상호작용 중심의 컴포넌트 인터페이스를 표현하기 위한 UML의 확장

박 성 호[†]·최 은 만[††]

요 약

이 논문에서는 소프트웨어 부품이 되는 컴포넌트를 설계하기 위한 요소 중 컴포넌트간 상호작용에 중점을 두어 이를 체계적으로 분류한 후, 이에 맞는 도해적 표기방법(graphical notation)과 이를 이용한 설계 방법을 제시하고 실제 설계 사례를 통해 이를 검증하며, 컴포넌트 설계 및 모델링을 효과적으로 표현할 수 있는 방법에 대해 논의하였다. 컴포넌트는 일반 클래스와는 달리 그 규모가 크며 인터페이스와 이를 이용한 컴포넌트 간의 상호작용에 대한 표현이 매우 부족하다. 따라서 이 연구에서는 컴포넌트 표현기법을 제시하기 위하여 UML을 확장하였고 이를 이용하여 실험적으로 설계해 보고 그 효용성을 검토하였다.

Extension of UML to Represent Components Interface Focusing on Interaction

Sungho Park[†]·Eun Man Choi[††]

ABSTRACT

This paper focuses components interactions which is important factor in designing software components. We classified several types of interaction between components and suggested appropriate graphical notation to extend UML and design process with extended method. Suggested notation and process was verified by practical experiment which is performed in travel agent component application. Representing interaction between components is not same to representing association between classes. UML in current version needs to be extended for representing this kind of components interaction. This research covers UML extension for components interaction and experiment for showing effectiveness.

키워드 : 상호작용(Interaction), UML(Unified Modeling Language), 객체지향 설계(Object-oriented Design), 컴포넌트 인터페이스 (Component Interface)

1. 서 론

소프트웨어 생산효율성의 저하로 인한 '소프트웨어 위기(software crisis)'의 해결방법으로써 소프트웨어 부품화는 오래 전부터 논의되어 왔으나 그 제반여건의 미비로 활성화되지 못하였다. 그러나 20세기 말 시작된 정보화의 물결은 광범위한 대상에 대한 컴퓨터의 이용을 촉진했으며 이에 따른 엄청난 양의 소프트웨어 수요를 만들어내었다. 효율적이며 신뢰할 수 있는 소프트웨어의 개발은 비단 컴퓨터공학뿐 아니라 사회전반에 걸쳐 중요한 문제로 대두되었으나 이에 비해 소프트웨어의 개발은 생산적이지 못한 특성을 아직 극복하지 못하고 수요에 미치지 못하는 실정이다. 이에 따라 빠르고 경제적인 소프트웨어 개발을 위한 다양한 방법론이 제시되고 있는데, 객체지향 개발방법론도 그 중 하나이다. 80년대 중반부터 주목을 받아온 객체지향 방법론은 소프트웨어의 개발에 재사용과 보호의 개념을 도입함으로써 소프트웨어 개발의 생산성을 높이고자 하였다. 그러나 객체지향 방법론 역시 재사용의 효과적인 적용에 실패하였다.

이에 따라 90년대 들어 새롭게 컴포넌트를 기반으로 한 소프트웨어 개발방법론(CBSD ; Component Based Software Development)이 주목을 받게 되었다. 컴포넌트는 객체지향의 클래스와 달리 범용적이며, 재사용이 쉽고, 구성객체간 결합도(coupling)가 낮기 때문에 필요한 컴포넌트를 찾아 통합하기만 하면 빠르고 경제적으로 소프트웨어를 만들 수 있을 것으로 기대되고 있다. 이것은 GUI를 기반으로 한 윈도우 시스템에서, 사용자 인터페이스에 대한 컴포넌트의 적용이 성공적으로 정착된 데 영향을 받았으며, 이미 VB 컴포넌트를 기반으로 하는 상업적인 시장이 성립되어 있다.

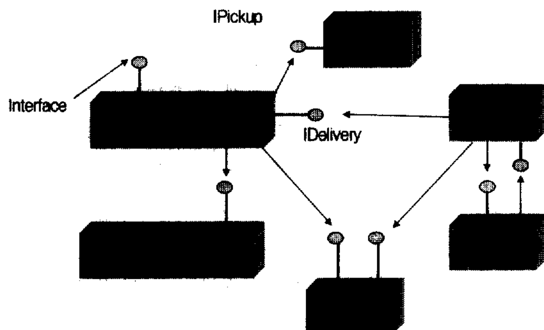
† 정 회 원 : 소프트포럼(주) 기술 연구소
†† 정 회 원 : 동국대학교 컴퓨터공학과 교수
논문접수 : 2001년 3월 24일, 심사완료 : 2001년 10월 30일

그러나 기존의 방법론을 이용한 모델링과 소프트웨어의 설계는 컴포넌트의 기본적인 특성인 인터페이스에 대한 표현이 단순하고 제한된 표현 방법으로 그 뜻을 나타내기 때문에 여러 가지로 해석이 가능할 수 있는 모호한 점이 있고, 내부 디자인의 규모가 객체지향 방법론의 중심이 되는 객체 혹은 클래스와, 컴포넌트 사이에 큰 차이점을 보이게 되므로 정확한 컴포넌트의 설계 및 표현에 한계가 있다. 또한 컴포넌트 기반 개발과정은 컴포넌트의 조합(composition)과 재사용성을 강조하기 때문에 전통적인 소프트웨어 개발과정뿐만 아니라 객체지향적 소프트웨어 개발과정과도 다른 몇 가지 특성을 갖고 있으므로 기존의 방법론으로는 이런 특성을 적절하게 반영하기 어렵다. 그러므로 컴포넌트를 표현하기 위한 방법론에서는 그 특성에 적합한 표현방법이 필요하다고 할 수 있다. 더욱이 컴포넌트 표현에서 중요성을 갖는 컴포넌트간 상호작용 측면에서는 좀 더 상세하고 효과적으로 이를 표현할 수 있는 수단을 제시하는 과정이 반드시 포함되어야 할 것이다.

2. 컴포넌트의 상호작용

컴포넌트는 특정한 목적이 있는 기능을 수행하기 위하여 상호 협동하여 작업하는 개체들의 모임을 말한다[10]. 따라서 컴포넌트 개발에서 컴포넌트의 상호 작용과 협동 작업은 매우 중요하다. 객체지향 프로그램의 경우 전체 시스템을 이루는 객체들의 상호작용보다는 개별적인 객체에 치중하여도 충분히 시스템을 만들 수 있으나 컴포넌트와 같이 규모가 커지는 경우 컴포넌트의 단순한 정적인 인터페이스에만 초점을 맞춘다면 컴포넌트를 설계하기 어렵다. 실제 객체지향 프로그램 설계에서 동적 모형을 나타내는 인터액션 다이어그램은 클래스에 속하여야 할 오퍼레이션을 찾는 목적으로 작성되는 경우가 많다.

