EJB 환경에서 컴포넌트의 Required 인터페이스 설계 기법

윤 희 윤† · 김 수 동††

요 약

다양한 정보기술의 등장과 소프트웨어 복잡성의 증가로 소프트웨어 개발 비용과 노력이 크게 증가하고 있다. 컴포넌트 기반 개발(CBD) 기술은 높은 재사용과 유지보수성을 제공하여 비용과 노력을 크게 절감하는 새로운 방법으로 각광을 받고 있다. CBD에서는 컴포넌트의 사용자가 컴포넌트를 이용해 고유의 도메인에 맞는 시스템을 구축하기 위한 컴포넌트의 가변적인 부분을 가지고 있다. 컴포넌트 내부의 가변적인 부분은 사용자가 컴포넌트에서 제공하는 기능인 Required 인터페이스를 통하여 쉽게 설정할 수 있다. Enterprise JavaBeans(EJB)은 Java 기반의 컴포넌트를 구현할 수 있는 상용 규격으로 활용되고 있다. 그러나 EJB에서는 Required 인터페이스를 직접 구현할 수 있는 장치가 제한적이다. 본 논문에서는 EJB 환경에서 Required 인터페이스를 효과적으로 정의하고 구현하기 위한 인터페이스 저장형 기법, 클래스 저장형 기법, 인터페이스 생성형 기법, Plug-in 기법을 제안한다. 인터페이스 저장형 기법은 가변적인 설정 값들은 Required 인터페이스에서 저장하고 있으며 클래스 저장형 기법은 컴포넌트 분석 모델의 수정 없이 가변성을 가진 빈들이 Required 인터페이스로 설정된다. 인터페이스 생성형 기법은 컴포넌트의 가변성을 위한 Required 인터페이스 역할의 빈을 새롭게 생성하며 Plug-in 기법은 사용자가 외부로부터 가변적인 부분을 플러그인 하여 가변성을 설정할 수 있다. 제시된 4가지 기법들은 CBD 컴포넌트의 Required 인터페이스의 의미를 보존하며 높은 품질의 컴포넌트 구현을 가능하게 한다.

Design of Required Interface for Components in EJB Environment

Hee Yoon Yoon† · Soo Dong Kim††

ABSTRACT

As new and diverse information technologies are being introduced and software complexity is increased, software development cost and efforts are also sharply increased. Component-Based Development (CBD) technology is appealing as a new way to reduce the cost and effort by increasing reusability and maintainability. Component in CBD has variability internally which enables customization of the component within the specific domain. A component user can easily set up internally variable parts through Required interface which is provided by the component. Enterprise JavaBeans (EJB) is utilized as a commercial standard to implement Java-based components. However, EJB constructs are limited in directly implementing Required interfaces of coarse-grained components[8]. In this paper, we define Required interface and propose interface-storage technique, class-storage technique, interface-generation technique, and Plug-in technique for implementing required interface of component. Interface-storage technique stores variable value in Required interface and class-storage technique take the Bean containg variability as Required interface without modification of component model. Interface-generation technique generates new Bean which takes the role of Required interface for component variability and Plug-in technique sets up component variability that component user plugged-in variable part externally. The proposed four techniques conform to the semantics of CBD component interface and enable the implementation of high quality components.

키워드 : EJB, 컴포넌트(Component), Required, 인터페이스(Interface), 구현(Implementation), CBD

1. 서 론

1.1 동기 및 배경

소프트웨어가 대형화되면서 과거의 방법으로 개발 및 유지 보수하기에 많은 비용과 시간을 요구하게 되어 소프트웨어의 위기가 초래되었다. 이런 위기의 대응책으로 소프트웨어를 기계 부품과 같이 제작하고자 하는 컴포넌트 방법론이 대두되었다. 컴포넌트는 객체지향적 소프트웨어 모듈로서 유사한 기능의 서비스 단위이며 컴포넌트 방법론이란 컴포넌트를 기반으로 하여 소프트웨어를 개발하는 방법 및 절차를 말한다. 컴포넌트는 모듈 단위로 이루어져 있어 한 번 개발된 컴포넌트는 여러 도메인에서 재사용성과 대치성을 제공함으로 소프트웨어 개발의 비용과 시간에 이익을 가져다 준다.

개발된 컴포넌트가 여러 도메인에서 사용되기 위해서는

본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.
† 준 회 원 : 숭실대학교 대학원 컴퓨터학과
†† 종신회원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2003년 10월 20일, 심사완료 : 2003년 12월 3일

여러 도메인의 공통적인 기능을 포함하고 있어야 하며 여러 도메인에서 컴포넌트를 사용할 수 있도록 설정 가능해야 한다. 고객 컴포넌트가 A회사와 B회사 소프트웨어에서 사용되기 위해서는 두 회사의 공통적인 고객의 기능을 제공해야 하며 컴포넌트가 각 회사의 특정 기능을 수행하도록 설정이 가능해야 한다. 이렇게 컴포넌트를 도메인에 따라 특정 기능을 수행하도록 컴포넌트를 변경하는 것을 컴포넌트의 가변성이라 한다. 컴포넌트가 가지는 가변성은 컴포넌트의 사용자에 의해 설정되며 사용자는 컴포넌트의 가변성을 Required 인터페이스를 통하여 설정할 수 있다.

컴포넌트의 설계는 EJB를 이용하면 보다 효과적으로 설계할 수 있다. EJB는 컴포넌트를 기반으로 하는 분산 어플리케이션 아키텍처로서 트랜잭션, 보안 및 데이터베이스 연결 등의 다양한 기능을 제공한다. 또한 컴포넌트 설계에 있어 필요한 많은 장치들을 제공해 주는데 특히 EJB 2.0에서 새로운 장치들의 추가로 컴포넌트의 구현을 보다 효과적으로 지원하고 있다.

컴포넌트를 EJB 환경에서 설계하기 위한 여러 기법들이 제시되고 있지만 컴포넌트의 가변성을 고려한 설계의 연구는 미흡한 실정이다. 본 연구에서는 EJB 환경에서 Required 인터페이스 설계 기법으로 인터페이스 저장형 기법, 클래스 저장형 기법, 인터페이스 저장형 기법, Plug-in 기법을 제안한다.

1.1.1 CBD 컴포넌트

소프트웨어 개발 시에 소프트웨어에서 필요한 기능을 제공하는 컴포넌트를 이용하여 개발하는 것을 컴포넌트 기반 개발(Component Based Developer)라고 한다[14, 15]. 일반적인 소프트웨어의 개발은 소프트웨어에서 제공하는 기능의 모든 부분을 새롭게 개발해야 했다. 이에 비해 CBD는 개발된 컴포넌트를 재사용함으로 개발 기간을 단축시킬 수 있다. 소프트웨어를 개발하기 위해서 우선 요구하는 기능을 제공하는 컴포넌트의 존재를 검색한다. 그 다음 검색된 컴포넌트들 중에서 소프트웨어에 적합한 컴포넌트를 추출한다. 추출된 컴포넌트를 조합하고 추가적인 모듈을 개발하여 소프트웨어를 개발한다.

CBD에서는 컴포넌트를 기반으로 하여 소프트웨어의 개발 생산성을 높이고 컴포넌트 단위의 높은 재사용성과 유지보수성을 제공한다. 한번 개발된 컴포넌트를 유사한 서비스를 제공하는 여러 소프트웨어서 공통적으로 사용할 수 있다. 따라서 동일한 기능에 대하여 재사용이 가능하고 중복된 개발을 막을 수 있다. 컴포넌트는 여러 도메인에 요구사항 중에서 공통적인 요구사항을 추출하여 공통 컴포넌트로 구현하게 된다[5, 7]. 그러나 각 도메인에서는 소프트웨어에 요구하는 요구사항이 동일한 기능이라 할지라도 부분적으로 다른 점을 가지고 있게 된다.

즉, 같은 컴포넌트를 이용하여 소프트웨어를 설계하더라도 도메인 특성에 따라서 컴포넌트 내부에 특별한 부분을 가지게 된다. 컴포넌트는 각 도메인에서 컴포넌트를 각 도메인의 특별한 부분에 맞추어 사용할 수 있도록 컴포넌트의 가변성을 제공한다[13]. 컴포넌트의 가변성은 컴포넌트를 사용자가 요구하는 조건으로 변경 가능하게 함으로 여러 도메인에 적합하도록 해준다. 따라서, 사용자는 가변성을 이용하여 컴포넌트를 도메인 환경에 맞추어 컴포넌트를 특화(Customization)시킬 수 있다. 컴포넌트의 가변성은 컴포넌트에서 제공하는 Required 인터페이스를 통하여 설정되며 설정된 가변성을 통하여 컴포넌트가 대상 소프트웨어에 적합하도록 특화된다[4, 6]. 컴포넌트에서 가변성과 가변성을 설정해 줄 수 있는 Required 인터페이스는 보다 범용적인 컴포넌트 개발에 있어 무엇보다 중요하다고 하겠다.

