The Journal of Korean
Institute of CALS/EC
Vol. 7, No. 3, Dec. 2002

J2EE 패턴기반의 컴포넌트 개발 프로세스

최일우*, 류성열**, 이남용**

J2EE Pattern Based Component Development Process

Il Woo Choi, Sung Yul Rhew, Nam Yong Lee

Abstract

The various software engineering techniques have been appeared in order to cope with the software crisis since 1980's. Currently, the research against the techniques likes the Design pattern, Component which improve the software's re-use are spread out. Also S/W Development Process are interested intensively which attempts the quality and a increasing productivity of software development with the basic policy. The design pattern is the solution against the problem which occurs repeat in a specific area. Many design pattern are developed and researched, but the method which accommodates the developed design pattern efficiently in the phase of analysis and design software development process is not good enough, so it is the actual applying technique is difficult.

In this paper we suggest and the "UML components+" which is a efficient component development process from customizing EJB based the J2EE using the "UML Components" which is a component development methodology. Applying J2EE pattern efficiently with UML components+, there is a possibility of efficiency in the component development based pattern.

key word : *pattern, Component, Reuse, Process, Process Customizing.*

* 숭실대학교 컴퓨터학과 대학원생
** 숭실대학교 컴퓨터학부 교수

1. 서 론

현재 비용 효과적인 소프트웨어 산업화를 위한 다양한 소프트웨어 공학적 시도들은 웹 환경하의 컴포넌트 기반 개발 프로세스(CBD: Component Based Development Process)로 점차 집약되고 있다. CBD는 객체 지향 개발 방법론을 이용, 재사용 단위 컴포넌트를 개발, 조립, 재사용하는데 발생하는 여러 문제점을 해결하기 위한 확장된 소프트웨어 개발 방법론이다.

CBD는 일반적으로 특정 컴포넌트 모델(Component Model)을 기반으로 컴포넌트들을 개발하는 컴포넌트 개발 프로세스(Component Development Process)와 기 개발된 컴포넌트를 획득, 조립 및 통합하여 어플리케이션을 개발하는 어플리케이션 개발 프로세스(Application Development Process)로 구성되어진다[1].

현재 제시되어진 컴포넌트 모델은 Sun의 EJB(Enterprise Java Bean), Microsoft의 COM(Component Object Model), OMG의 CCM(CORBA Component Model)등이 있으며, 현재의 컴포넌트 개발 프로세스는 이러한 표준 컴포넌트 모델을 이용 시스템을 구축할 수 있도록 지원한다.

컴포넌트 개발을 지원하는 RUP, Catalysis, UML Components등의 많은 수의 개발 프로세스들이 개발되어 있으나 현실적으로 이러한 프로세스를 통한 개발에 많은 비용이 소모되어지고 있다. 이 문제를 해결하기 위한 다양한 방법들이 계속 제시되어지고 있고, 프로세스들은 점차 수정, 보완되어지고 있는 실정이다.

현재 소프트웨어 혹은 컴포넌트 설계 단계의 비용을 감소시키는 방법으로 디자인 패턴(Design Pattern)을 도입하려는 노력이 증가하고 있으며 이미 GoF패턴[2] 및 J2EE (Java 2 platform, Enterprise Edition)패턴[3] 등 여러 다양한 디자인 패턴들이 소개, 적용되어지고 있다.

디자인 패턴이란 "자주 발생되는 설계상의 문제를 해결하기 위한 반복적인 솔루션"[2][3]이라 정의되어진다. 패턴을 연구하고 사용하는 가장 큰 목적은 디자인의 재사용이며, 디자인 패턴은 각각의 클래스 라이브러리 등의 부품을 이용하여 각각의 부품이 어떻게 조립되어 있고, 어떻게 연관되어 하나의 기능 혹은 더 큰 재사용 부품을 구성하는지를 표현하며, 하나의 재사용 단위를 구성, 어떻게 기능을 확장해 나아가고 변경해 나아가는지에 관심을 둔다[3]. 그러나 소프트웨어 개발 프로세스의 분석, 설계 단계에 디자인 패턴들을 효율적으로 식별, 적용 하는가는 개발자의 능력에 한정되어지고 이에 대한 세부 지침은 매우 미약한 상태이다.

본 논문에서는 컴포넌트 개발 방법론인 UML Components[4]의 명세 및 공급단계에 J2EE 디자인 패턴을 효율적으로 식별, 적용하는 기법을 포함하여 확장한 UML Components+를 제시한다. 본 논문을 통하여 제시하는 UML Components+의 J2EE 패턴 적용 기법을 통하여 패턴기반의 효율적인 EJB 컴포넌트 개발을 수행 할 수 있다.

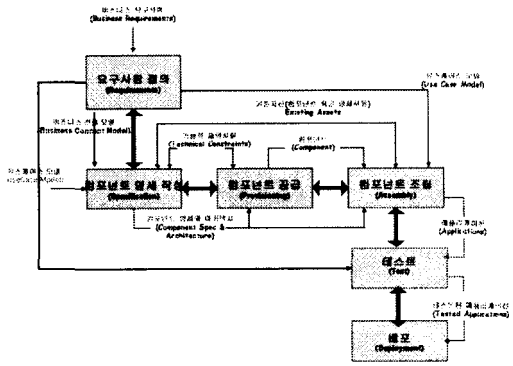
2. 관련 연구

2.1 UML Components

UML Components는 John Cheesman과 John Daniels이 제시한 컴포넌트 기반 시스템의 아키텍처를 설계하고, 또 그것을 명세화 하는데 있어서의 실제적인 문제를 정면에서 다루고 있는 방법론[4]이다. Catalysis, RUP, Advisor를 기반으로 전체적인 프로세스를 다루고 있으며, 현재 대표적 컴포넌트 개발 방법중의 하나로 자리 잡고 있다.

UML컴포넌트의 전체 워크플로우는 그림 1과 같다. UML 컴포넌트 명세단계의 산출물은 크게 인터페이스 명세와 컴포넌트 명세로 분류되며 그 세부 내용은 다음 <표 1>과 같다.

<그림 2>는 UML Components에서 제시하는 시스템 아키텍처를 나타낸 것이다. 이런 접근은 프로그램을 구성하는 각각의 소프트웨어 단위가 가지고 있는 목적을 설명하기에 적합하다. 최종적인 시스템의 전체 윤곽을 드러내고 우리가 필요한 시스템을 조립하기 위하여 다양한 기술요소들을 어떻게 적용할 것인지를 설명한다.

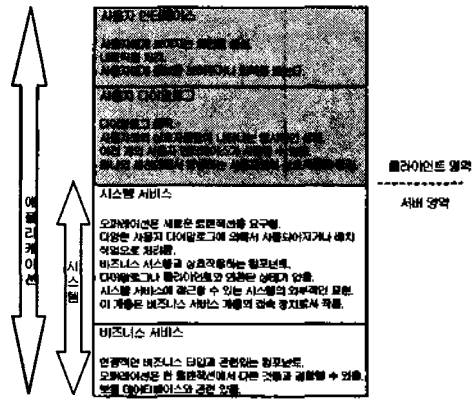


<그림 1> UML Components의 워크플로우

UML Components는 최근의 EJB, COM, CCM과 같은 컴포넌트 기술의 활용을 원하거나 UML을 사용하여 객체 지향 모델링을 좀 더 풍부하게 적용하고자 하는 많은 사람들에게 쉬운 접근법을 제시한다. 반면 UML

<표 1> UML Components의 명세(Specification)단계의 산출물과 세부 내용

인터페이스 명세	인터페이스 명세 다이어그램	인터페이스 오퍼레이션 시그네처 표현. 사용하는 비즈니스 객체 정보 표현. Data Type 표현. (인터페이스 정보 모델)
	오퍼레이션 명세	오퍼레이션 정의. 오퍼레이션 입,출력 매개변수 정의. 오브젝트 상태 변화. 오퍼레이션 제약(사전, 사후 조건등)사항. (불변식, 스냅샷)
	업무규칙 정의	인터페이스가 지켜야 할 업무 제약사항 표현. 인터페이스가 수행할 업무규칙 정의. (인터페이스 정보 모델에 일부 포함 가능)
컴포넌트 명세	컴포넌트 명세 다이어그램	컴포넌트가 사용하는 인터페이스 식별. 컴포넌트가 사용하는 인터페이스간의 제약사항 표현. (제공되는 인터페이스와 사용되는 인터페이스)
	컴포넌트 상호작용 다이어그램	컴포넌트 메소드와 컴포넌트가 사용하는 컴포넌트의 메소드와의 상호작용 및 제약사항 표현. 컴포넌트 메소드의 수행 범위 결정. (콜라보레이션 다이어그램으로 표현)
	인터페이스간의 제약조건	컴포넌트와 관련된 모든 인터페이스간의 제약조건 표현. 각 인터페이스가 사용하는 정보를 명확하게 구분.



<그림 2> UML Components의 시스템 아키텍처

Components는 프로세스 중 컴포넌트 명세 단계만을 중심으로 기술하고 있으며 그 외의 부분은 "RUP와의 적절한 매핑을 통하여 완벽한 프로세스로의 구축이 가능하다"[4]라 정의, 미완의 상태로 남아 있다. 또한 컴포넌트 설계의 효율성, 재사용성 및 확장성을 높일 수 있는 디자인 패턴의 수용 부분은 고려하고 있지 않다.

2.2 J2EE와 디자인 패턴

J2EE는 EJB 컴포넌트 스펙을 기본으로 한 멀티 티어(Multi-tier)로 구성된 Sun사의 엔터프라이즈 어플리케이션 개발을 위한 스펙[3]이다. EJB는 서버 상에서 자바 플랫폼으로 운영되는 컴포넌트 기술로 컴포넌트 개발에 필요한 스펙들로 구성되어 있다.

