# 소프트웨어 컴포넌트 이해를 위한 데이터 북 구성

김 선 희[†] · 최 은 만[††]


## 요 약

소프트웨어 위기를 극복하기 위하여 컴포넌트 기술이 제안되어 개발에 적용되고 있다. 소프트웨어 컴포넌트는 하드웨어의 집적회로와 같은 블랙박스로 취급되지만 사용자가 효과적으로 이해할 수 있도록 지원되지 않는다면 이용할 수가 없다. 이 논문은 하드웨어 컴포넌트의 이해를 돕기 위한 데이터 북 형식이 소프트웨어 컴포넌트를 표현하는데 잘 적용될 수 있다는 것을 보였다. 데이터 북의 내용으로 UML과 API 모형을 도입하여 컴포넌트를 이해하는 접근 방법을 채택하였으며 소프트웨어 컴포넌트의 중요한 부분인 아키텍처와 인터페이스 부분을 추가하였다. 실증적인 실험을 위하여 제안한 데이터 북을 EJB의 배치 디스크립터에 확장 포함하여 컴포넌트 데이터북을 웹 브라우저에서 볼 수 있도록 하였고 그 효용성을 실험하였다.


# Construction of Data Book for Understanding Software Components

SunHee Kim[†] · Eun Man Choi[††]


## ABSTRACT

Component technology was proposed and applied to software development to overcome software crisis. Software component is a black box like an integrated circuit in hardware but it can not be utilized without good support specially for helping users understand efficiently. This paper shows that data book format for understanding hardware component can be well applied to representing software component. We selected an approach to understand component by matching the contents of data book with UML and API model technique. Besides, we added the architecture part and the interface which are the most important property of software component to the data book for software components. In order to verify effectiveness of components data book we extended batch descriptor in EJB and performed an experiment providing data book to programmers with components.


**키워드 : 컴포넌트(components), 프로그램 이해(program understanding), 재사용(reuse), 컴포넌트 데이터북(components databook)**

## 1. 서 론

컴포넌트는 프로그램 개발에 있어서 하나의 부품과 같은 독립적인 역할을 하는 단위이다[7]. 컴포넌트를 기반으로 시스템을 개발한다는 것은 독립적으로 제작된 컴포넌트를 조립하여 하나의 시스템으로 합성해 나가는 것을 의미한다. 컴포넌트 기반 소프트웨어 개발 기술은 소프트웨어 위기의 해결 방안으로 제시되어 주목받고 있다. 객체지향 패러다임을 기반으로 하는 컴포넌트의 프러그 앤 플래이 기술로 대규모의 응용 소프트웨어를 쉽게 조립하여 완성할 수 있는 기술이 컴포넌트 기술이다. 그러나 제 3자에 의하여 만들어진 컴포넌트를 이해하여 조립한다는 것이 그리 쉬운 일은 아니다. 왜냐하면 컴포넌트 제작자와 컴포넌트를 조립하여 시스템을 완성하고자 하는 사용자가 서로 다르기 때문이다. 컴포넌트 사용자는 컴포넌트의 내부 구현에 대하여 자세히 알 필요는 없다. 그러나 컴포넌트를 사용하기 위해서는 컴포넌트의 이해는 반드시 필요하다.

지금까지 IWPC(International Workshop on Program Comprehension)을 중심으로 프로그램 이해를 위한 여러 가지 모델이 제시되고 효과적인 이해 프로세스의 지원 방향이 연구되었지만 거의 모든 연구의 배경은 객체지향이나 컴포넌트 개념을 떠난 것이다[15]. 컴포넌트 이해를 위한 연구는 많이 부족하다. 이미 다른 사람에 의해 만들어진 컴포넌트를 잘 사용하기 위해서 그리고 유지보수를 위해서도 컴포넌트의 이해는 매우 중요하다. 소프트웨어의 컴포넌트 개념은 하드웨어의 부품단위로 생성하고 이를 조립하여 판매하는 방법에서 제시된 것이다. 작게는 반도체 칩을 생각해 볼 때 이를

† 정 회 원 : 모트롤라 코리아 연구원
†† 정 회 원 : 동국대학교 컴퓨터공학과 교수
논문접수 : 2001년 10월 5일, 심사완료 : 2002년 3월 8일

사용하기 위하여 데이터 북(data sheet)을 참조한다. 소프트웨어에서도 컴포넌트를 사용하기 위한 데이터 북과 같은 것이 필요하다.

하드웨어 컴포넌트의 이해를 위한 데이터 북과 매칭 시키는 작업을 통해 소프트웨어 컴포넌트 데이터 북에는 어떤 요소가 들어가야 하는지를 살펴보았다. 소프트웨어 컴포넌트의 이해를 위해 UML(Unified Modeling Language) 기반 모델과 API 기반 모델을 사용하였다. UML 기반 모델을 사용하여 프레임워크 클래스나 인터페이스 등을 설명한다. 그리고 API 기반 모델로는 인터페이스를 프로빙하는 방식을 사용하여 설명하고 있다.

## 2. 관련 연구

### 2.1 소프트웨어 컴포넌트 이해 모델

프로그램 이해의 측면에서 다음 네 가지 접근 방향이 컴포넌트 이해에 적용될 수 있다.

- 컴포넌트 구현 모델[12] - 블랙 박스인 컴포넌트에 대하여 자세한 구현 사항을 요약하여 데이터베이스로 제공함으로써 컴포넌트의 이해를 돕는 방법이다.
- API 모델[4] - 이 방법의 기본 아이디어는 블랙박스 컴포넌트는 API에 의하여만 접근 가능하므로 컴포넌트의 인터페이스를 이용하여 이해해 보자는 것이다. 컴포넌트 인터페이스인 파라미터에 어떤 값을 주었을 때 나오는 리턴 값이나 출력 결과를 분석하여 컴포넌트의 기능을 파악하는 방법이다.
- 인터페이스 기반 모델[1] - 컴포넌트를 이해하고 이를 기반으로 시스템을 형성하는 기초는 커넥터이다. 커넥터는 컴포넌트 사이에 정보나 제어가 전달되는 메커니즘이라 할 수 있다[10]. 따라서 이러한 커넥터 중심으로 컴포넌트를 표현하고 이해하자는 모델이다.
- 패턴 기반 모델[5, 8] - 패턴을 이용한 프레임워크를 정의한 것을 컴포넌트로 사용할 때 이를 확장하거나 상속하여 사용하는 것으로 보고 프레임워크 클래스나 인터페이스를 UML과 같은 모델로 제공하여 컴포넌트를 이해하려는 방법이 있다. 다시 말해 Java의 클래스나 인터페이스로 정의한 프레임워크를 컴포넌트로 본다면 컴포넌트의 내부 구성을 UML의 클래스 다이어그램이나 순서 다이어그램으로 보여주어 이해를 도모하고 프레임워크 컴포넌트 사이의 인터페이스를 UML의 패키지 다이어그램과 같은 형태로 제공한다.