1.1.2 Enterprise JavaBeans(EJB)

EJB는 Sun에서 제공하는 컴포넌트 플랫폼이다. EJB는 분산 어플리케이션을 위한 아키텍처를 제시하며 트랜잭션과 보안과 같은 서비스를 제공한다. 따라서 EJB 개발자는 트랜잭션 처리, 데이터베이스 연동과 같은 하위 레벨 프로그램에 대하여 고려하지 않고 순수 비즈니스 로직 개발에 몰두할 수 있다. 이와 같은 서비스는 EJB 컨테이너에서 담당하게 되며 개발자는 요구되는 환경에 따라 설정만 해주면 된다. 즉, 비즈니스 로직과 하위 레벨 서비스를 분리시킴으로써 개발 생산성을 향상시키며 유지보수성을 제공한다[16, 17].

EJB의 빈은 외부 사용자의 직접적인 접근을 막고 인터페이스를 통하여 호출하도록 하고 있다. EJB에서 제공되는 인터페이스는 크게 홈 인터페이스(Home Interface)와 EJB 컴포넌트 인터페이스(Component Interface)가 있다. 홈 인터페이스는 빈 객체들의 생명 주기와 관련하여 빈 객체의 생성, 삭제를 담당하며 빈 객체들을 검색하는 기능을 제공한다. 그리고 EJB 컴포넌트 인터페이스는 특정 빈 객체에 대한 비즈니스 로직을 수행하는 기능을 제공한다[9, 10].

EJB 2.0 명세서에서는 환경에 따라 사용자 관점인 리모트 사용자 관점(Remote Client View)과 로컬 사용자 관점(Local Client View)을 제공한다. 리모트 사용자 관점은 사용자와 서버가 다른 JVM안에 배치되어 상호작용 하는 경우에 말한다. 다른 JVM안에 사용자와 서버가 배치되어 있으므로 사용자 환경은 서버의 환경에 독립적으로 존재하지만 사용자와 서버간의 상호작용 시에 특정한 값(Call by Value)의 형태로 이동한다. 반면, 로컬 사용자 관점은 사용자와 서버 동일한 JVM안에 배치되어 상호작용 하는 경우를 말한다. 사용자는 서버의 환경에 종속적이기는 하지만

상호 작용 시에 참조 값(Call by Reference)을 통하여 이루어진다. 따라서 리모트 사용자로 빈에 접근하는 것 보다는 로컬 사용자로 빈에 접근하는 것이 실행 속도의 향상과 시스템 부하를 줄일 수 있다[11, 12].

1.2 논문의 구성

본 논문에서는 2장에서 관련 연구로 EJB 환경의 컴포넌트 개발과 Catalysis의 플러그인에 대하여 기술하며 3장에서는 컴포넌트의 가변성과 인터페이스에 대하여 논의한다. 4장에서는 논문에서 제시하는 4개의 컴포넌트 Required 인터페이스 설계 모델에 대하여 제시하며 5장에서는 제시하는 설계 모델에 대한 적용 범위 및 평가를 기술한다.

2. 관련 연구

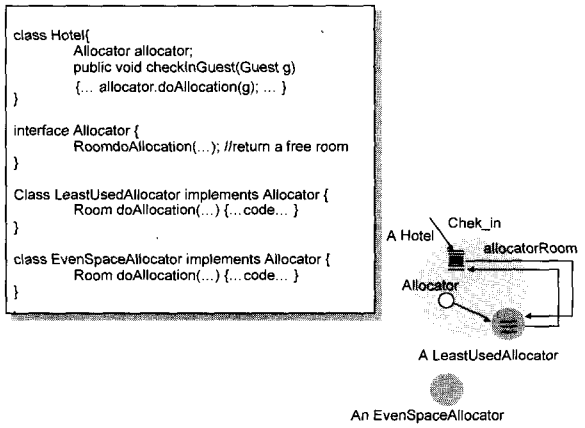
2.1 EJB 환경의 컴포넌트 개발

EJB의 기능을 이용한 컴포넌트 개발에 대하여 Matena[1]는 하나의 컴포넌트에 대하여 EJB로 구현할 때 빈의 타입과 트렌젝션, 비즈니스 및 프로세스 등을 제시하고 예제를 들어 EJB 구현에 대한 방법을 제시하고 있다. 제시된 예에서는 하나의 빈이 다른 객체 또는 헬퍼 클래스와 상호작용을 나타내며 사용자의 접근 방법을 기술하고 있다. 그러나 빈 자체의 인터페이스에 대하여 기술할 뿐 컴포넌트의 인터페이스 및 컴포넌트의 Required Interface에 대한 별다른 언급은 하지 않고 있다. 즉 컴포넌트를 EJB로 구현하는 상세한 과정은 나타내고 있지만 컴포넌트의 인터페이스의 전반적 내용 및 가변성에 관한 연구에 대해서는 다루지 않고 있다.

또한 Johannes Kuah[2]는 컴포넌트 인터페이스 명세에 대하여 정의하고 있다. 여기서는 컴포넌트 인터페이스에 대한 여러 정보들의 기술을 정의하고 있는데 이 명세서를 통하여 컴포넌트의 상세 내용과 상태를 알 수 있다. 그러나 EJB 환경에서 어떠한 기능이 구현에 적용되는 것에 대한 내용보다는 일반 컴포넌트 인터페이스의 명세로서 EJB를 기반으로 하지 않는 구현에 관한 기법 제시하고 있다. 이렇게 기존의 연구에서는 EJB 환경의 컴포넌트 구현은 빈의 인터페이스에 대하여 기술하거나 EJB 환경을 기반으로 두지 않는 컴포넌트 명세에 불과하며 가변성에 대해서는 다루어지지 못하고 있다. 따라서 EJB 환경을 기반으로 두고 있는 컴포넌트 Required 인터페이스를 개발하기 위해서는 상세한 개발 지침이 요구된다.

2.2 Catalysis의 플러그인

Catalysis에서는 컴포넌트의 재사용을 위한 플러그 기법을 제시하고 있다. 다형성(Polymorphism)과 전송(Forwarding)을 통하여 외부의 기능을 플러그 할 수 있도록 하였다. 다음 (그림 1)은 플러그 기법의 모습과 구현 소스를 나타낸다. 호텔의 체크인 시에 방을 할당하는 방법에 대하여 외부로부터 플러그인 되는 가변성을 가지고 있다. 방을 할당하는 방법으로는 제일 마지막으로 사용된 방을 할당하는(LastUsedAllocator) 방법과 균일하게 방을 할당하는(EvenSpaceAllocator) 방법을 가지고 있다. Hotel 클래스에서는 Allocator 인터페이스의 doAllocation 함수를 이용하여 체크인 할 때에 방의 할당을 수행한다. 그리고 방을 할당하는 방법을 플러그인 하기 위하여 Allocator 인터페이스를 구현해 주게 된다. LastUsedAllocator 클래스와 EvenSpaceAllocator 클래스는 Alloactor 인터페이스의 doAllocation 함수를 구현함으로써 기능을 플러그인 하도록 하고 있다.



(그림 1) Catalysis의 플러그인

3. 가변성과 Required 인터페이스

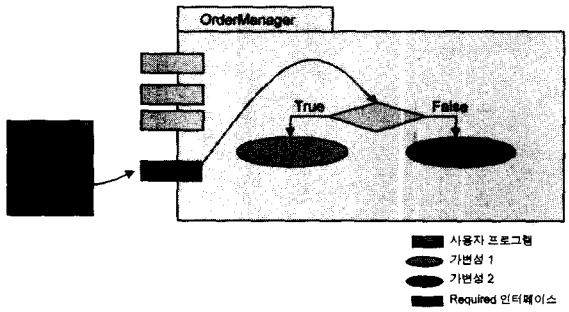
3.1 가변성 분류

컴포넌트의 가변성은 가변성의 성격에 따라 3가지 타입으로 분류할 수 있다. 이러한 3가지 타입의 컴포넌트 가변성 성격은 컴포넌트 내부에 가변성이 존재하는가에 따라 범위를 나눌 수 있다. 즉, 컴포넌트 내부의 컴포넌트가 가지는 가변성의 타입들을 내포하고 있으면 'Known Scope'에 해당되며, 컴포넌트 내부의 컴포넌트가 가지는 가변성의 타입들을 내포하지 않으며 컴포넌트 내부에서 가변성의 타입이나 종류를 파악할 수 없는 경우는 'unKnown Scope' 범위에 해당되게 된다.