디자인 패턴(Design Pattern)이란 "주어진 환경에서의 반복되는 문제들에 대한 재사용이 가능한 해결책이다"[2]라 정의 한다. 패턴을 연구하고 사용하는 가장 큰 목적은 재사용이다. 패턴은 여러 시행착오를 거쳐서 만들어진 개발자들의 경험들을 토대로 하여 주어진 상황에 따라 최적의 효과를 낼 수 있는 설계를 제공한다.

<표 2> J2EE 각 계층간의 패턴 요약

Presentation	Intercepting Filter	사용자 요청의 전처리와 후처리를 쉽게 처리.
	Front Controller	사용자 요청제어를 한곳에서 관리를 하기 위한 컨트롤러를 제공.
	View Helper	Helper컴포넌트의 Presentation formatting에 관계되지 않는 로직을 분리, 즉 화면과 Business Delegate에 접근하는 Helper를 분리.
	Composite View	독립적인 View를 모아서 새로운 View를 생성.
	Service To Worker	Front Controller와 View Helper 패턴의 결합형태.
	Dispatcher View	화면에 보여주는 행위들을 제외한 Front Controller와 View Helper 패턴의 결합형태.
Business	Business Delegate	Presentation계층과 Business계층과의 결합도를 완화하는 기능.
	Value Object Assembler	네트워크간의 상호작용을 감소시키는 계층간의 데이터 교환기능.
	Value List Handler	프레젠테이션 계층으로 다양한 요소의 리스트를 제공하는 기능.
	Session Facade	비즈니스 객체의 복잡도를 감추는 중앙 집중적인 워크플로우를 다루는 기능.
	Composite Entity	단일 Entity Bean으로 의존 객체들을 그룹지어서 실행 성능을 높이는 기능.
	Service Locator	비즈니스 서비스를 찾고, 생성하는 것과 같은 복잡한 서비스를 묶는 기능.
Integration	Data Access Object	추상 데이터 소스. 데이터의 접근의 투명성을 보장하는 기능.
	Service Activator	EJB 컴포넌트의 비동기적인 처리 기능.

J2EE 패턴은 J2EE 플랫폼을 기반으로 소프트웨어 어플리케이션을 개발하는 디자이너에게 일반적으로 발생하는 문제들의 해결을 제공하는 디자인 패턴이다. J2EE 패턴은 J2EE가 제시하는 시스템 아키텍처에 따라 각 계층별로 프리젠테이션 계층, 비즈니스 계층 그리고 인테그레이션 계층에 해당하는 패턴들을 제공하며 그 개략적인 내용은 표2[3]와 같다.

3. UML Components+의 구성

컴포넌트 개발 방법론인 UML Components를 적용하여 EJB 컴포넌트 모델로 컴포넌트 기반의 분산 시스템을 구축하는 것은 개발에 큰 효율성을 제공한다. 그러나 UML Components는 세부 스텝이 명세 워크플로우에만 치중 되어 있을 뿐 공급, 조립, 테스트, 배포 등의 워크플로우는 세부적으로 명시되어 있지 않다. 또한 설계의 효율성, 재사용성 및 확장성을 높일 수 있는 디자인 패턴의 수용 부분은 고려 되어 있지 않다.

본 논문에서는 이러한 한계점을 극복하기 위해 UML Components의 공급(Provisioning) 단계에 J2EE디자인 패턴을 수용한 컴포넌트 설계 기법을 추가하여 UML Components+로 확장한다.

본 논문에서 제시하는 UML Components+는 UML Components에서 반약하게 다루고 있는 공급단계를 J2EE패턴을 수용한 EJB 컴포넌트 모델로 구현하는 방법 및 세부 스텝을 정의한다.

본 논문에서 제시하는 UML Components+는 크게 다음과 같이 구별된다.

첫째, UML Components에 J2EE패턴을 적용한 EJB 컴포넌트 구현 모델을 디자인하기 위한 새로운 UML Components+ 시스템 아키텍처와 이에 따른 패턴 프레임워크를 제시한다.

둘째, UML Components+의 공급 단계에 J2EE 패턴을 수용하여 EJB컴포넌트를 구현하는 세부적 방법 및 절차를 제시한다.

본 논문에서 제시하는 UML Components+와 기존 방법론을 통한 EJB 컴포넌트 모델을 디자인하는 기법을 사례연구를 통하여 비교 그 효율성을 검증한다.

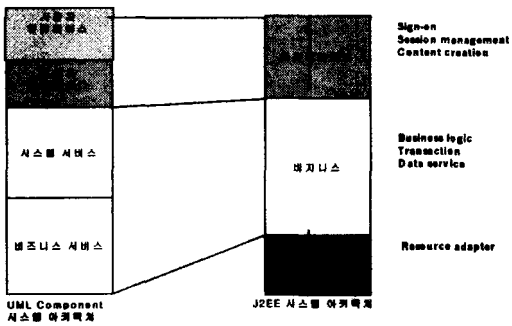
3.1 UML Components+의 시스템 아키텍처와 패턴 프레임 워크

시스템 아키텍처(System Architecture)란 완전한 소프트웨어를 구성하고 있는 소프트웨어 부품들의 구조로서 각각의 부품이 가지는 책임과 그들의 상호 연계성 및 적절한 적용 기술까지를 포함하는 개념이다[5].

UML Components를 따라 J2EE 패턴을 적용한 EJB 컴포넌트 모델을 구현하기 위해서는 시스템 아키텍처를 EJB를 포함한 J2EE 시스템 아키텍처로의 효율적인 매핑이 필요하다.

본 논문의 UML Components+ 에서는 UML Components의 시스템 아키텍처를 J2EE 시스템 아키텍처로 매핑, 그림3, 표3과 같이 적용하여 수용한다. 그림3에서와 같이 J2EE의 비즈니스 계층을 UML Components의 시스템서비스 계층과 비즈니스 서비스 계층으로 분할한다면 UML Components의 시스템 아키텍처와 J2EE의 시스템 아키텍처가 적절히 매핑된다.

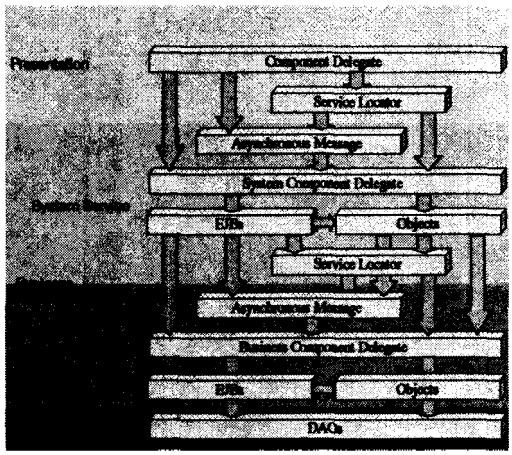
제시하는 UML Components+의 시스템 아키텍처는 UML Components의 프레젠테이션, 비즈니스, 통합의 3계층과 J2EE 시스템 아키텍처의 시스템 서비스와 비즈니스 서비스의 2계층을 매핑 하였다.



<그림 3> UML Components와 J2EE의 시스템 아키텍처 매핑

3.1.1 패턴 프레임워크 정의

J2EE 패턴에서는 패턴을 적용한 특성 있는 객체들이 존재한다. 여기서 특성 있는 객체란 패턴을 적용하기 위해 특별한 속성과 의무를 부여받은 객체들이다. 본 논문에서는 J2EE 패턴을 적용하여 시스템에 포함하는 모든 객체들을 각각의 특성별로 분류하고, 이것을 "시스템 구성 요소" 즉, 객체들의 집합이란 의미로 정의한다. 각각의 패턴들은 다음 그림4와 같은 구성요소의 연관 관계를 가지며 UML Component+의 패턴 프레임워크를 구성한다.



<그림 4> 패턴 프레임 워크

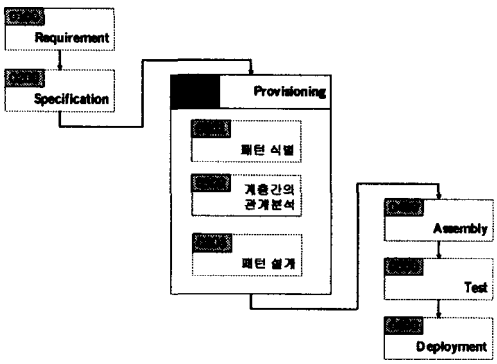
본 논문에서 제시하는 J2EE패턴의 프레임워크는 그림4와 같다. 각종 EJB관련 패턴들을 수용하기 위하여 J2EE의 패턴 중 Business Delegate, Value Object, Session Facade, Composite Entity, Service Locator, Data Access Object, Service Activator등 대표적 패턴만을 시스템 구성요소에 포함 하였으며 추가적으로 각 계층별로 다양한 패턴의 수용이 가능하다. 각 시스템 구성 요소와 패턴간의 정의는 표4와 같다.

<표 3> UML Components와 J2EE의 시스템 아키텍처 계층 설명

User Interface User Dialog	시스템에 접근하는 클라이언트에게 서비스하기 위한 모든 프리젠테이션 로직을 담고 있다. 이 계층은 클라이언트의 요청을 받아들이고, 인증이나 세션관리, 그리고 비즈니스 서비스로 접근하고, 사용자의 요청에 대한 응답을 되돌려 준다.
System Service Business Service	클라이언트 어플리케이션의 요청에 대하여 비즈니스 서비스를 제공한다. 이 계층은 비즈니스 데이터와 로직을 포함하며, 모든 비즈니스 처리는 이 계층에서 처리 한다.