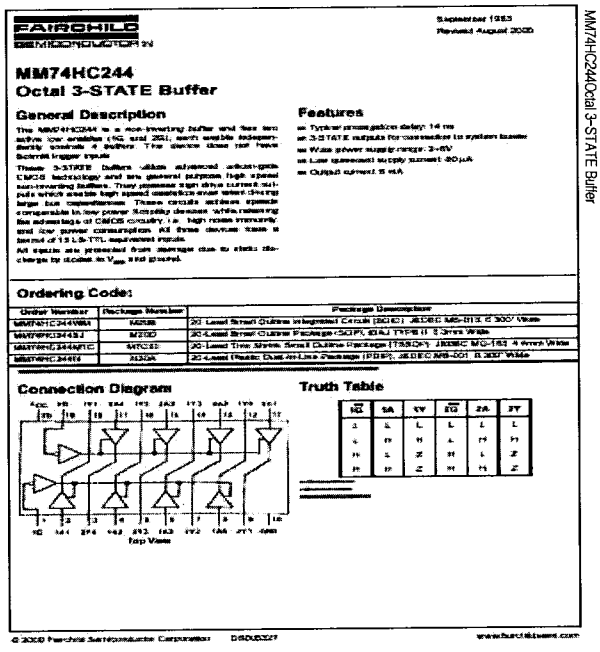
### 2.2 이해 모델의 문제점

이제까지 발표된 관련 연구 중에 API 모델에 인터페이스 프로빙 방식 사용하는 방법이 블랙 박스 방식의 이해를 충실히 따르고 있다. 이 방법은 컴포넌트에서 중요한 인터페이스에 값을 바꾸어 주면서 그 출력 값을 통하여 기능을 잘 파악하는 방법이다. 하지만 이 방법은 COTS(Commercial-Off-The-Shelf) 컴포넌트의 기능과 제한을 이해하는 데 효과적이나 가장 큰 단점은 만들고 분석해야 할 테스트 케이스가 너무 많다는 것이다. 물론 어떤 컴포넌트는 단순한 인터페이스 프로빙을 통하여 쉽게 이해될 수도 있지만 규모가 큰 컴포넌트들은 상당한 량의 인터페이스를 모두 프로빙하는데 많은 노력과 시간, 비용이 소요될 것이다.

규모가 큰 컴포넌트의 경우 서비스적인 측면만이 아니라 컴포넌트를 조립하기 위한 통합 측면의 정보, 즉 데이터 북과 같은 도큐먼트를 통하여 그 기능을 파악할 수 있도록 제공하고 파라메타 값의 범위나 기능, 컴포넌트의 적용 아키텍처, 인터페이스를 이해하는 데 도움을 주는 방법이 효과적일 것이다.

## 3. 하드웨어 데이터북의 관찰

하드웨어 시스템에서 컴포넌트의 예로 집적회로를 생각해 볼 수 있다. 집적회로는 블랙박스이며 하드웨어 엔지니어가 집적회로를 사용하려면 데이터 북을 찾아보고 그 구조와 기능 등, 사용에 필요한 모든 정보를 얻을 수 있다.


FAIRCHILD SEMICONDUCTOR

MM74HC244
Octal 3-STATE Buffer

General Description

Features

Ordering Code:

Connection Diagram

Truth Table

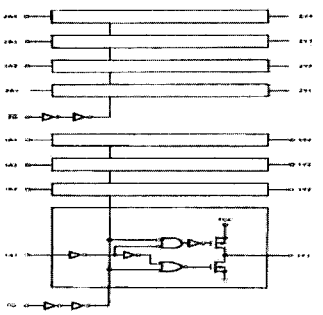
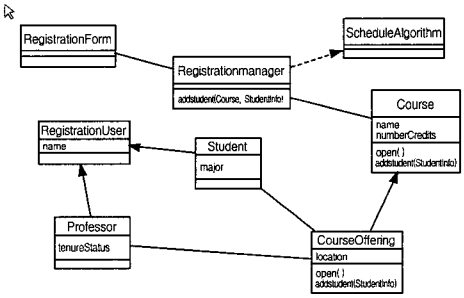
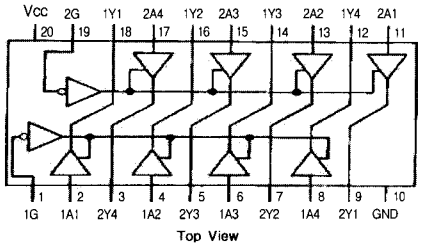
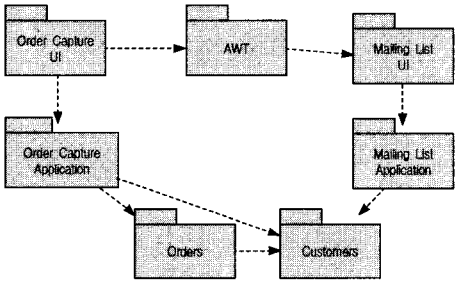
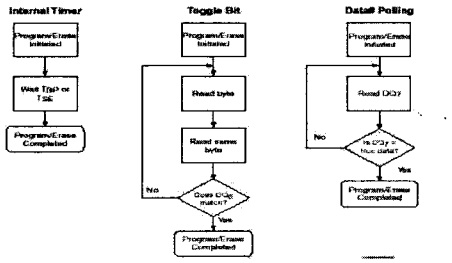
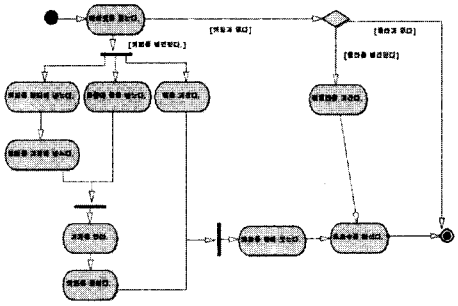
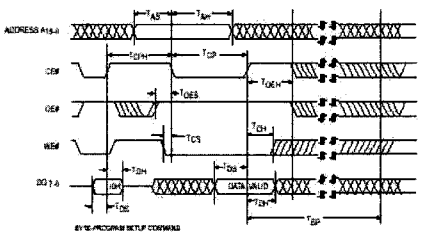
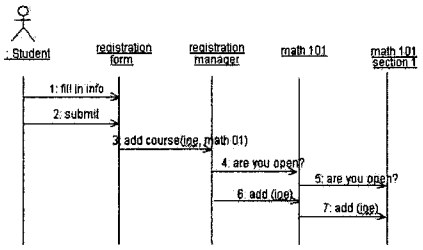
MM74HC244Octal 3-STATE Buffer