3.1.1 Binary 타입

가변성의 분류에서 Binary 타입은 컴포넌트 내부에서 두 가지 경우의 가변성을 내포하고 있는 타입이다. 또한, 컴포넌트 내부에서 가변성 종류를 파악할 수 있는 Known Scope에 해당하며 두 가지의 가변적 실행 경우를 가지고 있다. 이런 Binary 타입의 가변성을 가지고 있는 컴포넌트는 가

변적 실행 경우를 True와 False로 구분하여 사용될 수 있다. 컴포넌트의 실행 시에 사용자로 하여금 가변성의 설정 값을 True 또는 False 중 하나의 값으로 얻어오게 된다. 사용자에게 얻어진 가변성의 값은 컴포넌트를 사용자 요구사항에 적합하도록 특화한다.

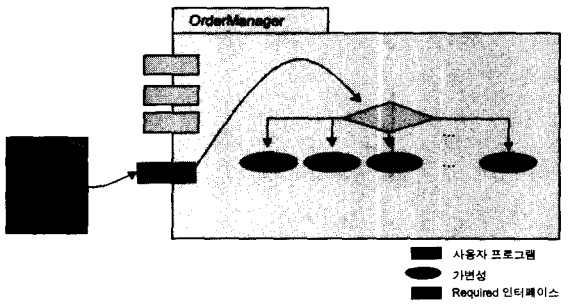


(그림 2) 가변성의 Binary 타입

3.1.2 Selection 타입

가변성 분류에서 Selection 타입은 컴포넌트 내부에 복수개의 컴포넌트 가변성의 경우를 포함하고 있는 타입이다. 즉, 컴포넌트 내부에서 가변적인 부분에 대한 경우의 수를 복수 개 가지고 있으며 사용자의 요구에 따라 선택되어 질 수 있다. Selection 타입은 컴포넌트 설계시에 복수개의 가변성 종류를 파악할 수 있는 Known Scope에 해당될 수 있다. 또한 컴포넌트 내부의 가변성의 종류를 가지고도 있지만 외부로부터 가변성의 종류를 추가로 받아 들임으로써 컴포넌트 설계시에 가변성의 종류를 파악할 수 없는 un-Known Scope에 해당하기도 한다.

Selection 타입의 컴포넌트는 컴포넌트 실행 시에, 복수개의 가변성 중에 하나를 설정하기 위하여 사용자로부터 가변성을 설정하는 값을 얻어오게 된다. 즉, 컴포넌트 내부에 여러 도메인에 적합 할 수 있는 복수개의 가변성이 존재하게 되는데, 그중 도메인 요구사항에 컴포넌트를 맞추기 하여 하나의 가변성을 선택하기 위한 가변성 설정 값을 사용자로부터 획득하게 된다. 따라서 사용자로부터 획득한 가변성 설정 값은 컴포넌트를 사용자가 요구하는 도메인에 적합하도록 특화 해준다.

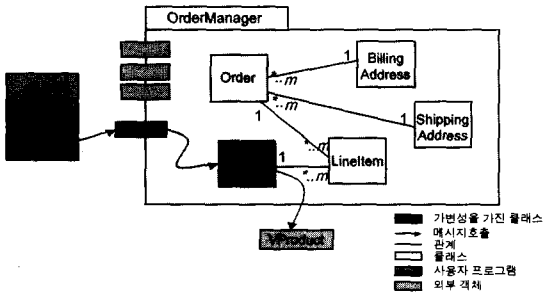


(그림 3) 가변성의 Selection 타입

3.1.3 Plug-in 타입

가변성 분류에서 Plug-in 타입은 컴포넌트 내부에서 가변성의 종류를 전혀 알지 못하고 존재하고 있지도 않은 타입이다. 따라서 컴포넌트 설계시에 가변성의 종류를 파악할 수 없는 unKnown Scope에 해당되게 된다. 이런 Plug-in 타입은 컴포넌트 내부에 가변성 경우가 존재하지 않으므로 컴포넌트 외부로부터 가변성인 부분을 플러그인하여 실행된다. 따라서, 플러그인된 컴포넌트는 사용자가 요구하는 가변성의 종류로 컴포넌트를 특화되게 된다.

Plug-in 타입의 컴포넌트는 컴포넌트 실행시에 사용자로부터 원하는 가변성을 플러그인하게 된다. 즉, 컴포넌트가 실행 시에 컴포넌트 내부에 존재하는 가변성의 종류로 실행되는 것이 아니라 사용자가 동적으로 끼어 맞춘 가변적인 부분이 실행하게 된다. 따라서 사용자는 컴포넌트를 도메인 환경에 보다 적합하도록 특화시킬 수 있다.

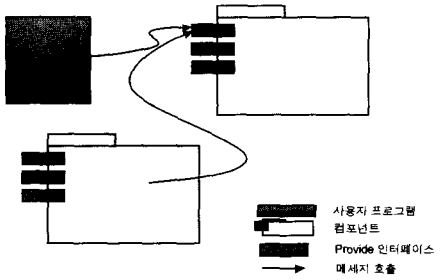


(그림 4) 가변성의 Plug-in 타입

3.2 인터페이스 분류

3.2.1 Provide 인터페이스

Provide 인터페이스는 컴포넌트의 기능을 제공하기 위한 인터페이스로 컴포넌트 외부의 사용자는 Provide 인터페이스를 통하여 컴포넌트에서 제공되는 서비스를 사용할 수 있다. 이런 인터페이스는 컴포넌트와 사용자간의 규약을 나타낸다. 즉, 컴포넌트는 인터페이스를 통하여 컴포넌트에서 제공되는 서비스를 나타내며, 사용자는 인터페이스를 통하여 컴포넌트에 요청할 서비스 항목을 파악할 수 있다. 인터페이스는 사용자에게 서비스를 제공해 주기 위한 함수들의 집합으로 Provide 인터페이스의 함수 이름 형식은 iP로 시작하여 다른 함수들과 차이를 둔다.

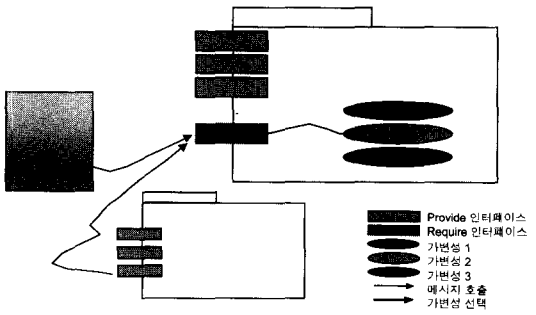


(그림 5) Provide 인터페이스

3.2.2 Required 인터페이스

유사한 도메인이라 할지라도 도메인마다 소프트웨어에 대한 요구사항이 다르며 각 도메인의 따른 약간의 차이를 가지고 있다. 컴포넌트가 다양한 도메인에서 사용되기 위해서는 컴포넌트를 각 도메인의 다양한 요구사항에 맞도록 설정할 수 있어야 한다. 즉, 도메인이 가지는 특성과 차이를 컴포넌트에 설정해 주기 위한 컴포넌트의 가변성을 가지고 있어야 한다. 이런 컴포넌트의 가변성을 설정해 주기 위하여 사용자는 컴포넌트의 인터페이스를 통하여 설정해 줄 수 있는데 이때, 사용되는 인터페이스가 Required 인터페이스이다.

컴포넌트의 사용자는 Required 인터페이스를 통하여 컴포넌트를 각각의 도메인에 적합하도록 가변적인 속성을 설정해 줄 수가 있다. 따라서, 컴포넌트의 사용자는 Required 인터페이스를 통하여 컴포넌트를 특화 할 수 있는 것이다. 이런 Required 인터페이스의 함수 이름 형식은 iR로 시작하여 다름 함수들과 차이를 둔다.



