특 집

컴포넌트 기반 미들웨어 기술

윤 은 영*, 윤 용 익**, 윤 석 환***

● 목 차 ●

1. 서 론

최근 인터넷과 초고속 정보통신망을 비롯한 여러 통신망의 구축이 확산되고 다양한 플랫폼(platform)이 통신망을 통하여 연계 이용됨에 따라 시스템의 환경이 분산 시스템 환경으로 변화하게 되었다. 그러나, 분산 환경에서 작업을 수행하기 위해서는 서로 다른 운영체제, 이종의 통신 프로토콜을 통합하여 단일한 사용자 환경처럼 만들어 주는 것이 중요하다.

미들웨어(Middleware)는 클라이언트/서버 환경에서 클라이언트와 서버 사이의 "/" 역할을 담당하는 소프트웨어라 할 수 있다. 즉, 클라이언트와 서버를 연결하기 위하여 클라이언트와 서버 중간에 위치하면서 분산 환경의 클라이언트와 서버가 단일 시스템 환경에서처럼 운영체제, 데이터베이스와 애플리케이션들을 연결해주는 소프트웨어를 말한다[2].

이러한 미들웨어는 분산 환경에서 애플리케이션과 애플리케이션, 애플리케이션과 데이터베이스, 데이터베이스와 데이터베이스를 연결함으로써, 분산 애플리케이션 환경을 하나의 로컬 애플리케이션 환경처럼 운용할 수 있도록 해준다.

또한, 다양한 하드웨어, 운영체제, 네트워크, 데이터베이스 차이에 의해 발생되는 데이터 변환 부분과 클라이언트와 서버간의 통신 부분을 담당함으로써, 하드웨어나 소프트웨어와는 무관하게 네트워크 상에서의 데이터 송수신을 가능하게 하는 등 상호 운용성을 제공하는 장점이 있다. 이에 따라 분산 환경의 다양한 애플리케이션들이 미들웨어 기술을 적용하여 개발하는 사례가 계속적으로 증가하고 있다.

미들웨어는 분산 환경의 클라이언트와 서버 사이에서 메시지 전송 서비스, 정보 서비스, 통제 서비스, 시스템 관리 서비스 등 다양한 서비스를 제공하며, 애플리케이션들의 특성에 따라 그 기능과 구현 환경에 적합한 미들웨어들을 적용하여 개발하는 것이 중요하다.

지금까지 객체와 컴포넌트 개념을 적용한 다양한 미들웨어 기술들이 개발되고 있다. 그러나 기존의 객체 지향 기술을 도입하여 개발한 미들웨어들은 재사용성과 확장성 등에서 문제점이 나타나고 있어 컴포넌트 개념을 도입한 미들웨어 기반 기술에 대한 관심이 높아지고 있다.

* 숙명여자대학교 정보과학부 박사과정
** 숙명여자대학교 정보과학부 교수
*** 정보통신연구진흥원 책임 연구원

분산 환경의 애플리케이션에 적용되어지는 대표적인 미들웨어 기술로 분산된 객체들의 요청을 연결해 주는 CORBA(Common Object Request Broker Architecture)가 있다. CORBA는 분산 컴퓨팅 환경에서 객체 지향으로 개발된 애플리케이션의 서로 다른 하드웨어 플랫폼, 이종의 통신 프로토콜을 통합하여 객체들간의 통신을 단일한 사용자 환경처럼 지원하기 위한 명세이다[5].

현재 CORBA 3.0 스펙의 표준화가 진행 중에 있으며, CORBA 3.0 스펙 가운데 컴포넌트 기반 모델을 제시하는 CCM (CORBA Component Model)에 대한 명세가 많은 관심을 받고 있다.

본 고에서는 기존의 미들웨어 기반 기술에 적용되었던 객체 지향 프로그래밍의 문제점을 제시하고 그 문제점을 보완하기 위한 컴포넌트 기반 기술 중에 하나인 CCM 모델에 대해 설명한다.

2. 객체와 컴포넌트

객체와 컴포넌트의 개념은 여러 책과 논문에서 혼용해서 쓰이기도 하지만 객체와 컴포넌트의 차이점을 살펴보면 다음과 같다.

