모델로 상세 설계 및 구현 할 때 고려할 사항들이 있다. 그 중 한 가지가 상속 문제이다. 본 논문에서는 객체지향 모델에서 나타날 수 있는 클래스 상속의 유형을 세 가지로 분류하고, 상속을 EJB 모델로 구현할 때 나타날 수 있는 문제점을 규명하며 상속을 지원하는 세 가지 패턴을 제시한다. 또한 제안된 패턴들에 대해서 객체지향 상속 유형별로 적용 가능한 패턴들과 지침을 제시한다.

Design Patterns for Realizing Object-Oriented Inheritance in EJB Environment

Si Won Choi† · Soo Dong Kim††

ABSTRACT

Software development methodology using object-oriented analysis and design techniques for improving productivity and maintainability has acquired a substantial acceptance in both academia and industry as a fundamental paradigm. Enterprise Java Beans (EJB) is widely used in both academia and industry because it provides high portability and rapid application development. EJB supports object-oriented paradigm, but there are several things to consider when designing detail model of object-oriented model or implementing object-oriented model with EJB model. One of them is inheritance problem. In this paper, we classify the types of class inheritance which is shown upon in object-oriented model into three types and identify the problems which can happen when implementing the inheritance mechanism with EJB model. And this paper proposes three patterns for realizing the inheritance in EJB. Moreover, applicable patterns and guidelines for each object-oriented inheritance types for the proposed patterns are suggested.

키워드 : 객체지향 설계(Object-Oriented Design), 상속(Inheritance), EJB

1. 서 론

1.1 동 기

객체지향 패러다임은 데이터와 기능성을 일체화하여 소프트웨어를 효율적으로 개발하고 소프트웨어의 품질을 높여 운용의 노력을 최소화하는 개념이다. 객체지향의 기본 장치 중 하나인 클래스 상속은 시스템 개발에 있어서 재사용성, 수정 용이성 그리고 확장성 등의 장점을 제공한다[1]. 객체지향 설계 모델의 상속 장치를 자바 기반의 서버 아키텍처인 EJB로 구현할 경우, 여러 가지 문제점이 발생한다. 객체지향의 클래스는 EJB의 홈 인터페이스(Home Interface), 컴포넌트 인터페이스(Component Interface), 빈 클래스(Bean Class) 그리고 배치기술서(Deployment Descriptor)등 네 부분으로 분리되어 매핑 된다[2]. 따라서 객체지향의 설계 모델이 EJB의 구현 모델과 불일치하게 되고, 객체지향 패러다임에서 상속 장치의 장점을 구현 모델에 반영하는데 어려움이 발생한다. EJB는 높은 이식성과 신속한 어플리케이션 개발 환경을 제공하기 때문에 산업계와 학계에서 보편화되고 있다. 하지만 EJB에서 상속 관계를 지원하는 장치나 패턴들의 연구가 미흡한 상태이고, 실무적인 지침 또한 충분히 제시되지 못하고 있다. 객체지향의 상속을 EJB에서 사용할 수 있게 될 경우에, 재사용성, 수정용이성 그리고 확장성을 기반으로 적은 비용과 시간으로 효율적인 EJB 기반 시스템을 개발 및 유지 보수할 수 있게 된다.

본 논문에서는 객체지향 설계 모델의 상속을 EJB로 구현할 때 발생하는 두 모델간의 의미적인 불일치를 규명하고, 설계 모델이 EJB 구현 모델로 매핑될 때 상속의 장점이 EJB구현 모델에 반영 될 수 있는 세 가지 패턴과 세부적인 지침을 제시한다.

※ 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음.
† 정 회 원 : 숭실대학교 대학원 컴퓨터학과
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2003년 7월 14일, 심사완료 : 2003년 11월 4일

1.2 논문의 구성 및 범위

본 논문에서는 객체지향 설계 모델에서 나타나는 클래스 간의 상속 관계를 EJB 모델로 구현 할 때 발생하는 두 모델간의 상속에 대한 의미적인 불일치를 규명하고, 개발 효율성과 고성능을 제공해 주는 컨테이너 관리 지속성(Container-Managed Persistence) 기반 엔티티 빈에서의 상속 설계 패턴을 제시한다. 본 논문의 2장에서는 관련 연구들의 성과와 미흡한 점에 대하여 알아보고, 3장에서는 객체지향 패러다임과 EJB 모델간의 의미적인 불일치에 대하여 기술한다. 4장에서는 객체지향 설계 모델의 상속 관계에서 나타날 수 있는 세 가지 유형을 규명하며, 5장에서는 EJB에서 구현될 수 있는 상속의 패턴과 세부 지침을 제안한다. 6장에서는 제안된 패턴에 대한 적용지침과 평가를 한다.

2. 관련 연구

2.1 Rumbaugh 연구[3]

Rumbaugh 연구에서는 코드를 상속하기 위한 방법을 세 가지로 분류하고 있다. 첫째로 서브루틴 방법은 각 메소드에서 호출되는 공통의 코드를 추출하여 하나의 메소드에 위치시키는 기법으로, 보통 공통 메소드는 상위 클래스에 할당될 수 있다. 둘째로 팩터링 방법은 공유해서 사용되는 메소드의 나머지 코드는 남겨두고, 다른 클래스들로 메소드들간의 차이점들을 추출하는 기법이다. 셋째로 델리게이션 방법은 상위 클래스/하위 클래스 관계가 존재하지 않는 경우에, 프로그램 안에서 코드 재사용을 향상시키기 위하여 사용되는 기법이다. 이 방법은 코드 재사용 메커니즘으로 코드 재사용을 원하는 클래스에서는 메소드를 요청하고 실제적인 실행은 다른 클래스로 전달되는 기법이다. Rumbaugh 연구에서는 세 가지 코드 상속에 대하여 개념을 제시를 하고 있으나, 각 기법이 EJB 또는 특정 플랫폼에서 적용될 수 있는 상세한 지침이나 구현 기법에 관한 연구가 필요하다.