반면 컴포넌트를 이용한 개발에서는 시스템을 이루는 컴포넌트들의 상호작용을 찾아내고 이를 나타내는 과정이 매우 중요하다. 예를 들어 물류회사의 창고관리 시스템의 예를 들어보자. 창고관리를 위하여 Warehouse, 상품 공급원을 위한 Supplier, 배달을 위한 Delivery 컴포넌트가 필요하다.



Focus on behavior: stored data, implemented procedures, other interfaces remain completely hidden

(그림 1) 컴포넌트의 인터페이스

각 컴포넌트에 대한 명세를 파악하려면 (그림 1)에 표현된 인터페이스만으로는 부족하다. 컴포넌트의 행위, 즉 어떤 다른 컴포넌트의 자료를 어떤 식으로 접근하여 어떤 워크플로우로 기능을 구현하고 다른 컴포넌트와 어떤 연관성이 있는지 파악할 필요가 있다.

결국 컴포넌트 단위의 설계를 나타낼 때는 정적인면과 동적인면을 망라하여 표현하는 방법이 필요하며 이런 측면에서 UML 표현 방법을 확장하려 한다.

3. 관련 연구

3.1 UML

UML 표기법에 있어서 컴포넌트를 이용한 소프트웨어를 모델링 할 수 있는 직접적인 수단은 포함하고 있지 않았다. 그렇기 때문에 이것을 모델링하기 위해서는 기존의 다이어그램을 이용해야 한다. 일반적으로 컴포넌트의 표현과 관련이 있는 다이어그램은 클래스 다이어그램(class diagram), 패키지 다이어그램(package diagram)과 배치 다이어그램(deployment diagram)등이 있다.

컴포넌트 다이어그램은 컴포넌트 사이의 의존성과 구조를 모델링 하여 컴포넌트간의 상호작용을 표현할 수 있도록 하였다. 여기에 나타나는 컴포넌트의 개념은 한 패키지에 포함될 수 있는 다양한 개념으로 사용가능한데 예를 들어 《documentation》 스테레오타입을 사용하여 문서를 구성요소로서 포함할 수도 있다[2].

배치 다이어그램의 경우 컴포넌트 기반 소프트웨어 동작 환경 등을 나타내며, 컴포넌트를 포함한 시스템의 물리적 구조를 나타냄으로써 시스템에 대한 이해를 증진시킨다.

컴포넌트 기반의 개발과정에서 필수적이라 할 수 있는 인터페이스에 대한 표기방법으로 UML에서는 클래스 다이어그램을 사용하여 나타낼 수 있다. 이것은 일반적인 객체에 적용되는 클래스 다이어그램의 변형으로써 클래스 다이어그램에 《Interface》 스테레오타입을 추가한다[4].

UML은 소프트웨어의 설계를 위한 언어지만 모든 상황에 대한 의미를 표현하기 위해서는 충분하지 않다. 또한 미묘한 의미의 차이를 표현하기에는 부족한 면이 있다. 이를 위해 UML은 다음과 같은 확장 기법을 통하여 확장할 수 있다.

3.1.1 꼬리표값(tagged value)

꼬리표값은 UML에서 어떠한 구성 단위(building block)의 특징들을 확장한다. 예를 들어 어떠한 제품이 있고 이 제품이 시간이 지남에 따라 확장된 제품임을 표시하고 싶어진다. 이때 꼬리표 값으로써 제품의 버전(version)과 어떤 부분의 저자(author)를 나타낼 수 있다.

3.1.2 제한요소(constraints)

제한요소는 구성 단위의 의미를 확장한다. 이러한 제한요소는 기존의 의미를 고치거나 새로운 규칙을 적용할 수 있도록 한다.

3.1.3 스테레오타입(stereotype)

스테레오타입은 UML의 어휘에 대한 확장이다. 어휘의 확장이란 스테레오타입을 사용하여 기존의 구성 단위를 새로운 종류의 구성 단위들로 만드는 것을 말한다.

3.2 컴포넌트 개발 방법론

현재 제안되었던 컴포넌트 기반 개발 방법론으로는 Catalysis와 RUP(Rational Unified Software Development Process)등이 있다.

3.2.1 Catalysis

컴포넌트 소프트웨어 설계를 위한 방법론으로 Platinum Technology에서 개발되어 적용되고 있다. Catalysis는 기본적으로 UML 표기법을 사용하여 Active-X를 기반으로 한 컴포넌트 응용 어플리케이션 제작을 위한 표준 공정을 제공함으로써 시스템 설계자나 제작자가 쉽고 빠르게 소프트웨어에 대한 설계를 진행할 수 있도록 돕는다. Catalysis는 프레임워크 수준의 시스템 구조도를 일관된 관점으로 발전시켜 소프트웨어의 설계를 이끌어내게 된다.

컴포넌트 설계를 위해 Catalysis는 다양한 표기법을 이용하여 컴포넌트와 인터페이스 및 상호작용을 표현한다. 기본적으로 Catalysis는 커넥터 심볼을 이용하여 컴포넌트간의 연계를 나타내며 각각의 커넥터 심볼 위에 컴포넌트간 연결방법을 스테레오타입으로써 표기함으로써 상호작용의 구조를 나타낸다.

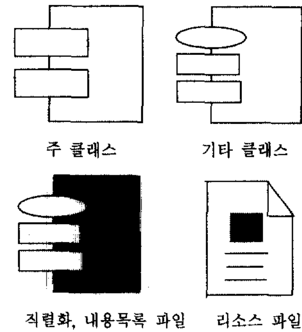
3.2.2 UML을 이용한 자바빈즈 컴포넌트 설계

"UML을 이용한 자바빈즈 컴포넌트 설계"[9]에서는 자바빈즈를 기반으로 하는 컴포넌트 설계를 위한 표기법과 그 의미를 기술하고 있으며 이것은 자바빈즈 컴포넌트의 설계와 배포 시에 효과적으로 사용될 수 있다.

