컴포넌트 참조 모델의 기술적 비교 평가
(Technical Assessment of Component Reference Models)

허 진 선 † 김 수 동 ††
(Jin Sun Hur) (Soo Dong Kim)

요 약 컴포넌트 기반 개발 (CBD) 기술은 객체 보다 더 큰 컴포넌트 단위의 재사용을 통하여 개발 생산성을 높이는 재사용 기술이다. 그러나, CBD 컴포넌트의 구성요소와 의미를 정의하는 참조 모델이 국제적으로나 산업계에서 표준화 되지 못하고 있어 CBD 플랫폼 간의 상호연동과 이식성 문제가 있으며, 또한 개발자에게 어떤 모델을 채택해야 할 것인지의 신중한 결정을 요구한다.

본 논문에서는 대표적인 컴포넌트 참조모델들에 대한 정형적 뷰(View)인 메타 모델을 정의하고, 이를 기반으로 기술적인 상호 비교를 제시함으로써 각 기술의 장·단점 및 특징을 규명한다. 또한, 비교 평가를 통해 모든 컴포넌트가 공통적이며 필수적으로 만족해야 하는 핵심(Essential) 컴포넌트 모델과 최대한 다양한 장치와 구성요소를 가지는 확장(Extended) 컴포넌트 모델을 제시하여 상용 컴포넌트 모델들과의 객관적인 연관 관계 및 각 모델에 대한 정확한 평가를 할 수 있게 한다.

키워드 : 컴포넌트 기반 개발 (CBD), 컴포넌트 참조모델, 메타 모델

Abstract Component-Based Development (CBD) is a reuse technology providing enhancement in productivity through using the unit of component which is larger-grained than an object. However, reference model defining the elements and semantics of CBD component is standardized neither internationally nor in industrial. This yields interoperability and portability problem between CBD platforms, and presents burden of choosing appropriate model to developers.

In this paper, we define meta-models for representative component reference models, and identify advantages, disadvantages, and features of each model through technical comparison of meta-models. Besides, through a proposal of essential component model containing common and essential elements that all component models must conform and a extended component model containing maximum elements and mechanisms, we can precisely assess candidate component models in practice.

Key words : Component-Based Development (CBD), Component Reference Model, Meta Model

1. 서 론

1.1 동기

컴포넌트 개발 방법론(CBD)은 재사용 가능한 소프트웨어 부품을 이용하여 어플리케이션을 신속하고 용이하게 개발하기 위한 패러다임으로 널리 알려지고 있다. 이러한 CBD 개발을 수행할 때 공통적으로 참조하게 되는 컴포넌트 모델이 존재하게 된다. 컴포넌트 참조 모델을 준수해야 컴포넌트와 컴포넌트 간의 조합이나 컴포넌트와 어플리케이션 간의 통합 시 연동이 수월하다. 현재 참조할 수 있는 대표적인 컴포넌트 모델들은 다양하나 이들 모델 가운데 실무 프로젝트에 적용할 컴포넌트 참조 모델을 선택하기 위한 지침의 제시가 부족하다. 본 논문에서는 어떠한 컴포넌트 모델이 고 품질의 컴포넌트를 생산할 가능성이 높은 지 등의 컴포넌트 모델들의 특성을 비교 평가하며 평가에 앞서 컴포넌트 모델들을 비교하기 위한 기준들을 제안한다.

컴포넌트 기반의 개발이 활발해지고 있으며 여러 컴포넌트 참조 모델이 제시되어 있으나, 표준이 되는 컴포넌트 참조 모델은 명확히 제시되지 않았다. 효과적이고 이상적인 컴포넌트 개발 방법론을 제시하기 위해서 우선 기본적으로 컴포넌트 참조 모델에 대한 연구가 더욱 필요하다. 본 논문에서는 현재 제시된 컴포넌트 참조 모델들 간의 특성을 비교 분석하여 이를 기반으로 새로운 컴포넌트 참조모델을 제시한다. 제시된 컴포넌트 참조 모델을 이용하여 더욱 이상적인 방법론 제시가 가능하다. 또한 기존의 여러 대표적인 컴포넌트 참조 모델을

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 비 회 원 : 숭실대학교 컴퓨터학과
jsher@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim@comp.ssu.ac.kr
논문접수 : 2003년 7월 21일
심사완료 : 2004년 3월 26일

비교하기 위해, 다른 표기법으로 표기된 모델을 메타 모델로 변환하여 표준화된 표기를 한다.

1.2 논문의 범위 및 구성

2장에서는 관련 연구로서 현재 대표적인 컴포넌트 참조 모델 다섯 개에 대해 간략히 기술하고 3장에서는 조사한 다섯 개의 컴포넌트 참조 모델에 대해 더욱 상세히 살펴본다. 또한, 각각의 컴포넌트 참조 모델을 비교하기 위한 표준화의 척도로써 메타모델을 사용한다. 4장에서는 분석한 여러 대표적인 모델에 대한 비교를 수행하고 5장과 6장에서는 이를 기반으로 새로운 컴포넌트 참조 모델을 제시하며 7장에서는 기존의 모델들과 제안된 모델의 구성 요소적 측면에서의 평가를 수행한다. 8장에서는 제시된 컴포넌트 모델과 기존의 컴포넌트 모델을 적용한 사례 연구를 수행한다.

2. 관련 연구

2.1 CCM(CORBA 컴포넌트 모델)[1]

CORBA 컴포넌트는 CORBA3의 3가지 주요 항목에, 인터넷 통합 서비스 제어의 높은 질, 코바 컴포넌트 구조와 함께 포함된다. CORBA 컴포넌트는 기존의 EJB와 비슷한 모형을 가지고 있다. 컨테이너 안에서 홈 객체에 의해서 객체들이 생성되고 컨테이너에서 관리되며, 각각의 컴포넌트들은 인터페이스만을 통하여 서로 호출을 할 수가 있다. CORBA 컴포넌트는 CORBA3의 3가지 주요 항목에, 인터넷 통합 서비스 제어의 높은 질, 코바 컴포넌트 구조와 함께 포함된다. CORBA 컴포넌트는 기존의 EJB와 비슷한 모형을 가지고 있다. 컨테이너 안에서 홈 객체에 의해서 객체들이 생성되고 컨테이너에서 관리되며, 각각의 컴포넌트들은 인터페이스만을 통하여 서로 호출을 할 수가 있다.

2.2 EJB 컴포넌트 모델[2]

EJB(Enterprise JavaBeans) 컴포넌트 모델은 CCM (CORBA Component Model) 컴포넌트 모델과 유사하다. 컴포넌트는 Container 안에서 실행되며 Home 객체를 통해 관리된다. 그러나 CCM 컴포넌트와 달리 EJB 컴포넌트는 하나의 Component 인터페이스를 통해서만 참조된다. 따라서 Bean은 여러 개의 인터페이스를 갖지 않는다. EJB 컴포넌트 모델은 Local 또는 Remote 인터페이스와 내부 구현 로직으로 분리된다. CCM이 ORB 안에 트랜잭션, 보안, 영구성에 대한 서비스를 포함하는 것과는 달리, EJB는 Container에서 서비스를 제공한다.

2.3 .NET 컴포넌트 모델[3]

.NET은 웹 서비스, 웹 어플리케이션 개발과 컴포넌트를 개발하기 위한 복합 아키텍처이다. 본 논문에서는 CBD에서 추구하는 소프트웨어 컴포넌트로서 .NET이 지원하는 부분을 .NET 컴포넌트 모델이라 명시하고 이를 도출한다. .NET 컴포넌트 아키텍처에는 .NET 컴포넌트 모델에 대한 명확한 명세가 제시되어 있지는 않으나, .NET의 핵심 구성 요소들을 통해 유추 가능하다. 3.3 절에서의 .NET 컴포넌트 모델 규명을 선행한 후에 컴포넌트 모델들을 비교한다.

2.4 CMU/SEI 컴포넌트 모델[4]

CMU/SEI의 컴포넌트는 기능성에 대한 불투명한 구현을 한 것으로서 Third-Party 구성요소에 종속적이고 컴포넌트 모델을 충실히 이행하는 구현물로 정의하고 있다. 전체적인 컴포넌트 참조 모델의 요소는 8가지로 이루어져 있으며 이들간의 상호작용을 통해서 컴포넌트 모델이 명시된다. 컴포넌트 구현물, 컴포넌트 타입에 따른 특정 인터페이스, 컴포넌트가 인터페이스를 구현하기 위한 규약, 독립적인 배치, 컴포넌트 타입과 규약, 컴포넌트 모델, 컴포넌트 프레임워크, 컴포넌트 프레임워크가 제공하는 서비스가 그것이다.

2.5 Perrone의 컴포넌트 모델[5]

Perrone의 연구에서는 컴포넌트를 하나 이상의 클래스들로 구성된 단위로서, 이러한 클래스들의 기능들을 서비스(인터페이스)로 제공하는 장치로 정의하고 있다. 기존의 클래스보다 큰 단위의 개념으로 묘사하고 있다. 또한 컴포넌트는 하나의 분할된 문제 영역을 캡슐화 한 것이라고 표현하고 있다. Perrone은 컨테이너는 컴포넌트가 운영 혹은 작동되는 환경이라고 정의하고 있다. 컨테이너는 동일 컴포넌트 모델로 만들어진 컴포넌트들이 표준화된 방식으로 메시지를 주고 받는데 필요한 서비스들을 제공한다.

3. 컴포넌트 모델들에 대한 메타 모델

3.1 CCM(CORBA 컴포넌트 모델)

3.1.1 CCM의 개요와 구성 요소

CORBA의 컴포넌트는 Basic과 Extended의 두 단계로 분류된다. 이 둘은 모두 컴포넌트의 home에 의해 관리되지만, 제공되는 기능에는 차이가 있다[6]. 컴포넌트는 클라이언트나 다른 어플리케이션 환경의 요소들이 컴포넌트와 상호 작용할 수 있도록 하는 장치로 다양한 표면 특성(Surface Feature)을 제공한다. 이러한 표면 특성을 포트(Port)라 부르며 컴포넌트 모델은 Facets, Receptacles, 이벤트 소스/싱크, 속성의 기본적인 네 가지 포트를 제공한다. Facet은 클라이언트와의 상호작용을 위해 컴포넌트가 제공하는 인터페이스를 말한다. Receptacle은 어떤 외부 대리자에 의해 제공되는 참조를 사용하는 컴포넌트의 기능을 설명하는 연결 포인트이다. 이벤트 소스는 하나 이상의 관심을 가지는 이벤트 소비자나 이벤트 채널에게 특별한 형태의 이벤트를 방출시키는 연결 포인트를 의미한다. 이벤트 싱크는 특별

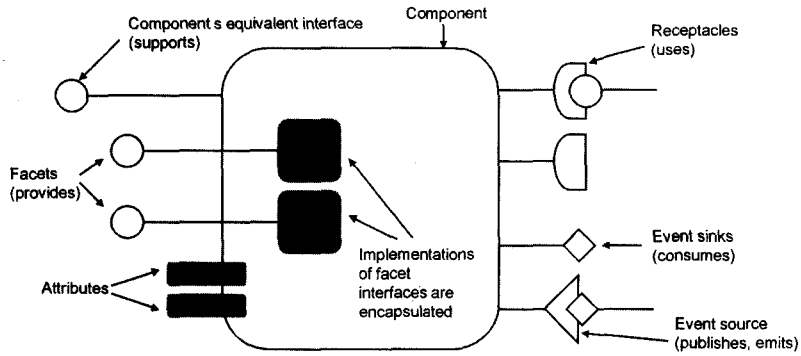


그림 1 CCM 컴포넌트 모델

한 형태의 이벤트를 받는 연결 포인트를 가리킨다. 속성은 주로 컴포넌트 환경설정을 위해 쓰이는 추가적이거나 변화하는 기능을 통해 표현되는 값들이다. Basic 컴포넌트는 Facet, Receptacle, 이벤트 소스/싱크를 제공하지 않고 단지 속성만 제공하는 반면, Extended 컴포넌트는 모든 종류의 포트를 제공할 수 있다[7].

컴포넌트는 Facets라 불리는 여러 개의 객체 참조를 외부에 제공할 수 있다. 클라이언트는 Equivalent 인터페이스를 통해 Facets의 참조를 얻게 되고 다른 컴포넌트의 포트에 연결될 수 있게 된다. Basic 컴포넌트는 Facet을 지원하지 않기 때문에 다른 Facet로의 항해는 불가능하며 Basic 컴포넌트의 Equivalent 인터페이스는 클라이언트와 상호 작용할 수 있는 유일한 객체가 된다. 위의 그림 1은 컴포넌트와 Facet의 관계를 설명하고 있으며, 여기서 언급한 Facet은 컴포넌트가 제공하는 모든 인터페이스를 의미한다. Facet 인터페이스의 구현은 컴포넌트에 의해 캡슐화되어 컴포넌트의 일부분을 이루고, 이러한 내부 구조는 클라이언트에게 감추어져 있다. 클라이언트는 어떠한 Facet에서든 Equivalent 인터페이스로 항해할 수 있으며, Equivalent 인터페이스로부터 어떠한 Facet이든 얻을 수 있다.

