특집

CBD 프레임워크

전 태 웅[1)]

목 차

1. 서 론

소프트웨어를 여러 응용 시스템의 개발에 재사용 가능한 컴포넌트의 형태로 개발하는 기술이 빠르게 발전하고 있다[1]. 특히, 컴포넌트의 설계 규격, 연결 방식 및 하부구조를 규정하는 컴포넌트 모델들과 이들의 실행 환경을 제공하는 플랫폼들이 다수 소개되어 있다. 예를 들면, OMG의 Corba 컴포넌트 모델[2], Microsoft의 COM+, 그리고 Sun Microsystems의 Java 컴포넌트 모델(Java Beans, EJB[3] 등)이 대표적인 컴포넌트 모델들이다. 이들은 .NET, Sun ONE 등과 같은 웹 기반의 컴포넌트 기술들로 발전하고 있다. 이러한 컴포넌트 기술의 발전에 힘입어 컴포넌트 기반 개발(CBD)이 보편적인 소프트웨어 개발 방식으로 자리 잡아가고 있다.

CBD는 소프트웨어를 컴포넌트 단위로 분해, 생성, 합성, 개조함으로써 개발하는 방식이다. 효과적인 CBD를 위해서는 컴포넌트의 생성과 합성이 잘 정의된 컴포넌트 표준 모델과 합성 아키텍처를 기반으로 이루어져야 한다. 그리고 CBD의 개발 과정 전반과 여러 측면에 걸쳐 "관심의 분리(separation of concerns)" 원칙이 잘 적용되어야 한다. 우선, 컴포넌트의 생성과 컴포넌트의 합성 작업이 분리되어야 한다. 잘 정의된 합성 아키텍처는 컴포넌트의 개별적인 구현과 구현된 컴포넌트들의 합성 작업이 서로 독립적으로 이루어질 수 있게 한다. 컴포넌트의 생성 시에는 컴포넌트의 명세와 구현, 그리고 개별 컴포넌트들에 분산, 구현될 부분과 컴포넌트 플랫폼에서 공통적으로 지원될 부분이 잘 분리되어야 한다. 또한 컴포넌트의 합성 시에는, 합성 아키텍처의 명세와 구현이 분리되고, 연결 대상(인터페이스)과 연결 관계(커넥션)가 서로 구분되도록 정의되어야 합성의 유연성이 높아진다. CBD는 컴포넌트 합성에 의한 소프트웨어 재사용을 지원한다. 높은 재사용성을 위해서는 합성 시 변경될 부분들이 고정 부분들과 분리되어 있어야 한다.

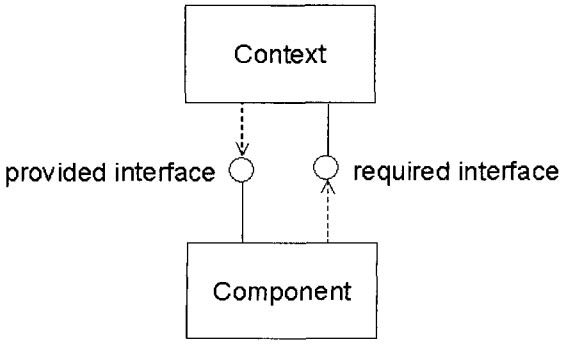
위와 같이 "관심의 분리" 원칙이 충실하게 적용된 CBD는 명세, 구현, 설치 및 실행의 전 과정에 걸쳐 다양한 측면과 형태를 갖는 개발 결과물들을 산출한다. 본 고에서는 CBD를 통한 컴포넌트의 생성과 합성에 관여된 핵심 개념들, 산출되는 결

1) 고려대학교컴퓨터정보학과 교수

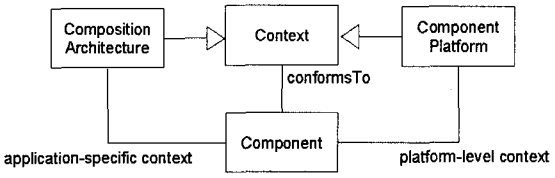
과물들, 그리고 이들의 상호 관계들을 고찰한다.