(그림 6) Required 인터페이스

4. 컴포넌트의 Required 인터페이스 설계 기법

본 논문에서는 Required 인터페이스를 설계하기 위한 4가지 설계 기법을 제시한다. 인터페이스 저장형 기법은 가변성에 대한 설정 값을 Required 인터페이스에서 저장하고 있는 기법이며 클래스 저장형 기법은 가변성을 가지고 있는 빈들이 직접 Required 인터페이스로 노출되는 기법이다. 인터페이스 생성형 기법은 Required 인터페이스 역할을 하는 새로운 빈을 생성하는 기법이며 Plug-in 기법은 외부부터 가변적인 부분을 받아들여 가변성을 설정하는 기법이다.

4.1 인터페이스 저장형 기법

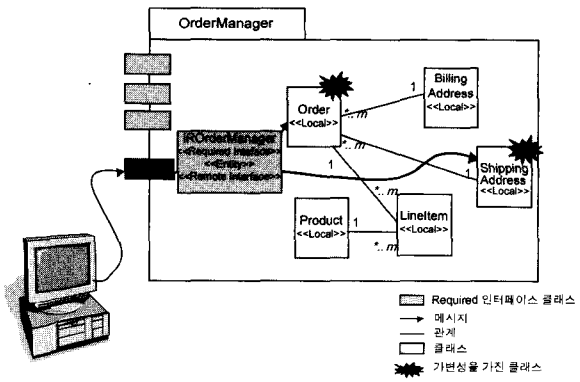
인터페이스 저장형 기법은 Know Scope 범위에 해당되는 가변성을 설계하기 위한 기법으로 Binary 타입과 Selection 타입 가변성에 적용할 수 있다. 컴포넌트는 도메인에 컴포넌트를 특화 시키기 위하여 컴포넌트의 가변성을 가지고 있으며 Know Scope 범위에서는 컴포넌트 내부의 가변성을 내포하고 있다. 이에 따라, 컴포넌트 내부의 가변성을 가지고 있는 빈이 존재하게 된다. 인터페이스 저장형 기법은 컴포넌트 내부에 가변성이 존재하고 있으며 가변성을 설정할 수 있는 빈을 새롭게 만들게 된다. 또한 새롭게 생성된 빈에서는 가변성을 설정할 수 있는 가변성 설정 값을 유지할 수 있다.

가변성을 설정하는 사용자는 생성된 빈을 사용하여 컴포넌트를 특화 시킬 수 있는데 이때, 빈은 Required 인터페이스가 되며 사용자에게 직접적으로 노출된다. 따라서, 사용자는 가변성을 가진 빈에 대하여 알지 못해도 Required 인터페이스를 통하여 가변성을 설정할 수 있다.

이와 같은 Required 인터페이스는 가변성 설정 값을 영구적으로 유지하게 되는데 EJB에서 Required 인터페이스를 엔티티 빈을 이용하여 설계한다. 사용자가 Required 인터페이스를 통하여 가변성 설정 값을 입력하면 Required 인터페이스 역할을 하는 엔티티 빈에서는 영구적으로 설정 값을 저장하고 있다. 컴포넌트 내부의 가변성을 가진 빈들의 객체가 생성될 때에는 가변성 설정 값을 필요로 하게 되는데, Required 인터페이스의 엔티티 빈에 있는 가변성 설정 값을 이용하게 된다.

이런 엔티티 빈은 컴포넌트의 사용자가 직접적으로 접근할 수 있도록 EJB에서 리모트 인터페이스로 설계한다. 또한, 컴포넌트 내부의 가변성을 가진 빈들은 로컬 인터페이스로 설계함으로 외부에서 컴포넌트의 직접적인 접근을 막는다.

다음 (그림 7)은 인터페이스 저장형 기법의 컴포넌트 설계를 나타낸다. OrdeManager 컴포넌트는 iROrderManager의 Required 인터페이스를 가지고 있으며 이 빈은 엔티티 빈과 로컬 인터페이스로 설계한다. 또한 가변성을 가진 빈은 Order 빈과 Shipping Address 빈이 존재하며 로컬 인터페이스로 설계된다.

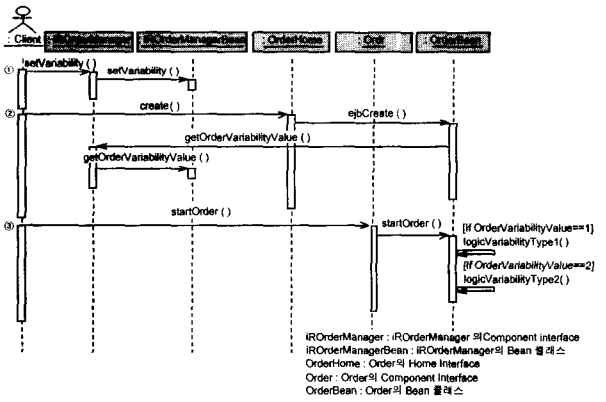


(그림 7) 인터페이스 저장형 기법

인터페이스 저장형 기법에서는 가변성 타입이 Binary인

경우에 가변성 설정 값이 True와 False의 Boolean 값을 가지며 Selection 타입인 경우 1~n의 속성값을 가진다. 이런 가변성 속성 값은 Required 인터페이스를 통하여 입력되며 영구적으로 저장된다. 사용자는 가변성을 가진 빈의 객체를 생성하기 위하여 빈의 홈 인터페이스의 Create() 메소드를 호출하게 되며 컨테이너는 ejbCreate() 메소드를 호출하게 된다. 이때, ejbCreate() 메소드 내에서 Required 인터페이스로부터 가변성 설정 값을 요청하게 되며 획득한 설정 값에 의해 가변성을 설정하게 된다.

다음 (그림 8)은 (그림 7)의 인터페이스 저장형 기법에 대한 순서도를 나타낸다. OrderManager 컴포넌트는 사용자로부터 가변성 설정 값을 입력 받으며 컴포넌트가 특화되어 실행된다. ① 항목에서는 컴포넌트의 사용자가 iROrderManager를 통하여 가변성 설정 값을 입력을 나타낸다. ② 항목에서는 가변성을 가진 Order 빈을 사용하기 위하여 객체를 생성된다. 빈 객체가 생성될 때, iROrderManager로부터 가변성 설정 값을 획득하여 OrderVariabilityValue 변수에 저장한다. ③ 항목에서는 가변성을 가진 기능인 stratOrder() 가 수행할 때, OrderVariabilityValue 변수의 값을 통하여 실행될 로직을 결정되는 것을 나타낸다. OrderVariabilityValue가 1이면 logicVariabilityType1( )를 수행하며 OrderVariabilityValue가 2이면 logicVariabilityType2( )를 수행된다.



(그림 8) 인터페이스 저장형 기법의 실행 순서도

인터페이스 저장형 기법의 특징으로는 컴포넌트 내부의 존재하는 모든 가변성에 대한 설정 값을 하나의 빈에서 관리해주며 하나의 빈이 컴포넌트의 Required 인터페이스 역할을 하는 것이다. 따라서 가변성 설정 값이 하나의 빈에 모두 모여 있으므로 가변성에 대하여 쉽게 관리해 줄 수 있으며 속성값에 대한 변경이 편리하다. 즉, Required 인터페이스를 통하여 컴포넌트 내부의 모든 가변성 설정 값을 체크할 수 있으며 가변성에 대한 파악 및 관리가 가능해진다.

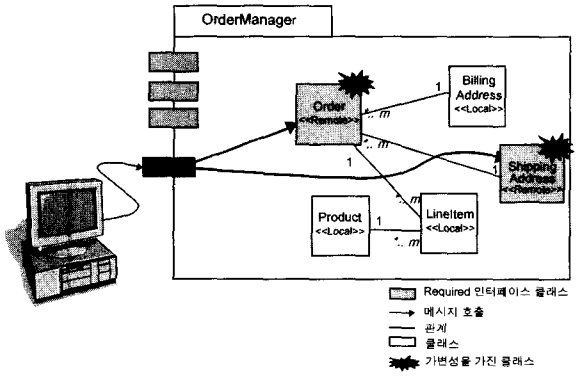
이 기법이 가지는 단점으로는 객체지향에서 클래스는 클래스가 가지는 고유한 속성들을 모두 포함하고 있어야 한다는 지침을 지키지 못하고 있다. 가변성에 대한 설정 값들이 모두 하나의 Required 인터페이스에 모여 있기 때문이다. 그렇기 때문에 가변성에 대한 관리는 용이하겠지만 각 빈에 대한 관리와 유지보수가 어려워질 수 있다.