2.2 Girdley 연구[4]

Girdley 연구에서는 일반적으로 사용되는 속성들과 비즈니스 메소드를 포함하는 원격 인터페이스(Remote Interface)를 상위 인터페이스로 분리함으로써 하위 빈의 원격 인터페이스에서 상속을 받는다. 홈 인터페이스의 상속에서는 서로 다른 원격 인터페이스와 'create' 메소드를 가져야 하므로, 서로 다른 홈 인터페이스를 갖는다. 그러나 홈 인터페이스에 홈 메소드 또는 'collection' 형태를 반환하는 'finder' 메소드를 포함하는 경우, 상위 홈 인터페이스로 추출하여 상속 관계가 된다. 원격 인터페이스와 홈 인터페이스에 있는 메소드들을 구현한 빈 클래스의 상속은 원격 인터페이스 상속과 같이 공통적인 부분을 상위 빈 클래스에 추가하고 상위 빈을 상속 받아 하위 빈 클래스에서는 각각의 빈에 특정한 메소드들을 추가하여 구현한다. Girdley 연구에서는 EJB 홈 인터페이스, 원격 인터페이스 그리고 빈 클래스의 상속에 대한 개략적인 소개는 되었지만 세부적인 적용 지침이나 구현 기법은 미흡하다.

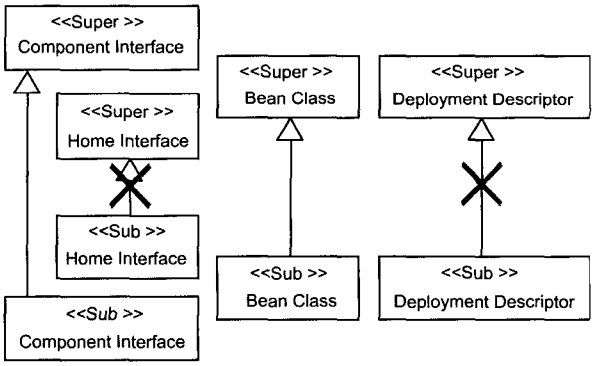
2.3 Marinescu 연구[5]

Marinescu 연구에서는 객체지향 설계 모델에서 상속 관계를 EJB에서 구현 시 제약사항이 많으며, 특히 엔티티 빈에서의 상속은 더욱 힘들다고 기술하고 있다. EJB에서 전체적인 상속은 제약사항이 많으므로 권장하지 않고, 부분적인 소스코드를 재사용 하는 것에 대하여 설명하고 있으나, 이 연구에서도 EJB에서의 상속 구현을 위한 실무적인 지침이나, 구현 사례에 대해서는 언급하고 있지 않다. Marinescu 연구에서는 부분적인 상속에 대한 지침과 상속 효과를 낼 수 있는 패턴들의 연구가 미흡하다.

3. EJB 상속 장치의 한계점

3.1 EJB 상속 장치의 제약사항

EJB는 자바 기반의 서버 아키텍처이기 때문에 객체지향 패러다임에서 제공하는 상속 장치를 동일하게 적용시킬 수 있어야 하지만, EJB 아키텍처상 (그림 1)과 같은 제약사항을 갖는다[6].



(그림 1) EJB의 상속

첫째로 홈 인터페이스의 'create' 메소드의 반환형이 하위 클래스 또는 상위 클래스가 아닌 원격 인터페이스 형태여야 한다. 둘째로 엔티티 빈의 'ejbCreate' 메소드 역시 반환형이 기본키 형태여야 한다. 셋째로 홈 인터페이스의 'findByPrimarykey' 메소드는 반드시 매개변수로서 기본키 클래스를 입력 받고 원격 인터페이스를 반환해야 한다[7, 8].

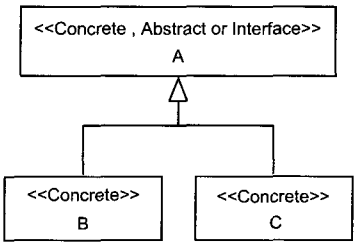
3.2 의미적 불일치

EJB와 객체지향 패러다임에서의 상속 장치 구현을 동일하게 1 : 1 매핑할 경우 문제점이 발생한다. 첫째로 홈 인터

페이스를 상속 받는 경우 EJB의 홈 인터페이스에 선언된 'create'와 'findByPrimaryKey' 메소드는 반환형이 컴포넌트 인터페이스이기 때문에 상위 홈 인터페이스를 상속 받으면 반환형의 충돌이 발생하게 된다. 둘째로 'findByPrimary Key' 메소드는 홈 인터페이스에 반드시 하나만 존재해야 한다. 셋째로 컨테이너 관리 지속성일 경우 상위 빈 클래스를 상속 받아도 속성 접근 메소드는 상속이 되지 않는다. 넷째로 배치기술서는 XML 파일이기 때문에 상속이 불가능하다. 따라서 객체지향 패러다임의 상속 장치를 EJB에서는 완전히 지원하지 못한다[9, 10].

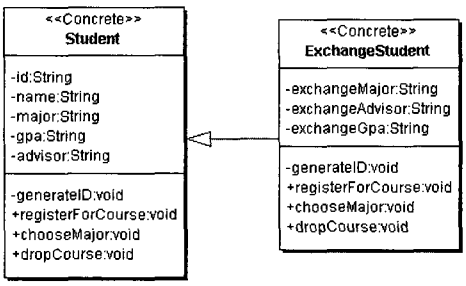
4. 객체지향 설계 모델의 상속 관계 유형

본 장에서는 객체지향 설계 모델에서 자바 기반의 구체(concrete) 클래스가 상속 받을 수 있는 관계를 세 가지 유형으로 분류한다. (그림 2)는 상속 관계에 있는 클래스들의 스테레오 타입을 표현한 것으로 하위 클래스는 모두 구체 클래스만 될 수 있고 상위 클래스는 구체 클래스, 추상 클래스 그리고 인터페이스 등이 될 수 있다.



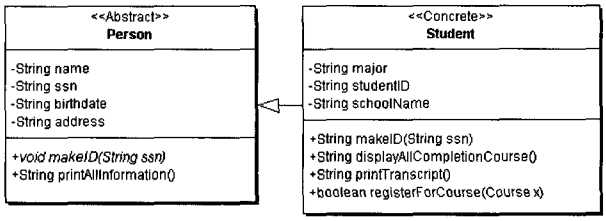
(그림 2) 상속 관계의 클래스 스테레오 타입

첫째 유형으로 상위 와 하위 클래스 모두 구체 클래스인 경우이다. (그림 3)은 학생과 교환학생의 상속 관계를 보여준다. 교환학생 클래스는 상위 클래스의 속성, 메소드 그리고 속성 접근 메소드를 모두 상속 받는다. 교환학생 클래스는 추가적으로 교환학교에서 받은 교환학점, 교환학교에서의 교환지도교수, 교환학교에서의 교환전공등과 같은 속성을 정의한다[11].