2. 컴포넌트 컨텍스트

컴포넌트가 합성되어 작동되는 환경을 컨텍스트라고 부른다. 컴포넌트는 인터페이스를 통해 컨텍스트와 상호 작용한다. 컴포넌트의 인터페이스는 컴포넌트 자신이 외부에게 제공하는 인터페이스(provided interface)와 외부로부터 제공이 필요한 인터페이스(required interface)의 두 가지 유형으로 구분된다. 컴포넌트는 명시된 제공 인터페이스와 요구 인터페이스만을 통하여 외부 환경과 연결, 상호작용한다(그림 1). 컴포넌트의 컨텍스트는 플랫폼 수준의 컨텍스트인 컴포넌트 플랫폼과 응용 수준의 컨텍스트인 컴포넌트 합성 아키텍처의 두 가지 컨텍스트로 나뉘어진다(그림 2). 컴포넌트 플랫폼은 합성, 설치되는 컴포넌트들에게 공통적으로 필요한 실행 환경과 시스템 서비스들을 제공한다. 합성 아키텍처는 컴포넌트가 사용된 응용 시스템의 합성 구조를 나타낸다. 합성 아키텍처는 컴포넌트가 응용 시스템의 일부로 작동하는데 필요한 다른 컴포넌트들과의 연결 관계와 상호작용 관계를 정의한다. 컴포넌트가 정확하게 작동하기 위해서는 컴포넌트 플랫폼과 합성 아키텍처가 강제하는 규칙과 제약 조건들을 준수해야 한다.



(그림 1) 컴포넌트 컨텍스트



(그림 2) 컨텍스트의 유형

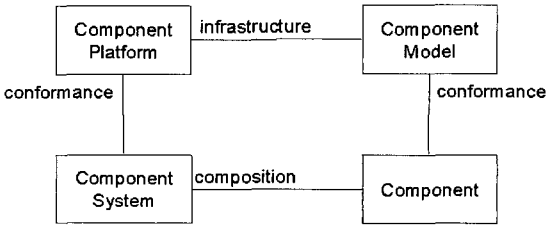
2.1 컴포넌트 플랫폼

컴포넌트 플랫폼은 컴포넌트 모델을 구현한 컴포넌트 하부구조다. 컴포넌트 모델은 컴포넌트들의 합성과 상호작용 표준을 의미한다. 컴포넌트 모델은 컴포넌트의 구현, 명명(naming), 상호운용성, 커스터마이제이션, 진화, 및 설치(deployment)를 위해 필요한 공통 규격들을 정의한다. 컴포넌트 모델은 두가지 수준의 컴포넌트 표준을 제공한다.

1) 개별 컴포넌트의 구현 시 준수해야 할 설계 규격
2) 컴포넌트 시스템의 컴포넌트들이 상호작용 시 공통적으로 준수해야 하는 행위

컴포넌트 모델을 준수하는 컴포넌트들이 합성된 컴포넌트 시스템은 컴포넌트 모델을 구현한 컴포넌트 플랫폼이 제공하는 실행 지원 환경에 맞게 구현된다(그림 3). 컴포넌트 플랫폼이 컴포넌트들에게 제공하는 런타임 서비스들은 다음과 같다.

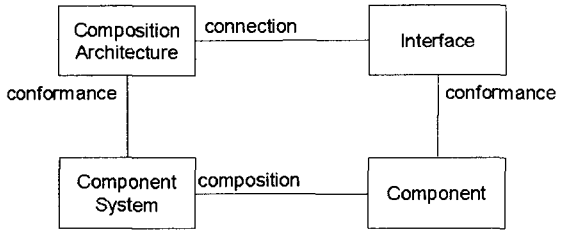
1) 기본 서비스: 통신, 생명주기, 저장, 병행처리
2) 불특정 다수 도메인에 유용한 서비스: 사용자 인터페이스, 정보관리, 시스템 관리
3) 특정 도메인 서비스: CIM, telecommunication