(그림 1) 하드웨어 데이터 북

하드웨어 컴포넌트인 집적회로를 설명하기 위한 데이터 북 (그림 1)은 다음과 같은 내용을 담고 있다. 칩의 이름, 입

출력과 구성에 대한 전반적인 설명을 하는 일반적 설명, 핀에 번호를 부여하여 내부 배치도를 나타내는 연결 다이어그램, 상이한 입력 값에 따른 출력 값을 나타내는 진위표, 연결 다이어그램에서 번호를 부여 한 핀에 대한 각각의 회로도를 그린 논리 다이어그램, 이 밖에 기본 환경에 대한 절대치 최대 등급(Absolute Maximum Ratings), 권고된 연산 조건(Recommended Operating Conditions), 칩을 여러 각도에서 바라본 물리적 차원이 있다. 그리고 주요한 부분의 기능에 대한 흐름도인 순서도와 시간의 흐름도를 나타낸 시간 다이어그램, 그리고 테스트를 통해 파라메타의 최대 값과 최소

〈표 1〉 하드웨어와 소프트웨어 데이터북 비교

| 기 능 | 하 드 웨 어 | 소 프 트 웨 어 |
|---|---|---|
| 논리적인 뷰<br>(정적인 관계) | Logic Diagram<br>  <br>논리 다이어그램 |  <br>클래스 다이어그램 |
| 상호 연결<br>구조 | Connection Diagram<br>  <br>연결 다이어그램 |  <br>패키지 다이어그램 |
| 사건 흐름도 |  <br>순서도 |  <br>상태 다이어그램<br>활동 다이어그램 |
| 시간 흐름도 |  <br>시간 다이어그램 |  <br>순차 다이어그램<br>협력 다이어그램 |
| 테스팅 케이스 | 운영 특성 | 인터페이스 프로빙 |

값을 알아보는 운영 특성(Operating Characteristics) 등이 있다.

소프트웨어 컴포넌트는 기본 취지는 같지만 하드웨어와 다른 점이 많다. 데이터 북에 기술된 다이어그램을 목적과 기능 면에서 매칭 시켜보자. 하드웨어 데이터북에서 논리 다이어그램은 연결 다이어그램에서 번호로 나타나 있는 핀에 대한 회로도이며 이것은 시스템 내에 존재하는 객체들의 유형과 그들 사이의 정적인 관계를 표현하는 클래스 다이어그램으로 매핑 된다.

클래스 다이어그램은 여러 다이어그램 중에서도 디자인 패턴을 이해하는데 필수적인 논리적인 뷰에 포함된 다이어그램으로 객체지향 방법론의 중심이 된다. 시스템에 존재하는 객체의 타입과 타입간의 정적인 관계-연관, 서브타입, 클래스 이름, 속성, 연산 등-를 기술한다. 중요한 클래스나 부가적인 설명이 필요한 클래스인 경우는 사전(dictionary)[1]을 추가해 준다.

집적회로 안에 물리적 연결 구조인 핀의 배치도를 나타내는 연결 다이어그램[10]은 패키지 다이어그램과 매칭 된다. 패키지 다이어그램은 클래스들로 이루어진 패키지와 그들 사이의 의존 관계를 보여준다. 의존관계는 한쪽의 정의에 대한 변화가 다른 쪽의 변화를 유발할 때 둘 사이에 존재하는 관계로서 클래스에서는 여러 가지 이유로 존재한다. 이를테면 한 클래스가 다른 클래스에게 메시지를 보낸다거나, 한 클래스가 다른 클래스를 데이터의 일부로 보유하고 있거나, 한 클래스가 다른 클래스를 오퍼레이션의 매개변수로 사용하는 경우 등이다. 만일 한쪽이 인터페이스를 변경하면 그것이 보내는 메시지는 더 이상 유효하지 않게 된다. 이처럼 클래스들 사이의 관계를 나타낸다.

하드웨어 데이터북의 데이터 흐름도[13]는 소프트웨어 컴포넌트의 상태 다이어그램(Statechart Diagram)과 활동 다이어그램(Activity Diagram)과 같은 역할을 한다. 상태 다이어그램은 시스템의 행동을 설명하는 방법이다. 이는 한 특정 객체가 가질 수 있는 모든 가능한 상태들과 객체에 접근하는 사건들의 결과로 객체의 상태가 어떻게 바뀌는지를 설명한다. 활동 다이어그램은 어떤 객체의 행동 과정에서 발생하는 진행 단계와 결정 위치(decision point)를 나타낸다. 이것은 일을 하는 순서를 선택할 수 있도록 해준다. 다시 말하면 병렬 처리를 다룰 수 있다는 것이다. 이 점에서 순차 처리인 흐름도와 차이가 있기는 하지만 이벤트에 따른 처리의 과정에 중점을 둔다면 같이 보아도 무방하다.

상태 다이어그램에 순차 표현(Sequence expression)[8]을 추가한다. 순차 표현은 다음과 같은 규칙을 적용하여 표현한다.

| | |
|---|---|
| S1 ; S2 | S1수행 후 S2수행 |
| S1 \| S2 | S1 또는 S2 |
| S1* | S1를 여러 번(0번 이상) 반복 |
| S1 \|\| S2 | S1과 S2를 함께 수행 |

순차 표현으로 바꾸어 보면 다음과 같다.

[Checking ; Dispatching | (Waiting* ; Dispatching) ; Delivered ]

시간 다이어그램[13]은 교류 다이어그램(Interaction Diagram)에 매칭 된다. 교류 다이어그램은 순차 다이어그램(Sequence Diagram)과 협력 다이어그램(Collaboration Diagram)의 두 가지 형태가 있다. 시퀀스 다이어그램은 특정한 행동을 수행하는 시스템에서 여러 개의 객체들이 서로 메시지를 주고받는 메시지의 순서를 시간의 흐름에 따라 보여준다. 협력 다이어그램은 순차 다이어그램처럼 시간의 흐름을 나타내고 있으며 메시지에 번호를 매기는 것으로 표시한다.