3.2 UML Component+의 구성

디자인 패턴은 특정 아키텍처에 독립적으로 적용할 수 있기 때문에 확장된 UML Components+의 명세단계의 산출물을 입력으로 받아 컴포넌트 공급 단계에서 적용된다. 패턴을 고려하여 정의된 UML Components+의 공급 단계는 다음 그림 5와 같다.



<그림 5> UML Components+의 워크플로우

UML Components+의 명세단계에서는 컴포넌트의 구현에 무관한 명세가 작성된다. 공급 단계를 통하여 컴포넌트의 명세를 만족하는 컴포넌트의 설계 및 구현이 이루어지며 이 단계에서 패턴의 식별 및 적용이 이루어진다.

3.2.1 패턴의 식별

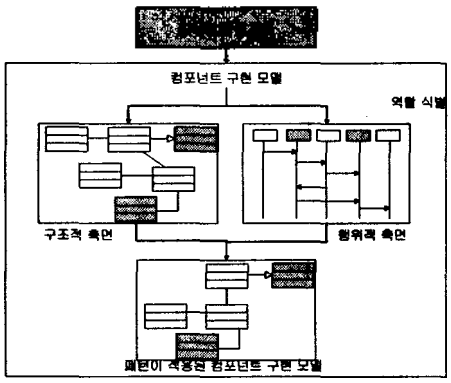
시스템을 모델링할 때는 정적, 동적, 그리고 기능적 측면 등 다양한 관점에서 모델링 수행한다[6]. UML Components+를 이용하여 개발된 명세를 만족하는 설계를 수행할 때, 필요한 패턴들을 식별하기 위해서는 구조적 측면과 행위적 측면을 고려하여 패턴을 식별 한다. UML Components+의 디자인 식별 기법은 그림6과 같은 형태로 이루어진다.

패턴 식별을 위해 클래스 다이어그램의 해당 클래스들의 역할들을 식별하고 서로 관련을 맺고 있는 클래스들과의 관계를 파악해야 한다. 이처럼 추출된 역할들을 바탕으로 J2EE 패턴 레포지토리에 있는 패턴 내의 각각의 클래스들의 역할들에 대해 추출된 역할과 매칭이 되는 패턴들을 식별한다.

<표 4> 비즈니스, 시스템 계층의 J2EE패턴과 UML Component+의 구성요소와의 매핑

패턴 이름	구성 요소	역할 및 설명
Business Delegate	Component Delegate	Presentation 계층에서 시스템 컴포넌트를 사용하기 위한 대표 객체로 구성되며 Presentation 계층에서 사용하는 시스템 컴포넌트의 오퍼레이션을 정의. 결합도 감소, 관리능력 개선, 수행능력 개선, 오류수정 용이, 비즈니스 계층의 접근 용이.
Value Object	Data Type	Entity Bean과 원격 인터페이스의 단순화, 적은 수의 원격 호출로 데이터 전송, 네트워크 부하 감소, 중복코드 감소, 명명성 증가, 트랜잭션 용이.
Session Facade	System Component Delegate, Business Component Delegate	시스템 컴포넌트 각각을 대표하는 EJB컴포넌트로 구성. 수행하는 시스템 서비스를 통합하여 이것을 통해서 시스템 서비스를 수행. 비즈니스 계층의 관리기능 감소, 결합도 감소, 관리능력 개선, 수행능력 개선, 중앙 집중적인 보안관리, 중앙집중적인 트랜잭션 제어.
Composite Entity	Business Component Delegate, 비즈니스 객체를 통합한 EJB	비즈니스 컴포넌트 각각을 대표하는 EJB컴포넌트로 구성. 수행하는 비즈니스 서비스를 통합하여 이것을 통해서 비즈니스 서비스를 수행.내부 엔터티 관계를 제거, Entity Bean을 감소시켜서 관리능력 개선, 네트워크 부하감소, 데이터베이스 스키마 의존성 감소.
Service Locator	Service Locator	사용하려는 컴포넌트 탐색, 생성등 복잡한 서비스를 묶어놓은 객체로 구성. 새로운 비즈니스 컴포넌트 추가 용이, 네트워크 수행 능력 개선, 클라이언트 수행능력 개선, 복잡한 코드를 추상화.
Data Access Objects	DAOs	데이터 접근의 투명성, 쉬운 데이터베이스 전환, 비즈니스 객체내의 복잡한 코드를 감소.
Service Activator	Asynchronous Message	비동기적인 메시지를 처리하기 위한 메시지 수신자 역할의 EJB컴포넌트로 구성. EJB 컴포넌트의 비동기적 처리 기능.

UML Components+ 공급단계의 세부 스텝에서는 이러한 구조적, 행위적 측면의 패턴들이 쉽게 식별 가능하도록 제시한다.



<그림 6> 역할 매칭을 통한 디자인 패턴 식별 방법

3.2.1.1 구조적 관점의 패턴 식별

구조적 관점의 패턴 식별 단계에서 설계 단계의 클래스 다이어그램에 본 논문에서 제시하는 패턴 프레임워크 적용한다. 우선 해당 계층의 패턴별로 클래스 다이어그램들을 세분화시킨다.

클래스 다이어그램에 패턴들을 적용할 때 추가적으로 표현되어야 할 정보들이 존재한다. UML에서는 의미를 확장 할 수 있는 메커니즘을 가지고 있는데, 그 대표적인 것이 스테레오 타입(Stereotype)이다. 스테레오 타입은 기존의 UML요소를 확장하여 새로운 메타클래스의 인스턴스를 만들기 위한 수단으로 고안되었다. 즉, 표준의 UML 요소로 표현할 수 없는 시스템 고유의 특성이나 도메인을 잘 표현할 수 있는 요소이다[7].

디자인 패턴은 UML의 스테레오타입 표기법을 적용, 디자인 패턴과 관련된 클래스들 간의 관계를 바인딩 시켜준다. 표5에서는 패턴 프레임워크를 이용한 컴포넌트 구현 모델의 구성 요소들을 표현하기 위한 스테레오 타입들을 정의한다.

3.2.1.2 행위적 관점의 패턴 식별

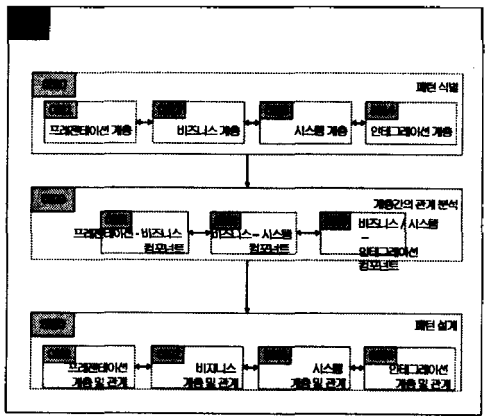
행위적 관점에서의 디자인 패턴을 적용하기 위해서는 해당 디자인 패턴 내의 클래스들 간의 메시지 흐름을 검토해야 한다. 이러한 메시지 흐름을 바탕으로 설계 단계의 순차도(Sequence Diagram)에 추가적으로 요구되는 객체들을 포함시킬 뿐만 아니라 객체들 간의 메시지 흐름을 정제시킨다. 디자인 패턴이 적용되기 이전의 순차도에는 메시지 흐름은 표현되어 있으나 메시지 흐름의 순서가 비효율적으로 표현되어 있을 수 있을 뿐만 아니라 필요한 메시지가 반영되지 않을 수도 있다. 따라서 디자인 패턴 내의 객체들 간의 메시지 흐름을 바탕으로 순차도의 메시지 흐름을 정제시키면 효율적인 메시지 흐름으로 완성된다.

3.3 공급 단계의 세부 절차

UML Components+는 명세 단계의 산출물인 컴포넌트 명세와 컴포넌트 인터페이스 명세 산출물을 제공한다. 가장 먼저 할 일은 컴포넌트 인터페이스 명세 산출물과 UseCase를 이용하여 시스템 컴포넌트에 접근하는 프리젠테이션 계층을 구현하는 것이다. 이것은 아키텍처의 프리젠테이션 계층의 구현에 해당한다.

그림7은 UML Components+의 공급 단계의

세부 워크플로우를 보여준다. 공급단계인 0300 단계의 각 워크플로우는 구현에 있어서 독립적이기 때문에 병행적으로 수행 될 수 있고, 각 계층간에도 Delegate를 통하여 상호작용을 하기 때문에 병행적으로 수행 될 수 있다.



<그림 7> UML Components+의 공급 단계(0300)의 세부 워크플로우

0310, 0320, 0330 웍크플로우는 순차적으로 수행해야 하고 각 워크플로우의 세부 워크플로우들은 각 워크플로우 상에서 병행적으로 수행될 수 있다.

3.4 계층별 패턴 식별 방법

본 절에서는 공급단계(0300)의 세부 워크플로우별로 설명한다.

3.4.1 프리젠테이션 계층(0311) 패턴 식별

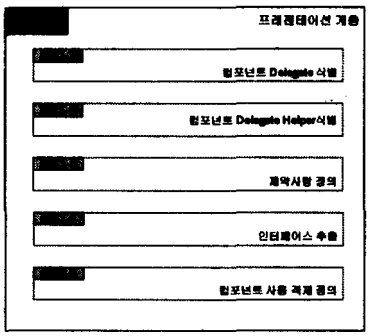
사용자는 프리젠테이션 계층을 통해서 시스템 서비스나 비즈니스 서비스와 인터페이스를 한다.

이 계층의 설계에 따라서 시스템의 성능에 많은 영향을 준다. Scriptlet 코드를 줄이고 재사용이 가능한 구현 설계를 목표로 한다. 그림8, 표6은 프리젠테이션 계층 패턴 식별 절차 및 구현 설계 방법을 나타낸다.

