객체 지향 기반 개발에서 재사용성 컴포넌트 설계 방법

안희수*, 박만곤
부경대학교 전자계산학과

Method for Design of Component Reusability in Object-Oriented Based Development

Hee-Soo Ahn, Man-Gon Park
Dept. of Computer Science, PuKyong National University*

요 약

소프트웨어를 개발하는데 기존의 개발된 시스템의 컴포넌트를 재사용하면 생산성 향상과 신뢰성 향상, 생산 원가를 절감할 수 있으며, 컴포넌트는 프로그램 설계시 소프트웨어에서 공통적으로 이용될 수 있는 부분들을 표준화하고 이들을 새로운 소프트웨어 개발 과정에서 재사용 함으로써 소프트웨어 개발 기간을 단축시킬 수 있다 소프트웨어 개발 초기에서 견고한 시스템 아키텍쳐를 정립하는 것이 프로젝트 성패에 중요한 요인으로 등장하고, 재작업을 줄이고, 재사용성, 확장성, 시스템 품질측면에서 많은 장점을 얻을 수 있다.

1. 서론

컴포넌트는 언제, 어디서나, 누구나 필요한 정보를 쉽게 얻을 수 있는 인터넷 환경이 보편화되고, 이 기종 컴퓨터간 연동 기술의 발전으로 인터넷상의 다양한 소프트웨어 부품을 기종에 관계없이 사용할 수 있게 되었으며, 소프트웨어 plug and play 기술의 발전으로 소프트웨어 조립 기술이 등장했다.

부품을 조립해서 상위 부품을 만드는 것처럼 부품화된 소프트웨어를 조립하여 완성된 소프트웨어를 만들어 낼 수 있는데, 이러한 독립된 단위 기능의 소프트웨어 부품들을 컴포넌트라 한다.

그룹웨어 패키지는 전자게시판, 워크 플로우, 사용자 인터페이스, 데이터 관리, 전자우편 등의 단위업무를 처리하는 소프트웨어 부품으로 구성된다. 좀더 기술적으로 정의한다면 컴포넌트라 하면 시스템을 구성하는 물질적이고도 대체가 가능한 부품들이라고 정의되는데, 소프트웨어 부품 자체적으로 독립된 업무기능을 수행할 수 있어야 하고, 다른 소프트웨어 부품들간에는 인터페이스를 통하여 정보를 주고받는 단위를 말한다.

2. 모듈성 개념

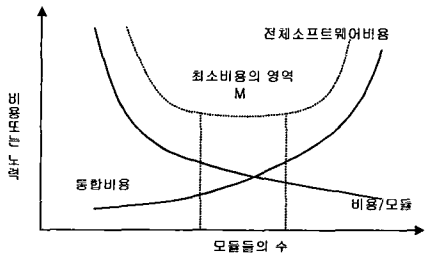
컴퓨터 소프트웨어에서 모듈성(Modularity)의 개념은 오래 동안이나 연구되었다. 소프트웨어 아키텍처는 모듈성을 구체화한 것이다. 즉, 소프트웨어는 개별적으로 이름과 주소를 부여할 수 있는 컴포넌트들로 나누어지고 이 컴포넌트들을 모듈(Module)이라고 부르며, 이 모듈들은 문제의 요구 사항을 만족시키기 위해 통합된다.

모듈성은 프로그램을 지능적으로 다루어지게 해주는 소프트웨어의 단일 속성을 나타낸다. 거대한 단일 소프트웨어는(한개의 모듈로 만들어진 대형 프로그램)는 쉽게 내용을 파악할 수 없다. 재어 경로의 개수, 참조의 간격(span), 변수의 수, 거리고 전체적인 복잡도 등이 이해를 거의 불가능하게 만든다. 이러한 점을 설명하기 위해 인간이 문제를 푸는 방식에 기초해서 다음과 같이 생각할 수 있다.

함수 C(x)는 문제 x의 인식된 복잡도를 정의하는 함수이고, E(x)는 문제 x를 푸는데 요구되는 노력(시간 단위)을 정의하는 함수라고 할 때, 문제 P1, P2에서 만약에 C(p1) > C(p2) 라면 다음과 같이 된다.