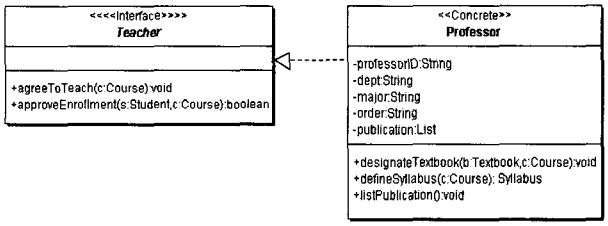
(그림 3) 구체 클래스에서 구체 클래스로 상속

둘째 유형으로 상위 클래스가 추상 클래스이고, 하위 클래스는 구체 클래스인 경우이다. (그림 4)에서의 사람 클래스는 객체를 생성할 수 없으며, 사람의 고유한 속성인 이름, 주민등록번호, 생년월일, 주소를 포함하고, 사람별로 아이디를 생성해주는 추상 메소드인 'makeId'와 각각의 속성을 접근하는 메소드들로 구성된다. 사람 클래스로부터 상속을 받는 학생클래스는 고유한 속성인 학생 아이디, 전공, 학교이름을 가지고 있으며, 반드시 상위 클래스의 추상 메소드를 구현해야 하고, 상위 클래스의 모든 속성과 메소드들을 상속받고 있다[12].



(그림 4) 추상 클래스에서 구체 클래스로 상속

셋째 유형으로 상위 클래스는 인터페이스이고 하위 클래스는 구체 클래스인 경우이다. (그림 5)에서 상위 인터페이스인 교사는 강의진행 여부를 결정짓는 메소드 'agreeTo Teach', 강좌에 등록한 학생들을 승인하는 메소드인 'appro veEnrollment'로 구성되어 있고, 하위 구체 클래스인 교수는 상위 인터페이스의 메소드들을 구현한다[13].



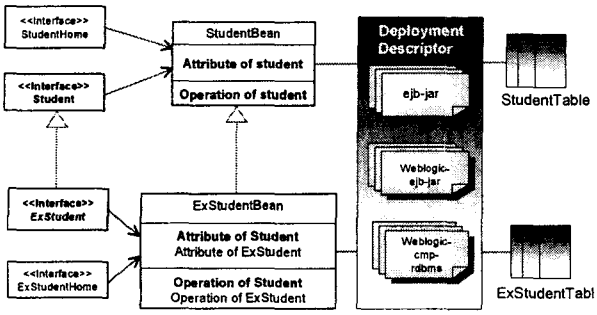
(그림 5) 인터페이스에서 상속 받은 구체 클래스

5. EJB에서 상속 설계 패턴

5.1 수직 복제 패턴

수직 복제 패턴은 상속 관계에 있는 두 클래스에 각각 독립적인 빈 클래스를 정의하고 각 빈 클래스와 하나의 테이블이 연동되는 구조이다. 각 빈의 배치기술서는 별도로 정의하지 않고, 하나의 배치기술서에 두 개의 빈을 기술한다. 이 패턴은 상위 컴포넌트 인터페이스와 빈 클래스는 상속을 받고, 빈 클래스의 속성은 하위 배치기술서에 중복 정의하여 구현함으로 상속의 효과를 간접적으로 표현하고 있다. (그림 6)은 수직 복제 패턴의 전체적인 구조를 표현한 것으로 하위의 교환학생 빈(ExStudentBean) 클래스에서 상위 학생 빈(StudentBean) 클래스의 속성과 메소드의 중복을 보여주고 있다. 빈 클래스의 속성들은 배치기술서의

<cmp-field>에 정의되고, 속성 접근 메소드들은 빈 클래스에 선언되며 <cmp-field>와 1 : 1로 매핑 된다.



(그림 6) 수직 복제 패턴 구조

배치기술서는 상위 배치기술서에 정의된 속성과 접근 메소드들을 하위 배치기술서에서도 중복하여 정의한다. 다음은 학생과 교환학생이 상속 관계에 있을 때 수직 복제 패턴을 사용한 경우이다. (프로그램 1)은 학생 빈 클래스를 상속 받은 교환학생 빈 클래스의 홈 인터페이스이다.

```
public interface ExchangeStudentHome extends EJBHome{
  public ExchangeStudent create(String ID, String name, ···,
    String exchangeMajor) throws CreateException,
    RemoteException ;
  public ExchangeStudent findByPrimaryKey(String primaryKey)
    throws FinderException, RemoteException ;
  ·····
}
```

(프로그램 1) 교환학생 빈의 홈 인터페이스

홈 인터페이스에 정의된 'create' 메소드는 교환학생 빈 인스턴스를 생성하고 'findByPrimaryKey' 메소드는 이미 생성된 교환학생 빈 인스턴스를 기본키로 검색하는 기능을 한다. 이 두 메소드들의 반환형은 원격 인터페이스 타입인 'ExchangeStudent'를 반환한다. 따라서 상위 빈 클래스를 상속 받을 경우, 반환형의 충돌이 발생하기 때문에 홈 인터페이스는 상속을 받지 않고 각각의 빈에 별도의 홈 인터페이스를 정의한다.

(프로그램 2)는 비즈니스 메소드들의 선언으로 구성된 교환학생의 원격 인터페이스다. 원격 인터페이스의 경우 홈 인터페이스와는 달리 'EJBObject'을 상속 받아야 한다는 것 이외에는 별도의 제약사항이 없기 때문에 상위 학생 빈의 원격 인터페이스를 상속 받는다. 이미 학생 빈의 원격 인터페이스에서 'EJBObject'을 상속 받아 구현했기 때문에 교환학생 빈의 원격 인터페이스에서는 상위 학생 빈의 원격 인터페이스를 상속 받는 것이 가능하다. (프로그램 2)에서 교환학생 빈의 원격 인터페이스는 학생 빈의 원격 인터페이스에서 정의된 메소드들을 중복 정의하지 않고, 교환학생 빈의 추가적인 고유 메소드와 재 정의된 메소드만 포함한다.

```
public interface ExchangeStudent extends Student {
  public void chooseMajor (String oldMajor, String newMajor)
    throws RemoteException ;
  ·····
}
```