<표 5> 패턴구성요소의 표현을 위한 스테레오 타입의 정의

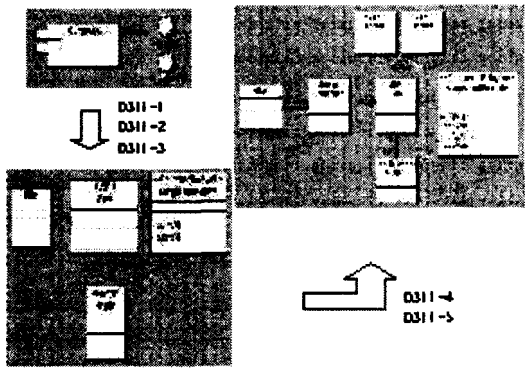
스테레오 타입	정의
<<JSP>>	Java Server Page 구현 객체
<<Servlet>>	Servlet 구현 객체
<<View>>	클라이언트에게 정보를 보여주기 위한 객체
<<Helper>>	Component Delegate에 접근하기 위한 객체
<<exception>>	예외처리 구현 객체
<<datatype>>	특정 데이타를 구조화한 구현 객체
<<DAO>>	Data Access Object. 데이터 베이스 연결 및 질의를 구현하는 객체
<<EntityEJB>>	EJB의 Entity Bean 컴포넌트
<<SessionEJB>>	EJB의 Session Bean 컴포넌트
<<StatefulSessionEJB>>	EJB의 Session Bean의 Stateful Session Bean 컴포넌트
<<StatelessSessionEJB>>	EJB의 Session Bean의 Stateless Session Bean 컴포넌트
<<MessageEJB>>	EJB의 Session Bean의 Message Driven Bean 컴포넌트
<<ComponentDelegate>>	Presentation계층에서 비즈니스 계층을 대표하는 객체
<<ServiceLocator>>	EJB 서비스를 찾고, 생성하기위한 기능을 구현하는 객체
<<BusinessObject>>	EJB컴포넌트의 기능을 보조하는 객체
<<comp spec>>	UML컴포넌트 방법에서 생성된 컴포넌트 명세 산출물
<<interface type>>	UML컴포넌트 방법에서 생성된 인터페이스 타입
<<DesignComponent>>	컴포넌트 명세에의한 하나의 컴포넌트에 해당하는 EJB 구현 모델

표5의 단계에 따라 인터페이스 명세에 위의 액티비티를 적용하면 각 인터페이스 별로 오퍼레이션에 접근하는 Helper와 화면의 View 가 만들어진다. 사용자 필터링이나 로그인 처리 같은 Filter클래스도 식별이 된다. 그 다음 각 요소의 통합을 통해서 공통적인 요소를 추출한 후에 통합하면 패턴이 적용된 클래스 다이어그램이 나온다.



<그림 8> UML Components+의 프레젠테이션 계층의 세부 단계

그림9는 UML Components+의 컴포넌트 명세단계에서 나온 인터페이스 명세를 가지고 프리젠테이션 구현 설계 웍플로우를 적용한 클래스 다이어그램을 나타낸다.



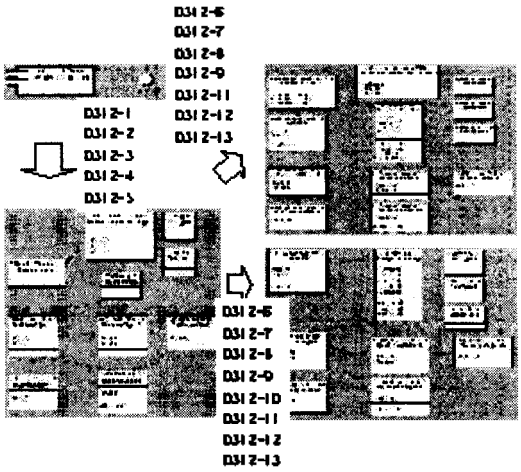
<그림 9> 프레젠테이션 계층(0311) 식별 방법 적용 클래스 다이어그램

3.4.2 비즈니스 계층(0312) 패턴 식별

비즈니스 계층의 패턴을 식별하는 방법은 표7과 같다. 적용과정의 클래스 다이어그램은 그림 10에서 보여준다.

<표 6> 프레젠테이션 계층 패턴 식별 세부 단계 및 액티비티

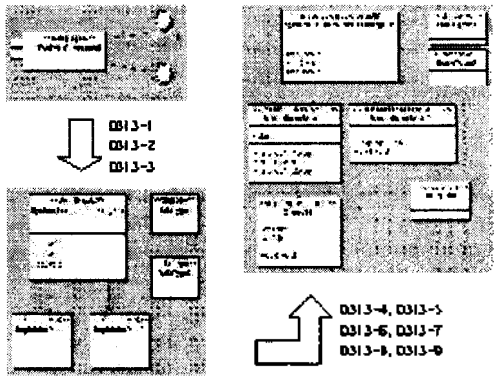
0311-1	컴포넌트 Delegate 식별과 개략 오퍼레이션의 정의.	비즈니스 컴포넌트 사용자가 직접 접근하는 오퍼레이션들로만 구성된 컴포넌트 Delegate를 식별, 각 오퍼레이션들의 이름과 간단한 파라미터 정도로 구성된 개략 오퍼레이션을 정의.
0311-2	컴포넌트 Delegate 에 접근하기 위한 Helper를 구성.	View에서 직접 시스템 서비스에 접근하지 않도록 Helper를 구성하여 시스템 서비스에 접근하도록 한다.
0311-3	제약 사항 및 추가사항 식별.	사용자 접근 제한이나 로그인 처리 같은 제약사항 및 추가사항을 식별.
0311-4	0311-1, 2, 3을 반복하여 각 인터페이스별로 공통적인 요소 추출 후 통합.	공통적인 요청들을 한곳에서 처리하기 위한 컨트롤러를 구성하고 시스템 서비스에 접근하기 위한 Helper를 재구성.
0311-5	각 요소를 패턴 구성에 적용하여 컴포넌트를 사용하기 위한 객체(Servlet / JSP)를 구성.	식별된 구성요소들을 패턴에 적용하여 구성하고, 컴포넌트를 사용하기 위한 UI 객체들을 구성 .



<그림 10> 비즈니스 계층(0312) 식별 방법 적용 클래스 다이어그램

3.4.3 시스템 계층(0313) 패턴 식별

시스템 계층의 패턴을 식별하는 방법은 표8과 같다. 적용과정의 클래스 다이어그램은 그림 11에서 보여준다.



<그림 11> 시스템 계층(0313) 식별 방법 적용 클래스 다이어그램

<표 7> 비지니스 계층 패턴 식별 세부 단계 및 액티비티

		상세설명
0312-1	Business Component Delegate 식별 및 오퍼레이션 구성	비즈니스 컴포넌트가 가지고 있는 모든 오퍼레이션으로 구성. 인터페이스 명세 다이어그램 참조. Session Bean으로 구성.
0312-2	비즈니스 컴포넌트의 비즈니스 로직을 수행하는 Business Session Bean 식별	비즈니스 컴포넌트가 수행하는 Business Session Bean의 이름만 식별. 인터페이스 명세 다이어그램 참조.
0312-3	비즈니스 컴포넌트의 구성 객체를 통합하는 Composite Entity Bean 식별	비즈니스 컴포넌트의 구성 객체를 통합하는 Composite Entity Bean의 이름만 식별. 인터페이스 명세 다이어그램 참조 .
0312-4	비즈니스 컴포넌트의 구성 객체 식별 및 구성	인터페이스 명세 다이어그램을 구성하는 모든 객체를 구현모델로 구체화. 인터페이스 명세 다이어그램 참조. Composite Entity Bean과의 관계설정.
0312-5	비즈니스 컴포넌트에서 사용하는 Data Type 식별 및 구성	비즈니스 컴포넌트 명세에서 식별된 Data Type을 구현모델로 구체화. 인터페이스 명세 다이어그램 참조 .
0312-6	Business Component Delegate의 추가적인 오퍼레이션 구성	비즈니스 컴포넌트의 오퍼레이션이 사용하는 추가적인 오퍼레이션을 구성. 오퍼레이션 명세, 업무규칙 정의 참조.
0312-7	Business Component Delegate의 추가적인 속성 식별 및 상태 정의	비즈니스 컴포넌트의 오퍼레이션이 사용하는 속성을 구성하고 그 상태를 정의. 오퍼레이션 명세, 업무규칙 정의 참조 . Session Bean이 Stateless Session Bean인지 Stateful Session Bean인지를 결정
0312-8	Business Session Bean의 오퍼레이션 구성	Business Component Delegate의 오퍼레이션이 수행하는 핵심 비즈니스 로직을 오퍼레이션으로 구성. 오퍼레이션 명세, 업무규칙 정의, 컴포넌트 상호작용 다이어그램 참조
0312-9	Business Session Bean의 속성 구성 및 상태 정의	Business Session Bean이 사용하는 속성을 구성하고 그 상태를 정의. 오퍼레이션 명세, 업무규칙 정의 참조 . Session Bean이 Stateless Session Bean인지 Stateful Session Bean인지를 결정 .
0312-10	Business Component Delegate의 재구성	Business Session Bean의 구성이 Composite Entity Bean을 호출하기위한 경로에 지나지 않거나 수행하는 비즈니스 로직이 미약한 경우 Business Session Bean을 제거하고 Business Component Delegate가 Composite Entity Bean으로 재구성. 오퍼레이션 명세, 업무규칙 정의 참조
0312-11	Composite Entity Bean의 오퍼레이션 및 속성 구성	비즈니스 컴포넌트의 구성객체를 대표하는 모든 오퍼레이션으로 구성. 비즈니스 컴포넌트의 구성객체를 구성하는 모든 속성으로 구성. 오퍼레이션 명세, 업무규칙 정의 참조
0312-12	추가적인 객체 식별 및 구성과 객체간의 관계 설정	Business Session Bean의 기능 또는 Composite Entity Bean에서 독립적으로 분리되어질 수 있는 객체를 식별 및 구성하고 관계를 설정. 오퍼레이션 명세, 업무규칙 정의 참조
0312-13	Exception 객체 식별 및 구성	생할 예외를 정의하여 Exception 객체를 식별하고 구성. 오퍼레이션 명세, 업무규칙 정의, 인터페이스간의 제약조건 참조