자바 빈은 상호작용하는 하나 이상의 클래스들을 사용하여 주어진 요구 사항들을 구현한다. 따라서 이러한 클래스들의 집합을 포함하는 단위로서 UML의 패키지를 사용하여 자바 빈을 표현하며, 자바 빈을 명확히 하기 위해 《Bean》 스테레오 타입을 사용한다. 그리고 패키지를 다음과 같이 세 부분으로 나누어 표현한다. 각각의 부분은 인터페이스, 기능명세(Use-Case로 표현), 클래스 다이어그램 부분으로 나타낸다[9].

자바 빈의 JAR 파일에 포함되는 여러 클래스 파일, 직렬화(Serialization) 파일(.ser), 리소스 파일 등의 관계를 설계할 필요가 있을 때 UML의 컴포넌트 다이어그램으로 형상화한다.

이때 각각의 파일들을 동일한 표기법을 사용하여 표현하는 경우 가독성이 떨어지므로 각 파일의 종류에 따라 다음과 같이 서로 다른 컴포넌트 표현을 사용한다. 그림2는 여러 가지 컴포넌트의 표현을 보여준다. 이들 중 클래스와 직렬화 파일 표현들은 UML 표준에는 없지만 권장하는 컴포넌트 스테레오 타입들로서 각 파일의 의미와 일치하는 표현들을 사용하였으며, 리소스 파일을 위한 표현은 컴포넌트 스테레오 타입을 의미에 맞게 수정한 것이다. 이러한 표기법들은 UML과 직접적인 관련성은 적지만 이전의 방법론들과 연계성을 갖는 것도 있다.



(그림 2) 여러 가지 컴포넌트의 표현

이 외에 컴포넌트기반 개발방법론으로 Rational사의 Grady Booch, James Rumbaugh, Ivar Jacobson이 제안한 UML 기반의 RUP(Rational Unified Software Development Process)가 있다. RUP는 소프트웨어의 구조와 Use-case를 중심으로 일관된 관점을 모델의 분석에서 구현과 테스트에 이르기까지 유지하고 각 반복단계에서의 개발활동에 대한 기준을 제공함으로써 복잡한 개발 프로젝트를 진행하는데 필요한 개발 프로세스의 역할을 담당한다.

위에서 살펴본 바와 같이 컴포넌트 기반의 개발과정의 설계 및 모델링을 위한 다양한 방법이 존재한다. 하지만 이러한 방법들은 다양한 정보를 포함하거나 사용자의 이해성을 높이는 등 각각의 장점과 함께 단점을 포함하고 있다.

<표 1>은 각각의 개발 방법론의 특징을 비교해 놓은 것이다.

〈표 1〉 컴포넌트 기반 개발 방법론의 특성 비교

	Catalysis	JavaBeans 설계	RUP
프로젝트 규모	일정규모 이상의 프로젝트에 적합하다.	소규모 소프트웨어 부품 개발에 적합하다.	중, 대규모 프로젝트에 적합하다.
개발 프로세스	객체지향 및 컴포넌트 기반 프로세스가 존재한다.	특별한 프로세스를 포함하지 않는다.	객체지향 프로세스 위주로 개발과정을 설명하고 있다.
컴포넌트의 지원	컴포넌트 기반 개발 과정을 지원한다.	컴포넌트를 위한 표기법 지원	특별한 언급이 없다.
표기방법	UML이외의 비표준적인 표현방법을 사용한다.	UML 위주의 표현 방법을 사용한다.	UML을 사용한다.
컴포넌트 모델 지원	Active-X 중심이다. (모든 컴포넌트 모델을 대상으로 확장중)	자바빈즈	설계 단계에서의 적용이기 때문에 특별한 제한은 없다.

4. 컴포넌트간 상호작용의 분류

본 논문에서는 컴포넌트간 상호작용을 그것이 갖고 있는 특성에 따라 크게 전체 어플리케이션과의 구조에 따른 분류와 컴포넌트와 상호작용을 이루는 다른 컴포넌트들 사이의 관계에 따른 분류로 나누었으며, 각각에 대해 다음과 같이 상호작용을 분류하였다.

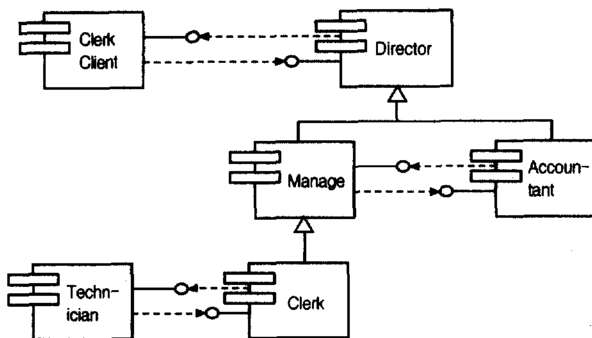
4.1 구조에 의한 분류

특정 도메인의 기능을 위해 개발된 컴포넌트는 그것을 사용한 시스템의 다른 부분이나 컴포넌트에서 사용되는 것을 전제로 개발하게 된다. 그러므로 컴포넌트는 특정한 방법과 인터페이스를 통해 다른 컴포넌트에서 참조되는 구조를 갖게 되며, 이것은 전체적으로 계층적 구조를 이루게 된다. 계층적 구조는 부분적으로 살펴볼 때, 다수의 수직적, 수평적 구조의 조합으로 볼 수 있다. 그러므로 이러한 컴포넌트간의 상호작용 역시 구조적 관점에서 다음과 같이 분류 할 수 있다.

4.1.1 수직적 상호작용

컴포넌트 기반 소프트웨어 개발과정에서는 하나의 어플리케이션 개발을 위해 다수의 컴포넌트가 협력하는 형태의 구조를 갖게 된다. 아울러 좀 더 일반적인 범위의 컴포넌트가 더 상세한 기능을 수행하는 다른 컴포넌트를 통하여 세부적인 기능을 포함하는 구조를 갖게 된다. 이러한 구조에서 두 컴포넌트 사이의 상호작용을 수직적 상호작용(vertical interaction)이라 한다. 수직적 상호작용은 의존적 관계에 기반하여 하나의 컴포넌트가 구성요소로써 내포되는 등의 관계를 유지한다. 이러한 관계에서 상위에 위치하는 컴포넌트는 자신의 인터페이스를 통해 중개자로써 제공하는 서비스를 교섭하는 역할을 맡게 된다.

예를 들어 (그림 3)과 같이 여러 계층을 가진 Mediator를 나타내는 컴포넌트 구조에서 Clerk과 Manager, Manager와 Director라는 컴포넌트는 수직관계를 맺고 있다. 패키지로 선언된 컴포넌트가 더 구체화된 컴포넌트로 확장됨을 나타낸다.