4.2 클래스 저장형 기법

클래스 저장형 기법은 인터페이스 생성형 기법과 마찬가지로 Known Scope 범위에 해당되는 가변성을 설계하기 위한 기법으로 Binary 타입과 Selection 타입 가변성에 적용할 수 있다. 인터페이스 생성형 기법은 Required 인터페이스 역할을 하는 빈을 생성하고 가변성 설정 값을 저장하는 반면 클래스 저장형 기법은 가변성 설정 값을 가변성을 가진 빈에서 가지고 있다. 또한 가변성을 가지고 있는 빈들이 컴포넌트의 Required 인터페이스로 직접 노출된다.

각 빈에서는 가변성 설정 값을 유지하고 있으며 설정 값을 이용하여 컴포넌트를 특화하게 된다. 이런 가변성 설정 값을 유지하기 위해서 빈에 static 변수나 데이터베이스를 이용한다. 즉, 가변성을 가진 빈이 세션 빈일 경우에는 static 변수를 이용하여 각 빈 클래스의 가변성 설정 값을 지속적으로 유지하며 앤티티 빈일 경우에는 데이터베이스를 이용하여 가변성 설정 값을 유지한다.

클래스 저장형 기법에는 가변성을 가지고 있는 빈들을 Required 인터페이스로 노출하기 위하여 EJB의 리모트 인터페이스로 선언한다. 따라서, 리모트 인터페이스로 선언된 빈들은 컴포넌트의 외부 사용자들이 직접적으로 접근할 수 있다. 또한 그 밖의 외부 사용자와 직접적인 상호작용이 없는 빈들은 로컬 인터페이스로 선언함으로 외부에서의 접근을 막는다.



(그림 9) 클래스 저장형 기법

(그림 9)는 클래스 저장형 기법의 컴포넌트 설계를 나타낸다. Order 빈과 Shipping Address 빈에서는 가변성을 가지고 있으며 컴포넌트의 Required 인터페이스의 기능을 수

행하게 된다. 따라서 컴포넌트 외부로 받은 가변성 설정 값을 유지하고 있으며 EJB의 리모트 인터페이스로 설계된다.

클래스 저장형 기법은 설계된 컴포넌트 모델에서 컴포넌트의 가변성을 가진 빈을 Required 인터페이스로 설정하게 된다. 따라서 컴포넌트 모델의 변경 없이 Required 인터페이스를 설계할 수 있다는 장점을 가지고 있다. 따라서, EJB 환경에서의 컴포넌트 설계시에, 별도의 빈 생성이나 모델의 변화를 요구하지 않는다. 또한 가변성을 가지고 있는 빈에서 가변성 설정 값을 유지하게 됨으로 클래스는 클래스의 고유한 속성들을 포함하고 있어야 하는 객체지향 지침을 유지할 수 있다.

그러나 가변성을 가지고 있는 빈이 늘어날수록 Required 인터페이스 역할을 하는 빈이 늘어나게 된다. 따라서 외부의 컴포넌트 사용자는 Required 인터페이스를 사용하기 위하여 여러 빈에 대한 정보를 유지해야 하는 단점이 있다. 또한 Required 인터페이스 역할을 하는 빈들이 가지고 있는 오퍼레이션이 컴포넌트 외부로 노출되게 된다. 그러므로 컴포넌트의 내부 기능이 컴포넌트 외부로 노출될 수 있는 단점을 가지게 된다.

4.3 인터페이스 생성형 기법

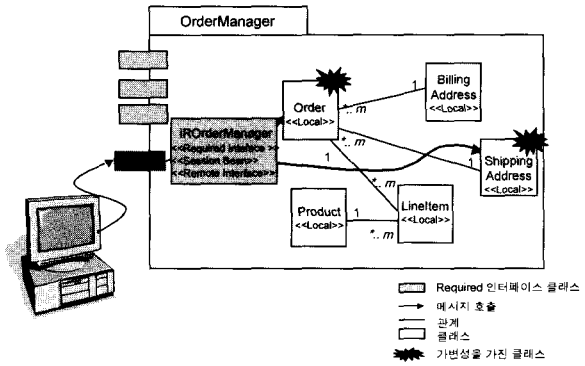
인터페이스 생성형 기법은 인터페이스 저장형 기법과 유사하게 Know Scope 범위에 해당되는 가변성을 설계하기 위한 기법이며 Binary 타입과 Selection 타입 가변성에 적용할 수 있다. 인터페이스 저장형 기법에서는 컴포넌트 내부에 가변성을 포함하고 있으며 Required 인터페이스 역할을 하는 빈을 생성하게 된다. 그리고 가변성의 설정 값은 가변성을 가지고 있는 빈에서 나눠서 저장하게 된다.

가변성에 대한 설정 값은 컴포넌트의 사용자가 Required 인터페이스를 통하여 설정하게 된다. Required 인터페이스를 통하여 설정된 값들은 가변성을 가진 빈에게 전달되어 지속적으로 유지된다. 따라서 Required 인터페이스는 가변성을 가지고 있는 빈들에 대하여 컨트롤러 역할을 하게 되고 컴포넌트 외부에서는 가변성을 가진 빈에 대한 정보 없이 Required 인터페이스의 정보만으로 컴포넌트를 특화하게 된다.

인터페이스 생성형 기법에서는 Required 인터페이스를 설계하기 위하여 Required 인터페이스 역할의 빈을 생성하고 이 빈을 EJB의 세션 빈으로 설정한다. 이 빈에서는 특정 값에 대하여 영구적으로 유지할 필요도 없으며 가변성을 가진 빈에 가변성 설정 값을 전달해 주는 역할을 하게 된다. 또한 컴포넌트 내부에서 가변성을 가진 빈들은 클래스 저장형 기법과 유사하게 가변성 설정 값을 유지하게 된다. 즉, 세션 빈일 경우에는 static 변수를 이용하여 각 빈에 속하는 가변성의 설정 값을 유지할 수 있으며 엔티티 빈일 경우 데이터베이스를 이용하여 가변성의 설정 값을 영구적으로 유지할 수 있다.

인터페이스 생성형 기법을 EJB 환경에서 설계하기 위해서는 새롭게 생성된 Required 인터페이스를 EJB의 리모트 인터페이스로 선언하여 컴포넌트의 사용자가 접근할 수 있도록 한다. 또한 그 외 컴포넌트 내부 빈들은 로컬 인터페이스로 선언함으로 외부에서 컴포넌트의 직접적인 접근을 막도록 한다.

(그림 10)은 인터페이스 생성형 기법의 컴포넌트 설계를 나타낸다. Order 빈과 Shipping Address 빈은 가변성을 가지고 있으며 가변성에 대한 설정 값을 유지하고 있다. 두 가변성을 설정하기 위해서 Required 인터페이스인 iROrder Manager가 존재하게 있으며 이 빈은 EJB의 리모트 인터페이스를 가진 세션 빈으로 설계된다.



(그림 10) 인터페이스 생성형 기법

인터페이스 생성형 기법은 인터페이스 저장형 기법과 유사하게 하나의 Required 인터페이스를 가지며 이를 통하여 컴포넌트를 특화시킬 수 있다. 또한 각각의 가변성을 가진 빈에서 가변성에 대한 설정 값을 유지하고 있다. 따라서 가변성에 대한 유지 보수가 간편하며 컴포넌트 외부의 사용자가 가변성을 가진 빈들에 대한 정보를 유지할 필요 없이 하나의 Required 인터페이스를 통하여 가변성을 설정 할 수 있다. 그러나 이 기법은 Required 인터페이스 역할을 하는 빈을 새롭게 모델링 해야 하며 이에 따른 컴포넌트 모델의 수정이 필요하게 된다.

4.4 Plug-in 기법

Plug-in 기법은 컴포넌트의 사용자가 가변성에 플러그인 되는 부분을 Required 인터페이스를 통하여 설정하는 기법이다. 이 기법은 Unknown Scope 범위에 해당되는 가변성을 설계하기 위한 기법이며 Plug-in 타입의 가변성에 적용할 수 있다. Plug-in 기법에서는 컴포넌트 내부에 가변성을 포함하고 있지 않으며 컴포넌트 외부로부터 가변성 부분을 끼워 맞추게 된다.

컴포넌트의 사용자는 플러그인 되는 부분에 대한 빈 정보를 획득하고 Required 인터페이스를 통하여 획득한 빈 정보를 넘겨준다. 넘겨진 빈의 정보는 가변성에 대치되는 부분에 대한 위치설정 값으로 Required 인터페이스에서 지속적으로 유지하게 된다. 가변성을 가진 빈이 실행될 때, Required 인터페이스로부터 위치 설정 값을 가져오게 된다. 그리고 가변성이 실행되면 가져온 빈의 위치 정보를 이용하여 플러그인되는 외부 빈을 호출하게 된다. 이때, 가변성을 가진 함수의 매개변수를 호출되는 빈에 매개변수로 넘겨준다. 따라서 플러그인되는 빈의 매개 변수 값을 이용하여 가변성의 기능을 수행 할 수 있다.