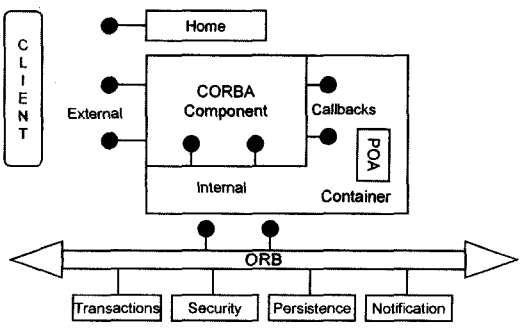


그림 2 CCM 컨테이너 모델

위의 그림 2는 구현된 CORBA 컴포넌트가 실행될 환경인 서버의 컨테이너를 나타내고 있다. 컨테이너는 일종의 프레임워크로써 컴포넌트가 실행될 때 트랜잭션, 보안, Notification, 영속성 관리 등과 같은 서비스를 제공한다. 컨테이너는 실행 시에 컴포넌트 인스턴스를 생성하고 관리하며, 다르게 구현된 컨테이너에서 같은 컴포넌트를 탑재할 수 있도록 표준적인 서비스들을 컴포넌트에게 제공한다. Internal 인터페이스는 지역성의 제약이 있는 인터페이스로써 local 인터페이스로서 정의되며 컴포넌트에게 컨테이너 기능들을 제공한다. Callback 인터페이스는 컨테이너에 의해 호출되고 CORBA 컴포넌트에 의해 구현되는 인터페이스이다.

3.1.2 CCM의 메타 모델

다음의 그림 3은 CORBA 컴포넌트 참조 모델의 구조적인 관점에 대한 메타 모델을 클래스 다이어그램을 이용하여 나타낸 것이다[8].

CORBA의 컴포넌트는 Basic 컴포넌트와 Extended 컴포넌트의 컴포넌트로 분리된다. Basic 컴포넌트와 Extended 컴포넌트 모두 컴포넌트 홈에 의해서 관리되며 각각의 컴포넌트는 하나의 Equivalent 인터페이스를 가지게 된다. Basic 컴포넌트는 포트라는 표면 특성 중에 Facet 인터페이스만을 여러 개 제공할 수 있다. 반면, Extended 컴포넌트는 Facet, Receptacle, 이벤트, 속성의 모든 포트를 제공할 수 있다. Facet 인터페이스는 CORBA컴포넌트에서 구현하게 되며, 이벤트는 이벤트를 방출하는 이벤트 소스와 이벤트를 소비하는 이벤트 싱크로 나누어진다.

3.2 EJB 컴포넌트 모델

3.2.1 EJB 컴포넌트 모델의 개요와 구성 요소

EJB는 다중 계층(Multi-Tier)의 분산형 객체지향 자바 어플리케이션을 개발하고 보급하기 위한 컴포넌트 아키텍처로서 선(Sun)사에서 개발하였다. EJB는 확장성 있는 어플리케이션 서버 컴포넌트들을 지원하는 여러

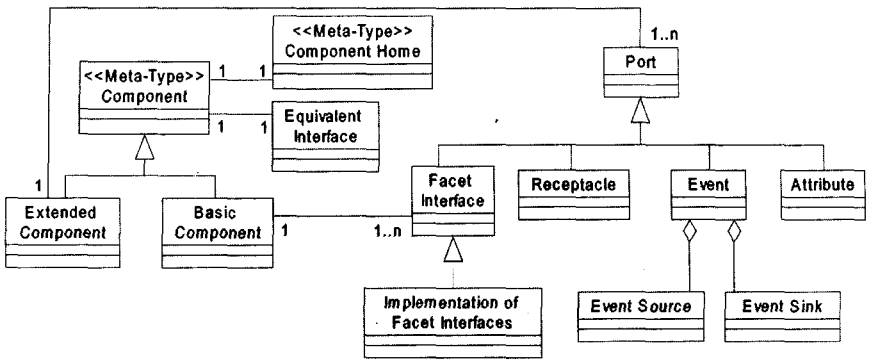


그림 3 CCM 컴포넌트 모델에 대한 메타 모델

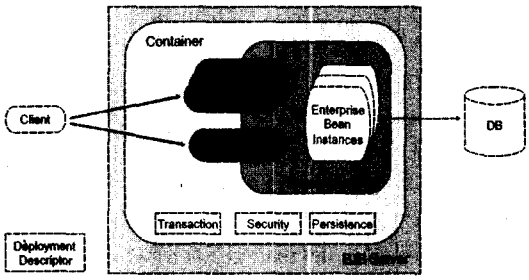


그림 4 EJB 컴포넌트 참조 모델

서비스들을 제공함으로써 비즈니스 어플리케이션들을 컴포넌트 단위로 쉽게 작성할 수 있도록 한다[9].

그림 4와 같이 EJB 컴포넌트 참조 모델은 크게 EJB 빈(컴포넌트), 컨테이너, EJB 객체, EJB 홈(Home)으로 나눌 수 있다. EJB 컴포넌트는 컨테이너에서 실행된다. 또한 컨테이너는 EJB 서버에서 실행된다. 컨테이너를 실행시키고 필요한 서비스들을 제공하는 모든 서버는 EJB 서버가 된다. EJB 빈은 개발자가 작성한 자바 클래스로서 비즈니스 로직을 구현한 것이다. 컨테이너는 EJB 컴포넌트를 실행시키는 것으로서 EJB 컴포넌트에 대한 트랜잭션 및 자원 관리, 버전관리, 지속성, 그리고 암호화 등과 같은 서비스들을 제공한다.

EJB컨테이너가 이러한 모든 기능들을 제공하기 때문에 컴포넌트 개발자는 비즈니스 로직 만 구현하면 된다. 하나의 컨테이너 안에 여러 개의 EJB 컴포넌트들이 존재할 수 있다. EJB 객체는 서버에 있는 EJB 컴포넌트의 Remote 인터페이스를 구현한다. 원격 인터페이스는 EJB 컴포넌트의 "비즈니스" 메소드들을 가리킨다. EJB 객체와 EJB 컴포넌트들은 분리된 클래스들이다. 이들은 같은 인터페이스 즉 EJB 컴포넌트의 원격 인터페이스를 구현하지만 서로 다른 일을 수행한다. EJB 컴포넌트는 EJB 컨테이너가 있는 서버에서 실행되고 비즈니스 로직을 구현한 것이다. 그러나 EJB 객체는 클라이언트 머신에서 실행되고 원격으로 EJB 컴포넌트의 메소드를 실행시킨다.

엔터프라이즈 빈은 세션 빈, 엔티티 빈, 메시지 드리븐 빈의 세 가지 종류로 구분된다. 세션 빈은 자료의 지속성이 일시적인 성격을 지니는데, 이는 엔티티으로 처리 되지 않는 부분들을 채워주는 역할을 하며, 서로 다른 빈 간의 상호작용, 즉 워크플로우를 표현해 준다. 세션 빈은 클라이언트와의 대화 상태를 유지 여부에 따라 두 가지의 기본 형태, 무상태(Stateless) 세션 빈과 상태 유지(Stateful)세션 빈으로 나뉘어 진다. 세션 빈에 비해 엔티티 빈은 보존의 성격을 가진다. 엔티티 빈은 데이타베이스의 데이타를 표현하고 데이타베이스를 기반으로 한 데이타를 갱신하고, 트랜잭션에 관여하여 동시에 여러 사용자들에게 사용될 수 있다. 자료의 지속성을 보장하는 방법에 따라 빈에서 직접 관리하는 빈 관리 지속성(Bean-Managed Persistence)과 컨테이너가 관리하는 컨테이너 관리 지속성(Container-Managed Persistence)으로 나뉠 수 있다. 메시지 드리븐 빈은 클라이언트로부터의 메시지를 비 동기적으로 처리해 준다.

3.2.2 EJB 컴포넌트 모델의 메타 모델

다음 그림 5는 EJB 컴포넌트 모델에 대한 구조적인 관점을 메타 모델로 그린 것이다.

그림 5에서 보는 바와 같이, 컨테이너는 EJB 서버에서 실행되며, 하나의 EJB 서버는 여러 개의 컨테이너를 포함할 수 있다. 컨테이너는 EJB 빈을 지원하기 위한 여러 시스템 레벨의 서비스를 제공한다. 컨테이너에는 여러 EJB 컴포넌트가 탑재될 수 있으며, 이 EJB 컴포넌트는 엔터프라이즈 빈, 홈 인터페이스, 컴포넌트 인터페이스, Deployment Descriptor의 묶음으로 구성된다. 엔터프라이즈 빈의 타입에는 세션 빈, 엔티티 빈, 메시지 드리븐 빈의 3가지 종류가 있으며, 인터페이스는 빈의 생명주기를 관리하는 홈 인터페이스와 빈의 고유 기능을 가지는 컴포넌트 인터페이스를 가진다. 또한 각 인

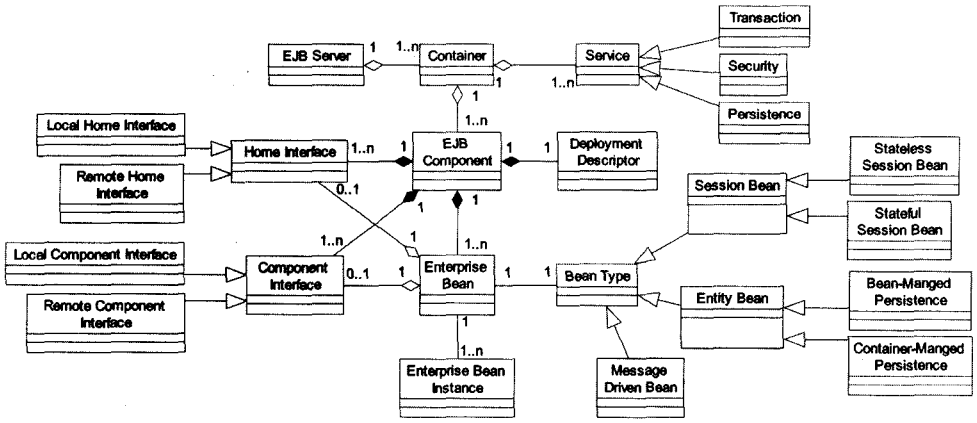


그림 5 EJB 컴포넌트 모델에 대한 메타 모델

터페이스는 접근 지역성에 따라 같은 가상 머신 내에서 접근 시에는 로컬 인터페이스, 가상 머신 밖에서 접근 시에는 리모트 인터페이스를 통해 접근한다.

3.3 .NET 컴포넌트 모델

3.3.1 .NET 컴포넌트 모델의 개요와 구성 요소

컴포넌트 참조 모델의 정적인 구성 요소는 컴포넌트 모델의 구체적이고 물리적인 측면에서의 구성 요소를 의미한다. .NET의 컴포넌트, 컨트롤, 컨테이너, 그리고 사이트 등이 정적인 구성 요소에 속하며 이에 대한 정의는 다음과 같다. .NET 프레임워크에서의 컴포넌트는 System.ComponentModel.IComponent 인터페이스를 상속 받아 구현한 클래스나 IComponent를 구현한 객체로부터 직접, 간접적으로 유도된 클래스를 의미한다.

컴포넌트 참조 모델의 동적 구성 요소는 정적인 구성 요소들 사이에서 런 타임 시에 컴포넌트가 동작할 때 발생하는 동적인 측면에서의 구성 요소를 의미한다. .NET 컴포넌트에서는 외부 자원에 대한 제어(Control over external resources)를 한다. 즉, IComponent 인터페이스에서 System.IDisposable 인터페이스 내의 Dispose 메소드를 구현할 때, 컴포넌트는 외부 자원을 명시적으로 릴리즈(Release)해서 컴포넌트가 가진 내부 요소들이 자원을 해제하도록 해야 한다.

컴포넌트는 원격에서 접근 가능하거나 비원격(지역적)일 수 있다. 원격 컴포넌트는 참조 또는 값에 의해 마셜(Marshal)될 수 있다. 마셜링(Marshaling)은 어플리케이션 도메인, 프로세스, 심지어는 머신과 같은 경계를 넘어서 객체를 전송하는 것을 말한다. 객체가 참조에 의해 마셜될 경우에는, 객체에 대한 원격 호출을 하는 프록시가 생성된다. 객체가 값에 의해 마셜될 경우에는, 관련된 경계 내에서 객체에 대한 직렬화된 복사본이 전송된다.

.NET 컴포넌트 모델을 정의하기 위해 추출된 정적 구성요소와 계층적 구성 요소에 대한 메타데이타를 이용하여 .NET 컴포넌트 모델을 추출하면 다음의 그림 6과 같다.