(프로그램 2) 교환학생 빈의 원격 인터페이스

'chooseMajor'는 학생 빈의 원격 인터페이스에 정의된 메소드로 전공을 선택하는 기능을 하며, 교환학생 빈의 원격 인터페이스에서 재 정의된 메소드이다. (프로그램 3)은 교환학생 빈 클래스로 원격 인터페이스의 구현 부분이다. 빈 클래스는 'EntityBean' 인터페이스를 구현해야 하는 제약사항이 있다. 그러나 교환학생 빈 클래스는 학생 빈 클래스를 상속 하고 있지만 위의 제약사항을 만족한다. 이미 학생 빈 클래스에서 'EJBEntity' 인터페이스를 구현했기 때문이다.

```
abstract public class ExchangeStudentBean extends StudentBean{
        abstract public String getExchangeGpa() ;
        abstract public void setExchangeGpa() ;
        ·····
        abstract public String getExchangeMajor() ;
        abstract public void setExchangeMajor(String
          exchangeMajor) ;
        ·····
  public String ejbCreate (String name, ···, String
    exchangeMajor) throws CreateException {  ·····
        setName(name) ;
        ···
        setExchangeMajor(exchangeMajor) ;
        return null ;
  }
  ·····
}
```

(프로그램 3) 교환학생의 빈 클래스

교환학생 빈 클래스에서는 상위 학생 빈 클래스를 상속 받았기 때문에, 교환학생 빈 클래스의 추가적인 부분만 기술한다. 'setName'과 'setGpa' 메소드는 상위 빈에서 정의된 메소드를 사용하고 있다. (프로그램 4)는 교환학생의 배치기술서중 빈의 속성들을 정의하고 있는 'ejb-jar' 파일이다. XML파일은 상속을 지원하지 않기 때문에 상위 학생 빈의 'ejb-jar' 파일을 상속 받을 수 없다. 따라서 교환학생의 ejb-jar.xml은 상위 학생 빈의 속성 값을 중복해서 정의하고 있으며, 교환학생의 추가적인 속성들을 같이 정의하고 있다.

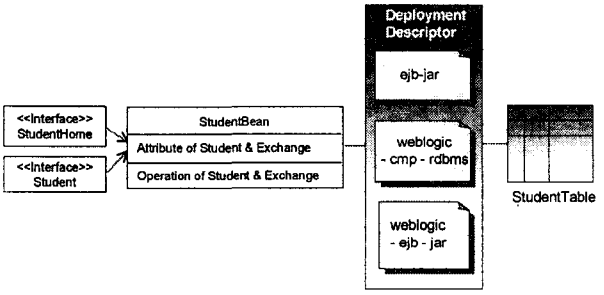
교환학생의 속성과 데이터베이스의 칼럼을 연결시켜 주는 weblogic-rdbms-jar 파일도 xml 형식이므로 역시 상위 학생 빈의 weblogic-rdbms-jar를 상속 받지 못한다. 따라서 학생 빈의 속성과 데이터베이스의 열을 연결시켜주는 부분도 같이 정의해야 한다.

```
<ejb-jar>
<enterprise-beans>
    <entity>
      .....
<abstract-schema-name> ExchangeStudentBean
</abstract-schema-name>
      <cmp-field>
        <field-name> id </field-name>
      </cmp-field>
      <cmp-field>
        <field-name> name </field-name>
      </cmp-field>
       .....
      <cmp-field>
        <field-name> exchangeAdvisor </field-name>
      </cmp-field>
  ...............
</ejb-jar>
```

(프로그램 4) 교환학생 빈의 ejb-jar.xml

5.2 통합 패턴

통합 패턴은 상속 관계에 있는 상위 클래스와 하위 클래스를 하나의 빈으로 합치고, 두 부분으로 나누어져 있는 속성과 메소드들을 하나로 통합하는 방법이다. (그림 7)은 객체 모델의 학생 클래스와 교환학생 클래스가 상속 관계에 있을 때 통합 패턴을 사용하여, 하나의 빈으로 합친 구조를 보여준다. StudentBean은 학생의 속성과 메소드를 가지고 있고, 교환학생의 속성과 메소드도 포함한다. 학생테이블과 교환학생으로 나뉘어져 있던 테이블도 하나의 테이블을 갖는다.



(그림 7) 통합 패턴의 구조

(프로그램 5)는 통합 패턴을 적용한 예로써, 상속 관계에 있는 학생 빈 클래스의 홈 인터페이스와 교환학생 빈의 홈 인터페이스를 하나의 학생 홈 인터페이스로 통합한 경우이다. 학생 홈 인터페이스의 'create' 메소드는 두 개의 빈이 하나로 통합되었기 때문에, 학생의 속성과 교환학생의 속성을 모두 인자로 입력 받고, 하나의 통합된 학생 빈 인스턴스를 생성한다. 또한 학생과 교환학생 두 개의 빈이 통합되었기 때문에 학생속성으로 인스턴스를 찾아오는 메소드와 교환학생 속성으로 학생 빈 인스턴스를 찾아오는 메소드가 포함되어 있다.

```
public interface StudentHome extends EJBHome {
   public Student create (String studentID, String name, ··· ,
      String exchangeMajor) throws CreateException,
      RemoteException ;
   public Student findByPrimaryKey (String studentId) throws
      FinderException, RemoteException ;
   public Student findByExchangeStudentId (String
      exchangeStudentId) throws FinderException,
      RemoteException ;
   ...
}
```

(프로그램 5) 학생 빈의 홈 인터페이스

(프로그램 6)은 통합된 학생 빈의 원격 인터페이스다. 원격 인터페이스에서는 학생 빈과 교환학생 빈에서 필요한 메소드가 선언되어 있다. 통합된 학생 빈의 원격 인터페이스에는 두 개 빈의 고유한 메소드가 선언되기 때문에 메소드 중복은 발생하지 않는다.

```
public interface Student extends EJBObject {
   public void generateId (String ssn) throws RemoteException ;
   public void registerForCourse (String newCourse, String
      newId) throws ProcessingErrorException, RemoteException ;
   public void dropCourse (String newCourse, String newId)
      throws RemoteException;
   public void chooseMajor (String newMajor) throws
      RemoteException;
   public void chooseMajor (String oldMajor, String newMajor)
      throws RemoteException;
   ...
}
```

(프로그램 6) 학생 빈의 원격 인터페이스

(프로그램 7)은 통합된 학생의 빈 클래스이다. 통합된 학생 빈 클래스에는 학생과 교환학생의 속성에 대한 접근 메소드, 콜 백 메소드 그리고 원격 인터페이스에서 정의된 비즈니스 메소드의 구현 부분이다.

```
abstract public class StudentBean implements EntityBean {
...
   abstract public String getExchangeMajor( );
   abstract public void setExchangeMajor (String
      exchangeMajor);
...
   public String ejbCreate (String id, String name, String ssn, ··· )
      throws CreateException {
      setExchangeAdvisor (exchangeAdvisor) ;
      setExchangeMajor (exchangeMajor) ;
      ...
   }
   public void chooseMajor (String newMajor) {
      setMajor (newMajor);
      System.out.println (newMajor +  is selected.) ;
   }
   public void chooseMajor (String oldMajor, String newMajor)
   ...
}
```