## 4. 소프트웨어 컴포넌트 데이터 북의 구성

소프트웨어 부품으로써 컴포넌트를 어플리케이션 개발에 사용할 때는 어플리케이션 개발자가 컴포넌트를 만든 것이 아니므로, 컴포넌트의 구조나 인터페이스에 대한 이해가 부족하게 된다. 그러므로 어플리케이션 개발자가 이러한 컴포넌트의 구조나 인터페이스에 대한 이해를 높이기 위해서는 컴포넌트 개발자가 컴포넌트를 적용하는 데 도움이 될 수 있는 컴포넌트 명세를 제공하는 것이 중요하다[14]. 컴포넌트 명세는 컴포넌트 데이터 북의 중요한 요소이다.

### 4.1 소프트웨어 컴포넌트의 특성 표현

소프트웨어 컴포넌트는 하드웨어에서 그 개념이 나왔기 때문에 하드웨어 데이터 북과 같은 연상을 가능하게 하도록 소프트웨어의 여러 다이어그램으로 매칭시켜 보면서 소프트웨어 컴포넌트 데이터 북의 구성 요소들을 구성하였다. 그러나 소프트웨어 특성으로 인하여 강조되고 추가되어야 할 부분이 있다. 인터페이스 명세와 컴포넌트 아키텍처 부분이다. 인터페이스는 컴포넌트를 어플리케이션에 적용함에 있어서 직접적으로 컴포넌트와 어플리케이션을 이어주는 중계자의 역할을 담당하여 의사소통을 원활하게 만들기 때문에 사전에 개발된 소프트웨어 부품으로써의 컴포넌트가 개발 이후에 얼마든지 어플리케이션의 한 부분으로써 이상 없이 제 기능을 발휘하도록 돕게 된다. 인터페이스는 컴포넌트의 구조와 의사소통 방법을 어플리케이션으로부터 넘어온 메시지를 적절한 형태로 컴포넌트에 넘겨줌으로써 컴포넌트가

해야 할 일을 정확히 인식할 수 있도록 한다. 따라서 실제적으로 인터페이스는 컴포넌트에서 중요한 구성요소가 되며 소프트웨어 컴포넌트 데이터 북에서도 중요한 구성요소이다.

컴포넌트 아키텍처는 관련된 다른 종류의 컴포넌트들을 연관시키기 위한 표준 계층으로 컴포넌트의 획득, 이해, 조립을 위한 레이아웃을 제시함으로써 사용자들이 필요로 하는 컴포넌트들을 식별하고 검색하며 커스터마이즈할 수 있는 가이드 라인을 제공한다. 컴포넌트 아키텍처는 소프트웨어 컴포넌트를 위한 구성 기술이다. 어떤 요소들로 시스템이 만들어졌는지, 그리고 그 요소 사이에서의 상호작용을 설명한다. 뿐만 아니라 복잡한 시스템에서 이해의 도구[9]가 되며 시스템 요구분석과 만들어진 시스템의 요소 사이의 대응을 알아볼 수 있다. 재사용에도 도움을 주며 IBM San Francisco 프로젝트와 같이 디자인 가이드의 역할을 한다. 또한 비구조적 요구사항에서 적합한 디자인을 선택하는 데 도움을 준다. 제공되는 미들웨어를 통해 상호작용하는 모듈들을 나타냄으로써 컴포넌트 통합에 있어 메카니즘도 제공한다.

〈표 2〉 아키텍처 종류에 따른 UML 도해

| 아키텍처 종류 | 내 용 | 정적 도해 | 동적 도해 |
|---|---|---|---|
| 설계 관점 (Design View) | ● 시스템이 최종사용자에게 제공해야할 서비스를 표현 문제 영역과 해법의 어휘를 형성하고 있는 Class, Interface, Collaboration으로 구성 | 클래스 다이어그램<br>객체 다이어그램 | 교류 다이어그램<br>상태 다이어그램<br>활동 다이어그램 |
| 프로세스 관점 (Process View) | ● 시스템의 성능, 신축성, 처리 능력을 표현<br>● 시스템의 동시성과 동기화 메커니즘을 형성하고 있는 Thread와 Process로 구성 | 클래스 다이어그램<br>객체 다이어그램<br>활성 다이어그램 | 교류 다이어그램<br>상태 다이어그램<br>활동 다이어그램 |
| 구현 관점 (Implementation View) | ● 시스템 배포의 형성관리 표현<br>● 물리적인 시스템을 조립하고 배포하는 데 사용되는 Component와 File들로 구성 | 컴포넌트 다이어그램 | 교류 다이어그램<br>상태 다이어그램<br>활동 다이어그램 |
| 배치 관점 (Deployment View) | ● 시스템을 구성하는 물리적 부분의 분산, 인도, 설치 표현<br>● H/W형태를 형성하는 Node로 구성 | 배치 다이어그램 | 교류 다이어그램<br>상태 다이어그램<br>활동 다이어그램 |

소프트웨어 아키텍처는 일차원적인 것이 아니라, 몇 개의 동시 발생적인 다중적인 관점으로 구성되어진다. 아키텍처는 단일의 통합적 관점에서 보는 것으로, 좀더 세분화된 관점에서 분석되어져야 한다. <표 2>는 소프트웨어 아키텍처의 상이한 관점들을 정리한 것이다. 아키텍처의 각 관점에 대한 요소들을 UML 다이어그램으로 표현할 수 있으며 추후 컴포넌트를 데이터 북에서 설명할 때 사용될 것이다[2].

### 4.2 데이터 북의 구성 요소

컴포넌트 데이터 북의 구성은 다음과 같다.