(그림 3) 컴포넌트 플랫폼

2.2 합성 아키텍처

컴포넌트 플랫폼의 준수는 컴포넌트의 정확한 구현과 합성에 필요 조건이지만 충분하지는 않다. 컴포넌트들이 응용 시스템에서 정확하게 동작하기 위해서는 구현, 합성된 컴포넌트들이 응용에 맞게 정의된 합성 아키텍처를 준수하여야 한다. 합성 아키텍처는 응용 시스템을 구성하는 컴포넌트들의 인터페이스 연결 구조를 의미한다. 합성 아키텍처를 준수하는 컴포넌트 시스템은 합성 아키텍처에 연결된 인터페이스들을 준수하는 컴포넌트들을 구현, 합성함으로써 만들어진다(그림 4). 즉, 컴포넌트 시스템의 합성 아키텍처는 시스템에 합성된 컴포넌트들이 준수하는 인터페이스들의 연결 관계들로 정의되며 합성되는 컴포넌트의 구현과는 독립적이다.

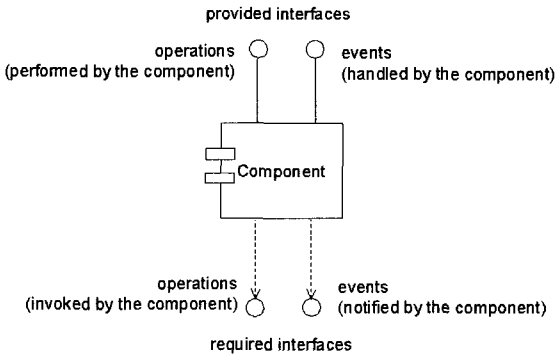


(그림 4) 합성 아키텍처

3. 컴포넌트 인터페이스

3.1 인터페이스의 유형

컴포넌트의 외부 접속 부위인 인터페이스는 컴포넌트가 외부와 상호 작용 시 사용되는 오퍼레이션들과 이벤트들을 정의한다. 컴포넌트의 외부로부터 호출 또는 수신받아 컴포넌트에 의해 처리되는 오퍼레이션/이벤트들이 정의된 인터페이스를 제공 인터페이스라 부른다. 컴포넌트가 호출 또는 발신하여 외부에 의해 처리되는 오퍼레이션/이벤트들이 정의된 인터페이스는 요구 인터페이스라 부른다(그림 5). 컴포넌트는 제공 인터페이스를 통하여 자신이 수행하는 서비스들을 외부에게 제공하고 요구 인터페이스를 통하여 필요한 서비스들을 외부로부터 제공받는다.



(그림 5) 컴포넌트 인터페이스

컴포넌트들의 상호작용이 오퍼레이션 호출이 아닌 이벤트나 메시지의 송수신 방식으로 이루어지는 경우, 그러한 상호작용을 지원하는 인터페이스는 이벤트/메시지의 흐름 방향에 따라 입력 인터페이스와 출력 인터페이스로 구분하여 정의할 수 있다. 입/출력 인터페이스는 오퍼레이션 호출 방식의 상호작용도 지원할 수 있다. 예를 들면, 입/출력 인터페이스를 갖는 컴포넌트가 상대방 컴포넌트의 오퍼레이션을 호출할 때 오퍼레이션의 호출은 출력 인터페이스를 통하여, 오퍼레이션 처리 결과의 수신은 입력 인터페이스를 통하여 이루어질 수 있다.

서로 관련이 높은 오퍼레이션/이벤트들을 제공/요구 인터페이스 또는 입/출력 인터페이스로 구분하지 않고 함께 그루핑하여 단일한 인터페이스로 정의하면 컴포넌트들의 연결 관계의 복잡도를 크게 줄일 수 있다. 그렇게 정의한 인터페이스를 포트 인터페이스라고 부른다[4].