(프로그램 7) 학생의 Bean 클래스

홈 인터페이스의 'create' 메소드에서 입력 받은 인자들을 사용하여 'ejbCreate' 메소드는 학생과 교환학생의 속성을 모두 포함한 학생 빈 인스턴스를 생성한다. 학생과 교환학생이 가지는 비즈니스 메소드들을 학생 빈 클래스에서 모두 정의를 한다. (프로그램 8)은 통합된 학생의 ejb-jar 파일이다. 학생의 속성과 교환학생의 속성들을 <cmp-field>로 선언하고, 홈 인터페이스에서 정의 되었던 'finder' 메소드의 EJB Query Language(EJB-QL)를 정의한다. (프로그램 8)의 하단 부분은 'findByExchageStudentId'의 EJB-QL을 나타낸 것이다. 'finder' 메소드를 통해 입력 받은 'studentId'와 일치하는 빈 인스턴스의 레퍼런스를 반환한다.

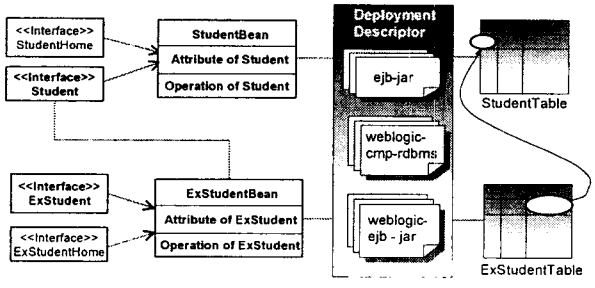
```
<ejb-jar>
  <enterprise-beans>
   <entity>
    <ejb-name> Student </ejb-name>  ...
     <persistence-type> Container </persistence-type>
     <cmp-field> <field-name> exchangeMajor
     </field-name> </cmp>  ...
    <query>
      <query-method> <method-name>
        findByExchangeStudentId </method-name> ···
      </query-method>
     <ejb-ql> <![CDATA[SELECT OBJECT(a) FROM
       StudentBean AS a WHERE a.studentId = ?1]]>
     </ejb-ql>
    </query>
  ...
</ejb-jar>
```

(프로그램 8) 학생 빈의 ejb-jar.xml

통합된 학생의 weblogic-rdbms-jar 파일에서는 ejb-jar에서 <cmp-field>로 정의된 속성과 데이터베이스의 칼럼을 연결시켜 준다. 'studentId'는 실제 데이터베이스의 'studentId'로 연결되고, 교환학생의 교환학교에서의 전공을 의미하는 'exchangeMajor'는 데이터베이스의 'exchangeMajor'에 연결된다.

5.3 위임(Delegation) 패턴

위임 패턴은 상속 관계에 있는 클래스들을 독립적인 빈 클래스로 매핑하고, 각 빈 클래스 별로 독립적인 테이블 구조를 갖는다. (그림 8)은 위임 패턴의 구조를 나타낸다. 상위 빈 클래스인 'StudentBean'에는 학생 고유의 속성과 메소드들을 선언한다. 하위 빈 클래스인 'ExStudentBean'에는 추가적인 교환학생의 속성과 메소드들을 선언한다. 수직 복제 패턴과는 달리 상위 원격 인터페이스와 빈 클래스를 상속 받지 않고, 합성관계를 이용하여 전체-부분의 개념을 사용한다.



(그림 8) 위임 패턴의 구조

위임 패턴은 객체지향 상속의 내부 메커니즘을 적용한 것으로, 실행 시 상위 학생 빈의 속성과 메소드를 사용할 수 있게 한다. 교환학생의 빈 인스턴스가 생성될 때 'ExStudentTable'에 레코드 하나가 생성되고 교환학생의 속성들은 새로 생성된 레코드의 필드에 추가된 후에는 교환학생 빈 인스턴스가 학생의 속성 값과 비즈니스 메소드를 사용가능해야 한다. 따라서 교환학생의 빈 인스턴스 생성이 완료되기 전에 학생의 빈 인스턴스가 생성 되어 있어야 한다. 상위 빈에서 생성된 인스턴스와 하위 빈에서 생성된 인스턴스는 물리적으로 분리 되어 있지만, 논리적으로는 교환학생 빈 인스턴스에 포함되는 하나의 인스턴스로 볼 수 있기 때문에 교환학생 빈 인스턴스에서 학생 빈 인스턴스 참조를 위한 컨테이너 관리 관계(Container-managed relationship)를 정의한다. 이 패턴에서는 상속 관계이기 때문에 상위 빈 인스턴스와 하위 빈 인스턴스가 1 : 1관계 이고, 방향성은 하위 빈에서 상위 빈으로 단 방향을 갖는다. 다음은 학생과 교환학생 클래스가 상속 관계에 있을 때 위임 패턴을 사용한 예이다. (프로그램 9)는 교환학생의 홈 인터페이스이다. 이 패턴에서는 홈 인터페이스에서 교환학생 빈 인스턴스를 생성할 때, 상위 학생 빈의 인스턴스를 생성하기 위하여 필요한 학생 속성을 입력 받는다. 이는 학생 빈으로부터 상속 받은 속성과 메소드들을 하위 빈인 교환학생이 가지고 있는 것이 아니라 상위 빈으로 위임하여 속성의 중복 선언을 안한다.

```
public interface ExchangeStudentHome extends EJBHome {
   public ExchangeStudent create(String studentId, ··· , String
      exchangeMajor) throws CreateException, NamingException,
      RemoteException ;
   public ExchangeStudent findByPrimaryKey (String
      exchangeStudentId) throws FinderException,
      RemoteException ;
 ···
}
```