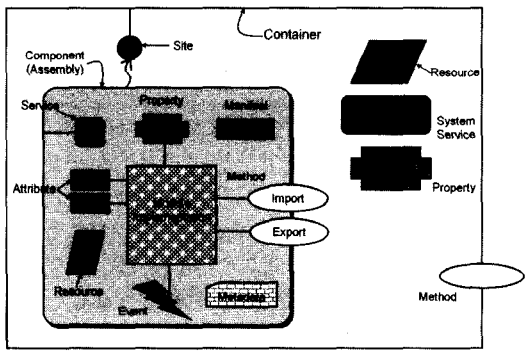


그림 6 .NET 컴포넌트 모델

컨테이너 안에는 0개 또는 그 이상의 컴포넌트가 존재하며 컴포넌트와 컨테이너의 상호작용은 사이트를 통해 이루어진다. 컴포넌트는 모듈로 구현되어 Property, Attribute, 이벤트 그리고 메소드를 제공하며 컴포넌트 자체의 타입과 서비스와 리소스를 제공한다. 컨테이너는 시스템 레벨의 서비스, 리소스, Property, 그리고 메소드를 제공한다.

3.3.2 .NET 컴포넌트 모델의 메타 모델

.NET 컴포넌트 모델을 정의하기 위해 분석된 정적 구성 요소, 동적 구성 요소, 계층적 구성 요소를 기반으로 메타데이타를 추출한다. .NET 클래스 라이브러리의 핵심 클래스와 인터페이스를 통해 .NET 컴포넌트 모델을 정의하기 위한 정적인 구성요소에 대한 메타데이타는 다음의 그림 7과 같이 추출된다.

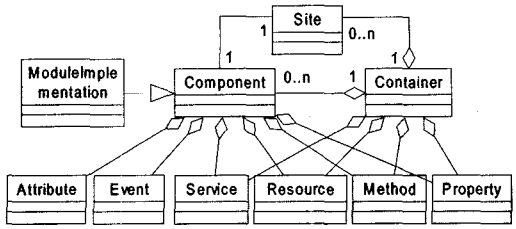


그림 7 .NET 컴포넌트 모델의 정적인 구성 요소에 대한 메타 모델

.NET 컴포넌트 모델을 정의하기 위한 계층적인 구성 요소에 대한 메타데이타는 다음의 그림 8과 같다. 컴포넌트는 원격 컴포넌트(Remotable Component)와 비원격 컴포넌트(Non-remotable Component)로 분류할 수 있으며, 원격 컴포넌트는 값에 의해 마셜(Marshal)되는 컴포넌트와 참조에 의해 마셜되는 컴포넌트로 분류될 수 있다. 컨트롤(Control)은 UI 기능을 제공하는 일종의 컴포넌트이며, 컨트롤에는 윈도우 폼(Windows Forms) 컨트롤과 웹 폼(Web Forms) 컨트롤이 있다. 디자이너(Designer)는 컨트롤을 위한 일종의 컨테이너가 된다.

3.4 CMU/SEI 컴포넌트 모델

3.4.1 CMU/SEI 컴포넌트 모델의 개요와 구성 요소

CMU의 SEI에서는 컴포넌트에 대한 명백한 두 가지 관점을 결합하여 컴포넌트에 대한 정의를 내리고 있다. 첫 번째 관점은 구현 측면으로서 소스코드로 이루어진 COTS 제품의 컴포넌트를 의미하고, 두 번째 관점은 구조적 추상화의 측면으로서 컴포넌트는 시스템의 전체적인 구조를 이해하기 위한 것으로서의 역할을 함을 의미한다. 이러한 두 가지 관점을 통합하여 컴포넌트는 컴포넌트' 판매 업체에서 구입하여 조립할 수 있는 구성요소로서 특정 컴포넌트 모델을 준수하면서 기능성을 구현한 프로그램, 즉 유통될 수 있는 소프트웨어 모듈로서 정의 내리고 있다.

다음 그림 9는 CMU의 SEI에서 참조하는 컴포넌트 모델로서 컴포넌트 기반 설계 패턴을 반영하고 있다. ①은 논리적 또는 물리적 장치에서 실행될 수 있는 소프트웨어 컴포넌트 구현 코드를 나타내고 있으며 이는 컴포넌트가 외부에 제공해야 할 기능인 컴포넌트 타입에 따른 특정한 인터페이스(②)를 구현하고 있다. 이는 컴포넌트가 규약(③)이라는 것으로 기술되는 특정한 기능을 책임져야 한다는 것을 의미하며 이러한 컴포넌트가 지켜야 하는 규약상의 책임은 전체적인 시스템에서 컴포넌트 간에 상호 작용할 때 적용되는 기준이 되기도 한다. 이렇게 구현된 컴포넌트는 표준화된 실행 환경에 배치되며(④) 전체 컴포넌트 기반 시스템은 시스템 내부에서 특정한 역할을 담당하며 인터페이스로 기술되는 여러 다양한 컴포넌트 타입을 기반으로 한다(⑤). 컴포넌트 모델(⑥)은 컴포넌트 타입과 그들의 인터페이스 집합, 그리고 특정 컴포넌트 타입 사이에서 적용되는 상호작용의 패턴들에 대한 명세로 구성된다. 컴포넌트 프레임워크는(⑦) 컴포넌트 모델을 강화하고 지원하기 위한 실행시의 다양한 서비스(⑧)를 제공한다. 이런 의미에서 컴포넌트 프레임워크는 특수한 목적을 위한 작은 의미의 운영체제라 할 수 있다.

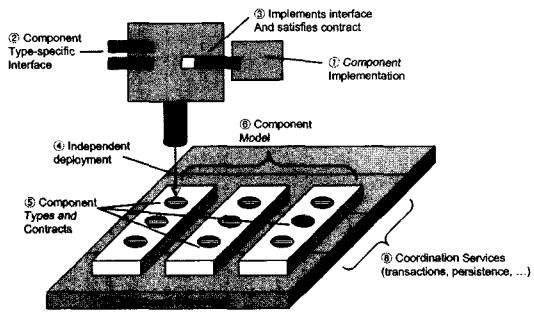


그림 9 CMU/SEI의 컴포넌트 참조 모델

3.4.2 CMU/SEI 컴포넌트 모델의 메타 모델

다음 그림 10은 CMU의 SEI의 컴포넌트 참조 모델에 대한 구조적인 관점을 UML의 클래스 다이어그램을 이용하여 메타 모델을 그린 것이다. CMU의 SEI의 컴포넌트 참조 모델을 구성하는 요소들과 이들 사이의 관계를 명시적으로 파악할 수 있다.

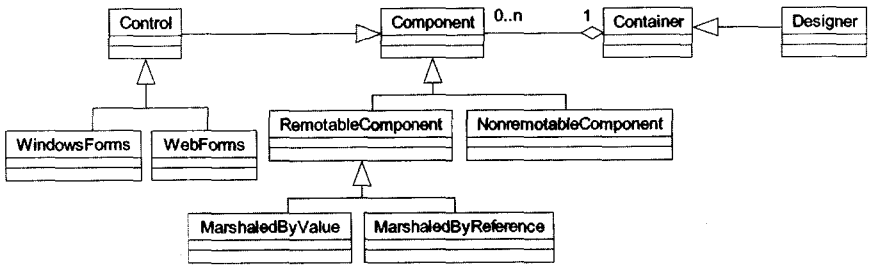


그림 8 .NET 컴포넌트 모델의 계층적인 구성 요소에 대한 메타 모델

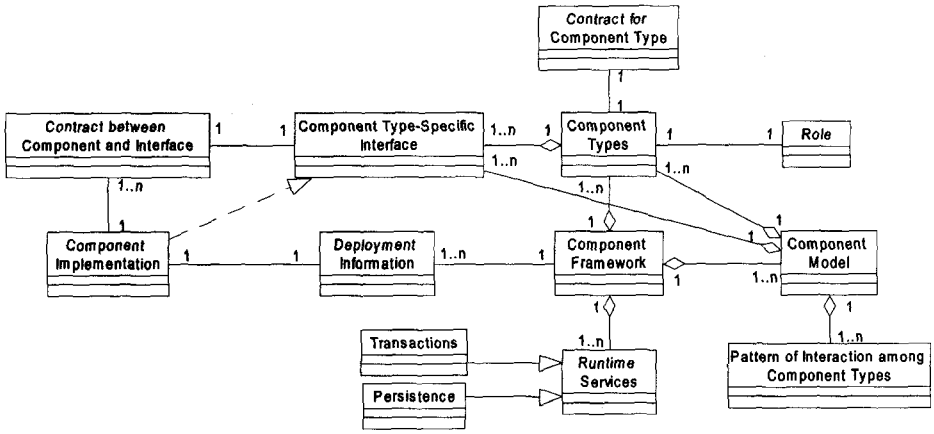


그림 10 CMU/SEI 컴포넌트 모델에 대한 메타 모델

실제적인 컴포넌트라 할 수 있는 컴포넌트 구현 코드는 컴포넌트 인터페이스를 구현한 것으로 컴포넌트와 인터페이스 사이의 규약을 준수해야 한다. 이러한 컴포넌트와 인터페이스 사이의 규약은 구현해야 할 인터페이스 당 하나씩 존재하며, 여러 규약을 반영하여 컴포넌트를 구현하게 된다. 구현된 컴포넌트는 실행시의 환경인 컴포넌트 프레임워크에 배치된다. 컴포넌트 프레임워크에는 여러 컴포넌트가 조합되어 하나의 컴포넌트 기반 시스템을 형성하게 된다. 따라서, 배치시 각 컴포넌트에 대한 배치 정보가 필요하게 된다.

컴포넌트는 역할에 따라 몇 가지 타입으로 분류가 될 수 있으며, 이러한 각기 다른 타입의 컴포넌트는 전체 컴포넌트 기반 시스템에서 각기 다른 역할을 담당하게 되며 컴포넌트 타입에 따른 특정한 인터페이스로서 기술된다. 컴포넌트 타입당 준수해야 할 규약이 있으며 이러한 컴포넌트 타입들의 집합과 인터페이스들, 컴포넌트 타입들 간의 상호작용 패턴에 관한 명세가 모여 컴포넌트 모델을 이룬다. 컴포넌트 프레임워크는 탑재된 컴포넌트들을 지원하기 위한 다양한 실행시의 서비스를 제공한다.

3.5 Perrone의 컴포넌트 모델

3.5.1 Perrone 컴포넌트 모델의 개요와 구성 요소

Perrone이 정의한 컴포넌트 참조 모델에서 컴포넌트는 상태와 함수의 집합들로 구성된 여러 클래스들을 모아놓은 단위로서, 이러한 클래스들이 제공하는 기능은 컴포넌트가 인터페이스라는 장치를 통해 제공하는 컴포넌트 서비스를 지원한다. 이러한 인터페이스는 내부 구현 로직은 감추고 클라이언트가 사용할 수 있는 서비스를 캡슐화하여 제공한다. 기존의 클래스보다 큰 개념으로 사용되며 여러 작은 단위의 클래스들의 논리적인 집합을 나타낸다. 그림 11은 Perrone이 정의한 컴포넌트

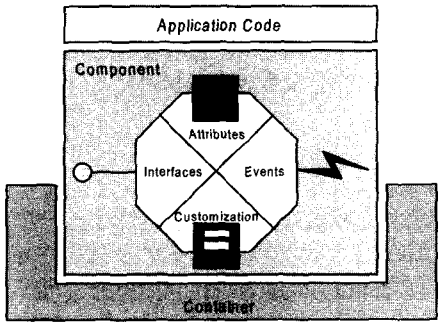


그림 11 Perrone의 컴포넌트 모델

모델의 기본적인 구성요소를 나타내고 있다. 어플리케이션에서 사용되는 코드를 캡슐화한 컴포넌트 자체가 컴포넌트 모델의 첫 번째 구성요소가 된다. 비록 컴포넌트 어플리케이션에 특정한 코드일지라도, 다름 그림 11에서는 어플리케이션 코드가 컴포넌트의 위에 위치하여 컴포넌트가 제공하는 서비스를 이용한다는 의미에서 컴포넌트와 어플리케이션 코드를 분리하여 명시하였다.

컨테이너는 컴포넌트가 동작하는 환경을 나타낸다. 컨테이너는 동일한 컴포넌트 모델로 만들어진 컴포넌트가 컴포넌트 모델에 따라 표준화된 스타일의 서비스를 제공 받도록 해준다. 컴포넌트는 여러 면(요소)들을 가지고 있다. 먼저 가장 중요한 요소는 컴포넌트가 제공하는 인터페이스 집합이다. 컴포넌트가 제공하는 인터페이스 뿐 아니라, 컴포넌트가 사용하도록 컨테이너에서 제공하는 인터페이스도 컴포넌트 모델에 정의되어야 한다. 컴포넌트로부터 생성될 수 있는 이벤트와 컴포넌트가 요청할 수 있는 이벤트에 대한 정의도 있어야 한다. 마지막으로, 컴포넌트는 명시적인 상태 속성(State Attribute)을 표현할 수 있는 표준 수단과 컴포넌트의 행위

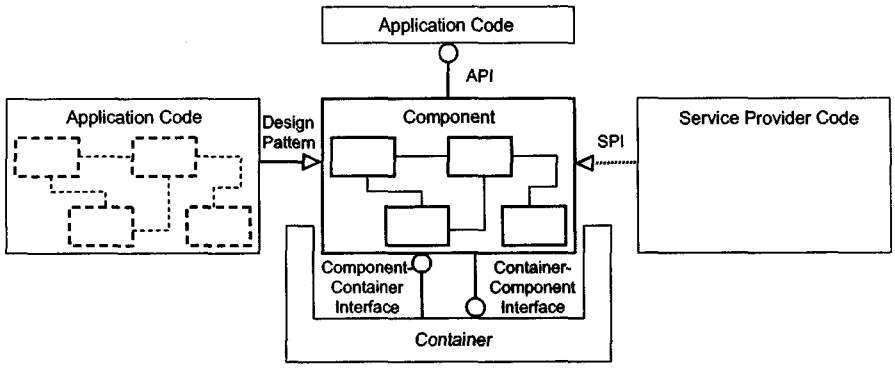


그림 12 컴포넌트 인터페이스 타입

와 속성을 커스터마이즈할 수 있는 수단을 제공한다.