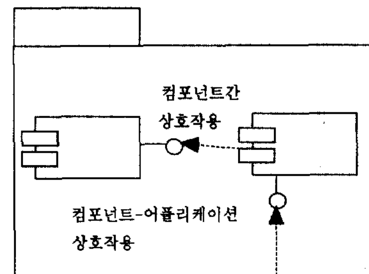
(그림 3) 수직 및 수평적 상호작용

4.1.2 수평적 상호작용

어플리케이션 개발에 사용되는 다수의 컴포넌트는 하나의 상위 모듈을 통해, 혹은 자율적으로 컴포넌트 상호간 참조를 이룰 수도 있다. 이러한 컴포넌트간의 상호작용은 일반적으로 특정한 상위모듈 혹은 중개자에 의해 동작한다. 이러한 상호작용을 수평적 상호작용(horizontal interaction)이라 한다. (그림 3)의 Clerkclient와 Clerk의 관계와 같이 수평적 상호작용은 조합 등의 방법을 통해 객체의 참조를 직접 호출하는 방법이나 메시지 전달 등의 방법을 통해 간접적으로 호출하는 방법이 사용될 수 있다.

4.2 관계에 의한 분류

컴포넌트 기반 소프트웨어 개발과정에 있어서 상호작용을 그 주체와 객체 사이의 관계로 살펴보면 다음의 두 가지로 분류할 수 있다. (그림 4)에 나타낸 것처럼 컴포넌트들끼리 인터페이스에 의하여 결합되는 경우와 어플리케이션 즉 Java 패키지 안에서 EJB와 같은 컴포넌트를 직접 사용하는 경우로 나눌 수 있다.



(그림4) 관계에 의한 분류

4.2.1 컴포넌트-컴포넌트 상호작용

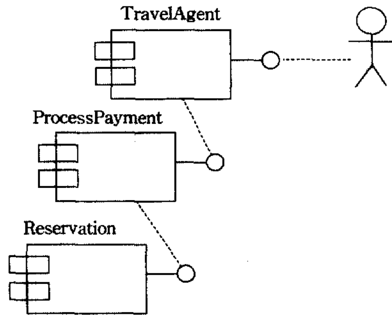
컴포넌트간의 상호작용은 같은 범주에 속하는 다수의 컴포넌트 사이의 상호작용이나, 독립적인 컴포넌트사이의 상호작용을 나타낸다. 구조에 의한 분류에서 일반적으로 수평적 상호작용을 나타내며 상위 모듈이 컴포넌트인 경우 역시 여기에 속하게 된다. 컴포넌트-컴포넌트 상호작용의 경우 소프트웨어의 내부 구조에 속하게 되며, 소프트웨어의 외부 사용자에게 인터페이스에 의해 캡슐화되어 은닉되는 특성을 갖는다.

4.2.2 컴포넌트-어플리케이션 상호작용

컴포넌트-어플리케이션 상호작용은 어플리케이션 수준에서 참조되는 컴포넌트의 상호작용을 의미한다. 이것은 구조적으로는 수직적 상호작용에 속하며, 외부의 사용자에게 공개되는 부분이기 때문에 일반적으로 인터페이스로써 서비스를 제공하는 역할을 담당하며 상호작용의 계층 구조의 최상위층을 이루게 된다.

예를 들어 (그림 5)에 나타낸 객실 예약 업무를 처리하는 컴포넌트들의 표현에서 TravelAgent 컴포넌트와 Process-

Payment 컴포넌트 사이의 상호 작용, ProcessPayment 컴포넌트와 Reservation 컴포넌트의 상호작용이 파악되어야 하며 TravelAgent 컴포넌트와 이를 사용하는 클라이언트 어플리케이션의 상호작용이 상세히 표현되어야 한다. 구체적인 인터페이스는 6장의 (그림 10)와 (그림 11)에 표현하였다.



(그림 5) 관계에 의한 상호작용 분류

5. 상호작용의 표기법

5.1 컴포넌트 상호작용의 표기법

컴포넌트 상호작용을 분류에 따라 효과적으로 표현하기 위해서 UML 표기법을 사용한다. 객체지향 설계 표현 방법의 표준이라고 할 수 있는 UML을 사용함으로써 개발자, 사용자간의 의사소통을 효과적으로 이룰 수 있으며, 재교육에 대한 비용과 노력을 절감할 수 있게 된다.

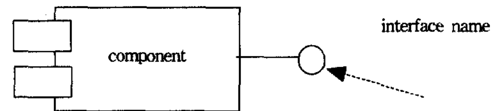
그러나 UML에서 기본적인 컴포넌트의 표기법은 소프트웨어에 적용 가능하고 실행 가능한 의미에서의 컴포넌트가 아니라, 일반적인 개념에서의 파일, 문서 등을 포괄하는 의미로 사용되며, 이는 컴포넌트 다이어그램에서도 마찬가지이다. 이러한 이유로 UML의 컴포넌트 표기는 기본적인 구조를 나타내는 목적으로 한정되어 있다. 이것을 소프트웨어 컴포넌트에 적절하게 확장하기 위해서 본 논문에서는 다음과 같은 표기법과 프로세스를 제시한다.

5.1.1 컴포넌트 및 클래스

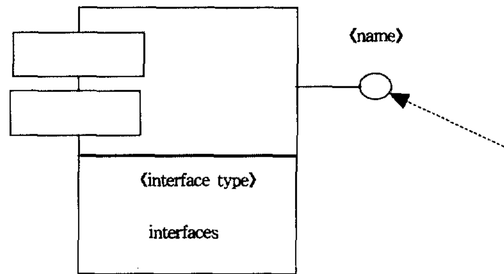
컴포넌트에 대한 기본적인 표기법은 UML 표준명세를 사용한다. 그러나 UML 명세에 나와있는 컴포넌트의 정의는 일반적인 것으로써 일반 파일이나 문서까지 포함하는 개념이라고 할 수 있다. 그러므로 본 논문에서 다루고 있는 컴포넌트와 기타의 요소들을 구분하기 위하여 컴포넌트에 대한 표기는 소프트웨어 부품으로써의 컴포넌트로 제한하며 UML을 위한 확장 기법을 사용하여 표기법을 확장한다. 이에 대한 기본 표기 방법은 [11]에 제안되어 있으며 이를 기초로 상호작용 및 인터페이스 표현 방법을 확장한다.