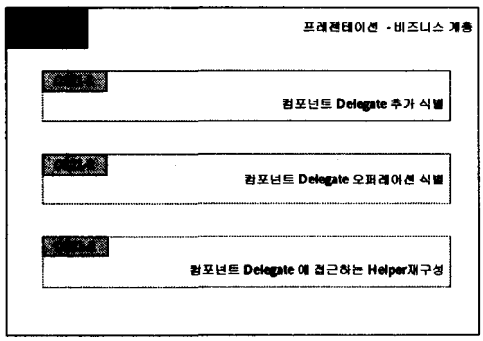
<표 8> 시스템 계층 패턴 식별 세부 단계 및 액티비티

0313-1	System Component Delegate 식별 및 오퍼레이션 구성	시스템 컴포넌트가 가지고 있는 모든 오퍼레이션으로 구성. 인터페이스 명세 다이어그램 참조. Session Bean으로 구성.
0313-2	컴포넌트 인터페이스별 Session Bean 식별 및 오퍼레이션 구성	시스템 컴포넌트가 가지고 는 인터페이스별로 각각온 인터페이스의 오퍼레이션으로 구성. System Component Delegate에서의 association 관계를 가진다. 인터페이스 명세 다이어그램 참조.인터페이스가 하나일 경우에는 System Component Delegate만 생성하고 인터페이스별 Session Bean은 구성할 필요가 없다.
0313-3	컴포넌트에서 사용하는 Data Type 식별 및 구성	시스템 컴포넌트 명세에서 식별된 Data Type을 구현모델로 구체화. 인터페이스 명세 다이어그램 참조.
0313-4	System Component Delegate의 추가적인 오퍼레이션 구성	시스템 컴포넌트의 오퍼레이션이 사용하는 추가적인 오퍼레이션을 구성. 오퍼레이션 명세, 업무규칙 정의 참조.
0313-5	System Component Delegate의 속성 구성 및 상태 정의	시스템 컴포넌트의 오퍼레이션이 사용하는 속성을 구성하고 그 상태를 정의. 오퍼레이션 명세, 업무규칙 정의 참조. Session Bean이 Stateless Session Bean인지 Stateful Session Bean인지를 결정.
0313-6	컴포넌트 인터페이스별 Session Bean의 추가적인 오퍼레이션 구성	컴포넌트 인터페이스별 Session Bean의 오퍼레이션이 사용하는 추가적인 오퍼레이션을 구성한다. 오퍼레이션 명세, 업무규칙 정의, 컴포넌트 상호작용 다이어그램 참조.
0313-7	컴포넌트 인터페이스별 Session Bean의 속성 구성 및 상태 정의	컴포넌트 인터페이스별 Session Bean의 오퍼레이션이 사용하는 속성을 구성하고 그 상태를 정의한다. 오퍼레이션 명세, 업무규칙 정의 참조 . 컴포넌트 인터페이스별 Session Bean이 Stateless Session Bean인지 Stateful Session Bean인지를 결정.
0313-8	추가적인 객체 식별 및 구성과 객체간의 관계 설정	컴포넌트 인터페이스별 Session Bean의 operation 수행에 있어서 필요한 객체 식별 및 구성하고, 객체간의 관계를 설정한다. 오퍼레이션 명세, 업무규칙 정의 참조. 컴포넌트 인터페이스별 Session Bean의 기능에서 독립적으로 분리되어질 수 있는 객체를 식별
0313-9	Exception 객체 식별 및 구성	발생할 예외를 정의하여 Exception 객체를 식별하고 구성. 오퍼레이션 명세, 업무규칙 정의, 인터페이스간의 제약조건 참조

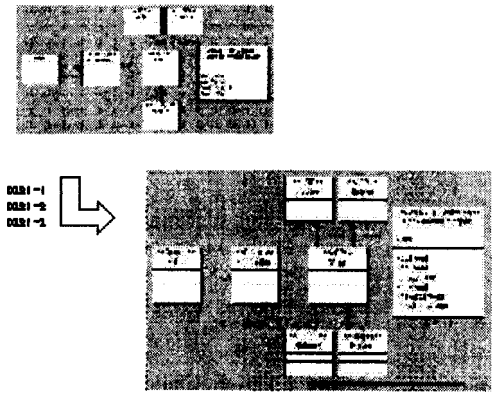
3.5 계층간의 패턴 식별 방법

아키텍처 계층간의 상호작용을 위한 패턴 식별 방법을 제공한다. 여기서 계층들 사용에 관한 엄격한 규칙을 만들지 않는다. 예를 들면 프리젠테이션 계층에서 직접 비스니스 서비스 계층을 접근할 경우도 있으며, 어떤 시스템에서는 비즈니스 서비스 계층의 컴포넌트를 관리하기 위한 기능이 불필요하기 때문에 시스템 서비스 계층에 비어 있기도 한다.

3.5.1 프리젠테이션 계층과 비즈니스/시스템 계층(0321)간의 패턴 식별



<그림 12> 프리젠테이션 계층과 비지니스/시스템 계층간의 패턴 식별

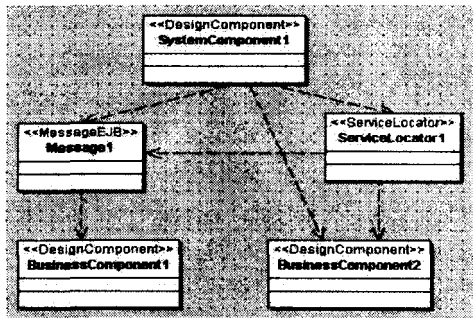


<그림 13> 프리젠테이션 계층과
비지니스/시스템 계층간의 구현

프레젠테이션 계층과 비지니스/시스템 계층간의 패턴 식별 방법은 그림 12와과 표9과 같다. 그림13은 프리젠테이션 계층과 시스템 컴포넌트간의 패턴 식별 워크플로우를 적용한 클래스 다이어그램을 나타낸다. 오퍼레이션이나 데이터 타입의 추가적인 식별에 대하여 접근하는 Helper를 새롭게 구성하거나 기능이 다른 Helper를 분리한다.

3.5.2 비지니스 계층과 시스템 계층(0322)간의 패턴 식별

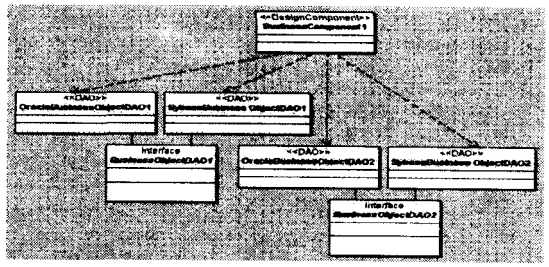
시스템 계층과 비즈니스 계층간의 패턴 식별하는 방법은 표10과 같다. 적용한 클래스 다이어그램은 그림14 에서 보여준다.



<그림 14> 시스템 컴포넌트와
비즈니스 계층간의 패턴 식별

3.5.3 비즈니스/시스템 계층과 Integration 계층(0323)간의 패턴 식별

비즈니스 컴포넌트와 Integration 계층간의 구현 모델을 디자인하는 방법은 표11과 같다. 적용한 클래스 다이어그램은 그림 15에서 보여준다.



<그림 15> 비즈니스 컴포넌트와
Integration 계층간의 구현 설계.

<표 9> 시스템 계층과 비지니스 계층 간의 세부 워크플로우

0321-1	컴포넌트 Delegate의 추가 오퍼레이션 식별.	두 계층간의 통합하는 과정에서 새롭게 추가되는 동작을 식별.
0321-2	컴포넌트 Delegate의 오퍼레이션 세부 스텝 정의.	각 오퍼레이션 별로 동작하는 세부 스텝들을 정의하여 서비스 요청시 응답할 수 있도록 구성.
0321-3	컴포넌트 Delegate 에 접근하는 Helper를 재구성.	객체(Servlet or JSP)가 사용하는 Helper를 재구성한다. 추가되는 오퍼레이션에 따라서 Helper를 분리하거나 추가로 구성.

<표 10> 시스템 계층과 비지니스 계층 간의 세부 워크플로우

0322-1	System에 하나의 Service Locator 구성.	Service Locator는 시스템에서 사용하고 있는 분산 서비스(EJB의 Home Object, Datasource등)와 레퍼런스를 찾아주는 역할을 하는 객체. 시스템에서 사용하는 모든 분산서비스의 환경을 설정하여 쉽게 분산 서비스를 레퍼런스 할 수 있게 구성.
0322-2	비동기적인 메시지에 의한 Component요청시 Message Driven Bean을 비즈니스 서비스 계층에 추가 설계.	비동기적인 메시지에의한 Component 요청이 필요할 때 비동기적인 메시지를 처리하는 Message Driven Bean을 통해서 Component를 요청. 오퍼레이션 명세, 업무규칙 정의, 컴포넌트 상호작용 다이어그램, 컴포넌트 명세 다이어그램 참조

<표 11> 비즈니스/시스템 계층과 인테그레이션 계층간의 세부 워크플로우.