컴포넌트 모델의 인터페이스는 컴포넌트와 다른 코드 개체들 사이의 규약이라 정의한다. 다른 코드 개체들은 컨테이너, 어플리케이션 코드, 다른 컴포넌트, 다른 컴포넌트 모델, 또는 어떠한 독점적인 서비스의 제공자가 될 수 있다. Perrone의 연구에서는 컴포넌트의 인터페이스를 다음의 그림 12와 같이 5 가지 타입으로 분류한다.

컴포넌트 인터페이스의 첫 번째 타입은 컴포넌트 API로서, 컴포넌트의 서비스들을 이용할 수 있는 표준 인터페이스의 집합을 정의하고 있다. 두 번째 타입은 설계 패턴(Design Pattern)으로서, 하나의 설계 패턴은 공통된 문제 영역에 대한 범용적인 설계 솔루션(Solution)을 제공하는 클래스들의 집합과 그들의 인터페이스, 그리고 클래스들간의 관계들을 의미한다. 어플리케이션 코드는 이러한 설계 패턴을 기반으로 보다 구체적이면서 어플리케이션에 한정된 설계를 제공함으로써 설계 패턴을 구체화시키게 된다. 세 번째 타입은 컴포넌트 SPI (Service Provider Interface)이다. 컴포넌트 SPI는 서비스 제공자가 자체적으로 구현한 표준 인터페이스 집합이다. 따라서 서로 다른 하부 서비스들이 어플리케이션 코드의 수정이나 컨테이너 코드 수정 없이도 어플리케이션이나 컨테이너에 플러그-인 될 수 있다. 네 번째 타입은 컴포넌트-대-컨테이너 인터페이스(Component-to-Container Interface)로서, 이것은 컴포넌트가 컨테이너와 연동하는데 필요한 인터페이스이다. 다섯 번째 타입은 컨테이너-대-컴포넌트 인터페이스(Container-to-Component Interface)로서, 이것은 컨테이너가 컴포넌트와 연동하기 위해 필요로 하는 인터페이스이다.

3.5.2 Perrone 컴포넌트 모델의 메타 모델

다음 그림 13은 Perrone의 컴포넌트 참조 모델에 대한 구조적인 관점의 메타 모델이다. Perrone이 정의한 컴포넌트는 여러 개의 클래스를 모아놓은 단위로서 컴포넌트 인터페이스, 속성, 이벤트, 커스터마이제이션의 구성요소로 이루어진다. 컴포넌트 인터페이스의 타입으로는 컴포넌트 API, 디자인 패턴, 컴포넌트 SPI, 컴포넌트 대 컨테이너 인터페이스, 컨테이너 대 컴포넌트 인터페이스가 있다.

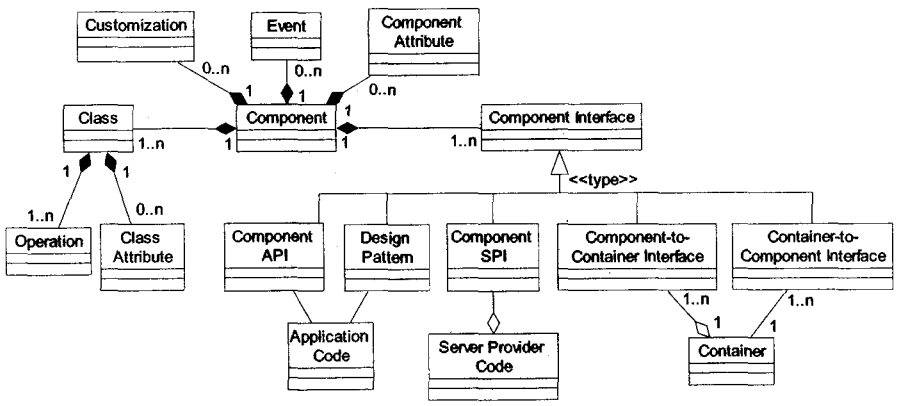


그림 13 Perrone 컴포넌트 모델에 대한 메타 모델

4. 컴포넌트 모델 비교

4.1 비교 기준

Heineman과 Councill은 소프트웨어 컴포넌트에 대해 컴포넌트 모델을 만족하는 소프트웨어 요소이며 조립 표준을 준수하기 때문에 수정하지 않고 조립할 수 있고, 독립적으로 배치될 수 있다고 정의하고 있다[10]. 이러한 컴포넌트를 이루는 컴포넌트의 하부구조는 소프트웨어 시스템의 디자인에서 상호 작용 하는 컴포넌트들의 집합이거나 컴포넌트를 구성하는 서브 시스템이라 명하고 있다.

이러한 컴포넌트들은 인터페이스를 가지게 되는데 인터페이스는 성능 명세서의 정의를 정확하게 만족하는 것으로 컴포넌트의 개념에서 기본이 되는 것 중에 하나라고 정의한다. 인터페이스의 표준은 고객의 의무적인 요구사항을 나타내며 다른 소프트웨어와 직접적인 상호작용을 하는 소프트웨어 요소이다. 컴포넌트는 인터페이스에 의해 정의된 모든 오퍼레이션의 구현을 컴포넌트가 포함하고 있을 때 Provide 인터페이스를 지원한다. 이러한 인터페이스는 컴포넌트 내부의 구현은 감추고 Provide 인터페이스의 오퍼레이션으로 표현된다.

Heineman과 Councill은 [10]에서 컴포넌트 모델은 특정한 상호작용 표준과 조립(Composition) 표준을 정의한다고 명한다. 즉, 컴포넌트 모델은 두 단계에서 작동된다. 첫 번째, 컴포넌트 모델은 개별적인 컴포넌트를 어떻게 구성하는지에 대해 정의하고 있다. 두 번째, 컴포넌트 모델은 컴포넌트 기반 시스템 내에서 컴포넌트의 집합이 상호 통신하고 상호 작용하는데 있어 전체적으로 적용되는 동작을 강요할 수 있다고 명하고 있다. 컴포넌트 모델은 애매하지 않고 구체적으로 명시된 인터페이스를 사용하도록 조성하는 상호작용 표준을 정의함으로써 조립이 가능하게 한다. 컴포넌트는 통합된 연결을 통해 다른 컴포넌트나 다른 소프트웨어 구성요소(예를 들어, 레가시 코드)와 조립될 수 있다. 컴포넌트 모델은 이러한 통합된 연결을 위해 허용된 메커니즘을 정의하고 있어야 한다. 컴포넌트 모델은 변경 없이 확장될 수 있는 방법을 설명하는 커스터마이제이션 메커니즘을 정의하는 것이 바람직하다. 컴포넌트 모델은 또한 코딩 형식, 문서 표준, 필수 공급자 독립적인 인터페이스와 같은 의무적인 컴포넌트의 속성들은 정의한다.

Heineman과 Councill은 앞서 설명한 컴포넌트 모델의 정의에 따라 8 가지의 표준적인 구성 요소를 제안하고 있다. 이러한 8 가지 구성 요소에 지원 언어와 시스템 서비스를 추가하여 다음 표 1의 10 가지를 컴포넌트 모델이 포함하고 있어야 하는 기본적인 구성요소로 정의한다. 즉, 컴포넌트 모델은 다음 10 가지의 구성요소를 포함하고 있는 것이 바람직하다는 것을 의미한다.

컴포넌트가 블랙 박스 타입의 재사용성을 제공할 때 컴포넌트 내부의 구현은 완전히 숨기고 인터페이스라는 설명이나 명세로 컴포넌트의 행위를 나타낸다. 엄밀히 말하면, 인터페이스는 컴포넌트의 구성요소라기 보다는 컴포넌트와 클라이언트 사이의 규약이라고 할 수 있다. 인터페이스에 대한 설명을 제공하는 Interface Specification은 컴포넌트 모델의 핵심적인 요소가 된다. 인터페이스는 의미적으로 연관된 함수들의 이름, 함수들의 매개변수, 매개변수 타입으로 구성된다. 인터페이스를 더욱 상세히 분류하면 컴포넌트가 가진 기능을 제공하는 Provide 인터페이스, 가변성을 설정하기 위한 Required 인터페이스, 다른 소프트웨어 부품의 기능을 사용하는 Uses 인터페이스를 가지는 것이 바람직하다. 이뿐 아니라, 인터페이스에 대한 시맨틱적 구성 요소와 명

표 1 컴포넌트 모델 비교 기준표

[illegible]	[illegible]
인터페이스 (Interfaces)	컴포넌트의 기능과 속성에 대한 명세; Provide, Required, Uses 인터페이스의 분류.
네이밍 (Naming)	인터페이스와 컴포넌트에 대해 전체적으로 유일한 식별 가능한 이름
메타데이타 (Meta Data)	컴포넌트, 인터페이스, 그리고 그것들의 관계에 대한 정보; 이러한 정보를 제공하는 API와 서비스
연동성 (Interoperability)	다른 언어로 구현된 다른 벤더들에서 만든 컴포넌트들 사이의 상호통신과 데이타 교환
특화성 (Customization)	컴포넌트를 특화하기 위해 필요한 인터페이스. 사용자 친화적인 커스터마이제이션 툴들이 이러한 인터페이스를 사용한다.
조합 (Composition)	더 큰 구조를 이루기 위해 컴포넌트를 조합하고 기존에 존재하는 구조에 컴포넌트를 대체하고 추가하기 위한 인터페이스와 규칙
업데이트 지원 (Evolution Support)	새로운 버전에 맞추어 컴포넌트나 인터페이스를 대체하기 위한 규칙과 서비스
패키징과 배치 (Packaging and Deployment)	컴포넌트 설치와 구성(configure)을 위해 필요한 리소스와 구현물에 대한 패키징
지원 언어 (Language Support)	컴포넌트 모델을 구현하기 위해 지원되는 언어
시스템 서비스 (System Service)	컴포넌트 실행을 위해 제공되는 서비스로 컴포넌트 객체 생성, 생명주기, 영구성, 트랜잭션, 보안 관리 서비스 등을 의미

세 방법을 제시하고 있는 컴포넌트 모델이 바람직하다. 시맨틱적인 구성요소에는 선조건(Precondition), 후조건(Postcondition), 불변조건(Invariant)이 있다.

모든 컴포넌트나 인터페이스는 충돌이 발생하지 않도록 하기 위한 유일하게 식별할 수 있는 이름을 가지고 있어야 한다. 그렇기 때문에 컴포넌트 모델의 한 부분으로 표준화된 명명 계획(Naming Schema)이 필요하다. 일반적으로 명명하는 대표적인 방법이 2가지 있는데, 하나는 유일한 ID를 이용하는 것으로 툴에서 생성하는 방법을 통하여 구현되며 Microsoft의 COM/DCOM/COM+/.NET 계열의 GUID가 이에 속한다. 두 번째 방법은 계층적인 네임 스페이스를 사용하는 것으로 Java 계열의 컴포넌트 모델이 주로 이러한 방식을 이용한다.

메타데이타는 컴포넌트와 인터페이스와 그들의 관계에 대한 정보를 의미하는 것으로 문서화나 원격 메소드 호출에 대한 기초 정보를 제공하여 컴포넌트 조합 툴에서 사용되게 한다. 컴포넌트 모델은 이러한 메타데이타가 어떠한 방법으로 기술되며 어떠한 방법으로 획득되는지를 명시하고 있어야 한다. 즉, 메타데이타는 인터페이스에 대한 정보를 기술하기 위한 방법과 컴포넌트의 정보를 기술하기 위한 방법, 그리고 컴포넌트와 인터페이스 사이의 규약을 기술하기 위한 방법으로 분류되어 제공되어야 한다.

다른 벤더에 의해 만들어진 컴포넌트들 사이에 연결을 제공하고 데이타 교환과 제어 공유를 가능하게 하는 잘 정의된 통신 채널이 있어야 컴포넌트 간의 조합이 가능하다. 통신하고자 하는 컴포넌트들이 같은 컴퓨터의 단일 프로세스에 있든지, 다른 프로세스에 있든지, 네트워크 상으로 연결된 다른 컴퓨터에 있든지, 동일한 컴포넌트 참조 모델을 준수하지만 다른 언어로 구현된 컴포넌트들이든지, 다른 컴포넌트 모델을 준수하는 컴포넌트들이든지 간에 바이너리 수준의 연결이나 통신을 제공해야 한다. 따라서, 컴포넌트 모델은 이러한 컴포넌트 간의 통신을 위한 호출 협정이 표준화되어 있어야 한다. 즉, 같은 컴포넌트 모델을 기반으로 구현된 컴포넌트들 사이의 연결 방법, 다른 컴포넌트 모델을 기반으로 구현된 컴포넌트들 사이의 연결 방법이 제공되어야 한다.