5.1.2 인터페이스

인터페이스는 컴포넌트의 상호작용을 기술하는 직접적인 수단이며, 이를 통해 설계자는 컴포넌트의 기능 명세를 제공하고 사용자는 컴포넌트에 대한 이해도를 높인다. 또한 인터페이스는 양자간의 독자적인 개발을 위한 기준의 역할을 담당하므로 중요성을 갖는다고 할 수 있다. (그림 6)에 나타낸 것과 같이 UML 명세에서는 인터페이스를 일반적으로 막대사탕 모양의 표기를 추가하여 나타내거나 클래스 다이어그램에 <<interface>> 스테레오타입을 추가하여 나타낸다. 결국 인터페이스의 프로퍼티는 나타낼 수 없다는 문제가 있다.

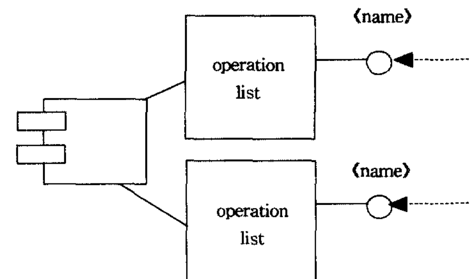


(그림 6) 일반적인 인터페이스 표기



(그림 7) 인터페이스를 위한 컴포넌트 표기의 확장 : 형태1

제안된 표기법에서 인터페이스는 각각의 컴포넌트에 포함되어 해당하는 상세한 명세를 나타내는 형태를 지닌다. (그림 7)의 경우 단순한 형태의 다이어그램으로, 컴포넌트의 인터페이스가 컴포넌트 다이어그램의 하단에 추가되는 형태를 갖는다. 이 부분에는 인터페이스에 해당하는 오퍼레이션의 목록을 적음으로써 외부와의 상호작용을 표현할 수 있다. 이 다이어그램에서 표현되는 인터페이스는 설계상의 유형으로서의 의미를 갖게 된다.



(그림 8) 인터페이스를 위한 컴포넌트 표기의 확장 - 형태2

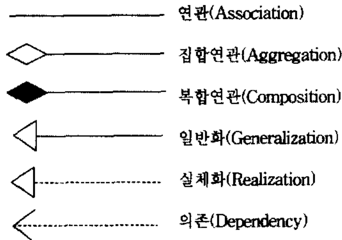
(그림 8)의 경우는 좀 더 복잡한 구조 혹은 목적으로 사용되는 컴포넌트를 위한 다이어그램이며, 다중 인터페이스를 갖는 컴포넌트를 위해 사용할 수 있다. 이것은 다수의 인터페이스 심볼을 포함하는 UML 컴포넌트에 박스를 덧붙여 각각에 해당하는 인터페이스를 표현한다. 박스 안의 목

록은 인터페이스가 제공하는 서비스의 목록을 제공한다. 각각의 인터페이스는 추상화 부분으로 볼 수 있으며 각각 참조하는 컴포넌트와의 상호작용을 표현하게 된다.

다중 인터페이스를 갖는 컴포넌트의 경우 하나의 컴포넌트 다이어그램이 다수의 인터페이스 심볼을 가질 수 있다. 이런 경우 외부 컴포넌트들이 각각의 인터페이스를 통해 독립적으로 해당 컴포넌트를 참조할 수 있다. 다중 인터페이스는 설계와 구현의 연계성을 위해서 확장표기로 표현하는 것이 좋다.

5.1.3 관계

UML에서 각 요소간의 관계는 기본적으로 화살표를 사용하여 표시한다. 컴포넌트간 상호작용에 있어서도 각 컴포넌트 사이의 관련성은 화살표를 사용하며, 상호작용의 종류에 따라 다른 방법으로 관계를 표시한다. 수직적 구조를 이루는 컴포넌트간 상호작용에서 조합의 방법을 사용한 경우에는 UML 표기법의 조합 화살표 심볼로 대체할 수 있다. 내포의 방법을 통한 컴포넌트간 상호작용의 구현은 하나의 컴포넌트 다이어그램이 다른 컴포넌트 다이어그램의 내부에 포함되는 방식으로 나타내어질 수 있다. 표현할 수 있는 화살표의 종류를 (그림 9)에서 나타내고 있다.



(그림 9) 컴포넌트간 의존성 표기법

관계에 의한 분류에서, 어플리케이션의 구성단위로 컴포넌트가 표현될 때, 컴포넌트-컴포넌트 상호작용과 컴포넌트-어플리케이션 상호작용을 표현할 수 있는데, 컴포넌트-컴포넌트 상호작용의 경우 일정 범위 내에서 컴포넌트간 참조관계를 나타내는 일반적인 표현에 따른다. 컴포넌트-어플리케이션 상호작용은 어플리케이션을 표현하는 다이어그램의 경계와 컴포넌트와의 연관성을 표기하여 그 관계를 표현한다.

5.1.4 기타 요소

컴포넌트를 사용하는 외부의 은닉된 사용자를 액터 심볼을 이용하여 나타낸다. 행위자를 표시하는 것은 컴포넌트 외부와의 상호작용을 표현하는 방법을 제시한다는 점에서 중요하다. 이것은 주로 인터페이스를 통해 나타내는 상호작용이라고 할 수 있다. 기본적인 UML 표현방법에서 행위자의 역할은 단지 시스템의 외적인 요소를 나타내는 일차적인 기능에 머물렀지만, 제안한 표기법에서는 인터페이스를 매개로 컴포넌트와 상호작용하는 모든 요소를 표현하기 위해 사용하고 있다.

또한 컴포넌트의 분류를 위해 패키지 심볼을 사용하며, 기타 구성요소로써 자료저장소나 문서의 경우 UML 표준명세에 따른다. 제시된 표기법은 일반적인 컴포넌트 표기를 확장한 것이다[9].

6. 활용사례

위에서 제시한 컴포넌트 표현방법의 유용성을 검증하기 위해 소프트웨어 컴포넌트를 설계하는 과정을 보이고, 그 과정에 반영된 표현방법상의 특징을 기술하기로 한다. 본 논문에서 사용된 소프트웨어 컴포넌트의 예는 선박여객회사의 관리 시스템의 일부로써 고객의 객실 예약과 지불과정을 담당하는 소프트웨어 컴포넌트이며 EJB(Enterprise JavaBeans)를 사용하여 구현하게 된다.

이를 위해 설계되는 컴포넌트는 객실 예약 업무를 처리하는 TravelAgent 컴포넌트와 지불과정을 담당하는 ProcessPayment 컴포넌트로 구분되며 이외에 이들이 이용하는 데이터베이스와 Java의 패키지들이 포함된다.