0323-1	비즈니스 컴포넌트 구성객체별로 DAO객체의 인터페이스를 구성.	오퍼레이션 명세와 인터페이스 명세 다이어그램을 참조. 각 구성객체의 속성과 오퍼레이션을 기초로 지속성 있는 데이터를 다루기 위한 오퍼레이션을 식별하여 인터페이스를 구성.
0323-2	상호 의존적인 DAO의 통합	DAO는 OR-Mapping의 중요한 요소로 식별된다. 실존하는 Data Model과 객체 모델과 적절한 조화를 이루며 DAO를 설계. 식별된 DAO간의 의존성이 있으면 DAO의 통합으로 하나의 DAO를 이룬다. 즉, DAO는 각 각 모두가 독립적이어야만 한다.
0323-3	사용하는 데이터 소스별로 DAO객체를 구성.	구성된 인터페이스를 기초로 DAO 객체를 구성. 사용하는 데이터소스를 식별하여 데이터 소스에 해당하는 데이터베이스 연결 및 질의를 위한 속성 및 메소드를 구성. 같은 DAO객체를 구성하는 데이터소스가 다수 일 경우 각 경우마다 DAO객체를 구성.

4. 사례 연구 및 평가

본 논문에서 제시한 방법을 적용하여 사례연구를 수행 한다. 제시한 방법의 효율성을 증명하기 위하여 논문에서 제시한 방법을 적용하지 않은 구현과 적용한 구현을 비교 평가 한다. 평가는 평가요소 별로 각 항목을 측정하며 비교결과는 다음절에서 언급한다.

4.1 사례연구

본 장에서 사례연구에 적용한 도메인은 고객 섭외 관리 시스템이다.

4.1.1 적용 산출물

UML Components+의 명세단계에서 나온 고객 섭외 관리의 산출물은 다음 표 12와 같다. 고객 섭외 관리 시스템은 금융시스템의

한 부분으로써 지점장은 섭외직원을 관리하고, 섭외 직원은 고객을 섭외하여 고객관리를 하고, 섭외 일지를 관리하는 활동 들을 시스템으로 표현 한 것이다.

<표 12> 고객 섭외 관리의 산출물

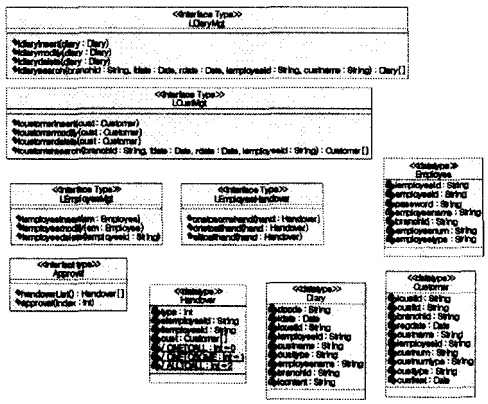
컴포넌트 명세	컴포넌트 인터페이스 명세	컴포넌트 형태
CustomerLiaisonMgt	LCustMgt	System Component
	LEmployeeMgt	
	LDiaryMgt	
	LEmployeeHandover	
	Approval	
LDiary	LDiary	Business Component
LCustomer	LCustomer	
LEmployee	LEmployee	

CustomerLiaisonMgt는 고객 섭외 관리 시스템의 시스템 컴포넌트로 식별되었다. 시스템 컴포넌트 중에서 LEmployeeHandover와 Approval은 데이터베이스와 관련이 없기 때문에 비즈니스 컴포넌트가 없다. 이 시스템 컴포넌트는 비즈니스 컴포넌트를 관리하고, 프리젠테이션 계층과 연결을 해주는 역할을 한다. 인터페이스 타입 모델을 통한 인터페이스 책임 다이어그램을 구성하고, 이를 기반으로 비즈니스 컴포넌트인 LDiary, LCustomer, LEmployee 가 식별되었다.

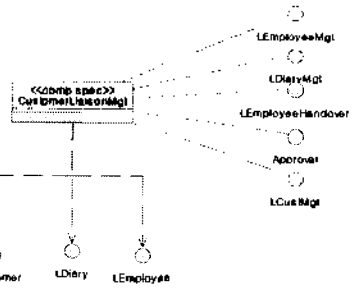
그림 16은 컴포넌트 명세 다이어그램을 나타내고, 프리젠테이션 계층 구현을 위한 산출물인 컴포넌트 인터페이스 다이어그램은 그림 17과 같다.

4.1.2 계층별 패턴 설계 방법

본 논문에서 제시하는 절차에 따라 사례 연구의 과정을 기술한다.



<그림 16> CustomerLiaisonMgt의 인터페이스 타입과 데이터 타입

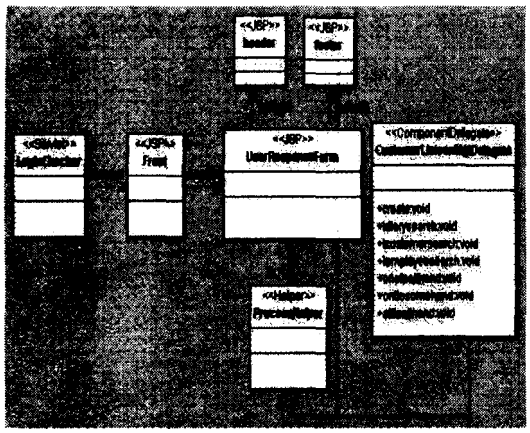


<그림 17> CustomerLiaisonMgt 컴포넌트 명세 다이어그램

4.1.2.1 프리젠테이션 계층 패턴 설계

본 논문의 3.4.1에서 제시하고 있는 프리젠테이션 계층 패턴 식별 워플로우에 따라서 아래의 절차를 적용한다.

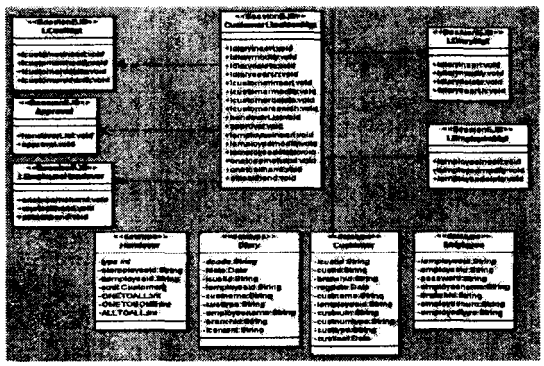
구성된 프리젠테이션 계층의 클래스 다이어그램은 그림18과 같다. 아래의 다이어그램은 개념적 클래스 다이어그램이며 상세 클래스 다이어그램은 표시하지 않는다.



<그림 18> 고객섭외관리 프리젠테이션 계층 구현 설계 클래스 다이어그램

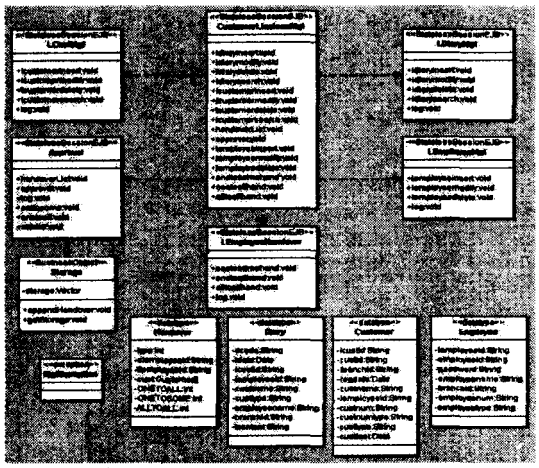
4.1.2.2 시스템 계층 구현 설계 방법

본 논문의 3.4.3에서 제시하고 있는 시스템 계층 패턴 식별 절차 0313-1,2,3을 따르면 그림19 에서 보여주는 결과를 얻을 수 있다. 여기서 System Component Delegate는 CustomerLiaisonMgt 이다. 그리고 각 인터페이스 별로 Session Bean을 구성하였고, 산출물을 참조로 datatype을 구성하였다.



<그림 19> 시스템 컴포넌트 구현 설계 (1)(2)(3)

본 논문의 3.4.3에서 제시하고 있는 시스템 계층 패턴 식별 절차 0313-4,5,6,7,8,9를 따르면 그림20에서 보여주는 결과를 얻을 수 있다.

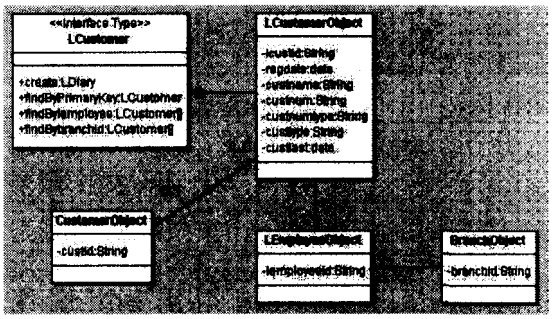


<그림 20> 시스템 컴포넌트 구현 설계 (4)(5)(6)(7)(8)(9)

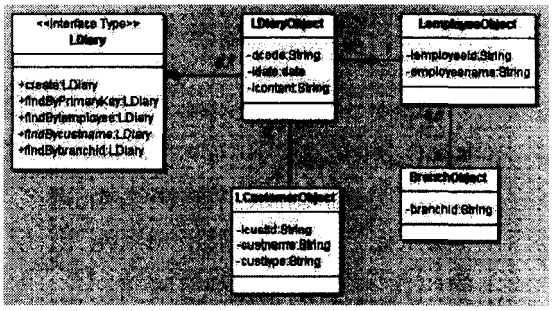
여기서 특별한 속성이 구성되지 않았으므로 모든 Session Bean은 Stateless Session Bean 으로 구성되었고 승인처리를 위한 인수객체를 저장하는 Storage 객체가 추가 구성되었고 모든 고객섭외 관리의 예외를 처리하기 위한 MgtExection이 추가되었다.