가변성 기능을 가지고 있는 함수의 수행 방법은 두 가지로 나눌 수 있다. 첫 번째 경우, 함수가 수행될 때, 매개변수로 받은 값만을 이용하여 로직이 수행할 경우이다. 예를 들어 간단하게 환전을 계산하는 함수라고 하면, 매개변수로 환율 비율과 금액을 입력 받아서 로직이 수행된다. 두 번째 경우, 함수가 수행될 때에 매개변수로 받은 값과 클래스의 변수를 사용하여 로직을 수행하는 경우이다. 즉, 함수 내부에서 매개변수 외의 다른 변수를 사용하는 경우이다.

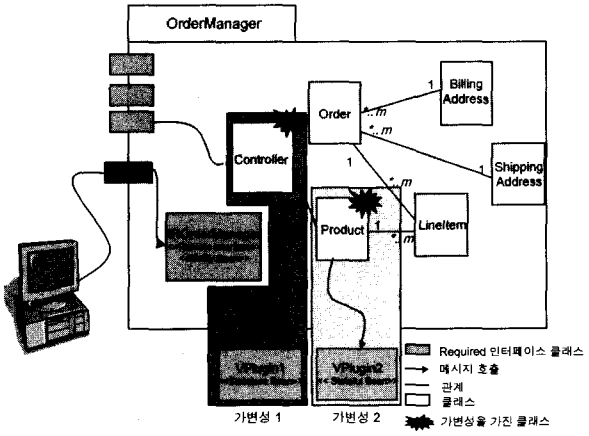
4.4.1 매개변수만을 사용하는 Plug-in 기법

첫 번째의 경우인 가변성이 실행될 때 매개 변수로 받은 값만을 이용하여 로직이 수행하는 경우에는 대치되는 빈에서 가변성을 가진 빈의 정보를 사용하지 않는다. 따라서 가변성에 대치되는 곳에서는 가변성을 가지고 있는 곳을 참조하거나 호출할 필요가 없게 된다. 즉, 가변성을 가지고 있는 곳에서는 대치되는 곳의 정보를 유지할 필요가 있지만 가변성에 대치되는 곳에서는 가변성을 가지고 있는 곳의 정보를 유지할 필요가 없는 단방향성이다.

EJB 환경에서 이와 같은 기법의 설계는 가변성에 대치되는 빈을 세션 빈으로 설계하며 스테이트리스(Stateless Session Bean) 세션 빈으로 설정한다. 이런 경우는 가변성 기능의 함수가 실행되면서 매개변수로 획득한 값만을 이용하여 로직이 수행되며 다른 특정 객체에 종속적이지 않으므로 스테이트리스 세션 빈의 성격을 가진다.

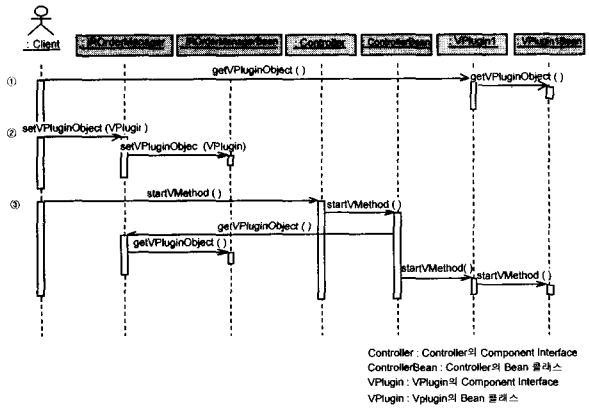
다음 (그림 11)은 Plug-in 기법의 OrderManager 컴포넌트를 나타낸다. 이 컴포넌트는 iROrderManager의 Required 인터페이스를 가지며 사용자로부터 입력 받은 빈의 위치 정보를 영구적으로 저장하게 된다. OrderManager 컴포넌트는 Controller 빈과 Product 빈에서 가변성을 가지고 있는데 가변성은 외부의 VPlugin1과 VPlugin2 빈으로 대치된다.

Required 인터페이스 역할의 빈은 사용자로부터 입력 받은 위치 정보를 영구적으로 저장하기 위하여 엔티티 빈으로 설계되다. 또한 외부 사용자들에게 노출되기 위하여 리모트 인터페이스를 가진다. Required 인터페이스에서 저장하고 있는 위치 정보는 EJB 빈을 찾아오기 위한 핸들이나 JNDI가 된다. 컴포넌트의 사용자는 가변성을 설정해 주기 위하여 플러그인 되는 빈의 위치 정보를 getEJBObject( ) 함수를 이용하거나 JNDI-name을 통하여 얻어오게 된다. 컴포넌트의 가변성이 실행될 때, Required 인터페이스에 저장되어 있는 플러그인 되는 빈의 위치 정보를 이용하여 플러그인 빈을 찾는다. 다음으로 플러그인 빈의 가변성 함수를 호출하도록 함으로 사용자가 지정하는 외부 함수가 실행되도록 한다.



(그림 11) Plug-in 기법

(그림 12)는 (그림 11)의 Plug-in 기법 모델에서 가변성을 가진 함수가 실행될 때의 수행과정을 보여주는 순서도이다. ① 항목에서는 사용자가 컴포넌트의 가변성을 설정하기 위하여 플러그인 되는 빈의 정보를 얻어온다. ② 항목은 Required 인터페이스인 IROrderManager를 통하여 가변성을 설정하는 것을 나타내며 ③ 항목에서는 가변성을 가진 빈의 가변성 함수가 Required 인터페이스로부터 위치 정보를 얻어와서 외부의 객체를 실행하는 과정을 나타낸다.



(그림 12) 매개 변수만을 사용하는 Plug-in 기법 실행 순서도

(그림 12)의 순차도를 이용하여 동적으로 컴포넌트 외부에

서 가변성을 플러그인 하는 과정을 설계할 수 있다. 다음 (그림 13)은 가변성을 가진 빈에서 플러그인 되는 빈을 호출하여 가변성을 실행하는 소스를 나타낸다.

```
import javax.ejb.CreateException ;
import javax.ejb.SessionBean ;
import javax.ejb.SessionContext ;
import javax.naming.InitialContext ;
import javax.naming.NamingException ;

public class ControllerBean implements SessionBean {
  ...

// 가변성을 가진 메소드
startOrder(shippingCart, int n) {

    Handle handle;

    // Required Interface로부터 획득한 Plug-in 되는 객체의 핸들을 저장
    handle=iROrderManager.getPluginEJBObject( ) ;

    // Plug-in 되는 메소드 핸들로 부터 객체을 얻어옴
    Variability variability = (Variability)handle.getEJBObject( ) ;

    // Plug-in 되는 메소드 호출
    // 가변성을 가진 메소드에서 받은 아규먼트를 넘겨줌
    variability.startOrder(shippingCart, int n) ;
    ...
}
```

(그림 13) Plug-in 기법에서 Plug-in 되는 빈의 소스

(그림 13)에서 ControllerBean의 startOrder( ) 함수는 Plug-in 가변성을 가지고 있다. 이 함수 내에서 동적으로 가변성을 실행시킬 수 있는데 이때, Required 인터페이스인 iROrderManager 빈으로부터 플러그인 되는 빈의 위치 정보를 얻어온다. 그 다음, 얻어온 빈의 위치 정보를 이용하여 플러그인 되는 가변성 함수를 호출함으로 가변성을 동적으로 실행시킬 수 있다.

4.4.2 클래스 변수를 사용하는 Plug-in 기법

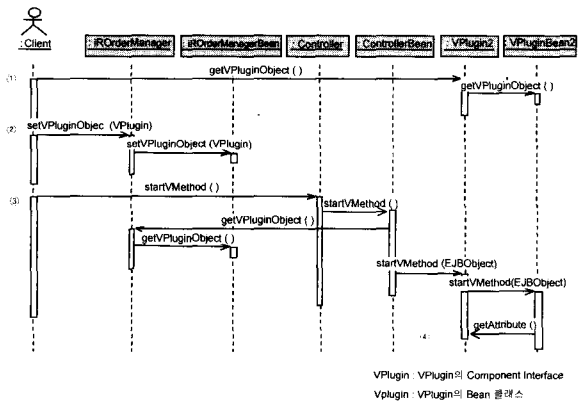
오퍼레이션이 수행될 때에 매개변수로 받은 값과 빈의 에트리뷰트를 사용하여 로직이 수행되는 두 번째의 경우에는 대치되는 빈에서 가변성을 가진 빈의 정보를 사용하게 된다. 따라서 가변성을 가지는 빈과 대치되는 빈에서 서로의 위치 정보를 가지고 서로 호출 및 속성 값을 가져 올 수가 있게 된다. 즉, 가변성을 가지고 있는 곳과 대치되는 곳에서 서로의 정보를 유지함으로 양방향성이 된다.