컴포넌트를 구입하여 설치하기 전에 사용자 환경에 맞게 적응시키는 선행 작업이 필요하다. 그러나, 컴포넌트는 일반적으로 내부를 감춘 블랙박스 형태로 제공되기 때문에 커스터마이제이션 인터페이스를 따로 두어서 이를 이용하여 컴포넌트의 속성이나 심지어는 복잡한 기능까지도 사용자 환경에 맞게 변형을 시킬 수 있다. 즉, 가변성을 제한하지 않으며 이러한 가변성의 종류에 대한 분류 체계가 있으며 각각의 가변성 종류에 대한 어느 정도의 설계 방법을 제공하고 있는 컴포넌트 모델이 바람직하다.

두 개 이상의 컴포넌트를 조합하거나 조립하여 새로운 기능을 제공하는 하나의 컴포넌트를 만들 수 있다. 이러한 컴포넌트의 조합은 프로그래밍 언어, Glue 언어, 조합 툴이나 컴포넌트 하부구조(Infrastructure)에 의해 가능하다. 그러나, 이 중에 컴포넌트 간의 조합을 위한 가장 효율적인 방법을 제공하는 것은 컴포넌트 하부구조이다. 컴포넌트 하부구조는 특정 도메인 내에서의 기능을 만족시키기 위한 컴포넌트와 그들 간의 상호 작용에 대한 설계를 미리 정의하여 표준화한 것을 의미하는 것으로 컴포넌트 하부구조에 정의된 상호작용 표준을 준수하는 컴포넌트를 삽입하거나 대체하여 개별적인 컴포넌트 뿐만 아니라 전체 설계에 대한 재사용성을 증가시키면서 컴포넌트 조합을 수행할 수 있다.

컴포넌트는 진화하는 기능에 맞추어 새로운 기능을 제공해야 한다. 이렇게 새로운 기능을 제공하기 위해서 인터페이스에 대한 구현만을 수정하는 데에 그칠 수도 있고 인터페이스를 수정하거나 새로이 추가해야 하는 경우도 있을 것이다. 그러나, 기존에 컴포넌트를 사용하던 클라이언트나 시스템으로의 영향을 최소화 하는 것이 바람직하다. 그래서, 이러한 영향을 최소화하기 위해 컴포넌트 모델은 진화하는 컴포넌트를 지원할 수 있는 규칙이나 표준을 제공해야 한다. 즉, 컴포넌트 대체, 컴포넌트의 내부적인 구현에 대한 변화, 인터페이스의 변화, 새로운 인터페이스 생성을 대비한 규칙과 표준을 정의하고 있어야 한다.

컴포넌트 모델은 컴포넌트의 패키징 방법과 독립적으로 배치(Deployment), 즉, 컴포넌트 하부구조에 설치하고 구성하는 방법을 기술하고 있어야 한다. 배치 표준에는 패키지의 구성요소와 배치 프로세스에 필요한 정보에 대해 기술한 배치 기술문서(Deployment Description)의 구조와 의미를 명시하고 있어야 하며 패키지의 형식을 정의해야 한다. 컴포넌트 모델은 또한 컴포넌트 등록과 배치 프로세스에 대한 정의를 포함하고 있어야 한다.

컴포넌트 모델들은 각각을 구현하기 위한 언어가 지원되어야 한다. 이러한 언어의 지원 정도가 컴포넌트 모델을 평가하는 또 하나의 기준이 된다. 다양한 언어를 통해 구현될 수 있는 컴포넌트 모델이 바람직하다. CMU와 Perrone의 모델은 지원되는 언어가 제시되어 있지 않은 반면, EJB, .NET과 CCM과 같은 경우에는 지원되는 언어가 제시되어 있어 실용적으로 적용될 수 있음을 나타내고 있다.

확장된 컴포넌트 모델의 경우 컴포넌트 실행 시에 지원되어야 하는 추가적인 시스템 서비스에 대한 정의가 있어야 한다. 컴포넌트는 분산 트랜잭션을 관리하기 위

한 서비스와 보안 시스템을 만들기 위한 공통 보안 하부 구조나 기능을 제공해야 한다. EJB나 CCM과 같이 영구적인 데이타를 저장하기 위한 서비스와 추가적으로 컴포넌트 리소스를 사용하여 컴포넌트 객체를 관리하는 것과 데이타베이스 커넥션 풀링, 쓰레딩, 그리고 상태 관리에 대한 서비스를 제공하는 것이 바람직하다.

4.2 비교 결과

본 논문에서 조사한 다섯 가지 컴포넌트 모델들 중에는 표 1에서 정의된 각각의 구성요소를 충분히 만족하고 있는 모델이 있기도 한 반면 부분적으로 만족하는 모델도 있다. 표 1의 구성요소에 대한 상세 분류는 4.1에서 설명하였으며 이러한 상세 구성요소를 비교 기준으로 삼아 각각의 모델이 얼마나 상세 기준을 만족하여 바람직한 컴포넌트 모델이 되고 있는지에 대한 비교를 수행한다. 표 2는 비교 기준에 따라 컴포넌트 모델을 비교한 결과이다.

CCM 모델은 Facet과 Equivalent 인터페이스라는 형태로 Provide 인터페이스를 제공하며 Receptacle이라는 Uses 인터페이스 장치를 제공한다. 컴포넌트 어셈블리 디스크립터에 Namingservice라는 항목에 컴포넌트와 인터페이스를 찾기 위한 이름을 명시한다. 메타데이타는 인터페이스와 구현 레파지토리를 통해 지원되며 다른 언어로 구현된 컴포넌트들 간에 원활한 상호 운영을 위한 표준을 정의하고 있다[11]. 또한, Composition이라는 단위로 여러 요소들을 하나의 일관성 있는 단위로 묶으며 Assembly라는 단위로 배포가 되며 Java, C++ 등과 같은 다수의 언어를 통해 구현 가능하다. 또한 트랜잭션, 보안, 영구성 관리, Notification과 같은 시스템 서비스를 제공한다.

EJB 컴포넌트 모델은 Component 인터페이스를 통해 비즈니스 로직을 제공하며 계층적인 네임스페이스의 메커니즘과 JNDI를 통해 네이밍을 지원한다. Introspection과 EJBMetaData 클래스를 통하여 메타데이타를 제공하며 RMI를 통해 연동성을 높여주고 IIOP를 이용한 EJB에서 CORBA로의 연동 표준을 제공한다. 디플로이먼트 디스크립터를 통해 배치 정보를 제공하며 bean 클래스, Home과 Component 인터페이스, 디플로이먼트 디스크립터를 하나의 Ejb-jar 파일로 묶어서 배포한다. EJB는 자바 계열 언어를 통해 구현되며 트랜잭션, 보안, 영구성 관리와 같은 시스템 서비스를 제공한다.

.NET은 Exported 메소드와 Imported 메소드를 통해 Provide와 Uses 인터페이스의 기능을 제공한다[12]. GUID (Globally Unique Identifier)를 부여함으로써 네이밍을 제공하며 Type Library의 형태로 메타데이타를 제공하며 다른 언어로 구현된 컴포넌트들 사이의 뛰어난 상호 연동성을 제공한다. .NET 컴포넌트 또한 Assembly라는 단위의 바이너리 형태로 배포되며 CLS를 만족하는 다수 언어로 구현 가능하다. .NET 모델 또한 트랜잭션, 보안, 영구성 관리와 같은 시스템 서비스를 제공한다[13].

컴포넌트 개발 플랫폼을 제공하는 EJB, .NET, CCM과 컴포넌트 모델에 대한 개략적인 제시만을 제공하는 CMU/SEI, Perrone의 모델과는 차이가 있다. CMU/SEI 모델은 Provide 인터페이스 성격의 인터페이스를 명시하고 있으며 구체적인 메커니즘 없이 네이밍 서비스를 지원해야 할 것을 언급해 놓고 있다. 또한, 여러 조합 형태들을 제공하고 있으며 패키징과 배치에 관한 구체적인 방법은 정의해 놓고 있지 않다. 가정하고 있는 구현 언어는 없으며 트랜잭션과 영구성 관리의 시스템 서비스를 제공해야 함을 명시하고 있다.

Perrone의 모델은 Component API와 Component SPI의 형태로 Provide와 Uses 인터페이스를 정의하며

표 2 컴포넌트 모델 비교 결과표

비교 기준		CCM	EJB	.NET	CMU/SEI	Perrone's
인터페이스 (Interface)	Provide	✓	✓	✓	✓	✓
	Required					
	Uses	✓		✓		✓
네이밍 (Naming)		✓	✓	✓	✓	
메타데이타 (Meta Data)		✓	✓	✓		
연동성 (Interoperability)		✓	✓	✓		
특화성 (Customization)						✓
조합 (Composition)		✓			✓	
업데이트 지원 (Evolution Support)						
패키징과 배치(Packaging and Deployment)		✓	✓	✓	✓	
지원 언어 (Language Support)		✓	✓	✓		
시스템 서비스 (System Service)		✓	✓	✓	✓	

다른 모델과는 달리 향상된 적용성을 제공하기 위해 컴포넌트가 가진 행위(behavior)와 속성(attribute)에 대한 커스터마이제이션을 해야 함을 기술하고 있다.

5. 핵심 참조 모델(Essential Reference Model)

5.1 정의

컴포넌트 참조 모델은 컴포넌트 방법론들과 컴포넌트 아키텍처들이 지원하는 다양한 개념들간의 관계를 보여주고, 컴포넌트 모델을 위한 표준 용어를 지원한다는 점에서 컴포넌트 기반 방법론의 중요한 부분이다. 컴포넌트 참조 모델을 이용해서, 다양한 컴포넌트 아키텍처상의 컴포넌트 모델을 설명할 수 있고, 기존의 컴포넌트 방법론이나 아키텍처에 독립적인 일반적인 컴포넌트 모델을 제공할 수 있다.

핵심 참조 모델은 최소 규모의 컴포넌트 참조 모델이다. 핵심 참조 모델은 컴포넌트, 인터페이스, 워크플로우 구성되어 있으며 컴포넌트 모델이 준수해야 하는 최소한의 요구사항을 정의한 모델이다.

5.2 기준

핵심 참조 모델은 3장에서 분석한 컴포넌트 모델들이 공통적으로 가지고 있는 구성요소들을 추출한 것과 컴포넌트의 특성상 최소한 포함하고 있어야 하는 요소들을 기준으로 구성된다. 3장에서 분석한 여러 컴포넌트 모델들이 필수적으로 포함하고 있는 구성요소는 컴포넌트 구현물과 인터페이스이다. 그리고 컴포넌트는 인터페이스가 제공하는 기능이 어떠한 내부 로직에 의해 수행되는 것인지를 숨긴 블랙박스 형태가 보편적이다. 이러한 컴포넌트의 특성상 워크플로우를 핵심 참조 모델의 구성요소로 정의한다.

5.3 구성요소

핵심 참조 모델은 아래의 그림 14와 같이 컴포넌트 단위, 인터페이스, 워크플로우의 세 가지 구성요소로 이루어져 있다.

• 컴포넌트 단위

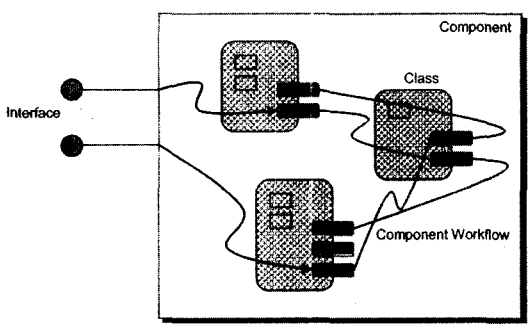


그림 14 핵심 컴포넌트 참조 모델

컴포넌트 단위는 컴포넌트가 가지고 있는 기능을 실제적으로 구현하고 있다. 이 컴포넌트 단위의 구현물이 물리적인 장치에 직접 설치된다. 컴포넌트는 컴포넌트를 구현한 실제 구현물을 의미하는 것으로 물리적인 장치에서 실행되는 이진 코드를 말한다. 컴포넌트는 관련성이 높은 클래스들의 묶음으로 이루어지며 이러한 클래스들은 어트리뷰트와 오퍼레이션으로 구성되어 있다. 컴포넌트 단위는 조사한 컴포넌트 모델들이 모두 포함하고 있는 구성요소이며 컴포넌트 참조 모델의 핵심 단위가 되는 요소이기 때문에 핵심 컴포넌트 모델의 구성요소로 제안한다.