E(p1) > E(p2) (식1) 일반적인 경우처럼 이 결과는 직관적으로 명백하다. 즉, 어려운 문제를 푸는 데 보다 많은 시간이 소요된다. 다른 흥미있는 특징은 인간이 문제를 푸는 실험을 통해 다음과 같이 발견되었다. C(p1+p2) > C(p1) + C(p2) (식2) (식3)은 p1과 p2를 합친 문제의 인식된 복잡도가 각 문제를 독립적으로 고려한 문제의 인지된 복잡도 보다 크다는 것을 의미한다. 즉, 식(2)와 식(1)이 의미하는 조건을 고려해 보면 다음과 같다. E(p1+p2) > E(p1) + E(p2) (식3) 식(3)은 복잡한 문제를 풀 때 문제를 관리하기 쉬운 작은 단위로 분할해서 푸는 것이 쉽다는, 즉, "분할해서 정복하라(divide and conquer)" 라는 결론을 이끌어 낸다. 식(3)의 결론은 모듈성과 소프트웨어에 관해 중요한 의미를 갖고 있다. 실제로 이것이 모듈성의 근거이다. 식(3)의 결론은 우리가 소프트웨어를 무한히 작은 단위로 분할하면 그것을 개발하는데 요구되는 노력은 무시할 정도로 아주 적다고 결론을 내릴 수 있다. 그러나 불행하게도 다른 인자들이 삽입되어 이러한 결론은 쓸모 없다. (그림: 1)은 개별 소프트웨어 모듈을 개발하는 데 드는 노력(비용)은 모듈의 전체 개수가 증가하면 감소한다. 같은 요구 사항이 주어진 경우, 모듈의 수가 증가하면 개별 모듈의 크기는 작아진다. 그러나 모듈의 수가 증가하면, 모듈을 통합하는데 연관된 노력(비용)은 증가한다. 이들 특성은 (그림: 1)에서 보여주는 전체 비용 또는 노력 곡선이 된다.

모듈의 개수가 M일 때 최소 개발비용이 들지만, 우리는 M을 정확하게 예측하는 데 필요한 기법을 갖고 있지 않다. 우리는 M의 근방에 있을 수 있도록 관심을 갖고 모듈화 시켜야 한다. M의 근방을 알 수 있는 방법을 평균 이하의 모듈성과 평균이상의 모듈성을 피해야 한다.



(그림: 1) 모듈성과 소프트웨어 비용

2.1 컴포넌트 기술

컴포넌트 관련 기술은 컴포넌트 아키텍처 기술, 컴포넌트 생성 및 조립기술, 컴포넌트 시험 및 검증 기술, 응용 컴포넌트 구축 기술, 컴포넌트 저장 및 운용 관리 기술 등으로 나누어진다.

컴포넌트 아키텍처 기술은 OMG(Object Manageme-nt Group)에서 CORBA(Common Object Request B-roker Architecture)를 표준 아키텍처로 제공하고 있고,Microsoft사에서는 COM(Component Object Mode-l)/DCOM(Distributed COM)을 개발하여 자사의 컴포넌트 표준으로 하고 있으며, SUN사가 주축이 되어 JAVA 표준 아키텍처를 EJB(Enterprise JabaBeans) 개발하였다.

컴포넌트 기반 모델링 도구 분야는 Rational사의 Roserk UML 기반으로 제품화하고, Sterling사의 COOL이 Catalysis 기반으로 상품화했다. Microsoft사가 기존의 OCX, ActiveX 컴포넌트 생성 및 개발도구인 Visual Basic을 COM/DCOM기반으로 상용화하였다. SUN, Inprise 등에서는 JavaBeans 컴포넌트를 활용하여 개발도구를 상품화하고 있으며 향후 CORBA, COM 및 JAVA와의 연동 기술을 모색할 것으로 보인다.