4.1.2.3 비즈니스 컴포넌트 구현 설계 방법

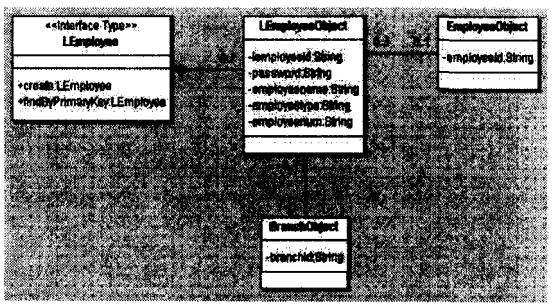
비즈니스 컴포넌트 구현을 위한 핵심적인 산출물인 각 인터페이스 명세 다이어그램은 그림21,22,23에서 보여주고 있다. 본 논문에서는 기타 산출물은 생략한다.



<그림 21> LCustomer 인터페이스 명세 다이어그램



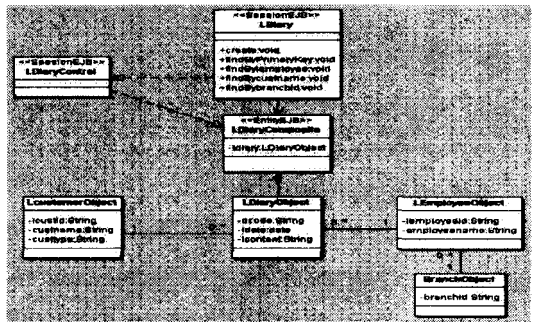
<그림 22> LDiary 인터페이스 명세 다이어그램



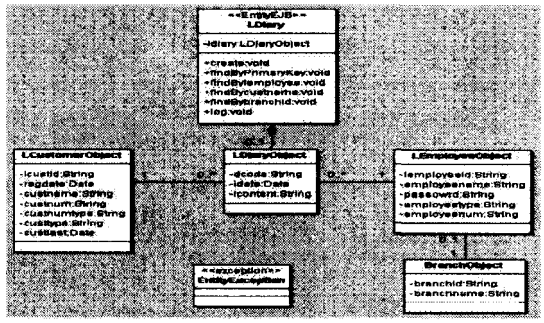
<그림 23> LEmployee 인터페이스 명세 다이어그램

본 장에서는 LDiary 비즈니스 컴포넌트를 실예로 기술한다.

본 논문의 3.4.2에서 제시하고 있는 비즈니스 계층 패턴 식별 절차 0312-1,2,3,4,5를 따르면 그림24에서 보여주는 결과를 얻을 수 있다. 여기서 Business Component Delegate는 LDiary이고, Business Control은 LDiaryControl이다. 그리고 Composite Entity는 인터페이스 타입의 소속 객체들을 통합한 LDiaryComposite 이다.



<그림 24> 비즈니스 컴포넌트 구현 설계 (1)(2)(3)(4)(5)



<그림 25> 비즈니스 컴포넌트 구현 설계 (6)(7)(8)(9)(10)(11)(12)(13)

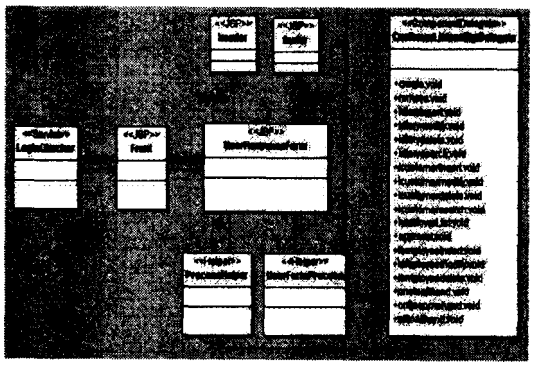
절차 0312-6,7,8,9,10,11,12,13을 따르면 그림25 에서 보여주는 결과를 얻을 수 있다. 여기서 LDiaryControl은 비즈니스 로직이 거의 없고 LDiaryComposite을 호출하는 경로에 지나지 않으므로 Business Component

Delegate는 LDiary와 LDiaryComposite의 통합 형태로 재구성되고 이름은 LDiary이다. 그리고 엔티티 간의 예외를 처리하는 EntityException을 추가 식별하였다.

4.1.3 계층간의 패턴 설계

4.1.3.1 프리젠테이션 계층과 비지니스/시스템 계층간의 패턴 설계

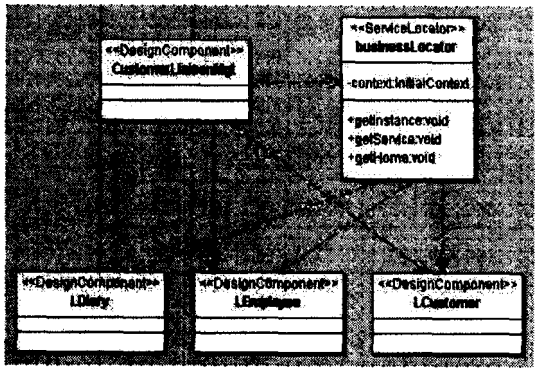
본 논문의 3.5.1에서 제시하고 있는 프리젠테이션 계층과 비지니스/시스템 계층간의 패턴 식별 절차를 적용하면 그림26에서 보여주는 클래스 다이어그램을 얻을 수 있다.



<그림 26> 고객센터관리 프리젠테이션 계층과 시스템 계층간의 절차 적용 클래스 다이어그램

4.1.3.2 시스템 계층과 비즈니스 계층간의 패턴 설계

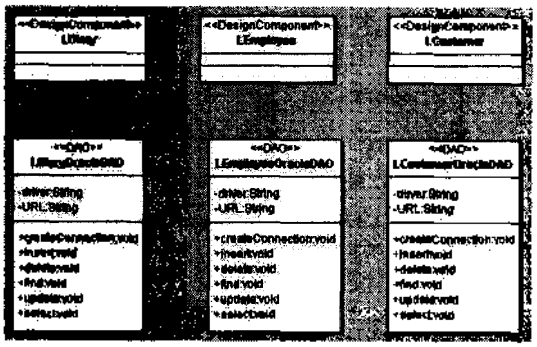
본 논문의 3.5.2에서 제시하고 있는 시스템 계층과 비즈니스 계층간의 구현 설계 방법을 따르면 그림27에서 보여주는 결과를 얻을 수 있다. 여기서 Service Locator 는 BusinessLocator 이다.



<그림 27> 시스템 계층과 비즈니스 계층간의 패턴 설계

4.1.3.3 비즈니스/시스템 계층과 Integration 계층간의 패턴 설계

본 논문의 3.5.3에서 제시하고 있는 비즈니스/시스템 계층과 Integration계층간의 패턴 식별 절차를 따르면 그림28에서 보여주는 결과를 얻을 수 있다. 여기서 데이터소스를 Oracle을 사용하였기에 한 종류의 DAOs 로 구성하였다.



<그림 28> 비즈니스 컴포넌트와 Integration 계층간의 구현 설계

4.2 평가

본 논문에서 제시하고 있는 UML Components+의 J2EE 패턴 수용을 위한 아키텍처와 절차에 대한 평가는 크게 두 가지 측면에서 접근한다.

첫째, J2EE 패턴의 효율적 수용 여부를 확인 한다. 본 논문에서 제시하는 세부 단계에 의해 디자인 된 결과가 J2EE 패턴을 수용한 디자인 모델이 나온다면 효율성 있는 패턴 수용 방법을 제시한 것이 된다.

둘째는 기존의 EJB 구현 모델을 디자인하는 방법과 비교, 분석하여 UML Components+의 방법이 효율적이라는 것을 제시한다.

그 외의 평가 항목은 표13에서 보는 것과 같다. 패턴 적용성, EJB설계 복잡성, 유지보수성 등의 평가 항목들은 UML Components+를 적용한 사례와 그렇지 않은 사례를 비교, 평가하도록 한다.

<표 13> UML Components+의 평가요소

Factor	내 용
활동 지원 Activity Support	개발 방법론의 각 단계의 세부지침이 지원되는가? 각 단계별 입,출력 산출물, 행위등이 명확하게 정의되어있는가?
패턴 지원 Pattern Support	개발 방법론의 분석, 설계 단계에서 디자인 패턴의 식별 및 적용이 용이한가?
EJB 설계 복잡성 EJB Modeling Complexity	EJB 컴포넌트 스펙을 지원하기 위한 구현 모델 구축의 단계가 얼마나 복잡한가?
유지 보수성 Maintainability	모듈을 추가하거나 내용을 변경하는 등의 작업을 쉽게 할 수 있는가?
컴포넌트 개발 General Component Model Support	다양한 컴포넌트 스펙을 지원하는 개발 가능한가?
효율성 efficiency	Cohesion, Coupling, LOC, Performance

● 패턴 적용성, EJB 설계 복잡성

기존의 EJB모델링 방법에는 클래스와 빈의 1:1 매핑 기법과 클래스를 상속하여 EJB를 형성하는 클래스 상속 기법이 있다[8]. 이 방법을 사용하기 위해서는 컴포넌트 명세 후 반드시 구현에 독립적인 클래스 다이어그램을 요구하게 된다. 하지만 UML Components 명세 단계에서는 구현 독립적인 클래스 다이어그램이 존재하지 않는다. 다만 인터페이스 명세 다이어그램에서 인터페이스와 동작하는 클래스를 표시하고 있다. 이것 역시 컴포넌트의 구현 독립적인 클래스 다이어그램으로 볼 수도 있겠지만 그것은 비즈니스 컴포넌트에 국한된다. 시스템 컴포넌트는 비즈니스 컴포넌트를 사용하는 시스템의 특징적인 요구사항을 수렴하는 컴포넌트이기에 인터페이스 명세 다이어그램은 비즈니스 컴포넌트에 존재하는 클래스들로 구성되어 있다. 따라서 기존의 방법을 UML Component 명세를 이용하여 적용을 하려면 구현 독립적인 클래스 모델링 수행 후 요구사항을 참고하여 컴포넌트 인터페이스별로 컴포넌트 스펙에 종속적인 구현 레벨의 클래스 다이어그램으로 재구성하는 과정이 필요하다.