• 인터페이스

인터페이스는 컴포넌트가 제공하는 기능성을 명시한다. 한 인터페이스는 복수 개의 오퍼레이션으로 구성되며 하나의 인터페이스는 여러 개의 컴포넌트에서 구현가능하다. 또한 하나의 컴포넌트는 여러 개의 인터페이스의 기능을 구현할 수 있다. 인터페이스를 이용하여 컴포넌트에서 제공하는 기능을 외부화시켜서 사용자는 인터페이스만을 보고 필요로하는 기능을 사용하고 어떻게 기능이 수행되는지는 숨기려는 것이 컴포넌트를 사용하는 중요한 목적이 된다. 또한, 조사된 여러 컴포넌트 모델에서 모두 공통적으로 인터페이스를 포함하고 있어 핵심 모델의 요소로 정의한다.

• 워크플로우

워크플로우는 인터페이스를 지원하기 위한 클래스간, 또는 컴포넌트 간의 메시지 흐름을 의미한다. 컴포넌트 워크플로우는 컴포넌트가 인터페이스를 통해 제공하는 기능의 동적 흐름을 볼 수 있게 하는 요소로서 오퍼레이션이 실행되는 경로인 런 타임 구성요소이다. OOP의 워크플로우는 단순히 객체 간의 상호 작용 정보를 나타내지만, CBD의 워크플로우는 컴포넌트 내(Intra-component)의 객체 간 상호 작용 정보에 컴포넌트 간(Inter-component)의 상호 작용 정보가 추가된다. 워크플로우는 핵심 컴포넌트 모델에 새로이 추가된 구성요소로서 블랙 박스화가 보편적인 컴포넌트가 제공하는 기능성의 실행 경로를 나타내기 위해 필요한 요소이다. 또한 워크플로우 모델링은 컴포넌트 설계에 있어 중요한 과정이 되며 워크플로우는 컴포넌트 오퍼레이션의 설계를 위해 필수적인 요소가 된다.

6. 확장 참조 모델(Extended Reference Model)

6.1 정의

확장 참조 모델은 풍부한 CBD 장치를 포함하고 있는 최대 규모의 컴포넌트 참조 모델이다. 확장 참조 모델은 핵심 참조 모델의 구성요소를 포함하고 있으며 인터페이스의 종류는 Provide, Required, Uses, Home 인터페

이스로 확장하였고 상태 속성, 이벤트, 컨테이너(컴포넌트 프레임웍), 콜백 인터페이스, 컨테이너에서 제공하는 인터페이스, 배치 정보, 커스터마이제이션, 가변성, 컴포넌트 타입, 컴포넌트와 인터페이스 간의 계약, 컴포넌트 간의 계약 등의 구성요소를 추가하였다. 확장 참조 모델은 CBD의 재사용성, 유지보수성, 유연성의 장점을 충분히 활용할 수 있는 모델이고 복잡한 도메인이나 패밀리 멤버 간에 가변성이 많은 도메인에서 유용하게 사용될 수 있다.

6.2 기준

확장 참조 모델은 3장에서 분석한 컴포넌트 모델들이 가지고 있는 구성요소를 모두 반영하고 CBSE (Component Based Software Engineering)의 특징을 고려하여 추가적인 구성요소를 포함하고 있다. 3장에서 조사한 대표적인 컴포넌트 참조 모델들의 비 공통적인 특징으로는 컴포넌트 타입을 위한 계약, 배치 정보, 실행 시 서비스(트랜잭션, 영구성, 보안, Notification), 컴포넌트 타입들 사이의 상호작용 패턴, Extended 컴포넌트, Basic 컴포넌트, 컴포넌트 홈, Equivalent 인터페이스, 포트, Receptacle, 이벤트, 이벤트 소스, 이벤트 싱크, 속성, 커스터마이제이션, 컴포넌트 SPI, 컨테이너를 위해 컴포넌트가 제공하는 인터페이스, 클래스, 클래스 함수, 클래스 속성, COM 라이브러리, 시스템 레지스트리, 클래스 Factory, Base 인터페이스 등이 있다.

3장의 컴포넌트 모델이 공통적으로 가지고 있는 정보는 컴포넌트 단위, 인터페이스, 컴포넌트 타입, 컴포넌트 모델, 컴포넌트와 인터페이스 사이의 규약, 컴포넌트 프레임워크 등이 있다. 이러한 필수적인 컴포넌트 요소에 이벤트, 속성, 커스터마이제이션, 콜백 메소드, 컴포넌트 홈 등의 요소들을 추가한다. 또한, CBSE의 특성상 재사용성이 높은 컴포넌트가 되기 위해 컴포넌트 워크플로우, Required 인터페이스, Uses 인터페이스, 가변성(속성, 로직, 워크플로우, 영구성에 대한 가변성) 등을 추가한다.

6.3 구성요소

본 논문에서는 확장 참조 모델을 아래의 그림 15와 같이 제안한다.

제안된 확장 컴포넌트 참조 모델의 구성요소는 크게 정적, 동적, 실행 환경, 적용적, 형태적, 커뮤니케이션 구성 요소의 6가지 영역으로 분류될 수 있다. 컴포넌트 모델의 정적인 구성요소에는 컴포넌트, 인터페이스, 상태 속성이 있으며, 컴포넌트 모델의 동적인 구성요소에는 컴포넌트 워크플로우, 이벤트가 있다. 컴포넌트 참조 모델의 실행 환경 요소에는 컨테이너, 콜백 인터페이스, 컨테이너에서 제공하는 인터페이스와 배치 정보 등이 있다. 컴포넌트 참조 모델의 적용적 구성 요소에는 커스터마이제이션과 가변성이 있으며 형태적 구성요소에는 컴포넌트 모델과 컴포넌트 타입이 있다[14]. 마지막으로,

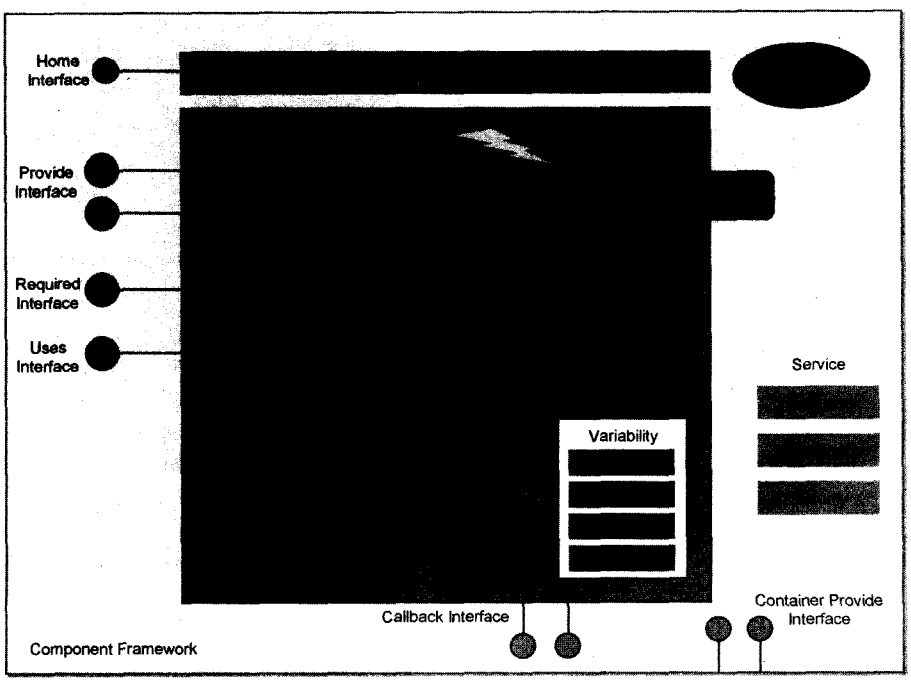


그림 15 확장 컴포넌트 참조 모델

컴포넌트 간 커뮤니케이션 구성 요소에는 컴포넌트와 인터페이스 간의 계약과 컴포넌트 간의 계약이 있다.

• 컴포넌트 단위(핵심 참조 모델)

컴포넌트는 물리적 혹은 논리적인 장치에서 실행할 수 있는 하나의 소프트웨어 구현물로 정의할 수 있다. 컴포넌트는 하나 이상의 클래스들로 구성된 단위로서, 이러한 클래스들의 기능들을 서비스(인터페이스)로 제공하는 장치로 기존의 클래스보다 큰 단위의 개념이다. 또한, 컴포넌트는 하나의 분할된 문제 영역을 캡슐화한 것이다. 즉, 컴포넌트는 하나의 응용 프로그램에서 활용되는 코드를 캡슐화시킨 것이다. 또한, 컴포넌트는 하나의 유용한 작업을 수행하는 능동적인 요소로서, 개별적인 객체, 객체들의 집합, 혹은 서브시스템을 지칭하기도 한다. 컴포넌트는 개별 단위로 독립적으로 개발되어서 배포될 수 있는 응집력 있는 소프트웨어 패키지로서 다른 컴포넌트들과 조립할 수 있도록 서비스를 제공하는 인터페이스와 사용하는 인터페이스가 정의되어 있어야 한다.

• 인터페이스

Provide 인터페이스만 정의하고 있는 핵심 참조 모델의 인터페이스를 확장 참조 모델에서는 Provide 인터페이스, Required 인터페이스, Uses 인터페이스, Home 인터페이스로 확장하여 정의한다.

- Provide 인터페이스 (핵심 참조 모델)

Provide 인터페이스는 컴포넌트가 가지고 있는 기능을 제공하기 위한 인터페이스로 외부의 사용자나 객체에서 이 인터페이스를 통하여 컴포넌트가 제공하는 서비스를 이용할 수가 있다. 이런 인터페이스는 객체와 클라이언트간의 규약으로 인터페이스를 호출하는 객체와 컴포넌트에서 제공하는 인터페이스에서는 동일한 메소드를 가짐으로 인터페이스의 메소드 호출에 의해 서비스를 제공한다.

- Required 인터페이스

Required 인터페이스는 가변성을 효율적으로 지원하기 위해 각각 어플리케이션 마다 가지는 약간의 차이점들을 설정할 수 있게끔 하는 인터페이스이다. 컴포넌트의 기능을 제공하는 Provide 인터페이스와는 달리 가변성을 설정해주기 위해 외부로부터의 결정 사항을 받아들이기 위한 인터페이스로서 컴포넌트의 재사용성을 높여주는 주요한 구성요소가 된다. [9]와 같은 문헌에서도 컴포넌트의 가변성과 이를 위한 Required 인터페이스의 중요성이 강조되고 있다.

- Uses 인터페이스

컴포넌트는 다른 컴포넌트들이나 다른 클라이언트 코드들과의 상호 작용 가운데 작동하게 되고 컴포넌트 내부에서 외부에 있는 서비스를 요청하는 경우가 생긴다. 인터페이스는 외부와의 연결 통로라는 의미를 가지게 되는데, 이렇게 외부에 있는 기능을 사용해야 하는 경우는 Provide 인터페이스, Required 인터페이스와 구분된다. Uses 인터페이스는 이와 같이 외부에 있는 기능을 써야 하는 경우 컴포넌트에서 호출하는 외부 API 들을 의미한다. Uses 인터페이스는 가변성을 설정하기 위한 Required 인터페이스와는 그 목적이 구분되며 컴포넌트가 가지는 의존성을 파악에 유용하게 이용된다.

- Home 인터페이스

홈 인터페이스는 다른 컴포넌트 모델의 구성요소를 반영한 요소로써 이를 통해 컴포넌트의 생명주기를 관리하고 특정 컴포넌트를 조회할 수 있는 메커니즘을 제공한다. 또한, 컴포넌트가 여러 개의 인터페이스를 가질 수 있기 때문에 컴포넌트에 접속할 수 있고 인터페이스들 사이의 네비게이션을 가능하게 하는 관리용 인터페이스로서의 역할을 담당한다.

• 상태 속성

상태 속성은 주로 컴포넌트 환경설정을 위해 쓰이는 추가적이거나 변화하는 기능을 통해 표현되는 값들이다. 상태 속성은 다른 컴포넌트 모델의 구성요소를 반영한 요소이다.

• 컴포넌트 워크플로우 (핵심 참조 모델)

컴포넌트 인터페이스의 기능을 수행하기 위해 한 컴포넌트 내의 클래스들 사이에서 일련의 메시지 흐름이 발생한다. 이러한 일련의 메시지 흐름을 컴포넌트 워크플로우라 한다. 일반적으로 컴포넌트 블랙 박스 형태를 기본으로 한다. 블랙 박스 형태의 컴포넌트에서는 인터페이스를 거치지 않고 컴포넌트 내부를 직접 접근할 수 없다. 컴포넌트의 내부는 객체들의 협력으로 이루어지며, 인터페이스의 호출은 컴포넌트 내부에 메시지 흐름을 발생시킨다.

• 이벤트

이벤트는 다른 컴포넌트 모델의 구성요소를 반영한 요소로써 한 컴포넌트가 생성할 수 있는 표준 사건(이벤트)들과 한 컴포넌트가 요청할 수 있는 이벤트들을 정의한다. 즉, 이벤트는 이벤트를 방출하는 이벤트 소스와 이벤트를 소비하는 이벤트 싱크로 나누어진다. 이벤트 소스는 하나 이상의 관심을 가지는 이벤트 소비자나 이벤트 채널에게 특별한 형태의 이벤트를 방출시키는 연결 포인트를 의미한다. 이벤트 싱크는 특별한 형태의 이벤트를 받는 연결 포인트를 의미한다.