2.2 컴포넌트 응용 구축 기술

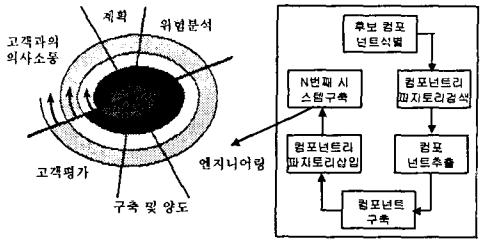
클래스의 재사용은 수정을 한 후 재사용하는 화이트 박스 재사용과 수정없이 재사용하는 블랙박스 재사용으로 나눌 수 있다. 블랙 박스 재사용의 경우는 클래스의 독립성이 중요한 품질 기준이 된다. 클래스의 독립성은 정보은닉, 클래스 사이의 결합도, 응집도 등으로 정량화 할 수 있다. 화이트 박스 재사용은 수정성이 중요한 품질 기준이 된다. 수정성은 클래스의 복잡도, 상속도, 결합도, 응집도 등에 의해 정량화 될 수 있다. 응용 컴포넌트 구축 기술은 도메인 분석 및 모델링 기술, 컴포넌트 추출 기술, 응용 컴포넌트 설계 및 구현 기술로 나누어진다. IBM사는 San Frascic-o 프로젝트를 통해 1200개의 비즈니스 응용 컴포넌트로 구성되는 프레임워크를 구축했으며, Microsoft의 ActiveX에 대하여 SUN사에서 JavaBeans가 등장하여 클라이언트 수준의 컴포넌트 개발 및 활용이 증가하고 있는데, Microsoft사의 MTS(Microsoft Transactio-n Server)와 SUN의 EJB(Enterprise JavaBeans)가 양대 표준으로 경합중으로 엄청난 소프트웨어 개발 형태에 일대 변혁을 가져올 것이다.

2.3 소프트웨어 컴포넌트의 종류

컴포넌트에는 세 가지 유형이 존재한다.

첫째: 배치; 컴포넌트(deployment component) 배

치 컴포넌트는 동적 라이브러리(DLL: dynamic Link Library) 및 실행(EXE: executable)등과 같 이 실행 가능한 시스템을 구성하는데 필요한 컴포넌트이고, 둘째: 작업 산출물 컴포넌트(work product componen -t)이다. 작업 산출물 컴포넌트는 개발 프로세스의 산출물로서 배치 컴포넌트를 작성할 수 있는 소스코드 파일, 데이터 파일로 구성된다. 세 번째: 실행 컴포넌트 COM + Object 등과 같은 시스템 실행 결과물로서 실행물(Executable), 라이브러리(Library), 테이블(Table), 파일(File), 도큐먼트(Document)의 다섯 가지의 대표적인 종류로 나누어진다.



(그림: 2) 컴포넌트 어셈블리 모형

2.4 모듈의 크기 정의

어떻게 적정한 모듈의 크기를 정의할 수 있는가 하는 방법은 시스템내에 있는 모듈을 정의하는 데 사용되는 방법이 있다. 우리가 효과적인 모듈과 시스템을 정의하는 능력과 관련된 설계 방법을 평가할 수 있는 5개 평가 기준을 다음과 같이 정의하였다.

■ Modular decomposability : 만약 설계 방법이 문제를 부프로그램으로 분할하는 체계적인 기법을 제공한다면, 전체 문제의 복잡도는 감소되고 이로 인해 효율적인 모듈과 해결 방안이 이루어진다.

■ Modular composability : 만약 설계 방법으로 기존(재사용 가능한)의 컴포넌트를 새로운 시스템에 조립시킬 수 있다면, 전체를 다시 고안하지 않는 모듈과 시스템이 된다.

■ Modular understandability : 만약 모듈이 독립 단위(다른 모듈을 참조하지 않는)로 이해된다면, 그것은 구축하기도 쉽고 변경하기도 쉽다.

■ Modular continuity : 만약 시스템 요구 사항의 작은 변경이 시스템 전체의 변경이기보다는 개별 모듈의 변경이라면, 변경 때문에 생긴 부작용은 최소화된다.

■ Modular protection : 만약 정도에 벗어난 조건이 모듈내에 발생하고 그 결과가 그 모듈내에 강요되면, 오류 때문에 생긴 부작용의 충격은 최소화 된다.

결과적으로 시스템이 단일체(monolithic)로 구현되었다면, 시스템은 모듈 형식으로 설계된다는 것을 아는 것이 중요하다. 부프로그램(예, 서브루틴, 프로시저) 때문에 발생된 최소 속도와 메모리 과부하(over head)가 수용되지 않은 상황(예, 실시간 소프트웨어, 마이크로프로세스 소프트웨어)이 있다. 이러한 상황에서 소프트웨어는 최우선 정책으로 모듈화를 설계할 수 있고, 설계되어야 한다. 비록 프로그램 원시 코드가 첫눈에 모듈화가 보이지 않을지라도, 이 정책이 유지된다면 이 프로그램은 시스템의 이점들을 제공해 줄 것이다.