- 컴포넌트 이름 : 컴포넌트의 이름은 컴포넌트의 내용과 특성을 가장 잘 표현 할 수 있는 설명적인 것으로 한다.
- 분류코드 : 분류코드는 컴포넌트를 체계적으로 관리하기 위해 필요하다. 그러므로 해당 응용 분야도 함께 기술한다. 이와 같은 분류코드는 컴포넌트와 컴포넌트 사이의 계층 구조를 표현하는 데에도 이용될 수 있다.
- 컴포넌트의 개요 : 컴포넌트의 개략적인 설명을 자연어로 기술한다.
- 컴포넌트의 구조
  - 컴포넌트 배경도 : 해당 컴포넌트와 상호 작용하는 외부 요소들과의 관계를 거시적인 관점에서 표현한 부분이다. 상호 작용하는 외부 요소로는 다른 컴포넌트와 액터(actor)가 있다. 이 다이어그램을 통해서 컴포넌트의 사용자와 개발자는 컴포넌트의 전체적인 모습을 파악할 수 있다.
  - 컴포넌트 상호작용 : 일반 시스템의 상호작용 다이어그램과 유사하다. 다른 점은 일반적인 경우에 객체를 중심으로 기술되는 데 반해 컴포넌트 중심으로 하는 상호작용을 파악하는 것이다. 이 다이어그램은 컴포넌트 배경도에 나타날 수 있는 여러 사용사례를 해당 컴포넌트와 외부 요소 사이의 상호작용을 통하여 표현한다.
  - 컴포넌트 인터페이스 : 인터페이스는 컴포넌트의 구조와 의사소통 방법을 어플리케이션에 알리고, 어플리케이션으로부터 넘어온 메시지를 적절한 형태로 컴포넌트에 넘겨줌으로써 컴포넌트가 해야 할 일을 정확히 인식할 수 있도록 한다.
  - 컴포넌트 아키텍처 : 사용 사례를 통해 얻어진 사용자의 요구사항을 통하여, 시스템이 제공해주어야 하는 기능을 찾아내고, 아키텍처를 통해서 시스템의 형태를 결정하는 것이다. RUP(Rational Unified Process)에서는 사용사례 중심, 아키텍처 중심, 반복점증 개발이라는 세 개의 핵심 요소를 가진다.

- 상세 설명 : 2장에서 기술한 UML로 표기하는 방법과 인터페이스 프로빙 방법이다.
  - 객체의 유형과 그들 사이의 정적인 관계를 표현한 클래스 다이어그램.
  - 클래스들로 이루어진 패키지와 그들 사이의 의존 관계를 보여주는 패키지 다이어그램.
  - 객체가 가질 수 있는 모든 가능한 상태들과 접근하는 사건들의 결과로 객체의 상태가 어떻게 바뀌는지를 설명하는 상태 다이어그램과 활동 다이어그램.
  - 시간의 흐름에 따른 메시지 교환의 순서를 보여주는 순차 다이어그램과 협력 다이어그램.
  - 실제 값을 대입하여 나온 결과 값을 참조하는 인터페이스 프로빙 결과표.

### 4.3 소프트웨어 컴포넌트 데이터 북의 종류

컴포넌트 제공을 위해서 필요한 것은 컴포넌트 명세, 컴포넌트 개발 명세, 구현 컴포넌트, 구현 컴포넌트 소스와 컴포넌트 테스트 데이터 등을 제공할 것이다.

그러나 위와 같이 언제나 똑같은 형태로 제공되는 것은 아니다. 컴포넌트에 대한 소유권을 갖고 싶어하거나, 컴포넌트를 변경하여서 새로운 컴포넌트를 개발하기를 원하는 사람들은 컴포넌트 관련 모든 것을 얻기를 바랄 것이다. 이와 같은 개발자들을 위한 것을 컴포넌트 개발 데이터 북이라고 하겠다. 여기에는 4.2에서 구성한 모든 내용을 포함시키면 된다. 또한 컴포넌트 구입해서 통합 개발 후, 어플리케이션으로 판매를 목적으로 하는 사람들은 컴포넌트 사용이 주된 목적이다. 이와 같은 개발자들을 위한 것을 컴포넌트 사용 데이터 북이라고 하겠다. 여기에는 위와 같이 모든 것이 필요하지는 않다. 그러므로 그 이상의 부분들을 제공한다는 것은 컴포넌트 사용자가 컴포넌트를 사용하는 목적인 캡슐화가 제공할 수 있는 단순함을 해치는 일이 된다. 단지 사용자를 위한 것이라면 컴포넌트 이름, 분류코드, 컴포넌트 개요, 요구분석(Requirements), 주된 설명(Main Specification)이다.

요구 분석은 문제를 설명하기 위한 단계이다. 여기서는 Use Case와 Business Process를 이용하여 표현한다. Use Case는 시스템의 사용에 대한 시나리오의 집합이다. 사용자의 관점에서 시스템을 모델링하기 위한 목적이 있다. 즉, 사용자가 시스템에 대하여 바라는 바를 표현한다. 사용자의 시점을 빨리 이해함으로써 쓸모 있고, 쓸 수 있는 시스템을 만들 수 있다. 이는 사용자의 시스템 사용 시나리오를 표현한 것이기 때문에 이를 따라 테스트할 수 있다.

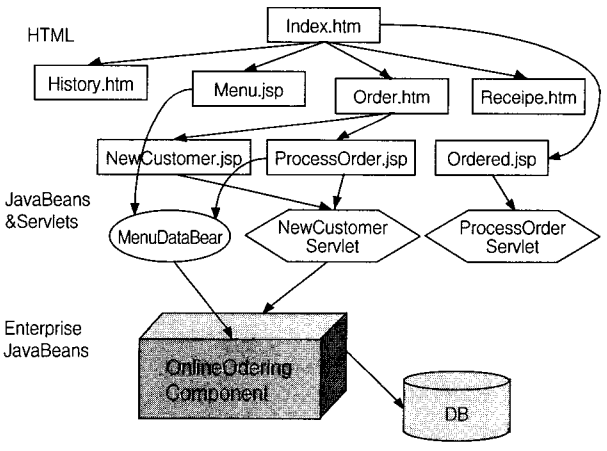
주된 설명에서는 컴포넌트에 가장 중요한 문서들이 포함된다. 인터페이스 명세(Interface Specification), 컴포넌트 아키텍처이다. 인터페이스 명세에서는 클라이언트와 컴포넌트사이의 인터페이스를 정의한다. 그리고 오퍼레이션 명세를 만든다. 그 내용은 컴포넌트 객체에 제공되거나 전달되는 정보에 대한 입력 파라메타, 컴포넌트 객체에 리턴 되거나 업데이트 되는 정보에 대한 출력 파라메타, 컴포넌트 객체에 대한 결과 상태 변화, 그리고 적용에 필요한 제약사항이다. 입출력 파라메타는 Interface Specification Diagram으로 표현한다. 또한 컴포넌트의 상태변화는 각 오퍼레이션의 precondition과 postcondition로 나타낸다. 필요한 제약사항은 OCL(Object Constraint Language)로 표기할 수 있다[3].