EJB 환경에서 이와 같은 기법의 설계는 가변성을 가지고 있는 빈을 엔티티 빈 또는 세션 빈으로 하며 플러그인 되는 빈은 세션 빈의 스테이트리스 빈이나 엔티티 빈으로 설계한다. 가변성이 플러그인 되어서 실행되는 동안 빈이 가지는 에트리뷰트를 사용하게 되는데 플러그인되는 빈과 에트리뷰트가 있는 빈이 다르게 된다. 따라서 서로 필요한 값을 호출하고 양방향 상호작용을 위하여 두 빈은 서로의 위치 정보를 유지하고 있어야 한다.

두 빈에서 서로의 위치 정보를 유지하기 위해서는 가변성이 실행될 때에, 매개변수 값과 함께 가변성이 있는 빈의 위치 정보도 함께 넘겨준다. 그러면 플러그인되는 빈에서는 위치 정보를 이용하여 빈의 에트리뷰트를 사용할 수가 있다. 따라서 가변성을 가진 빈은 플러그인 되는 빈을 호출할 수 있으며 플러그인 되는 빈은 가변성을 가지는 빈을 호출할 수 있으므로 양방향 호출이 가능해 진다.

컴포넌트의 사용자는 플러그인 가변성을 설정해 주기 위해서 플러그인 되는 빈의 위치 정보를 얻어오고 컴포넌트의 Required 인터페이스를 통하여 위치 정보를 설정해 준다. 가변성을 가진 함수가 실행될 때, Required 인터페이스에서 저장하고 있는 플러그인되는 빈의 위치정보를 획득하게 되며 그 정보를 이용하여 플러그인 되는 빈을 동적으로 실행시킬 수가 있다.

EJB 환경에서 플러그인 되는 빈의 설계는 세션 빈으로 설정한다. 플러그인 되는 빈에서는 특정한 속성값을 영구적으로 유지할 필요가 없이 가변성을 가진 빈의 에트리뷰트를 이용하여 실행하게 된다. 따라서 특정한 값을 데이터베이스에 영구적으로 저장할 필요가 없게 된다. 또한 플러그인 되는 빈에서는 가변성을 가진 빈의 위치 정보를 저장하고 있으며 두 빈간의 상태 정보를 유지하고 있어야 함으로 스테이트풀(Stateful) 세션 빈으로 설정한다.



(그림 14) 클래스의 변수를 사용하는 Plug-in 기법 실행

위 (그림 14)는 (그림 11)의 컴포넌트 모델에 있는 가변성의 특화와 실행 과정에 대한 순차도이다. ① 항목에서는 사용자가 컴포넌트의 가변성을 설정하기 위하여 플러그인 되는 빈의 정보를 얻어온다. ② 항목은 Required 인터페이스인 IROrderManager를 통하여 가변성을 설정하는 것을 나타내며 ③ 항목에서는 가변성을 가진 빈의 가변성 함수가 Required 인터페이스로부터 위치 정보를 얻어와서 외부의

객체를 호출하게 된다. 이때 매개변수로 가변성이 있는 빈의 주소 정보인 EJBObject도 함께 넘겨준다. ④ 항목에서는 VPluginBean2의 빈이 실행되면서 빈의 에틀리뷰트가 필요한 경우, 주소정보를 이용하여 가변성을 가진 빈을 호출하게 된다.

5. 적용 범위 및 평가

5.1 가변성 적용 범위

본 논문에서는 가변성을 설정하기 위한 Required 인터페이스의 기법들을 제시하였다. 다음 <표 1>은 가변성 분류에 따른 적용 가능한 기법에 대한 분류이다.

〈표 1〉 가변성 분류에 따른 적용 가능 기법

가변성 분류	Binary 타입	Selection 타입	Plug-in 타입
적용 가능 기법	인터페이스 저장형 기법 클래스 저장형 기법 인터페이스 생성형 기법	인터페이스 저장형 기법 클래스 저장형 기법 인터페이스 생성형 기법	Plug-in 기법

Binary 타입에서는 인터페이스 저장형 기법, 클래스 저장형 기법과 인터페이스 생성형 기법이 적용가능 하다. 인터페이스 저장형 기법에서는 인터페이스에서 가변성에 대한 두 가지 경우에 대한 가변성의 값을 저장하고, 클래스 저장형 기법과 인터페이스 생성형 기법에서는 빈에서 가변성의 대한 두 가지 경우에 대한 설정 값을 저장하고 있으므로 가변성의 대한 설정이 가능하다.

Selection 타입에서는 인터페이스 저장형 기법, 클래스 저장형 기법과 인터페이스 생성형 기법이 적용가능 하다. 인터페이스 저장형 기법에서는 인터페이스에서 가변적의 다양한 경우데 대하여 설정할 수 있는 가변성 설정 값을 저장하고, 클래스 저장형 기법과 인터페이스 생성형 기법에서는 빈에서 가변성의 대한 다양한 경우에 대하여 설정 할 수 있는 가변성 설정 값을 저장하고 있으므로 가변성의 대한 특화가 가능하다.

Plug-in 타입에서는 다른 타입들과 다르게 컴포넌트 외부로부터 가변성에 대한 설정을 플러그인 되어야 한다. 그러므로 가변성을 설정할 수 있는 Required 인터페이스가 다른 타입과는 다른 조건을 요구한다. 본 논문에서 제시된 Plug-in 기법에서는 이런 Plug-in 타입의 Required 인터페이스 구현 조건을 만족한다. 따라서 Plug-in 타입은 Plug-in 기법에 의해서 적용 가능하다.

5.2 제시된 기법 비교

아래 <표 2>는 컴포넌트를 평가 하기 위한 평가 기준에 의해 각 기법들을 비교 평가한다.

〈표 2〉 제시된 기법들 간의 비교 평가

	인터페이스 저장형 기법	클래스 저장형 기법	인터페이스 생성형 기법
효 율 성	○	●	○
유지보수성	○	○	●
생 산 성	▲	●	○
재사용성	●	▲	○
Black-box	●	○	●
인터페이스지원	●	○	●

▲ : 약함 ○ : 보통 ● : 최적

인터페이스 저장형 기법은 컴포넌트의 인터페이스 역할을 하는 새로운 빈을 가지며 컴포넌트의 서비스를 처리해 주기 위해서 인터페이스가 해당 클래스로 메시지를 호출해 주는 작업이 필요하다. 이에 따라 직접적으로 해당 클래스에서 메시지 처리가 되는 것에 비해 효율성 면에서는 보통이다. 또한, 인터페이스 역할을 하는 한 개의 빈이 별도로 유지됨으로 인터페이스 측면에서 유지보수성이 뛰어나지만 가변성이 해당클래스에 존재하는 것이 아니라 하나의 빈에 모아져 있기 때문에 가변성의 유지보수 측면에서는 약하므로 유지보수성에 대하여 보통이다. 생산성 측면에서는 별도의 인터페이스 빈을 생성과 가변성에 대한 관리의 필요로 다른 기법에 비하여 많은 시간과 노력을 필요로 한다.

반면, 컴포넌트의 인터페이스가 별도로 있기 때문에, 컴포넌트 인터페이스를 재사용할 수 있으며, 가변성 또한 한 빈에 있기 때문에 재사용이 가능함으로 재사용 측면에서 최적이라 할 수 있다. 이런 인터페이스 저장형 기법은 별도로 생성된 한 개의 인터페이스를 가짐으로 컴포넌트 내부를 숨길 수 있기 때문에 Black-Box와 인터페이스 지원에서 최적이다.