(그림 5)에 나타낸 TravelAgent 컴포넌트의 외부사용자는 관리 시스템을 구성하는 다른 컴포넌트가 될 수 있으며, 컴포넌트의 성격에 따라 직접적인 사용자를 의미할 수도 있다. 여기서 액터 심볼의 의미는 유효한 범위 밖에 있으면서 컴포넌트의 기능을 사용하는 구성요소를 나타내는 것이다.

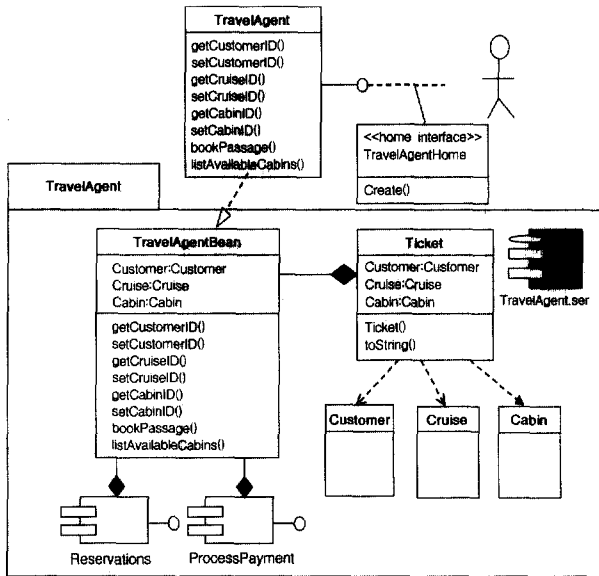
TravelAgent 컴포넌트는 사용자에게 고객ID, 선박ID, 객실ID와 가능한 객실의 목록을 제공하고 탑승권을 예약하는 기능을 제공한다. 컴포넌트 외부의 사용자와는 EJB 컨테이너를 통해 의사소통하기 때문에 이를 위한 몇 가지 요소들이 포함되어야 한다.

이 시스템에서 ProcessPayment 컴포넌트는 TravelAgent에 의해 사용되지만, 경우에 따라서 지불과정이 포함되는 모든 컴포넌트에 의해 이용될 수 있다. 이것이 소프트웨어 컴포넌트의 특징이며, EJB의 경우 jar파일로 제공되는 컴포넌트 패키지를 포함시키기만 하면 바로 사용할 수 있다.

TravelAgent 컴포넌트는 TravelAgentBean 클래스를 활성클래스로 하여 Ticket, Customer, Cruise, Cabin 등의 클래스로 구성되며 ProcessPayment, Reservations 컴포넌트 등을 사용한다.

(그림 10)은 TravelAgent 컴포넌트를 나타낸 것이다. TravelAgent.ser 파일은 EJB를 위한 Manifest 파일로써 제공되어야 하는 파일이다.

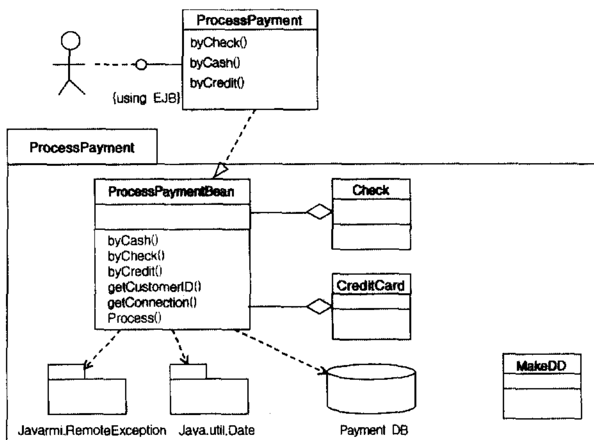
활성 클래스와 TravelAgent 인터페이스를 중심으로 다이어그램을 구현 코드로 바꾸는 것은 부록에 나열한 자바코드처럼 거의 일대일로 변환 가능하다. 다이어그램에 나타난 TravelAgent 인터페이스는 기본적으로 TravelAgentBean 클래스의 오퍼레이션으로 구현된다.



(그림 10) TravelAgent 컴포넌트 다이어그램

클래스 내부에서 참조하는 패키지나 컴포넌트는 EJB 상에서 import 예약어를 사용하여 나타나게 된다. 컴포넌트 또한 실제 코드에서는 package로 나타나게 된다. 인터페이스에 대한 참조는 EJB 컨테이너에서 담당하게 되므로 실제 코드에서는 나타나지 않게 된다. 이외에 Home 인터페이스도 소스에 포함되지만, 설계 상에는 나타나지 않는다.

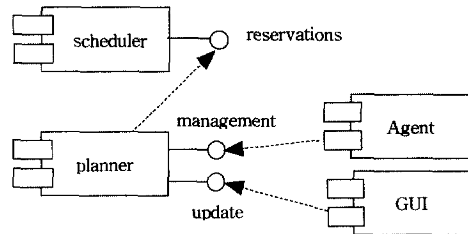
(그림 11)은 ProcessPayment 컴포넌트를 나타낸 것이다. MakeDD 클래스는 EJB 컨테이너의 파라미터를 설정하기 위한 환경설정 클래스로써 (그림 10)에 포함된 *.ser 파일로 대체할 수 있는 부분이다. (그림 12)에 나타난 외부 사용자는 이 시스템에서 TravelAgent 컴포넌트가 된다. (그림 10)을 보면 TravelAgent 컴포넌트 내에서 TravelAgentBean 클래스가 ProcessPayment를 사용하는 것을 볼 수 있다. 본 논문에서 제한한 표기방법을 통해 각각의 컴포넌트는 적절한 범위 내에서 분리되며, 액터 심볼에 의해 연결됨을 나타낼 수 있다.



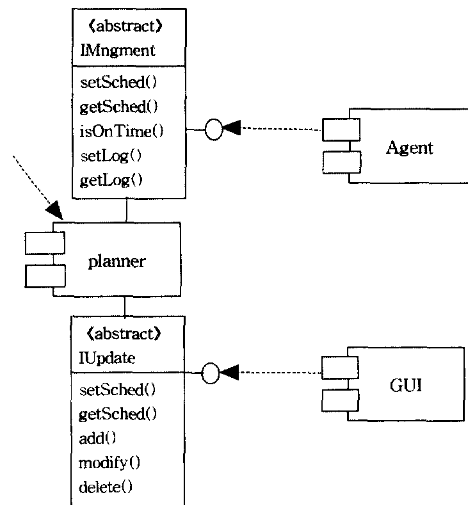
(그림 11) ProcessPayment 컴포넌트 다이어그램

ProcessPayment 컴포넌트는 ProcessPayment 인터페이스를 기본으로 제공한다. 기본적인 인터페이스 이외의 다른 서비스를 제공하기 위하여 다중 인터페이스로 확장될 수도 있다. 예를 들어 GUI를 이용하여 고객에게 직접 지불과정을 제공하기 위한 인터페이스를 만들 수도 있고, ATM이나 온라인 청구서를 위한 인터페이스를 만들 수도 있다. 이러한 다중 인터페이스를 잘 표현할 수 있도록 표현 방법을 제안하였다.