컴포넌트 아키텍처는 시스템 전체에 대한 형태를 이해하기 위해 필요하다. 시스템 전체에 대한 설계도는 시스템 개발 단계에서 고려되어야 하는 각각의 관점에 대한 전체적인 통합을 제공한다. 그래서 아키텍처란 바로 시스템 전체에 대한 관점이다.

## 5. 사용 사례

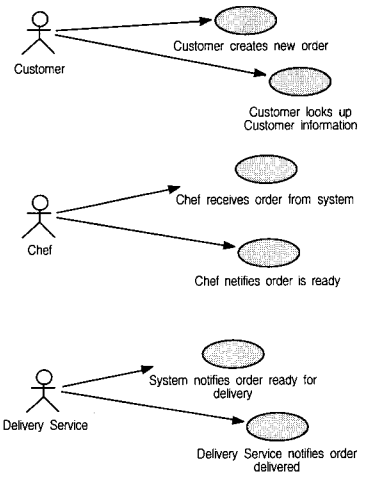
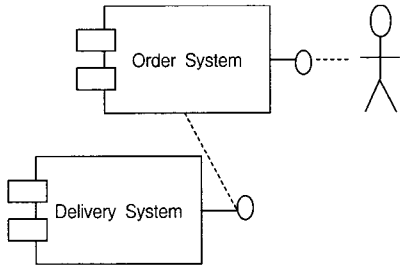
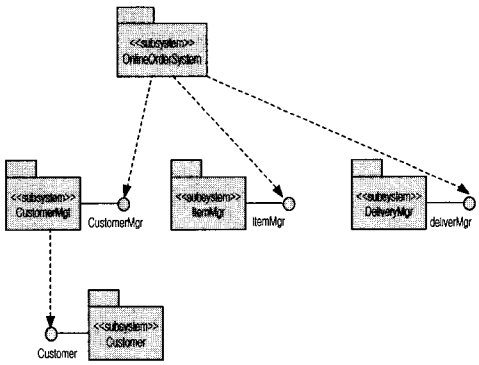
컴포넌트 데이터 북은 컴포넌트의 이해를 충분히 도울 수 있을 만큼 상세하고 분명해야 한다. 이를 위하여 컴포넌트의 특성을 살린 컴포넌트 데이터 북을 실험적으로 구성하여 보았다.

Henri라는 음식점은 인터넷 온라인 주문 시스템을 통해 Stuffed Pepper 메뉴를 제공함으로써 새로운 시장을 개척하려고 한다. 고객은 인터넷을 통해 온라인 주문 시스템에 접속한다. 고객은 다양한 메뉴를 고를 수 있다. 고객이 선택한 메뉴는 서버에 전달된다. 서버는 선택한 품목에 대해서 가격을 계산하고 이것을 고객에게 제시한다. 고객은 이를 확인하거나 취소할 수 있다. 고객이 확인한 주문은 레스토랑에 출력되어 전달된다[16].




(그림 2) 전체 아키텍처

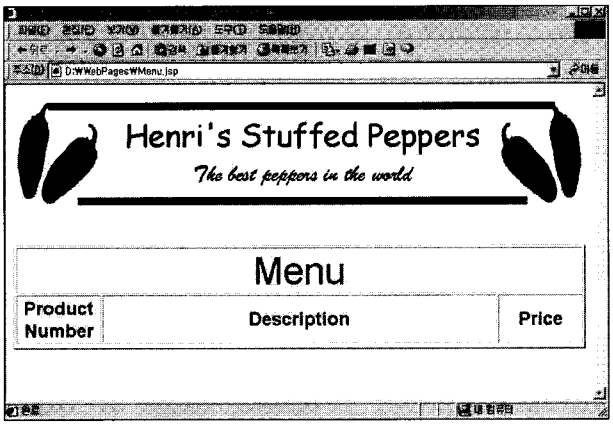
〈표 3〉 컴포넌트 데이터 북

<table>
<tr><td colspan="4">컴포넌트 명세</td></tr>
<tr><td>1. 이 름</td><td>온라인 주문<br>배달 시스템</td><td>4. 요구 분석</td><td>Use Case Model</td></tr>
<tr><td>2. 분류코드</td><td colspan="3">xxx.xxx</td></tr>
<tr><td colspan="2">3. 개 요<br>온라인 상에서 주문을 받아 주문 내역을 가게에 전달한다. 주문 품목이 완성되었으면 가게에서는 배달업체에 전달하여 배달되는 소프트웨어 컴포넌트</td><td colspan="2" rowspan="2"> 
</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td colspan="2">5. 인터페이스<br>명세 3. 주문취소</td><td colspan="2">6. 컴포넌트 아키텍처</td></tr>
<tr><td colspan="2">3. 주문 취소<br>3.1 관련 정보<br>3.1.1 입력정보 : 고객식별정보, 주문 내역정보<br>3.1.2 출력 정보<br>3.1.3 처리 정보 : 주문한 품목<br>3.2 선행조건 : 고객의 주문 내역에서 취소를 요구하는 주문 품목이 존재한다.<br>3.3 처리 조건<br>3.4 후행 조건 : 고객의 주문 내역에 취소한 주문 품목이 존재하지 않는다.</td><td colspan="2" rowspan="2"> 
</td></tr>
<tr><td colspan="2">OCL(Object Constraint Language)<br>ICustomerMgr : : cancelMenu( c : CustomerID, o : ItemID) : Boolean<br>pre : self.OderingItem → exit(io|ItemOffering | io.io_id = o.io.id)<br>post : self.OderingItem → excluding<br>(io|ItemOffering | io.io_id = o.io.id)</td></tr>
</table>

개발자는 이러한 요구사항을 이해하고 무엇보다도 비용을 절감하고 개발기간을 단축하기 위해 재활용 가능한 EJ Bean을 사용하기로 하였다면 적합한 EJ Bean을 데이터 북에서 선택하였을 것이다. <표 3>은 개발자가 사용하려고 하는 컴포넌트 사용 데이터 북의 내용이다.