컴포넌트가 갖는 인터페이스들이 제공/요구 인터페이스, 입/출력 인터페이스, 또는 포트 인터페이스 중 어느 형태로 정의되는지는 컴포넌트 모델의 제약을 받는다. 일반적으로, 합성 아키텍처에 연

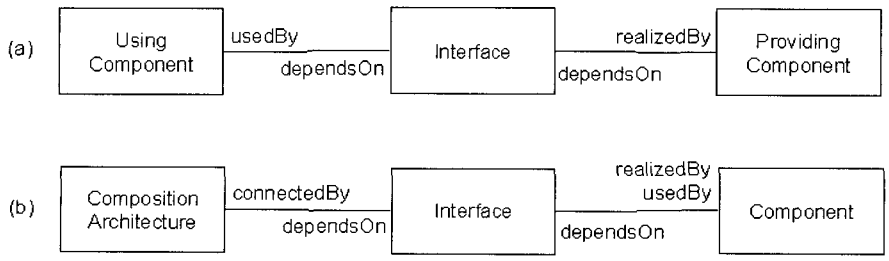
결된 인터페이스들의 컴포넌트들은 컴포넌트 플랫폼이 지원하는 컴포넌트들보다 추상화 수준이 높다. 이 경우 아키텍처 수준의 컴포넌트 모델과 실행 환경인 컴포넌트 플랫폼을 준수하는 컴포넌트 모델을 다르게 채택할 수 있다. 예를 들면, 아키텍처 수준의 컴포넌트 모델은 포트 인터페이스 방식을 따르고 플랫폼 수준의 컴포넌트 모델은 제공/요구 인터페이스 방식을 따른다. 그리고 이들 사이의 의미적인 격차는 매핑 규칙에 의거한 정제, 변환을 통하여 좁힌다.

3.2 인터페이스의 역할

인터페이스는 컴포넌트의 구현과 컴포넌트의 연결을 분리한다. 컴포넌트의 구현은 인터페이스에만 의존한다(그림 6-a). 즉, 컴포넌트는 이와 상호작용할 다른 컴포넌트들의 구현에 의존하지 않고 자신에 정의된 인터페이스들만을 참조하여 이들을 준수하도록 구현하면 된다. 컴포넌트들의 연결 관계도 인터페이스에만 의존하여 정의된다(그림 6-b). 즉, 컴포넌트들의 합성 아키텍처는 합성 대상 컴포넌트들의 개별적인 구현에 의존하지 않고 컴포넌트들의 인터페이스들만 참조하여 이들 사이의 연결 관계들로 정의하면 된다[4]. 합성 아키텍처를 준수하는 컴포넌트 시스템은 합성 아키텍처에 정의된 인터페이스들의 연결 관계들과 이에 참조된 인터페이스들을 준수하는 컴포넌트들을 각각 독립적으로 구현, 통합함으로써 구현할 수 있다. 이와 같이 인터페이스는 컴포넌트들의 생성과 합성을 서로 독립적으로 가능하게 하는 중요한 역할을 갖는다.

3.3 인터페이스 명세

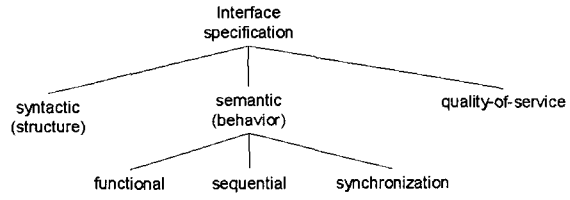
인터페이스에만 의존한 컴포넌트 연결을 위하여 인터페이스 명세에 기본적으로 필요한 것은 인터페이스를 구성하는 오퍼레이션/이벤트들의 시그너처들로 표현되는 구문 정의다. 인터페이스의 구문 명세는 컴포넌트들의 일차적인 연결 가능성을 결정한다. 하지만 인터페이스에만 의존하여 구현, 연결된 컴포넌트들이 서로 정확하게 상호작용 위해서는 오퍼레이션/이벤트들의 시그너처 명세만으로는 불충분하다. 연결의 정확성을 위해서는 인터페이스의 구문 매칭만으로는 부족하고 인터페이스를 통해 이루어지는 행위 즉, 의미 매칭도 함께 고려되어야 한다. 인터페이스의 의미는 기능적 행위뿐만 아니라 동기화나 서비스 품질 등과 같은 기능 외적인 성질들도 포함한다[5]. Beugnard는 Meyer가 제시한 '계약에 의한 설계'(Design-by-Contract) [6] 개념을 확장, 응용하여 컴포넌트 명세에 포함될 성질들을 컴포넌트들 사이의 협상 가능 정도에 따라 4가지 수준의 계약들로 구분하였다[7]. Beugnard가 제시한 4개의 계약 수준은 계약 내용에 대한 협상의 여지가 가장 적은 구문 수준으로부터, 행위 수준, 동기화 수준, 그리고 서비스 품질 수준으로 계약 협