(프로그램 9) 교환학생 빈의 홈 인터페이스

(프로그램 10)은 교환학생 빈의 원격 인터페이스이다. 위임 패턴에서는 상위 학생 빈의 원격 인터페이스를 상속 받지 않고 각각 독립적으로 구현한다. 상위 원격 인터페이스를 상속 받지 않기 때문에, 메소드 상속의 효과를 주기 위해서 상위 학생 빈의 원격 인터페이스에 정의된 'registerForCourse', 'dropCourse' 그리고 'chooseMajor' 메소드등을 교환학생 빈의 원격 인터페이스에 중복 정의한다.

```
public interface ExchangeStudent extends EJBObject {
    public void chooseMajor (String oldMajor, String newMajor)
        throws RemoteException ;
    public void registerForCourse (String newCourse, String
        studentId)
        throws CreateException, NamingException,
        RemoteException, FinderException, processingException ;
    public void dropCourse (String newCourse, String studentId)
        throws CreateException, NamingException,
        RemoteException,
        FinderException, processingException ;
    public void chooseMajor (String newMajor) throws
        RemoteException ;
    ...
}
```

(프로그램 10) 교환학생 빈의 원격 인터페이스

(프로그램 11)은 교환학생의 빈 클래스이다. 빈 클래스 역시 상위 학생 빈 클래스를 상속 받지 않고 각각 별도의 빈 클래스로 구현한다. 엔티티 빈은 반드시 엔티티 빈 인터페이스를 구현해야 하는 제약 조건이 있다. 엔티티 빈을 구현한 교환학생 빈에서는 콜 백 메소드와 원격 인터페이스의 실제 구현 부분이 포함된다. 교환학생 빈 클래스의 'ejb Create' 콜 백 메소드는 원격 인터페이스에서 입력받은 'cre ate' 메소드의 인자 값들을 가지고 교환학생 빈과 학생 빈 인스턴스를 생성한다.

```
abstract public class ExchangeStudentBean implements
    EntityBean {
...
    public String ejbCreate(String studentId, String name, Strin
        ssn, ··· ) throws CreateException, NamingException,
        RemoteException{
        StudentHome studentHome ;
        Context ctx = new InitialContext() ;
        studentHome = (StudentHome) PortableRemoteObject.nar
            row (ctx.lookup (StudentHome), StudentHome.class) ;
        studentHome.create (studentId, name, major, gpa, advisor) ;
        setExchangeStudentId (exchangeStudentId) ;
        ...
        setExchangeMajor (exchangeMajor) ;
    }
    public void registerForCourse (String newCourse, String
        studentId) throws CreateException, NamingException,
        RemoteException {
        StudentHome studentHome ;
        Context ctx = new InitialContext() ;
        studentHome = new Initial Context() ;
        studentHome = (StudentHome) PortableRemoteObject.nar
            row (ctx.lookup(StudentHome), StudentHome.class);
        Student student = studentHome.findByPrimaryKey
            (studentId) ;
        Student.registerForCourse (newCourse, studentId) ;
        System.out.println (studentId + Student is registered to +
            newCourse) ;
    }
...
}
```

(프로그램 11) 교환학생의 Bean 클래스

엔티티 빈의 'registerForCourse' 메소드는 상위 학생 빈에 정의되어 있는 메소드이다. 상위 엔티티 빈을 상속 받지 않기 때문에, 중복 정의를 통하여 상속 효과를 낼 수 있다. 'ExchangeStudent.registerForCourse' 메소드를 호출 할 경우 교환학생 빈에 정의된 'registerForCourse' 메소드가 호출되고, 메소드 내부에서 상위 학생 빈의 리모트 객체를 참조하여 상위 원격 인터페이스에 정의된 'registerForCourse' 메소드를 호출한다.

```
<ejb-jar>
...
 </enterprise-beans>
 <relationships>
   <ejb-relation>
    <ejb-relation-name> Student-ExchangeStudent
    </ejb-relation-name>
    ...
      <multiplicity> one </multiplicity>
      <relationship-role-source> <ejb-name>
        ExchangeStudent </ejb-name>
      </relationship-role-source>
    <cmr-field> <cmr-field-name> student </cmr-field-name>
    </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role>
    ...
    <multiplicity> one </multiplicity>
    <cascade-delete/> <relationship-role-source>
    <ejb-name> Student </ejb-name>
    ...
</ejb-jar>
```

(프로그램 12) 교환학생 빈의 ejb-jar

(프로그램 12)는 교환학생 빈의 배치기술서 중에서 ejb-jar 파일이다. ejb-jar 파일은 빈의 속성과 빈과의 관계를 표현한다. 위임 패턴에서는 메소드 중복은 허용하지만, 속성은 중복하지 않는 기법이다. 교환학생의 ejb-jar에서는 교환학생의 속성들만 <cmp-field>로 정의하고 상위 빈의 속성들은 정의하지 않는다. 또한 교환학생의 빈이 생성될 때 동시에 생성 되었던 상위 학생 빈의 인스턴스를 연결하기 위하여 <relationships>을 정의한다. 상위 학생 빈과 하위 교환학생 빈은 1 : 1 관계이고, 개념적으로 교환학생이 학생을 포함하고 있다. 또한 교환학생 빈 인스턴스가 생성될 때 함께 생성된 상위 학생 빈 인스턴스는 물리적으로는 서로 다른 상위 학생 빈, 하위 교환학생 빈이지만, 논리적으로는 교환학생 빈 인스턴스에 연결되어있는 하나의 교환학생 인스턴스로 볼 수 있다. 따라서 교환학생 빈 인스턴스가 생성될 때 같이 생성되고, 삭제될 때에도 같이 삭제되어야만 한다. (프로그램 12)에서 <cascade-delete/>가 교환학생 빈 인스턴스와 학생 빈 인스턴스가 동시에 삭제될 수 있게 해주는 부분이다.

(프로그램 13)은 교환학생의 weblogic-rdbms-jar 파일이다. 이 파일은 교환학생의 속성과 데이터베이스의 칼럼을

연결시켜 주는 부분(field-map)과 데이터베이스의 칼럼간의 관계를 표현하는 부분으로 구성된다. <foreign-key-column> 부분은 데이터베이스 테이블의 참조키를 설정해 주는 부분이다.

```
<weblogic-rdbms-jar>
 <weblogic-rdbms-bean>
  <ejb-name> ExchangeStudent </ejb-name>
  <data-source-name> dataSource-oraclePool
  </data-source-name>
  <table-name> ExchangeStudent </table-name>
   ...
   <field-map> <cmp-field> exchangeMajor </cmp-field>
  <dbms-column> exchangeMajor </dbms-column>
  </field-map>
 <weblogic-rdbms-bean>
 <weblogic-rdbms-relation>
  <relation-name> Student-ExchangeStudent </relation-name>
   <weblogic-relationship-role> <relationship-role-name>
     ExchangeStudent-Has-Student </relationship-role-name>
    <column-map> <foreign-key-column> studentId
    </foreign-key-column> <key-column> studentId
    </key-column> </column-map>
   </weblogic-relationship-role>
  </weblogic-rdbms-relation>
  <create-default-dbms-tables> True
  </create-default-dbms-tables>
</weblogic-rdbms-jar>
```

(프로그램 13) 교환학생 빈의 weblogic-rdbms-jar

6. 적용 지침 및 평가

본 장에서는 앞서 제안된 EJB 상속 관계 설계 패턴과 기존 관련 연구들과의 비교 평가를 하며, 객체지향의 상속 유형별 적합한 EJB 상속 관계 설계 패턴을 제안한다.