• 컨테이너(컴포넌트 프레임웍)

컨테이너는 컴포넌트가 운영 혹은 작동되는 환경으로서 컴포넌트 프레임워크라 부르기도 한다. 컨테이너에서는 동일 컴포넌트 모델로 만들어진 컴포넌트들이 표준화된 방식으로 메시지를 주고 받는데 필요한 서비스들을 제공한다. 또한, 컨테이너에서는 트랜잭션, 보안, 영

구성 관리, Notification과 같은 시스템 레벨의 서비스를 제공한다. 이렇듯 컨테이너에서 시스템 레벨의 서비스를 제공하기 때문에 컴포넌트에서는 트랜잭션, 보안, 영구성 관리, Notification에 대한 로직을 구현하지 않고 순수 비즈니스 로직 구현에 노력을 기울일 수 있다. 컨테이너는 컴포넌트가 작동되기 위한 실행환경적인 요소로써 컴포넌트 참조 모델에 정의되어야 한다.

• 콜백 인터페이스

컨테이너는 컴포넌트의 콜백 인터페이스를 통해 컴포넌트를 관리한다. 콜백 인터페이스는 컨테이너가 컴포넌트와 연동하기 위해 필요로 하는 인터페이스로서 컨테이너가 컴포넌트를 호출할 수 있도록 컴포넌트가 컨테이너에게 제공하는 API이다. 콜백 인터페이스는 컨테이너와 함께 기존의 컴포넌트 모델에 정의되어 있는 구성요소이다.

• 컨테이너에서 제공하는 인터페이스

컨테이너에서 제공하는 인터페이스는 컴포넌트가 컨테이너와 연동하는데 필요한 인터페이스로서 컴포넌트 참조 모델에 포함된다.

• 배치 정보

구현된 컴포넌트는 실행시의 환경인 컴포넌트 프레임워크에 배치된다. 컴포넌트 프레임워크에는 여러 컴포넌트가 조합되어 하나의 컴포넌트 기반 시스템을 형성하게 된다. 따라서, 배치 시 각 컴포넌트에 대한 배치 정보가 필요하게 된다. 이러한 배치 정보의 예로 EJB의 Deployment Descriptor를 들 수 있다.

• 커스터마이제이션

컴포넌트 참조 모델은 한 컴포넌트의 행위와 속성을 커스터마이즈Customize)할 수 있는 수단을 제공해야 한다. 한 컴포넌트가 재사용성이 높아지기 위해서는 여러 유사 응용 프로그램들 간에 공통적인 부분들이 구현되어 있어야 할 뿐만 아니라 각각의 응용 프로그램마다 필요에 따라 특화될 수 있도록 하는 메커니즘을 제공해야 한다. 커스터마이제이션은 컴포넌트의 재사용성을 향상 시키기 위해 필수적으로 정의되어야 하는 요소이다.

• 가변성

가변성은 효율적인 CBD를 위해 중요한 요소이다. 재사용성은 컴포넌트가 가지는 중요한 존재 동기가 되는데 이러한 재사용성을 제공하기 위해서는 컴포넌트가 하나의 어플리케이션에만 사용되는 것이 아니라 여러 어플리케이션에서 사용될 수 있어야 한다. 여러 어플리케이션에 사용되기 위해서는 각각 어플리케이션 마다 가지는 약간의 차이점에 맞게 컴포넌트를 변화시켜 사용할 수 있어야 한다. 바로 가변성이 이러한 차이점들을 수용하여 컴포넌트가 재사용될 수 있게끔 하는 중요한 메커니즘이다. 제안된 모델에서는 가변성의 종류를 4 가지로 분류하였다. 속성의 타입이나 개수가 다를 경우에는 속성 가변성, 하나의 오퍼레이션 내에서의 로직이 다를 경우에는 로직 가변성, 인터페이스의 오퍼레이션을 실행하기 위해 실행되는 메시지의 흐름이 다를 경우에는 워크플로우 가변성, 그리고 컴포넌트 영구 스키마와 어플리케이션의 영구 스키마가 다를 경우에는 영구 가변성에 속한다.

• 컴포넌트 타입

컴포넌트 타입은 컴포넌트가 외부에게 어느 레벨까지의 가시성을 제공하는 지에 대한 사항과 관련된 구성요소이다. 컴포넌트 타입에는 Black-Box, White-Box, Hybrid 형태가 있다. Black-Box는 컴포넌트의 내부 로직을 숨기며 외부에서 컴포넌트 내부를 직접 접근하지 못한다. 그렇기 때문에 외부의 사용자나 외부 컴포넌트들이 Black-Box 컴포넌트를 이용하려 할 경우 컴포넌트의 인터페이스를 통하여 접근할 수 있다. 반면 White-Box 컴포넌트는 컴포넌트의 내부 로직을 숨기지 않으며 외부에서 컴포넌트 내부를 직접 접근 할 수 있다. 그렇기 때문에 컴포넌트는 특별한 인터페이스를 가지지 않는다. Hybrid 컴포넌트는 일부는 숨기고 일부는 외부에 드러난 컴포넌트를 말한다.

• 컴포넌트와 인터페이스 간의 계약

컴포넌트와 인터페이스 사이의 계약은 인터페이스를 다른 코드 개체들과 컴포넌트들 간의 주된 약정(계약)으로 정의하고 있다. 여기서 다른 코드 객체들은 컨테이너, 어플리케이션 코드, 다른 컴포넌트, 다른 컴포넌트 모델, 그리고 독자적인 서비스를 제공하는 제공자들을 의미한다. 실제적인 컴포넌트라 할 수 있는 컴포넌트 구현 코드는 컴포넌트 인터페이스를 구현한 것으로 컴포넌트와 인터페이스 사이의 규약을 준수해야 한다. 이러한 컴포넌트와 인터페이스 사이의 규약은 구현해야 할 인터페이스 당 하나씩 존재하며, 여러 규약을 반영하여 컴포넌트를 구현하게 된다.

• 컴포넌트 간의 계약

컴포넌트 간의 계약은 컴포넌트 기반의 개발에서, 여러 컴포넌트들이 모여 하나의 어플리케이션을 구축할 때 시스템을 구성하는 컴포넌트의 역할에 대해 명시한다. 즉, 어떠한 어플리케이션을 구축하기 위해서는 어떠한 역할의 컴포넌트들이 조합되어야 하는지를 명시해야 한다.

7. 구성 요소 평가

앞에서 분석한 5개의 대표적인 컴포넌트 모델과 본 논문에서 제안한 핵심 참조 모델과 확장 참조 모델에 대해 구성요소의 측면에서 평가를 수행한다. 본 장에서 수행된 평가를 이용하여 프로젝트 관리자는 컴포넌트

표 3 컴포넌트 모델 간 구성 요소 비교표

구성 요소 \ 컴포넌트 모델	CCM	EJB	.NET	CMU /SEI	Perrone's	제안된 핵심 모델	제안된 확장 모델
컴포넌트	√	√	√	√	√	√	√
Provide 인터페이스	√	√	√	√	√	√	√
Required 인터페이스							√
Uses 인터페이스	√		√		√		√
Home 인터페이스	√	√					√
컴포넌트 내부 클래스					√	√	√
컴포넌트 상태 속성	√		√		√		√
컴포넌트 워크플로우						√	√
이벤트	√		√		√		√
비 기능적인 속성			√		√		√
컨테이너	√	√	√	√	√		√
컨테이너가 제공하는 서비스	√	√	√	√			√
콜 백 인터페이스	√	√			√		√
컨테이너가 제공하는 인터페이스	√	√			√		√
배치 정보		√	√	√			√
가변성							√
커스터마이제이션					√		√
컴포넌트와 인터페이스 간의 계약	√			√			√
컴포넌트 간의 계약	√			√			√

기반의 개발 프로젝트를 수행할 경우, 각각의 프로젝트에 합당한 컴포넌트 참조 모델을 채택할 수 있다. 본 논문에서 제안하는 핵심, 확장 참조 컴포넌트 모델은 필수적인 선택 사항은 아니지만, 이를 이용하여 프로젝트를 진행시켜 나가면 효과적일 것이다.

위의 표 3에서는 3장에서 분석했던 5개의 대표적인 컴포넌트 모델과 본 논문에서 제안한 핵심 참조 모델과 확장 참조 모델의 구성요소에 대한 비교를 수행하였다. 본 논문에서 제안한 핵심 참조 모델은 5개의 대표적인 컴포넌트 모델이 필수적으로 포함하고 있는 최소의 구성요소로만 구성되어 있고, 확장 참조 모델은 5개의 컴포넌트 모델들이 포함하고 있는 구성요소와 추가적으로 몇 개의 구성요소가 더 포함되어 구성된다.

8. 사례 연구

사례 연구를 통해 본 논문에서 제안한 최소, 최대 모델에 대한 검증과 기존의 컴포넌트 모델들과의 비교를 수행한다. 기존의 컴포넌트 모델 가운데서는 EJB 모델을 이용한 사례 연구를 적용한다. 본 논문에서는 뱅킹 컴포넌트 구축 사례 연구 중에서 그림 16에 명시된 수신 업무 부분에 대해 보여주고자 한다. 컴포넌트의 재사용성 높이기 위해 본 사례 연구의 요구사항에는 가변성을 포함한다.

8.1 EJB 모델 적용

EJB 모델을 적용하여 뱅킹 중 수신 컴포넌트를 설계하고 구현하였다. 설계 과정에서 수신 컴포넌트는 그림 17과 같이 DepositMgrBean, DepositTransactionBean, DepositAccountBean의 3개의 빈으로 구성됨을 알 수 있었다. 일반적으로 빈 단위로 하나의 컴포넌트가 되나

요 구 사 항

DP01	계좌신규	고객이 새로운 계좌를 신청하여 계좌번호 및 이율 등의 관련사항을 저장	계좌번호 생성 방식이 다르다
DP02	계좌입금	고객이 맡긴 일정금액을 희망 계좌에 입금	
DP03	계좌출금	고객의 계좌에서 일정금액을 출금	
DP04	계좌해지	고객의 계좌를 해지	
DP05	계좌조회	생성된 계좌를 조회	
DP06	이자계산	각각의 고객에게 이자를 계산 및 입금	이자를 부여하는 대상에 대한 기준이 다르다.

그림 16 뱅킹 도메인의 요구사항

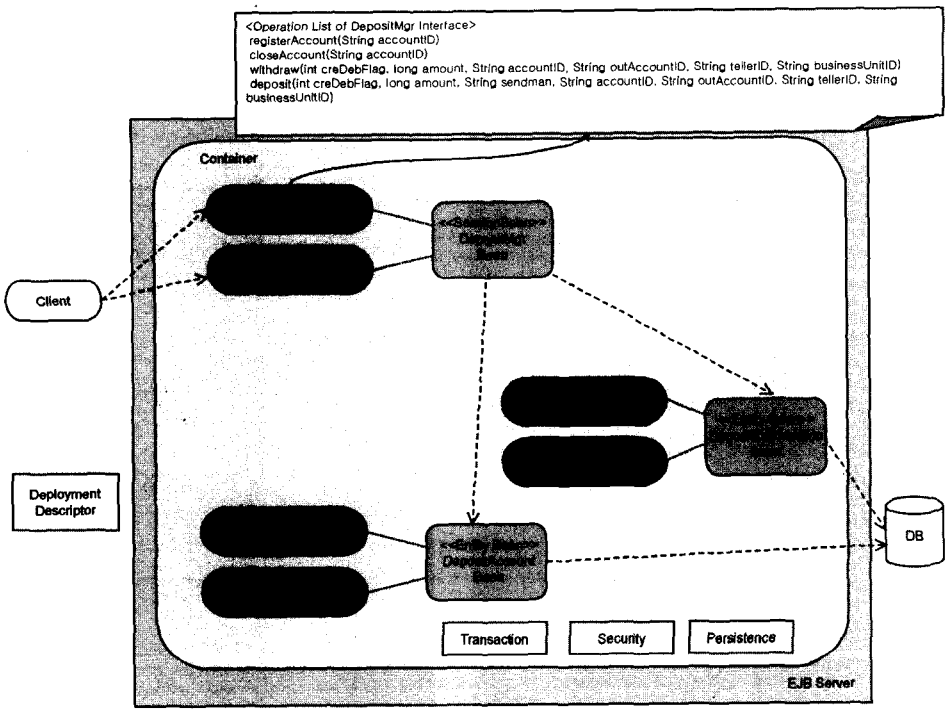


그림 17 EJB 모델을 적용한 수신 컴포넌트

사례 연구 시 컴포넌트 하나가 정확히 빈 하나에 매핑이 되지 않아 DepositMgrBean 세션 빈을 통해 DepositTransactionBean과 DepositAccountBean을 이용하여 수신 기능을 제공하도록 설계하였다. Operation List of DepositMgr Interface는 EJB 수신 컴포넌트가 제공하는 기능성을 의미한다.