(그림 6) 인터페이스 의존관계

상의 여지가 많아진다.

(그림 7)은 구문 명세, 의미 명세, 그리고 서비스 품질 명세의 3가지로 Beugnard가 제시한 컴포넌트 계약 수준들을 토대로 한 인터페이스 명세 범위를 보여준다. (그림 7)에서 의미 명세는 Beugnard의 행위 수준의 계약 명세를 기능적 행위와 순차적 행위로 구분하고, 여기에 동기화 수준의 계약을 포함한다. 기능적 행위는 인터페이스의 개별 오퍼레이션/이벤트 처리에 기대되는 사전, 사후, 및 불변 조건으로 표현할 수 있다. 순차적 행위는 일단의 오퍼레이션/이벤트들에 기대되는 처리 순서를 의미하며 상태 전이 조건들로 표현할 수 있다. 동기화 행위는 병행 처리가 허용되는 오퍼레이션/이벤트들에 대한 상호배제와 같은 동기화 조건으로 표현할 수 있다. 서비스 품질(quality-of-service)은 서비스에 결부된 성능, 정확도, 신뢰도 등과 같은 기능 외적 성질들을 의미한다. 서비스 품질의 계약 명세는 응답 시간, 단위시간 처리량(throughput), 결과값 정밀도(precision), MTBF(mean time between failures) 등과 같은 정량적인 품질 속성 값들로 정의된다.

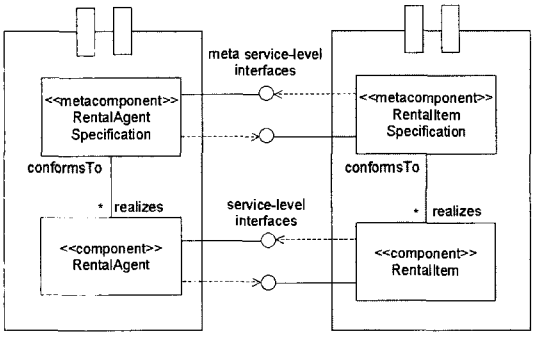


(그림 7) 인터페이스 명세의 구성 요소

3.4 메타 인터페이스

계약의 협상이란 계약 당사자 컴포넌트들이 상호작용에 필요한 계약 내용을 사전에 고정하지 않고 서비스를 주고 받기 전에 상호 조정하여 결정함을 의미한다. 계약의 협상이 가능하려면 컴포넌트가 자신 및 상대방이 지원할 수 있는 계약의 범위를 인지하고 조정할 수 있어야 한다. Beugnard는 그러한 컴포넌트를 계약인지 컴포넌트(contract-aware component)라고 지칭하였다[7]. 컴포넌트들이 계약을 인지하고 조정하려면 계약 명세 정보를 주고 받고 처리할 수 있어야 한다. 이를 위해서는 컴포넌트가 서비스 레벨의 통상적인 인터페이스 외에 서비스 자체에 대한 정보의 접근을 위한 인터페이스 즉, 메타서비스 수준의 인터페이스를 지원해야 한다.