<표 3>의 컴포넌트 사용 데이터 북에 있는 컴포넌트를 사용하여 Henri라는 음식점의 온라인 주문 시스템을 구축하면 다음과 같이 된다.

```
public double getPrice() throws java.rmi.RemoteException {
    EJSDeployedSupport s = new EJSDeployedSupport() ;
        double result ;
        try {
            s.setTx(preInvoke(0)) ;
        com.ibm.ejbatwork.beans.ProductBean beanRef
            = (com.ibm.ejbatwork.beans.ProductBean) ebRef ;
        result = beanRef.getPrice() ;
        } catch (java.rmi.RemoteException ex) {
                s.setUncheckedException(ex) ;
 생략 …
```




(그림 3) 컴포넌트 실행 결과

## 6. XML 기반의 데이터 북 구현에 의한 검증 실험

이제까지 설명한 컴포넌트 데이터 북은 일반적인 객체 컴

포넌트에 적용될 수 있는 요소들로 구성되어 있다. 따라서 제안한 테이터 북의 항목들이 과연 EJB[11]와 같은 실제 소프트웨어 컴포넌트를 이해하고 사용하는데 적합한지 검증하기 위하여 다음과 같은 실험을 추가하였다.

EJB에는 배치 디스크립터라는 것이 있다. 소프트웨어 컴포넌트인 EJB의 런 타임 속성들, 예를 들면 서버 위치, 지속성(persistence) 타입, 빈의 자원, 보안, 트랜젝션 속성 등을 XML로 기술하는 것이다. 배치 스크립터의 속성에서 컴포넌트 사용자가 얻을 수 있는 정보는 빈에 대한 간략한 기술과 컴포넌트 이름, 컴포넌트 인터페이스와 구성 클래스, 컴포넌트 접근권한 정도이다. 이러한 정보만으로는 컴포넌트 은행으로부터 추출된 컴포넌트가 사용자의 요구사항에 적합한지를 판단할 수 없으며, 또한 사용자가 해당 컴포넌트에 대해 정확히 이해하기 힘들다.

```
<!ELEMENT cluster (cluster-code)>
<!ELEMENT component-description (version?, date?, author?,
os?, language?, middleware?, descript?)>
<!ELEMENT architecture diagram (description?,
component-diagram?, interaction description?, interface)>
<!ELEMENT interface (description?, hierach?, constraints?,
method-name, method-params?, rolename?)>
<!ELEMENT role (description?, role-name, method-name?)>
<!ELEMENT detail diagram (description?, class-diagram?,
interaction-diagram?, state-diagram?, package-diagram?,
interface-probing table?)>
<!ELEMENT cluster-code(#PCDATA)>
<!ELEMENT version(#PCDATA)>
<!ELEMENT date(#PCDATA)>
<!ELEMENT author(#PCDATA)>
<!ELEMENT descript(#PCDATA)>
<!ELEMENT keyword(#PCDATA)>
<!ELEMENT constraints(#PCDATA)>
```

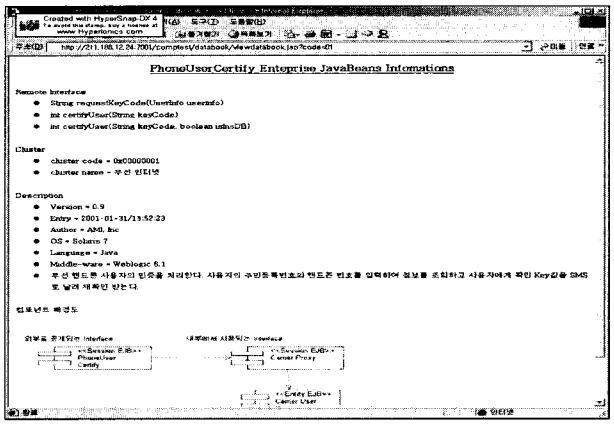
(그림 4) 확장된 EJB 배치 디스크립터의 DTD

따라서 사용자가 컴포넌트의 구조나 인터페이스에 대한 이해를 높이기 위하여 4장에서 제안한 컴포넌트 데이터 북의 항목들을 확장된 배치 스크립터로 제공하고 이를 XML로 제공할 수 있도록 (그림 4)와 같이 DTD를 정의하였다. 정의한 XML로 논문에서 제안한 항목들을 배치 디스크립터로 나타내면 (그림 5)와 같이 된다.

위와 같은 방법으로 인터넷 북 서점을 위한 컴포넌트 뱅크를 데이터 북으로 제공하였을 때 얻을 수 있는 효과를 <표 4>와 같이 측정하였다. 자세한 측정 및 분석 방법은 [6]의 방법을 따랐다. 정확성과 기억재생성(recall)은 [23]에서 정의된 바와 같이 적합한 컴포넌트를 선택한 비율을 나타낸다. 여러 차례의 실험결과의 평균을 점수로 나타내었으며 컴포넌트 이해 실험에는 전문가에서 초보가까지 여러 단계의 수준차를 가진 실험 대성을 선정하였다.

```
<? xml version = "1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-// Sun Microsystems,
Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http : //java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
   <enterprise-beans>
      <entity>
         <description>
              This Cabin enterprise bean entity represents a
               cabin on a cruise ship.
         </description>
<ejb-name>CabinBean</ejb-name>
         <home>com.titan.cabin.CabinHome</home>
         <remote>com.titan.cabin.Cabin</remote>
<ejb-class>com.titan.cabin.CabinBean
          </ejb-class>
          <component-interface>
          <remote-method>
          <Object cast = String>getName</Object>
          <void>setName(String)</void>
          <constraint>self.name.length <= 30
          </constraint>
          <int>getDeckLevel()</int>
          <void>setDeckLevel(int)</void>
          <constraint>
          </constraint>
          <int>getShip() </int>
          <void>setShip(int)</void>
          <int>getBedCount()</int>
          <void>setBedCount(int)<void>
          <Object cast = UserInfo>getUserInfo()</Object>
          <void>setUserInfo(UserInfo)</void>
          </remote-method>
          </component-interface>
          · 중략 ·
<persistence-type>Container</persistence-type>
<prim-key-class>com.titan.cabin.CabinPK
        </prim-key-class>
</entity>
</ejb-jar>
```

(그림 5) 확장된 배치 스크립터로 나타낸 컴포넌트




(그림 6) 브라우저로 본 컴포넌트 데이버 북

〈표 4〉 컴포넌트 데이터북의 효과 분석

| | 점 수 | 평 균 | 편 차 |
|---|---|---|---|
| 컴포넌트 선택 효율성 | | 68.85 | 0.46 |
| 정확성 | 69.17 | | |
| 기억 재생성 | 68.53 | | |
| 컴포넌트 이해도 | | 79.98 | 1.95 |
| 만족도 | 82.25 | | |
| 적합성 | 80.09 | | |
| 완전성 | 77.49 | | |
| 구조성 | 82.25 | | |

위 표에서 컴포넌트 선택 효율성은 컴포넌트 데이터 북의 이해를 토대로 선정한 후보 컴포넌트가 최종 소프트웨어 완성에 채택되는 비율을 퍼센트로 나타낸 것이다. 이해도는 컴포넌트를 재사용하는 프로그래머들이 주관적으로 판단할 때 데이터북의 내용에 대하여 얼마나 만족하며 적합한지, 완벽한 내용이 포함되어 있는지, 구조적으로 표현되었는지를 평가한 것이다.