클래스 저장형 기법은 컴포넌트의 인터페이스가 외부와 직접적인 상호작용이 있는 빈으로 추출됨으로 별도의 인터페이스를 가지지 않고 외부로부터의 호출이 해당되는 클래스에 직접 전달 된다. 그렇기 때문에 다른 기법과 다르게 중간 과정을 거치지 않고 서비스를 처리할 수 있으므로 높은 효율성을 나타낸다. 컴포넌트의 유지보수성에 대한 측면에서 이 기법은 가변성에 대한 값을 해당 클래스 내에 존재함으로 가변성의 유지보수 측면에서 뛰어나지만 별도의 인터페이스 역할을 하는 빈이 없으므로 유지보수성에 대하여 보통이다. 생산성 측면에서는 인터페이스를 추출하기 위한 별도의 모델링의 변화를 요구하지 않으며 가변성이 해당 클래스 내부에 있으므로 시간과 노력을 단축시킬 수 있으므로 최적이다.

또한, 이 기법은 컴포넌트의 인터페이스를 별도로 가지고 있지 않기 때문에, 컴포넌트의 인터페이스의 재사용 측면이

떨어지며 또한 컴포넌트의 가변성이 해당 객체들로 분산되어 있기 때문에 가변성을 재사용 하기 위해서는 노력이 필요함으로 컴포넌트의 재사용성에 있어 다른 기법에 비해 약하다. 또한, 컴포넌트의 인터페이스를 별도의 한 개의 빈에서 지원해 주는 것이 아니라 외부와 상호작용을 가진 빈들이 인터페이스로 설정되면서 외부로 노출된다. 그러므로 컴포너트의 Black-Box와 인터페이스 지원이 다른 기법에 비해 부족하다.

인터페이스 생성형 기법은 인터페이스 저장형 기법과 같이 컴포넌트의 인터페이스 역할을 하는 새로운 빈을 가짐에 따라 효율성 측면에서 직접적인 해당 클래스 호출하는 방법에 비교하면 보통 수준이다. 또한, 인터페이스 역할을 하는 한 개의 빈이 별도로 유지됨으로 인터페이스 측면에서 유지보수성이 뛰어나며 가변성이 해당 클래스에 존재함으로 유지보수성에 최적이다. 생산성 측면에서는 별로의 인터페이스 빈의 생성에 대한 모델링의 변화 등의 시간과 노력이 필요하지만 가변성에 대하여 해당 클래스에 존재하게 인터페이스 저장형 기법에 비해 빠른 생산성을 보인다.

인터페이스 생성형 기법은 별도의 인터페이스가 존재하기 대문에, 컴포넌트의 인터페이스를 다른 컴포넌트에 재사용 할 수 있으나 컴포넌트의 가변성을 재사용하기 위해서는 노력이 필요함으로 재사용성에 있어 보통이다. 그러나, 컴포넌트의 인터페이스가 별도의 한 개의 빈에서 관리됨에 따라 컴포넌트 내부 클래스를 숨길 수 있으며 컴포넌트의 인터페이스를 보다 정확히 지원해 줄 수 있으므로 Black-Box와 인터페이스 지원에서 최적이다.

6. 맺음말(Conclusion)

새로운 기술의 개발과 소프트웨어가 거대해짐에 따라서 소프트웨어 시스템의 개발 비용과 노력이 크게 증가하고 있다. 컴포넌트 기반 개발 기술은 컴포넌트를 이용하여 높은 재사용성과 유지보수성을 제공함으로 시스템의 개발 비용과 노력을 크게 절감하는 새로운 방법으로 대두되고 있다. 컴포넌트의 사용자는 컴포넌트를 구입하여 고유 도메인에 적합한 시스템을 구축하게 되는데, 이때 각각의 도메인이 가지는 가변적인 부분을 컴포넌트 내부의 가변성으로 설정해 줌으로서 고유 도메인에 맞는 시스템을 구축하게 된다. 이런 컴포넌트 내부의 가변적인 부분을 사용자가 컴포넌트에서 제공하는 Required 인터페이스를 통하여 쉽게 설정할 수 있다. EJB는 Java 기반의 컴포넌트를 구현할 수 있는 사용 규격으로 활용되고 있다. 그러나 EJB 규격에서 컴포넌트 단위의 Required 인터페이스를 직접적으로 구현할 수 있는 장치가 제한적이다.

본 논문에서는 EJB 환경에서 컴포넌트의 Required 인터페이스를 효과적으로 정의하고 구현하기 위한 4가지 기법인 인터페이스 저장형 기법, 클래스 저장형 기법, 인터페이스 생성형 기법 그리고 Plug-in 기법을 제시하고 있다. 인터페이스 저장형 기법은 컴포넌트의 Required 인터페이스 역할을 하는 새로운 빈을 가지며 인터페이스에서 가변성의 설정값을 저장함으로써 인터페이스를 구현하며, 클래스 저장형 기법은 컴포넌트 분석 모델의 수정 없이 가변성을 가지고 있는 빈들을 직접 Required Interface로 설정함으로써 인터페이스를 구현한다. 또한 인터페이스 생성형 기법은 컴포넌트의 가변성을 설정하기 위한 Required 인터페이스 역할을 수행할 수 있는 빈을 새롭게 생성하며 가변성의 값은 해당 클래스에서 유지하도록 하고 있다. Plug-in 기법에서는 사용자가 외부로부터 가변적인 부분을 플러그인하여 가변성을 설정할 수 있도록 기능을 구현하고 있다. 제시된 4가지 기법은 컴포넌트 기반 개발에서 컴포넌트의 가변성을 설정하기 위한 Required Interface의 의미를 보존하며 높은 품질의 컴포넌트 구현을 가능하게 한다. 또한, 높은 품질의 컴포넌트를 생산을 통하여 시스템의 개발과 경쟁력 향상에 기여할 것으로 기대된다.

참 고 문 헌

[1] Matena, V., *Applying Enterprise JavaBean Component-Based Development for the J2EE Platform*, Addison-Wesley, 2001.

[2] Kuah, J, "Component-Based Development using EJB" 2000.

[3] D'Souza, D. and Wills, A., *Objects, Components, and Frameworks with UML*, Addison Wesley, 1999.

[4] Sterling Software "The COOL : Gen Component Standard 3.0," Sept., 1999.

[5] Sterling Software "Advisor 2.00," 1998.

[6] Castek "Exploring a Comprehensive CBD Method : Use of CBD/e in Practice," at URL : http://www.cbd-hq.com/PDFs/cbdhq_000615_cbde_for_ICSE. pdf, p.6, February, 2000.

[7] Compuware Corporation, "Building Component-Based Application with UNIFACE," at UML : http://www.compuware.com/products/uniface/station/control/buildcbd.htm, Compuware Corp.

[8] Sun Microsystems, *Enterprise JavaBeans Specification, Version 2.1*, http://java.sun.com/products/ejb/docs.html, Sun, Jun., 2002.

[9] Roman, E., *Mastering Enterprise JavaBeans Second*

Edition Wiley, 2001.

[10] Adatia, R., *Professional EJB*, Wrox Press, 2001.

[11] Horstmann, C., *Core Java 2 Volume I -Fundamentals Sun Microsystems*, 1999.

[12] Eckel, B., *Thinking in Java Second Editions*, Prentice Hall, 2000.

[13] D'souza, D. Component with Catalysis : Cost-effective Software Development, at URL : http://www.platinum.com, http://www.catalysis.org. http://www.iconcomp.com.

[14] Kurt, C. W., Scott, A. H., Rebert, C. S., *Building Systems from Commercial Components*, Addison Wesley, 2002.

[15] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthing, D., Paech, B., Wust, J., Zettel, J., *Component-based Product Line Engineering with UML*, Addison Wesley, 2002.

[16] Alr, D., Crupi, J., Malks D., *Core J2EE Patterns Bast Practices and Design Stratiegies*, Prentice Hall, 2001.

[17] Spell B., *Professional Java Programming*, Wrox Press, 2000.

윤 희 윤

e-mail : shakiss@hanmail.net

2001년 상명대학교 정보과학과(학사)

2004년 숭실대학교 대학원 컴퓨터학과 (공학석사)

관심분야 : CBD, 컴포넌트 개발 방법론, 컴포넌트 커스터마이제이션

김 수 동

e-mail : sdkim@ssu.ac.kr

1984년 Truman State University (전산학 학사)

1988년 The University of lowa (전산학 석사)

1991년 The University of lowa (전산학 박사)

1991년~1993년 한국통신 연구개발단 선임연구원

1993년~1994년 The University of lowa 교환교수

1994년~1995년 현대전자 소프트웨어연구소 책임 연구원

1995년~현재 숭실대학교 컴퓨터학부 부교수

관심분야 : 객체지향 방법론, 분산 객체 컴퓨팅, 컴포넌트 개발 방법론