(그림 8)은 계약 인지 능력을 갖춘 렌털 시스템 컴포넌트의 인터페이스를 보여준다. (그림 8)에서 RentalAgent 컴포넌트와 RentalItem 컴포넌트는 각각 자신에 대한 컴포넌트 메타 정보를 구성 요소로 갖는다. 그리고 서비스 처리를 위한 통상적인 인터페이스 외에 메타 정보를 교환할 수 있는 인터페이스를 지원한다. 메타 서비스 수준의 인터페이스는 자신의 계약 명세를 외부로 제공하고 외부로부터 상대방 계약 명세를 요청한다. 그리고 렌털 서비스 처리의 계약 내용을 상호 조정, 합의할 수 있는 능력을 제공한다.

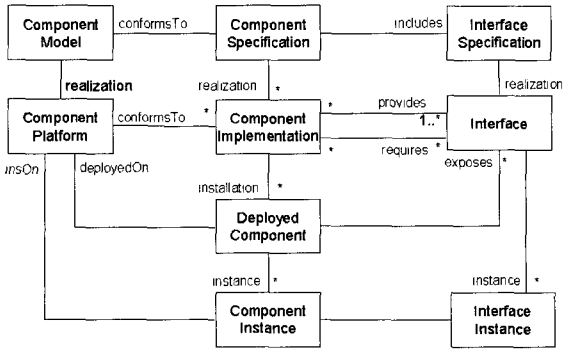


(그림 8) 메타 인터페이스

4. 컴포넌트 생성 프레임워크

컴포넌트의 생성은 명세, 구현, 설치 및 실행 시점에 따라 다양한 측면을 갖는다(그림 9). 컴포넌트 명세는 인터페이스 명세를 포함하여 외부로 드

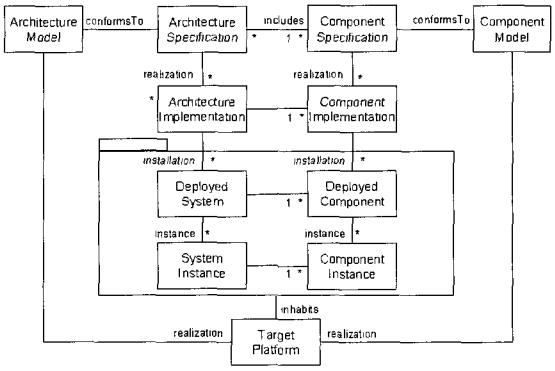
러난 컴포넌트의 접속과 행위에 대한 명세다. 컴포넌트 구현은 컴포넌트 명세를 컴포넌트 플랫폼에 설치 가능한 형태로 실현한 것이다. 컴포넌트 구현은 인터페이스 명세에 정의된 제공 인터페이스와 요구 인터페이스를 갖는다. 컴포넌트 명세는 컴포넌트 모델을 준수하고, 이에 대응하는 컴포넌트 구현은 컴포넌트 플랫폼을 준수하여야 한다. 구현된 컴포넌트는 이를 실행할 플랫폼에 설치된다. 플랫폼에 설치되어 실행 가능한 상태인 컴포넌트를 설치 컴포넌트(deployed component)라고 부른다. 컴포넌트에 구현된 기능은 설치 컴포넌트로부터 생성된 컴포넌트 인스턴스들이 수행한다. 컴포넌트 플랫폼은 구현된 컴포넌트의 설치 및 실행 환경을 제공한다. 컴포넌트가 설치되는 플랫폼과 실행되는 플랫폼은 보통 동일하지만, 다를 수도 있다. 예를 들면, 애플릿 형태의 컴포넌트는 웹 서버에 설치되지만 실행 플랫폼은 웹 클라이언트인 애플릿 컨테이너다.