본 사례 연구에서의 수신 컴포넌트는 계좌 번호 생성 방식과 이자를 부여하는 대상에 대한 기준에 있어 가변성을 포함하고 있으나 EJB 모델은 계좌 번호 생성 방식과 이자 부여 대상 기준에 대한 가변성을 설계하기 위한 가변성 종류, 가변점, 가변치, 가변성 범위에 대한 정보를 설계하기 위한 요소와 계좌 번호 생성 방식 설정, 이자 부여 대상 기준 설정을 위한 Required 인터페이스와 이들을 커스터마이즈 시키기 위한 요소를 EJB 모델에서 제공하지 않아 가변성을 EJB 수신 컴포넌트에 구현하지 못하였다. 또한, EJB 모델은 Uses 인터페이스가 없어 DepositMgrBean에서 DepositAccountBean, DepositTransactionBean의 어떠한 오퍼레이션을 필요로 하는지에 대한 정보를 알기가 어려웠다. 또한, 워크플로우라는 요소가 없어 컴포넌트가 제공하는 기능에 대한 런 타임 경로를 나타내는데 어려움이 있었다.

8.2 핵심 참조 모델 적용

핵심 참조 모델을 적용한 수신 컴포넌트는 아래의 그림 18과 같이 모델링 될 수 있다. Deposit Component 내에는 DepositController, Transaction, DepositTX, Account, DepositAccount의 5개 클래스가 있으며 IDeposit 이라는 인터페이스를 통해 기능성을 제공한다. IDeposit 내에 정의된 기능성은 내부 클래스들 간의 워크플로우를 통해 기능성을 제공하게 된다. 워크플로우는 그림에서는 나타나 있지 않지만, IDeposit 내의 각각의 오퍼레이션들에 대한 시퀀스 다이어그램 형태로 명시된다.

핵심 참조 모델을 적용한 수신 컴포넌트 또한 EJB 모델과 같이 가변성을 구현하기 위해 필요한 요소를 제공하지는 않는다. 따라서, 본 수신 컴포넌트에서도 계좌 번호 생성 방식과 이자 부여 대상 기준에 대한 가변성을 지원하지는 못한다. 그러나, 인터페이스와 내부 클래스와 워크플로우라는 요소를 통해 가장 간결하면서도 핵심적인 수신 컴포넌트를 구현할 수 있었으며 핵심적인 정보만을 가지고 있는 만큼 이 수신 컴포넌트가 뱅킹의 다른 컴포넌트인 고객 및 상품 컴포넌트와 쉽게 호환이 되었다.

8.3 확장 참조 모델 적용

확장 참조 모델을 적용한 수신 컴포넌트는 아래의 그림 19과 같이 여러 개의 내부 클래스와 'IDeposit'-Provide 인터페이스, 'IDepositCustomization' Required

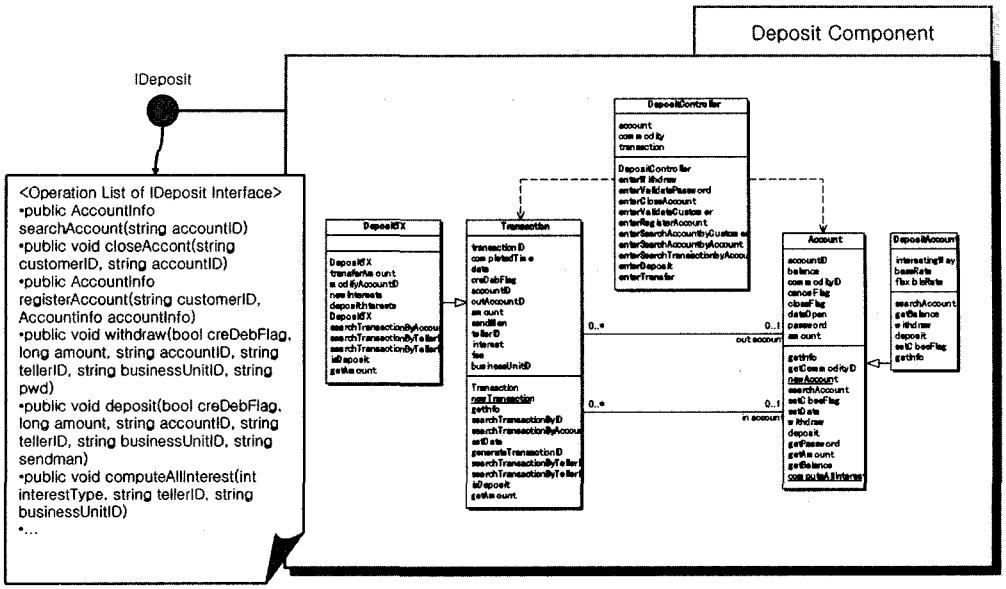


그림 18 핵심 참조 모델을 적용한 수신 컴포넌트

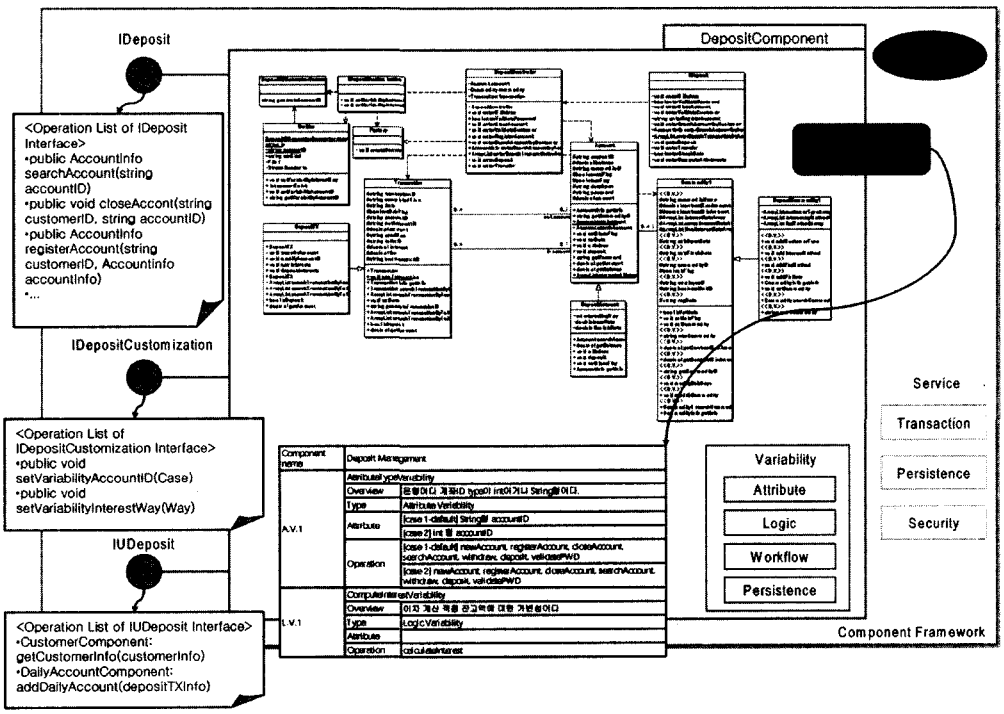


그림 19 확장 참조 모델을 적용한 수신 컴포넌트

인터페이스, 'IUDeposit'Uses 인터페이스를 가지며 'IDepositCustomization' 인터페이스를 통해 계좌 번호 생성 방식과 이자를 부여하는 대상에 대한 기준을 설정한다. 요구된 가변성을 커스터마이즈하기 위한 커스터마이즈 스킴도 제공되며, 아래의 그림에서는 나타나지 않았지만, 가변성 설정 값을 저장하고 있는 컴포넌트의 상태

정보, 여러 시스템 서비스를 제공하기 위한 정보, 배치 정보, 인터페이스의 기능성을 제공하기 위한 워크플로우 정보가 함께 모델링되어 구현에 반영되었다.

확장 참조 모델을 적용한 컴포넌트는 본 사례 연구의 요구사항에서 정의한 요구사항을 충분히 만족하였다. 모델에서 정의된 4가지 가변성 타입에 따라 모델링 하고자 하는 가변성 정보를 요구사항에 맞게 설계할 수 있었으며 Required 인터페이스를 통한 가변성 설정, 상태 속성을 통한 설정 값 저장, 커스트마이즈 스킴을 통한 수신 컴포넌트의 커스터마이즈 방법을 모델에 반영할 수 있었다. 또한, Uses 인터페이스를 통해 본 수신 컴포넌트에서 외부의 어떠한 API를 필요로 하는지, 즉, 다른 컴포넌트로의 의존성 정보를 알 수 있었다.

8.4 결 론

사례 연구를 통해 EJB 모델, 핵심 참조 모델, 확장 참조 모델을 사용할 때의 특성들을 알아 볼 수 있었다. EJB 모델을 사용할 시에는 모델을 지원하는 구현 환경이 잘 되어 있는 반면, 컴포넌트가 하나의 빈에 정확히 매핑이 되지 않으므로 인한 어려움과 모델 자체에서 가변성을 구현하기 위한 요소들을 제공하지 않아 컴포넌트 재사용성이 떨어진다는 결점을 가지고 있다. 핵심 참조 모델은 최소한의 요소만을 가지고 있어 요구사항을 지원하는 정도는 낮고 가 변성을 지원하지 못해 컴포넌트의 재사용성을 비교적 낮지만, 다른 모델과의 호환성이 높고 컴포넌트라면 최소한 만족해야 하는 기준을 설정해주고 있다. 확장 참조 모델은 다양한 요구사항을 충분히 만족할 수 있도록 풍부한 장치들을 제공하며 특별히 가변성 관련 요소들을 통해 컴포넌트의 재사용성이 높은 컴포넌트를 구축할 수 있다는 장점이 있다.

9. 결 론

CBD의 재사용성과 효율성은 학계와 산업계에서 널리 인정되고 있다. 이러한 CBD의 적용에 앞서 현재까지 제안된 다양한 컴포넌트 참조 모델 가운데 프로젝트에 적합한 참조 모델을 선택해야 하는데, 이를 위해 본 논문에서는 컴포넌트 모델들 간에 비교를 수행하였다. 현재까지는 컴포넌트 참조 모델 자체를 비교하기 위한 비교 기준의 제시가 명확하지 않았으나, 본 논문에서는 이를 위한 열 가지의 비교 기준을 제시하였다. 다양한 모델들의 비교를 위해 명시적으로 컴포넌트 모델이 제시되지 않은 컴포넌트 모델의 경우, 비교를 위해 각각의 메타 모델을 명시적으로 도출하였다.

컴포넌트 참조 모델의 표준화는 개발된 컴포넌트들 간에 원활한 연동의 가능성을 높여주는 장점을 가지나 현재까지 제안된 모델들 가운데 표준화된 모델의 제시가 부족하였다. 따라서, 본 논문에서는 모든 컴포넌트 모델들이 필수적으로 만족해야 하는 핵심 참조 모델과 가장 확장된 형태의 확장 참조 모델을 제안하였다. 마지막으로 본 논문에서 분석한 다섯 개의 참조 모델과 제안된 두 개의 모델 간의 구성 요소 측면에서의 비교 평가를 수행하고 사례 연구를 통해 제안된 모델을 검증하였다.

참 고 문 헌

[1] Object Management Group, "CORBA Components, Version 3.0," OMG, June. 2002.

[2] Sun Specification, "Enterprise JavaBeansTM Specification, Version 2.1," Sun Microsystems, June 14, 2002.

[3] Microsoft, http://msdn.microsoft.com/library/, 2003.

[4] Bachman F., Bass L., Buhman C., Comella-Dorda S., Long F., Robert J., Seacord R., and Wallnau K., "Volume II: Technical Concepts of Component-Based Software Engineering," CMU/SEI-2000-TR-008, May 2000.

[5] Perrone, P., Building Java Enterprise Systems with J2EE, Sams Publishing, 2000.

[6] Ruiz, D., "CORBA & Components," http://www.ditec.um.es/~dsevilla/ccm/, Nov. 12, 2000.

[7] Cobb, E., "CORBA Components: The Industry's First Multi-Language Component Standard," BEA Systems, June 16, 2000.

[8] Object Management Group, "OMG Unified Modeling Language Specification," OMG, Sept, 2001.

[9] Roman, E., Ambler, S., and Jewell, T., Mastering Enterprise JavaBeans, Wiley, 2002.

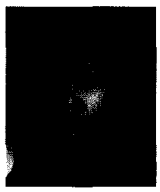
[10] Heineman, G. T., and Council, W T., Component-based Software Engineering, Addison Wesley, 2001.

[11] WhiteHead, K., Component-based Development: Principles and Planning for Business Systems, Addison-Wesley, 2002.

[12] Crnkovic, I., and Larsson, M., Building Reliable Component-Based Software Systems, Artech House, 2002.

[13] Szyperski, C., Gruntz, D., and Murer, S., Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 2002.

[14] Coplien J., Hoffman D., and Weiss D., "Commonality and Variability in Software Engineering," IEEE Software, pp37-45, November 1998.



허 진 선

2001년 숭실대학교 컴퓨터학부 공학사 2003년 숭실대학교 컴퓨터학과 공학석사 2003년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 컴포넌트 개발 방법론, 모델 기반 아키텍처, 제품 계열 공학

김 수 동

정보과학회논문지 : 소프트웨어 및 응용 제 31 권 제 1 호 참조