또 다른 예로 일정관리 어플리케이션의 일부분을 모델링 한 예는 (그림 12)와 같다. UML의 기본 표현 방법으로 모델링 한 후 상호작용을 반영하여 확장된 인터페이스를 표시하면 (그림 13)과 같이 된다.



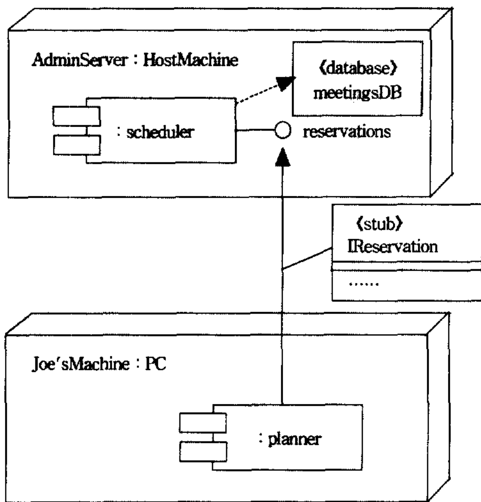
(그림 12) UML 컴포넌트 다이어그램



(그림 13) 상호작용을 고려한 컴포넌트 다이어그램의 예

(그림 12)에 포함된 인터페이스는 reservations, management, update의 세 가지 종류가 있고 planner 컴포넌트는 다중 인터페이스를 포함하고 있다. 이것을 참고하여 개괄적인 시스템의 구조는 파악할 수 있으나, 이것을 바탕으로 상호작용을 파악하기에는 정보가 부족하다. (그림 13)은 planner를 중심으로 한 상호작용의 확장된 표현방법을 보여주고 있다. 우선 management와 update 인터페이스에 대한 abstract 명세를 자세하게 보여주고 있는데, setSched()와 getSched()는 일반적인 planner 컴포넌트의 기능으로써 두 인터페이스에 모두 포함되어 있다. 그리고 management에는 inOnTime(), setLog(), getLog()가 포함되었고, undate

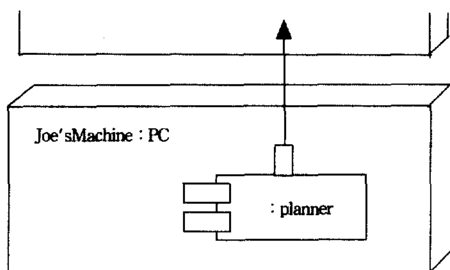
인터페이스는 add(), modify(), delete()가 포함되었다. 물론 각각의 함수들은 모두 planner 컴포넌트에 포함되어 있는 함수이다. 각각의 인터페이스의 명세를 상세히 기술해 줌으로써 설계자는 컴포넌트의 협동(collaboration)관계 등을 설계하는데 도움이 될 수 있고, 분석가 혹은 사용자의 이해도를 높이는 데 도움이 된다.



(그림 14) 분산환경에서의 표기법 사용예

또한, 기존의 UML에는 분산환경에 대한 특별한 표기방법이 제시되지 않았다. 확장된 UML 표기법에서 분산환경을 표현하기 위해서는 배치 다이어그램의 노드 심볼을 사용하여 각각의 도메인을 표시하고 원격호출이 발생하는 경우 실선 화살표로, 호출되는 서버측 컴포넌트의 인터페이스를 지시해 줌으로써 이를 나타낼 수 있다. 또한 Server/Client 간 인터페이스를 위해 원격호출에 인터페이스를 위한 box를 추가하고 《stub》 스테레오타입을 사용하여 표시할 수 있다.

원격호출임을 좀 더 확실하게 나타내고 싶은 경우에 (그림 15)와 같은 방법으로 표시하여 구분할 수도 있다.



(그림 15) 원격호출 notation의 다른 예

이러한 방법을 통해 컴포넌트간 상호작용을 좀더 상세하게 기술할 수 있으며, 이것을 통해 설계를 분명하게 할 수 있고, 다른 다이어그램으로 발전시키는 데 도움이 되며, 분석가와 사용자의 이해도를 높이는 데 도움이 된다.

7. 결론 및 향후 연구

본 논문에서는 표준적인 표현방법에 기반하여 컴포넌트 구조에 적절한 확장 표기법을 제안하고, 이를 위한 표현 프로세스를 제시하였다. 제안된 컴포넌트 상호작용에 대한 표현방법은 UML 표기법을 기반으로 하기 때문에 상호전환이 쉬우며 UML을 이용한 모델링 방법에 익숙한 소프트웨어 설계자, 제작자, 사용자가 컴포넌트 기반의 소프트웨어를 설계하고 개발하는 데 있어서 이해도를 높이고 시스템을 설계하는 데 도움을 준다. 그러나 상호작용 표현 프로세스는 가이드라인 수준으로써, 실제 적용에 있어서는 각 프로젝트나 개발 컴포넌트의 특성에 맞는 내용이 포함되어야 할 것이다.

컴포넌트 다이어그램을 이용한 컴포넌트의 상호작용에 관한 표현방법은 특정한 컴포넌트간의 상호작용을 주로 표현하기 때문에 정적인 표현방법이라고 할 수 있다. 그러나 어플리케이션 내의 컴포넌트는 동적인 환경에 기반을 두고 동작하므로 이에 대한 표현방법도 함께 제시되는 편이 더욱 효과적이다. 이를 위해서 상태 다이어그램 등을 이용한 상호작용의 표현방법이 함께 제시되어야 할 것이다.