〈표 1〉 EJB 상속 관계 설계 패턴 별 특징

○ : 지원, △ : 간접지원

	수직복제패턴	통합패턴	위임패턴
속성 재사용성			○
메소드 재사용성	○		
속성 수정 용이성		○	○
메소드 수정 용이성	○	○	○
속성 확장성			△
메소드 확장성	○		△
상위클래스 무결성	○		○
하위클래스 무결성			

<표 1>은 제안된 EJB 상속 관계 설계 패턴에 대하여 속성 재사용, 메소드 재사용, 속성 수정 용이성, 메소드 수정 용이성, 속성 확장성, 메소드 확장성, 상위클래스 무결성 그리고 하위클래스 무결성등 총 여덟 가지 기준으로 평가한다.

수직복제패턴은 메소드에 대하여 재사용성과 수정 용이성은 지원하고, 속성에 대하여 재사용성과 수정 용이성은 XML에서 상속의 제약으로 지원하지 않는다. 또한 상위 클래스에 잘 정의된 메소드와 속성이 패턴적용 후에도 지켜지는지의 여부를 상위 클래스 무결성이라 하고 하위 클래스에 잘 정의된 메소드와 속성이 패턴적용 후에도 잘 지켜지는지의 여부를 하위 클래스 무결성이라고 한다. 수직복제 패턴은 상위 빈 클래스에 있는 속성들을 하위 빈 클래스에서 중복 정의 하므로, 상위클래스 무결성을 제공하고, 하위 클래스의 무결성은 제공하지 않는다.

통합 패턴은 상위 빈 클래스와 하위 빈 클래스가 하나의 빈 클래스로 통합되기 때문에 속성 재사용성과 메소드 재사용성 측면에서 의미적인 차이가 있다. 상위, 하위 빈 클래스가 통합되었기 때문에 속성과 메소드를 수정 할 경우 한 번만 수정하면 되므로 속성 수정 용이성과 메소드 수정 용이성을 지원한다. 속성 확장성과 메소드 확장성의 경우에 객체지향의 확장성과 같은 유연성을 제공하지 않는다. 통합 패턴에서의 상위 클래스 무결성과 하위 클래스 무결성은 상위 빈 클래스와 하위 클래스가 통합되었기 때문에 상위 빈 클래스 측면에서 보면 하위 빈 클래스의 정보가 포함되었기 때문에 잘 정의되었다고 볼 수 없고, 하위 빈 클래스 측면에서 보면 상위 빈 클래스의 정보가 포함되었기 때문에 역시 잘 정의되었다고 볼 수 없기 때문에 상위 클래스와 하위 클래스의 무결성은 유지하지 않는다.

위임 패턴은 상위 빈 클래스와 하위 빈 클래스에 각각 잘 정의된 속성들을 정의 하므로 속성 재사용성을 지원한다. 메소드 재사용성의 경우 하위 빈 클래스에서 상위 빈 클래스를 상속을 받지 않으므로, 하위 빈에서 상위 빈의 메소드들을 다시 정의해 주어야 하기 때문에 메소드의 재사용성은 지원하지 않는다. 각 빈 클래스에는 잘 정의된 속성들이 있기 때문에 상위 빈의 속성을 수정할 경우 하위 빈에서도 변경 내용이 적용 되므로 수정 용이성 측면에서 유연성을 갖는다. 메소드 수정 용이성의 경우에는 상위 빈의 메소드를 하위 빈에서 정의하면, 하위 빈에서는 메소드를 위임하는 형태로 사용하기 때문에 메소드 수정 용이성이 지원 가능하다. 속성 확장성은 하위 빈 인스턴스가 생성될 때 상위 빈의 인스턴스를 생성시키는 상위 빈의 'create' 메소드로 위임해야 하고, 메소드 확장성 역시 하위 빈에서 상위 빈으로 위임되어야 하므로, 간접적인 확장성을 제공한다. 상위 빈 클래스의 경우 잘 정의된 속성과 메소드만 존재 하므로 상위클래스 무결성은 지원하며, 하위 빈 클래스는 잘 정의된 속성을 가지고 있지만 메소드의 경우 상위 빈 클래스의 메소드를 재정의 하여 잘 정의된 메소드로 구성되지 않았으므로 하위클래스 무결성은 지원하지 않는다. 엔티티 빈의 상속 관계를 구현할 때 각 패턴 별 특징들을 고려하여 상속 관계를 구현할 수 있다.

<표 2>는 기존에 연구된 상속 설계 기법과 본 논문에서 제안한 세 가지 상속 설계 기법을 비교 평가한 것으로, EJB

〈표 2〉 상속 설계 기법들과의 비교평가

○ : 지원, △ : 간접지원

관련연구 \ 평가기준 / 관련기법		속성재 사용성	메소드재 사용성	속성수정 용이성	메소드 수정용이성	속성확장성	메소드 확장성	상위클래스 무결성	하위클래스 무결성
Rumbaugh연구	서브루틴				△		△	△	△
	팩터링				△		△	△	△
	델리게이션				○		△	○	
Girdley연구					△		△		
Marinescu연구					△		△		
본 논문의 연구	수직복제패턴		○		○		○	○	
	통합패턴			○	○				
	위임패턴	○		○	○	△	△	○	

에서 상속지원 여부를 여덟 가지 평가항목으로 나누어 각 기법 별로 지원 또는 간접지원 여부를 나타낸다. Rumbaugh의 연구에서 제안된 세 가지 기법은 EJB 환경을 고려하지 않은 상속 기법으로 주로 메소드내의 코드 재사용을 주목적으로 제안되었기 때문에 메소드 수정 용이성, 메소드 확장성, 상위 클래스 무결성 그리고 하위 클래스 무결성의 항목에서 간접지원 또는 지원 형태를 보여주고 있다. Rumbaugh의 연구기법중의 하나인 델리게이션 기법과 본 연구의 위임 패턴과의 차이점은 Rumbaugh의 연구에서는 객체지향기반에서 상속 관계가 존재하지 않을 때, 메소드의 소스코드 재사용 측면을 다룬 기법이고, 본 연구의 위임 패턴은 EJB 기반에서 메소드의 소스코드 재사용 측면뿐만 아니라, 두 빈 클래스간의 속성 재사용성, 속성 수정 용이성 그리고 속성 확장성 측면을 고려한 점에서 차이가 있다. Girdley와 Marinescu 연구에서는 EJB 환경에서 메소드 재사용 측면을 다루고 있기 때문에 메소드 수정 용이성과 메소드 확장성 측면에서 부분적으로 지원하고 있다.