3. 컴포넌트 기반 소프트웨어 개발 기법

컴포넌트 자체를 개발하는 컴포넌트기반 개발(CBD : Component Based Development)기법과 기 개발된 컴포넌트를 가지고 컴포넌트 기반 소프트웨어를 개발(CBSD: Component Based Software Development) 기법의 두 가지가 있다.

■ 컴포넌트 기반 개발 기법(CBD)은 어떻게 하면 좋은 컴포넌트를 개발할 것인가에 초점이 맞추어져 있다.

■ 컴포넌트 기반 소프트웨어 개발 기법(CBSD)은 어떻게 하면 이미 구축되어 있는 컴포넌트를 잘 조합하여 좋은 애플리케이션을 개발할 것인가 하는 문제를 다룬다. 컴포넌트는 인터페이스를 통하여 외부 컴포넌트나 개체와 통신하는데, 컴포넌트의 인터페이스는 자신이 외부에 표출하는 Provided Interface와 외부에 요구하는 Required Interface가 있다.

① 컴포넌트 기반 소프트웨어 개발 기법은 후보 컴포넌트를 찾는 것으로부터 시작한다. 애플리케이션에서 관리될 데이터와 그 데이터를 관리하는 데 적용될 알고리즘을 찾는 것에서부터 시작한다. 후보 컴포넌트가 식별되면 컴포넌트 리파지토리(Repository)가 검색되고 그러한 컴포넌트가 존재하는 지를 찾는다. 만약 존재한다면 그 컴포넌트는 추출되어 재사용되고 후보 컴포넌트가 존재하지 않으면 새롭게 개발된 컴포넌트를 사용하여 애플리케이션을 개발한다.

컴포넌트 기반 소프트웨어 개발 모델은 소프트웨어의 재사용을 가져오며, 재사용성은 소프트웨어 개발자에게 여러 가지 장점을 가져다 준다.

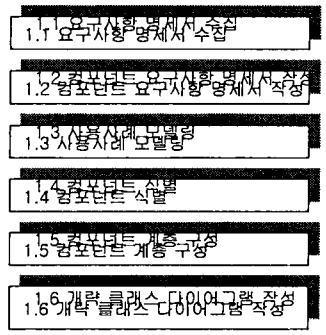
3.1 EJB기반 컴포넌트 설계 기법

본 논문에서 소프트웨어 컴포넌트를 활용한 정보시스템 구축 방법은 SUN사의 EJB를 이용한 정보 시스

템을 구축하는 방법을 예로 들었다.

1) 분석 단계

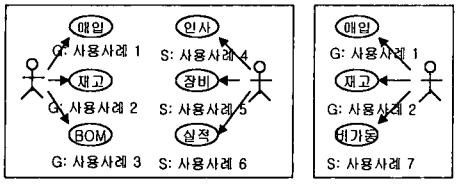
분석 단계는 도메인에 대한 이해 및 컴포넌트에 대한 요구사항을 추출하는 단계로서 6개의 업무로 구성되었다.



(그림: 3) 분석 단계

1.2 요구사항 명세서 작성: 요구사항 명세서로부터 기능적 요구사항 들만 추축한다. 공통적 요구사항과 몇몇 애플리케이션에만 명시된 기능적 요구사항 구분 표시한다.

1.3 사용사례 모델링 : 컴포넌트 요구사항 명세서에 대해 사용사례 모델링한다.



(그림: 4) 사용사례 다이어그램 A 사용사례다이어그램 B

1.4 컴포넌트 식별 : 사용사례 모델링 단계에서의 산출물을 바탕으로 컴포넌트 단위를 식별한다.

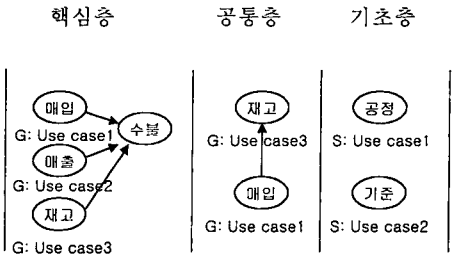
1.6 도메인에 대한 비즈니스 객체들을 식별하고 객체들간의 관계와 속성을 식별한다.

2) 설계 단계

컴포넌트 설계 단계는 8개의 업무로 구성되어 있다.