또한, 컴포넌트 상호작용을 포함한 효율적인 모델링이 가능하기 위해선 기본이 되는 컴포넌트에 대한 표현방법이 좀 더 확장될 필요성이 있으며, 이것 역시 표준적인 표기법에 기반하여 그 표현방법을 확장하여 함께 제안되어야 효과적인 컴포넌트 모델링이 이루어질 것이다. 제안된 컴포넌트간 상호작용의 표현방법은 구현에 대한 연계방법 등에 대한 언급은 하지 않았다. 그러나 제안된 방법이 적절하게 사용되기 위해서는 표현방법과 프로세스를 자동화하기 위한 도구의 개발 또한 함께 진행되어야 할 것이다. 또한 컴포넌트 개발 과정에서 전체적으로 일관된 관점을 유지하고, 프로젝트를 조율하기 위한 통합 프로세스에 대한 연구가 진행되어야 할 것이다.

참 고 문 헌

[1] David Chappell, "The Next Wave, Component Software Enters The Mainstream," http : //www.rational.com/product/whitepaper, 1997.

[2] OMG, "OMG Unified Modeling Language Specification (Revision 1.3)," http : //www.rational.com/product/whitepaper, 1999.

[3] Grant Larsen, "Designing Component-Based Frameworks using Patterns in the UML," Communications of the ACM, pp.38-45, October, 1999.

[4] Grady Booch, James Rumbaugh, Ivar Jacobson, "The Unified Modeling Language Reference Manual," Addison-Wesley, 1998.

[5] Grady Booch, James Rumbaugh, Ivar Jacobson, "The Unified Software Development Process," Addison-Wesley, 1998.

[6] Allen Cameron Wills, "Modeling for Components with Catalysis," http://www.trireme.com, 1998.

[7] Desmond D'souza, "Component with Catalysis," http://www.catalysis.org, 1998.

[8] 최은만 외, "컴포넌트 품질평가 방안 및 매트릭스 개발", 한국전자통신연구소, 1999.

[9] 김동현 외, "UML을 이용한 자바빈즈 컴포넌트 설계", 산·학·연 소프트웨어공학기술 학술대회논문집, pp.263-268, 1999.

[10] C. Szyperski, Component Software-Beyond Object-Oriented Programming, Addison-Wesley/ACM Press, 1998.

[11] UML Components, Addison-Wesley, 2001.

부록 : TravelAgent 컴포넌트 다이어그램을 구현한 자바 프로그램

<TravelAgentBean.java>

```
package com.titan.travelagent ;

import com.titan.cruise.* ;
import com.titan.cabin.* ;
import com.titan.customer.* ;
import com.titan.reservation.* ;
import com.titan.processpayment.* ;
...
public class TravelAgentBean implements javax.ejb.SessionBean {
 public Customer customer ;
    public Cruise cruise ;
    public Cabin cabin ;

    public void ejbCreate(Customer cust){
        customer = cust ;
    }
 public int getCustomerID( )throws RemoteException{
        if(customer == null)
            throw new RemoteException() ;
        return ((CustomerPK)customer.getPrimaryKey()).id ;
    }
    public int getCruiseID( )throws RemoteException{
        ...
    }
    public int getCabinID( )throws RemoteException{
        ...
    }

    public void setCabinID(int cabinID)
        throws RemoteException, javax.ejb.FinderException{
         CabinHome home =
            (CabinHome)getHome("jndiName_CabinHome") ;
        CabinPK pk = new CabinPK() ;
        pk.id = cabinID ;
        cabin = home.findByPrimaryKey(pk) ;
    }
    public void setCruiseID(int cruiseID)
        ...
    }

    public Ticket bookPassage(CreditCard card, double price)
        throws RemoteException, IncompleteConversationalState,
         DoubleBookingException{
         try{
           ReservationHome resHome =
              (ReservationHome)
              getHome("jndiName_ReservationHome") ;
Reservation reservation =
              resHome.create(customer, cruise, cabin,price) ;
           ProcessPaymentHome ppHome =
              (ProcessPaymentHome)

getHome("jndiName_ProcessPaymentHome") ;
           ProcessPayment process = ppHome.create() ;
           process.byCredit(customer, card, price) ;

           Ticket ticket =
              new Ticket(customer, cruise, cabin, price) ;
           return ticket;
        }catch(javax.ejb.DuplicateKeyException dke){
              ...
          }
    }
    public String [] listAvailableCabins(int bedCount)
    throws RemoteException, IncompleteConversationalState{

        int cruiseID = ((CruisePK)cruise.getPrimaryKey()).id ;
        int shipID = cruise.getShipID() ;
        try {
          con = getConnection() ;
          ps = con.prepareStatement() ;
          result = ps.executeQuery() ;
          Vector vect = new Vector() ;
          while(result.next()){
               StringBuffer buf = new StringBuffer() ;
               buf.append(result.getString(1)) ;
               buf.append(result.getString(2)) ;
               buf.append(result.getString(3)) ;
               vect.addElement(buf.toString( )) ;
          }
    String [] returnArray = new String[vect.size( )] ;
          vect.copyInto(returnArray) ;
          return returnArray ;
        }
        catch (SQLException se) {
              ...
        }
    }
        ...
}
```

<TravelAgent.java>

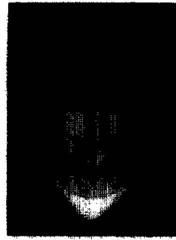
```
package com.titan.travelagent ;

import java.rmi.RemoteException ;
import javax.ejb.FinderException ;
```

```
import com.titan.cruise.Cruise ;
import com.titan.customer.Customer ;
import com.titan.processpayment.CreditCard ;
mpleteConversationalState,
                    DoubleBookingException ;

                public String [ ] listAvailableCabins(int bedCount)
            throws RemoteException, IncompleteConversationalState ;
}
public interface TravelAgent extends javax.ejb.EJBObject {
public void setCruiseID(int cruise) throws RemoteException ;
                public void setCabinID(int cabin)
                                throws RemoteException ;
                public int getCabinID() throws RemoteException ;

                public Ticket bookPassage(CreditCard card,
                        double price)
            throws RemoteException,IncompleteConversationalState,
                    DoubleBookingException;
                public int getCruiseID( ) throws RemoteException ;
                public String [ ] listAvailableCabins(int bedCount)
            throws RemoteException, IncompleteConversationalState ;
}
```



박 성 호

e-mail : shpark@softforum.com
1999년 동국대학교 컴퓨터공학과(학사)
2001년 동국대학교 컴퓨터공학과(공학석사)
2001년~현재 소프트포럼(주) 기술 연구소
관심분야 : 소프트웨어 설계 방법론, 소프트웨어 개발 프로세스, 분산 객체 시스템

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
관심분야 : 객체지향 테스트, Program Understanding, 소프트웨어 품질 메트릭, 웹 기반 소프트웨어 테스트