또한 기존의 방법을 이용하여 EJB 구현 모델을 디자인 하였다면 패턴을 고려하지 않은 디자인이기 때문에 이것을 효율적인 디자인으로 바꾸기 위하여 J2EE 패턴을 적용하여야 한다. 이러한 작업은 분석, 설계 시 많은 노력을 필요로 하게 된다.

그러나 UML Components+에서 제시하는 공급 단계의 세부절차를 따라 모델링을 수행하게 되면 개념적 모델링 단계에서 EJB구현으로 재구성과정이 필요치 않을 뿐만 아니라

다양한 J2EE 패턴을 효율적으로 수용한 모델을 구축 할 수 있다.

● 활동 지원, 컴포넌트 개발

UML Components+에서는 각 단계별 세부 산출물을 정의하였을 뿐 아니라 각 세부 스텝에 따른 패턴이 적용된 디자인 모델을 제시, 패턴의 장점을 최대로 수용한다.

반면 UML Components+는 UML Components에서 지원하는 다양한 컴포넌트 스펙을 지원하는 것이 아니라 EJB컴포넌트 스펙으로만 그 범위가 국한되어지는 단점이 있다.

●유지 보수성, 효율성

UML Components+를 통하여 만들어진 소스 자체의 효율성을 검증하기 위한 방법으로 개발 최종산출물인 소스의 LOC, 응답 성능 등을 측정, 결과를 도출해 낼 수 있다. 표 14는 LOC 측정과 응답성능 측정 파일에 포함되는 계산코드를 보여준다.

<표 14> 응답 성능측정 코드

```
// 파일명 : da_list.jsp
<%!
static long sumTime = 0;
// 시간의 합을 나타내는 전역변수
static int call = 0;
// 호출회수를 나타내는 전역변수 %>
<%
long startTime = 0;        // 호출 시작시간
long endTime = 0;          // 호출 종료시간
startTime = System.currentTimeMillis();
                           // 시스템 현재시간
endTime = System.currentTimeMillis();
call++;                    // 호출회수 증가
sumTime += endTime - startTime; %>
```

코드의 복잡성은 해당 모듈이나 파일이 얼마만큼의 분기가 일어나느냐에 따라 결정된다. 효율적인 디자인에 의해 좌우되는 부분이 상당하므로 패턴의 장점을 최대한 수용 한다. 그러나 패턴의 이용이 시스템의 성능에 미치는 영향은 상황에 따라 매우 상이하므로 직접 측정하여 제시하였다.

아래의 표 15는 각 평가 항목을 수행하여 나온 결과이다.

5. 결론 및 향후연구

EJB 구현 모델을 디자인 하는 방법은 대부분 개발자의 경험에 의해서 행해져왔다. 기존의 존재하는 알려져 있는 디자인 방법은 클래스와 빈과의 1:1 매핑을 하는 방법과 클래스를 상속하여 EJB로 구성하는 클래스 상속 모델이 있다[8]. 이러한 방법은 컴포넌트별 구현 독립적인 클래스 모델링이 이루어진 후에 가능한 방법이다. 이것은 UML Components 명세를 적용하기에는 적합하지 않다. 그리고 클래스 모델링이 이루어 졌다고 하더라도 EJB 특성만을 고려하고 시스템 성능을 고려한 방법이 아니기 때문에 추가적인 J2EE 패턴 적용이 필요하게 된다.

UML Component는 명세화에 중점을 둔 방법으로 컴포넌트 명세단계만을 중점으로 다룬다.

본 논문에서 제시하는 UML Components+는 컴포넌트의 명세를 입력으로 컴포넌트 공급 단계를 J2EE 패턴을 적용한 EJB컴포넌트로 구현하는 방법을 제공하였다. 이 절차와 방법은 본래 UML Component의 산출물을 충분히 반영하고 아키텍처의 특성을 고려하여 시스템을 디자인 할 수 있게 Component+로 구성되었다.

UML Component의 시스템 아키텍처를

<표 15> UML Component와 UML Component+와의 비교 평가

평가항목	[illegible] Component	UML Components+
[illegible]	명세단계만을 중점적으로 기술. 세부단계 미비	명세와 공급단계 모두를 세부 절차까지 지원.
[illegible]	패턴을 분석, 설계 단계로 수용하기 위한 가이드 부재. 개발 프로세스와 패턴 수용부분은 별개의 작업.	자세한 세부 액티비티를 제공 하므로서 분석, 설계 단계를 통한 자연스런 J2EE 패턴의 수용.
[illegible]	구현 독립적인 컴포넌트 명세만 만족되면 되므로 모델링에 제약이 없다. 자유로운 만큼 복잡도는 증가한다.	정의된 액티비티를 따라 특정 스펙에 종속적인 분석, 설계를 수행하여야 하므로 제약사항은 증가되나 모델링 복잡도는 감소된다.
[illegible]	복잡도가 증가하므로 유지보수가 상대적으로 어려움.	정해진 룰에 의해 컴포넌트가 모델링되므로 유지보수가 상대적으로 용이함.
[illegible]	개발 지침이 특정 컴포넌트 모델에 제약을 두지 않으므로 다양한 스펙의 컴포넌트 개발이 지원 가능.	개발 지침이 EJB모델로 컴포넌트 스펙이 한정되어 있어 다양한 컴포넌트 개발을 지원하지 못함 .
[illegible]	동일한 도메인을 적용 코딩시 33 파일 2344 LOC. 응집도, 결합도 측면에서 불리.	동일한 도메인을 적용 코딩시 36 파일 2151 LOC. 응집도, 결합도 측면에서 유리.
[illegible]	동일한 도메인 개발결과의 응답속도 246 MSec. 개발자의 숙련도에 따라 성능 혹은 확장성등이 감조됨	동일한 도메인 개발결과의 응답속도 272 MSec. 적용 패턴의 특성에 따라 성능 혹은 확장성등이 감조됨

J2EE 시스템 아키텍처에 맞게 변환하였고, 효율적인 시스템의 구현을 위해 J2EE 패턴을 적용하여 아키텍처의 각 계층에 구성요소들을 포함시켜 구성요소간의 상호작용을 표현하였다. 이렇게 구성된 아키텍처는 UML Component+의 명세 산출물을 이용하여 EJB 구현 모델을 디자인하는데 초석이 된다. 제시되는 세부 액티비티에 따라 J2EE 패턴이 올바르게 적용되었기 때문에 시스템 성능을 최적화 시켜주는 역할을 하게 된다.

향후 연구과제로는 다양한 사례연구를 통하여 다양한 패턴을 수용한 스펙에 독립적인 컴포넌트 구현에 대한 연구가 이루어져야 할 것이다.

참고 문헌

[1] Peter Herzum, and Oliver Sims, *Business Component Factory A Comprehensive Overview of Component-Based Development for the Enterprise*, Wiley, 2000

[2] Erich Gamma, *Design Patterns,Addison Wesley*,1995

[3] Deepak Alur and John Crupi and Dand Malks, *Core J2EE Patterns*, Sun Microsystems,2001

[4] John Cheesman and John Daniels, *UML Components*, Addison Wesley,2000

[5] Clemens Szyperski, *Component Software, Addison Wesley, 1998.*

[6] Craig Larman, *Applying UML and Patterns, Prentice Hall PTR*, 2001.

[7] CT Arrington, Enterprise Java with UML, Jhon Wiley&Sons,2001.

[8] *Enterprise Java Beans*, Sun Microsystems, 2001

저자소개

최일우

1995년 숭실대학교 전자계산학과 학사(공학사). 1997년 숭실대학교 컴퓨터학과 석사(공학석사). 1997년~2000년 숭실대학교 컴퓨터학과 박사 과정 수료. 관심분야는 개발 방법론, 유지보수, 재공학, 역공학, 재사용, 컴포넌트 기반 소프트웨어 공학.

류성열

1997년 2월 아주대학교 컴퓨터학부(공학박사). 1997년 3월~1998년 3월 George Mason University 교환교수. 1981년 3월~현재 숭실대학교 정보과학대학 컴퓨터학부 교수. 1998년 3월~2001년 2월 숭실대학교 정보과학대학원 원장. 1998년 3월~현재 숭실대학교 전자계산원 원장. 관심분야는 소프트웨어 유지보수, 소프트웨어 재사용, 소프트웨어 재공학/역공학

이남용

숭실대학교 컴퓨터학부 교수
바산네트워크(주) 대표이사
숭실대학교 컴퓨터학부(공학사). 고려대학교 경영학(MIS)석사. 미시시피주립대학교 경영학박사학위(MIS).관심분야는 컴포넌트기술(OOAD, CBD), 소프트웨어 테스팅기술(Functionality, Performance, Reliability Test)