<표 3>은 자바 플랫폼 기반에 구체 클래스가 상속받을 수 있는 모든 유형별로 적합한 EJB 패턴으로, <표 1>에서 상속의 여덟 가지 장점과 객체지향 패러다임에서의 구체 클래스가 상속 받을 수 있는 모든 상속 유형별 특징을 고려하여 각 유형에 적합한 패턴을 제안한다.

〈표 3〉 상속의 유형별 적합한 EJB 패턴

①: 최적합, ②: 적합, ③: 적용가능

OOP 상속유형 \ EJB 패턴	수직복제패턴	통합패턴	위임패턴
구체클래스에서 구체클래스로 상속	②	③	①
추상클래스에서 구체클래스로 상속	②	①	③
인터페이스에서 구체클래스로 상속	②	①	③

상위 클래스가 구체 클래스로부터 상속받는 경우는 일반적으로 위임 패턴을 사용하는 것이 가장 적합하다. 객체지향의 상속 개념과 가장 근접한 방법이기 때문이다. 상위클래스가 추상 클래스 또는 인터페이스인 경우에는 통합패턴이 가장 적합하다. 추상 클래스 또는 인터페이스의 경우 실제로 빈 인스턴스를 생성하지 않기 때문에, 위임 패턴이나 수직복제패턴을 사용할 경우 테이블과 1 : 1 매핑 되므로 저장장치 낭비가 발생하게 되므로, 일반적으로 통합 패턴이 적합하다.

7. 맺 음 말

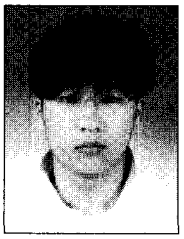
객체지향의 상속으로 인하여 재사용성, 수정 용이성, 확장성이 증가 하면서 고품질의 소프트웨어를 단기간에 개발할 수 있게 되었다. 그러나 객체지향 모델을 EJB 모델로 상세설계 및 구현할 때 고려할 제약사항들이 있다. 본 논문에서는 객체지향 설계 모델에서 나타나는 상속 관계를 명세하였고, 클래스간의 상속 관계를 EJB 컨테이너 관리 지속성기반 엔티티 빈으로 구현할 때 발생하는 문제점을 규명하였으며, EJB에서 객체지향의 상속의 장점을 사용할 수 있는 수직복제패턴, 통합패턴 그리고 위임패턴을 제안하였다. 비록 EJB가 자바 기반의 서버 아키텍처이지만 EJB의 특성상 순수 객체지향에서의 상속을 동일하게 지원하지 않는다.

그러나 본 논문에서 제안한 세 가지 패턴을 적용함으로써, 코드의 재사용으로 소프트웨어 생산성을 향상할 수 있고, 하위 빈으로 상속된 속성과 메소드는 상위 빈에서 수정할 경우 변경된 내용이 상속 받은 하위 빈으로 적용가능하므로 수정 용이성을 제공하여 도메인의 기능성이 변경되어도 빠른 수정이 가능하다. 또한 도메인이 확장될 경우 상위 빈을 상속 받는 하위 빈을 추가할 수 있기 때문에 유연한 확장성을 제공한다. 따라서 EJB 기반 소프트웨어를 개발할 때 본 논문에서 제안된 패턴을 적용할 경우 속성과 메소드의 재사용성, 수정 용이성 그리고 확장성의 장점을 사용하여 EJB 기반 고품질의 어플리케이션을 신속하게 개발할 수 있다.

참 고 문 헌

[1] OMG, UML Specification v1.4, OMG, Inc., September, 2001.

[2] Roman, E., *Mastering Enterprise JavaBeans Second Edition*, John Wiley and Sons, Inc., 2002.
[3] James, R., *OBJECT-ORIENTED MODELING AND DESIGN*, Prentice-Hall, Inc., 1991.
[4] Girdley, M., *J2EE Applications and BEA Weblogic Server*, Prentice Hall PTR, 2002.
[5] Marinescu, F. *EJB Design Patterns*, WILEY, 2002.
[6] Matena, V., *Applying Enterprise JavaBeans*, Addison Wesley, 2001.
[7] Gomez, P., *Professional Java 2 Enterprise Edition with BEA Weblogic Server*, Wrox Press, 2000.
[8] Perrone, P., *Building Java Enterprise Systems with J2EE*, Sams Publishing, 2000.
[9] Enterprise JavaBeans Specification, Version 2.0 Final Release, Sun Microsystems, 2001.
[10] Alur, D., *CORE J2EE Patterns*, Sun Microsystems Press/ Prentice Hall PTR, 2001.
[11] Larman, C., *Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall PTR, 2002.
[12] Arrington, C., *Enterprise Java with UML*, John Wiley and Sons, Inc., 2001.
[13] Gamma, E., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.



최 시 원

e-mail : swchoi@otlab.ssu.ac.kr
2000년 삼육대학교 컴퓨터과학과(학사)
2002년 숭실대학교 대학원 컴퓨터학과 (공학석사)
2002년~현재 숭실대학교 대학원 컴퓨터학과 박사과정
관심분야 : 객체지향방법론, CBSE, 소프트웨어 아키텍처, MDA

김 수 동

e-mail : sdkim@ssu.ac.kr
1984년 Northeast Missouri State University 전산학 학사
1988년~1991년 The University of Iowa 전산학 석사/박사
1991년~1993년 한국통신 연구개발단 선임연구원
1994년~1995년 현대전자 소프트웨어연구소 책임연구원
1995년~현재 숭실대학교 컴퓨터학부 부교수
관심분야 : 객체지향 개발 방법론, 컴포넌트 개발 방법론, 소프트웨어 아키텍처, MDA