2.1 객체 순차 다이어그램 작성

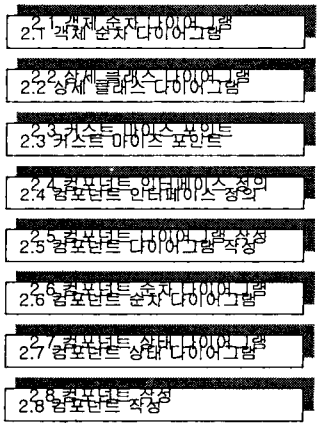
컴포넌트 내에 포함되는 객체들 간의 메시지 흐름을 식별하는 것이다.



(그림: 5) 컴포넌트 계층

2.3 커스트 마이즈 포인트 식별

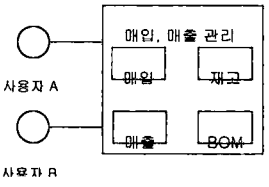
컴포넌트 사용자가 자신의 목적에 맞도록 커스트 마이즈 할 수 있는 부분들 추출하여 컴포넌트 인터페이스로 정의한다.



(그림: 6) 분석 설계 단계

2.5 컴포넌트 다이어그램 작성

컴포넌트 단위로 인터페이스를 정의하고, 인터페이스를 통한 컴포넌트내의 객체들 간의 제어 흐름을 정의하는 업무이다.

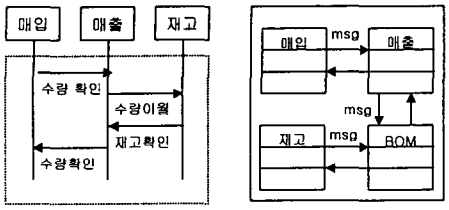


(그림: 7) 컴포넌트 표기법

2.7 컴포넌트 상태 다이어그램 작성

상태 다이어그램은 컴포넌트 내의 객체들에 대한

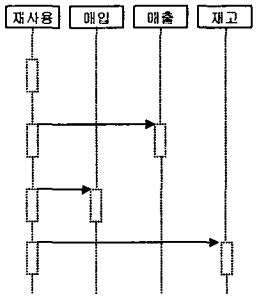
상태변화를 표현하기 위한 것으로 컴포넌트 내에 발생하는 이벤트를 표현한다.



(그림: 8) 컴포넌트 인터페이스 정의

2.8 컴포넌트 Contract 작성

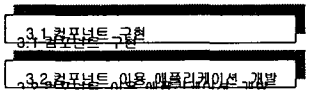
컴포넌트 인터페이스들은 컴포넌트들이 연결될 수 있는 수단이자 클라이언트가 호출할 수 있는 오퍼레이션의 집합이다.



(그림: 9) 카테고리 검색에 관한 객체 상호작용 다이어그램

3) 구현 단계

구현 단계는 크게 2개의 업무로 구성된다.



(그림: 10) 구현 단계

3.1 컴포넌트 구현

컴포넌트를 인터페이스와 구현 부분으로 분리 해야 한다. 컴포넌트 사용자에게 컴포넌트에 대한 인터페이스 부분만을 제공함으로써 컴포넌트 구현 부분을 숨기기 위함이다.

4) 테스팅 단계

컴포넌트를 기반으로 하여 구축된 애플리케이션은 소프트웨어 라이프사이클에서의 테스팅 기법을 사용하여 테스팅될 수 있다.

4. 재사용 메트릭스

다양한 소프트웨어 메트릭스는 소프트웨어 시스템 개발에서 재사용의 이점을 측정하기 위한 시도로 개발되었다. 시스템 S내에 재사용과 연관된 이점은 다음과 같은 비율로 표시할 수 있다.

$R_b(S) = [C_{noreuse} - C_{reuse}] / C_{noreuse}$ (식: 1)

$C_{noreuse}$는 재사용없이 S를 개발하는 비용이다. C_{reuse}는 재사용에서 S를 개발하는 비용이다. $R_b(S)$는 다음 치역(range)에서 차원의 값을 갖지 않는 것으로 표현할 수 있다. $0 \leq R_b(S) \leq 1$ (식: 2) Devanbu 와 그의 동료는 다음과 같이 제안했다. ① R_b 는 시스템의 설계에 영향을 받을 것이며, ② R_b가 설계로 인해 영향을 받기 때문에 설계 대안의 평가는 R_b를 구하는 것이 중요하다. ③ 재사용과 관련된 이점들은 각 개별적인 재사용 가능한 컴포넌트의 비용 이점과 밀접하게 직결된다. $R_b(S)$ 값이 크면 클수록 재사용은 더욱더 흥미롭다. *Reuse leverage*란 용어로 객체-지향 시스템에서 재사용의 일반적인 측정은 다음과 같다.