## 7. 결론 및 향후 연구

컴포넌트를 조립하여 소프트웨어를 생산하는 방식으로 패러다임이 변해감에 따라 소프트웨어 컴포넌트 개발, 판매에 못지 않게 재사용 과정에 대한 지원이 필요하다. 부품으로써 개발된 컴포넌트를 어플리케이션 조립에서 쉽게 이용하려면 컴포넌트의 이해가 중요하다.

하드웨어 컴포넌트인 집적 회로를 사용하는 사람들을 위해 필요한 모든 정보를 담고 있는 데이터 북에서 착안한 소프트웨어 컴포넌트 데이터 북을 제안하였다. 하드웨어와 소프트웨어는 서로 상이한 점들이 많으나 설명하고자 하는 목적과 기능을 중심으로 서로 매칭시켜 소프트웨어 컴포넌트에서 강조되어야 하는 부분인 인터페이스와 아키텍처를 추가하였다. 컴포넌트 제공을 위해서 필요한 것은 컴포넌트 명세, 컴포넌트 개발 명세, 구현 컴포넌트, 구현 컴포넌트 소스와 컴포넌트 테스트 데이터 등을 제공할 것이다.

이와 같은 것들의 제공은 언제나 똑같은 형태를 갖추지는 않는다. 컴포넌트에 대한 소유권을 갖고 싶어하거나, 컴포넌트를 변경하여서 새로운 컴포넌트를 개발하기를 원하는 사람들에게는 컴포넌트 관련 모든 것을 제공하면 된다. 소프트웨어 컴포넌트 데이터 북에는 소프트웨어 컴포넌트의 동적 상세 정보가 UML로 자세히 표현되어야 한다.

이 연구에서 제안한 컴포넌트 데이터북의 내용은 EJB의 배치 디스크립터의 내용으로 확장될 수 있다. EJB로 구현된 소프트웨어의 컴포넌트의 런타임 속성을 기술하는데 머무르지 않고 컴포넌트 이해를 위한 카달로그로 확장된다면 컴포넌트의 자세한 기능을 JAR 파일만의 제공으로 이해하고 재사용할 수가 있다. 앞으로 컴포넌트 뱅크에서 원하는 컴포넌트를 쉽게 찾을 수 있도록 하기 위하여 추가되어야 하는 부분과 이러한 과정에서 자동화를 위한 CASE 툴에 대한 연구도 함께 진행될 필요가 있다.

## 참 고 문 헌


[1] D'Souza, D., Wills, A., Objects, Components, and Frameworks with UML, Addison-Wesley, 1998.

[2] John Cheesman, John Daniels, "UML Components," Addison-Wesley, 2000.

[3] Jos Warmer, Anneke Kleppe, "The Object Constraint Language, precise modeling with UML," Addison-Wesley, 1998.

[4] Korel, B., "Black-Box Understanding of COTS Components," in Proceeding 7th International Workshop on Program Comprehension, IEEE, pp.92-99, 1999.

[5] Larson, G., "Designing Component-Based Frameworks Using Patterns in the UML," Comm. of the ACM, Vol.42, No.10, pp.38-45, 1999.

[6] Pighin, M, Brajnik, G. "A Formative Evaluation of Information Retrieval Techniques applied Software Catalogues," Information Software Technology, Vol.52/2-3, Elsevier Science Publications, pp.131-138, 2000.

[7] MM74HC244 Octal 3-STATE Buffer, Silicon Storage Technology Inc, 2000.

[8] Monica Ferreira da Sliva, Claudia Maria Lima Werner, "Packaging Reusable Components Using Patterns and Hypermedia," 4th International Conference on Software Reuse, 1996.

[9] Philippe B. Kruchten, Architectural Blueprints-The "4+1" View Model of Software Architecture, IEEE Software, pp.42-50, November, 1995.

[10] Richard A. DeMillo, "Test Adequacy and Program Mutation," Software Engineering, 11th International Conference ACM, pp.355-356, 1989.

[11] Richard Monson-Haefel, "Enterprise JavaBeans(3rd ed.)," O'Reilly, 2001.

[12] Sneed, H., Dombovari, T., "Comprehending a Complex, Distributed, Object-Oriented Software System : A Report from the Field," in Proceeding of 7th International Workshop on Program Comprehension, IEEE, pp.218-225, 1999.

[13] 4Megabit(512K x8) SuperFlash EEPROM, SST28SF040A/SST28VF040A, Silicon Storage Technology Inc., 2000.

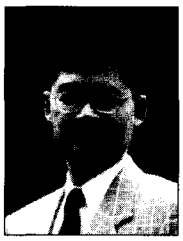
[14] 김동현, "UML을 이용한 자바빈즈 컴포넌트 설계", SETC'99, 1999, pp.263-268.

[15] 신인철역, "Enterprise JavaBeans 예제로 배우기", 인터비젼, 2000.

[16] 최은만, "컴포넌트 기반 소프트웨어 개발에서 프로그램 이해 문제", 소프트웨어공학회지, pp.71-78, 2000.

## 김 선 희

e-mail : ester@motrola.com
1999년 서울여자대학교 전산학과(학사)
2001년 동국대학교 컴퓨터공학과(공학석사)
2001년~현재 모트롤라 코리아 연구원
관심분야 : 소프트웨어 이해, 컴포넌트 설계 방법론, 소프트웨어 품질 보증, CMM



## 최 은 만

e-mail : emchoi@dgu.ac.kr
1982년 동국대학교 전산학과(학사)
1985년 한국과학기술원 전산학과(공학석사)
1993년 일리노이 공대 전산학과(공학박사)
1985년~1988년 한국표준연구소 연구원
1988년~1989년 데이콤 주임연구원
1993년~현재 동국대학교 컴퓨터공학과 부교수
2000년~2001년 콜로라도 주립대 전산학과 방문교수