(그림 9) 컴포넌트 생성 측면

5. 컴포넌트 합성 프레임워크

컴포넌트들의 합성도 컴포넌트의 생성과 마찬가지로 명세, 구현, 설치 및 실행 시점에 따른 여러 측면들이 있다. (그림 10)은 컴포넌트 합성 아키텍처의 여러 측면들과 이에 대응하는 컴포넌트 생성 측면들과의 상호 관련을 보여준다. (그림 10)에서 아키텍처 명세는 이와 별도로 정의된 컴포넌트 (인터페이스) 명세들을 참조하여 컴포넌트들의 연결 및 상호작용 관계를 정의한다. (그림 10)에서와 같이, 아키텍처 명세는 아키텍처 모델을 준수하고, 컴포넌트 명세는 컴포넌트 모델을 준수한다. 아키텍처 구현은 아키텍처 명세에 정의된, 컴포넌트(인터페이스)들의 연결 관계들을 실현하는 glue code이다. 아키텍처 구현과 이에 참조된 인터페이스들의 컴포넌트 구현들이 합쳐져서 컴포넌트 시스템이 구현된다. 구현된 컴포넌트 시스템은 해당 플랫폼에 설치, 실행된다. 컴포넌트 시스템이 설치될 때, 아키텍처 구현이 합성 대상 컴포넌트들과 분리된 형태로 설치될 수 있다. 이 경우, 구현된 아키텍처(즉 glue code)와 컴포넌트가 서로 다른 플랫폼에 설치될 수 있고, 컴포넌트들의 바인딩 시점도 유연하게 결정할 수 있게 된다.



(그림 10) 컴포넌트 합성 측면

6. 결 론

컴포넌트의 설계 표준을 정의하는 컴포넌트 모델들과 이들의 실행 환경을 제공하는 미들웨어들의 보급에 힘입어 응용 컴포넌트들과 응용 컴포넌트 시스템들의 개발이 활성화될 수 있는 여건이 현재 조성되어 있다. 향후의 컴포넌트 산업은 응

용 영역 별로 도메인 지식을 도메인 분석 모델, 도메인 아키텍처, 응용 프레임워크, 응용 컴포넌트 라이브러리 등의 형태로 축적하는 방향으로 발전할 것으로 예상된다. 이와 같이 컴포넌트 기반 소프트웨어 개발의 응용 범위가 방대해지고 응용 수준이 고급화함에 따라 CBD의 핵심원리, 기본 틀, enabling technology, 그리고 발전 추세를 잘 이해하고 이를 효과적으로 활용할 수 있는 능력을 갖추는 것이 더욱 중요해지고 있다. 본 글에서는 CBD의 기본 틀 구조를 이해하는데 도움이 되는 핵심 개념들과 이들의 상호 관계들을 기술하였다.

참고문헌

[1] C. Szyperski, Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 1998

[2] CORBA Components, Version 3.0, OMG, June 2002

[3] Sun Microsystems, Enterprise JavaBeans Specification, Version 2.0, May 2000

[4] D. C. Luckham, J. Vera, and S. Meldal, "Key Concepts in Architecture Definition Languages", Foundations of Component-Based System, G. T. Leavens and M. Sitaraman (Editors), Cambridge Univ. Press, 2000

[5] F. Bachmann, L. Bass, C. Buhman, et al., Volume II: Technical Concepts of Component-Based Software Engineering, 2/e, Technical Report, CMU/SEI-2000-TR-008, May 2000

[6] B. Meyer, "Applying Design by Contract", IEEE Computer, October 1992, pp. 40-52

[7] A. Beugnard, J. Jezequel, N. Plouzeau, and D. Watkins, "Making Components Contract Aware", IEEE Computer, July 1999, pp. 38-45

저자약력

전 태 웅

1981년 서울대학교 계산통계학과 (학사)

1983년 서울대학교 계산통계학과 계산학 전공 (석사)

1992년 Illinois Institute of Technology, Computer Science (박사)

1983년~1987년 금성통신 주임연구원

1992년~1995년 LG산전 책임연구원

1995년- 현재 고려대학교 컴퓨터정보학과 교수

관심분야: 소프트웨어 테스팅, 소프트웨어 아키텍처, 객체지향 방법론, 컴포넌트 기반 개발

이 메 일: jeon@korea.ac.kr