$R_{lev} = OBJ_{reused} / OBJ_{built}$ (식:3)

OBJ_{reused}는 한 시스템에서 재사용된 객체의 수이다. OBJ_{built}는 한 시스템에서 구축된 객체의 수이다. R_{lev}가 증가하면, R_b역시 증가한다.

구 분	구조적C/S시스템	EJB 컴포넌트 기반
처리 방식	제한적인 분산 처리 방식	완전한 분산 처리 방식
특 징	개발난이도 높음, 운영 난이도 높음, 집중 데이터처리 복잡	개발난이도가상대적으로 낮음 운영난이도가상대적으로 로 낮음, 집중데이터처리 간편
향후 시스템확장성	확장 가능	확장성 뛰어남
장애 복구	어려움	장애원인 식별용이, 빠른 복구지원
라이트 사이징	비교적 용이	규모와 성능의 최적화 용이
유지보수성	구조적 방식의 한계로 유지보수 어려움	모듈성 높고 유지보수성 뛰어남
트랜잭션	처리 가능	완벽한 트랜잭션 지원
정보 기술	기존 기술의존	최신 기술의 지속적 수용 가능

(표: 1) 컴포넌트 기반 C/S 시스템과 절차적 시스템의 비교

5. 결론

본 논문에서 UML 기반 객체지향 설계에서 재사용 가능한 컴포넌트 구축 방법에 대한 모델을 제안하였다. 소프트웨어 부품을 재사용하는 방법에는 화이트박스 재사용성과 블랙박스 재사용성이 있는데 이 두 가지 방법은 서로 다른 성질을 갖는다. 임의의 클래스가 화이트박스 재사용에는 부적합하나 블랙박스 재사용에는 적합할 수 있으며 그 반대의 경우도 있기 때문이다. 즉 재사용 가능한 부품인 컴포넌트들을 재사용 함으로써 생산성 향상, 신뢰성 향상, 생산 원가를 절감할 수 있다. 또한 소프트웨어 시스템을 개발하는데 제안된 모델을 이용하여 재사용이 높은 부품을 이용하고 재사용 가능한 컴포넌트들을 리파지토리(Repo-sitory)하여 개발해 봄으로써 이 모델의 문제점과 장단점을 분석하여 모델의 신뢰성을 높이는 작업을 계속해야 할 것이며 재사용 가능한 컴포넌트의 품질을 정량화하고 관련정보를 자동적으로 분석하고 평가하여 재사용을 지원하는 자동화 도구의 개발이 필요하다. 객체 지향 방법론이 추구하는 소프트웨어 부품의 재사용을 극대화하기 위한 분석, 설계, 구현 단계를 보다 체계적이고 일관성 있게 관리할 수 있는 버전 관리 및 자동화 도구를 사용하여 분석, 설계 모델의 가시화함과 동시에 코드 생성까지 이루어질 수 있는 연구가 필요하다.

[참고문헌]

[1] 'Objects, Components and Framework with UML', D' Souza Wills 저.

[2] 'Object-Oriented Design in Java', Mitchell Waste and Robert Lafore 저.

[3] 'Object-Oriented Programming in Java', Mitchell Waste and Robert Lafore 저.

[4] 'Object-Oriented Method', Ian Graham 저.

[5] 'Applying UML and Patterns', Craig Larman 저.

[6] 'Software Engineering A practitioner' Approach' Roger S. Pressman.

[7] '통합 객체지향 방법론 실무' 김상하 저.

[8] 'UML 객체지향 분석 설계' 조완수 저.

[9] '실용 소프트웨어 공학론' 이주헌 저.

[10] '객체지향 설계 모듈의 결합 방법' 정보처리학회 제3권 제4호 (96.7)

[11] 객체지향 프로그램에서 클래스 재사용성 측정 모델링' 정보처리학회 제6권 제3호(99.3)