- 컴포넌트는 객체지향 언어로 작성하지 않아도 된다. 하지만 일반적으로 객체지향 언어로 작성한다. 그러나 객체는 반드시 OO(Object Oriented) 프로그래밍만 할 수 있다. 객체지향 언어로 작성한 한 개의 클래스가 객체 단위이다.
- 한 개의 컴포넌트는 많은 객체를 가질 수 있다. 여러 객체들이 서로 연동하여 동작하지만 컴포넌트를 사용하는 클라이언트 입장에서 이것은 하나의 컴포넌트다.
- 객체 개념은 컴포넌트 개념에 적용될 수 있다. 하나의 컴포넌트는 객체가 갖는 특성을 가진다. 따라서, 한 개의 컴포넌트를 한 개의 객체로 볼 수도 있다.
- 객체는 일반적으로 소스 레벨에서 재사용을 의미하고, 컴포넌트는 바이너리 레벨에서의 재사용을 의미한다. 소스 레벨에서 객체를 수정할 때 재컴파일이 필요하지만 바이너리 레벨에서 컴포넌트를 변경할 때는 재컴파일이 필요 없다.

기존의 CORBA 객체 및 일반적인 객체 지향 프로그래밍(Object Oriented Programming:OOP)을 통해 생성된 객체는 재사용을 표방하였으나 실제 컴포넌트로 사용되는데 다음과 같은 여러 문제점이 제기 되었다.

- 소스 형태의 클래스/객체는 컴파일을 해야 하므로 컴포넌트로 사용하기 어렵다.
- 객체들간의 관계가 복잡한 대규모 프로젝트에서 확장성이 떨어진다.
- 서브 클래스가 수퍼 클래스로부터 상속받기 위해서 수퍼 클래스의 내부를 알아야 하므로 클래스의 캡슐화가 보장되지 않으며 확실한 정보은폐(information hiding)를 제공하지 않는다.

이러한 객체의 한계를 극복하기 위해 새로운 패러다임으로 컴포넌트에 기반을 둔 소프트웨어 개발 방법론(Component-Based Software Development)이 등장하였다. 컴포넌트는 여러 객체로 구성되며 그 내부는 외부에서 볼 수 없도록 캡슐화가 되어 있고 소스 코드 수준이 아닌 실행 파일 형태로 제공된다. 컴포넌트는 포트(port)를 통해 외부와 교류한다. 한 개의 컴포넌트만 사용하여, 또는 여러 컴포넌트를 하나로 패키징하여 원하는 애플리케이션을 만들 수 있다. 컴포넌트 또는 패키징된 컴포넌트는 대상 시스템에 배치되어 실제 작업을 수행한다.

3. 컴포넌트 기반 미들웨어 기술

컴포넌트 기반의 대표적 기술로는 마이크로소프

트사의 COM+, OMG의 CORBA, 썬마이크로시스템의 EJB 등을 들 수 있는데 본 고에서는 CORBA3.0 스펙에 포함되어 있는 CORBA 컴포넌트 모델을 중심으로 설명하고자 한다.

CORBA는 OMG(Object Management Group)에서 발표한 분산 객체 환경으로서 클라이언트나 서버 객체의 동작 환경에 관계없이 클라이언트가 구현 객체에서 제공하는 메소드를 자유롭게 요청해서 사용할 수 있는 대표적인 분산 객체 미들웨어 프레임워크 명세이다[5].

CORBA 서비스 객체는 컴포넌트 개념과 유사하다. 즉, 서비스 객체의 인터페이스와 그 구현부는 별개로 작성, 운영된다. 그러므로, 어떤 언어로 구현하든지 간에 외부에서 인터페이스를 쓸 수 있도록 알려줘야 한다. 이를 위해 CORBA에서는 IDL (Interface Definition Language)이라는 인터페이스를 기술하기 위한 언어를 별도로 제공한다. 그러므로 컴포넌트, 즉 서비스 객체를 만드는 프로그래머는 먼저 IDL을 이용해 외부에서 이 객체를 어떻게 사용할 것인지, 즉 인터페이스를 명시하며 이렇게 만들어진 IDL은 IDL 컴파일러를 통해 클라이언트에서 사용하는 스텁과 서비스 객체를 만들 때 사용되는 스켈리톤을 만드는데 사용된다.

3.1 CCM 특징

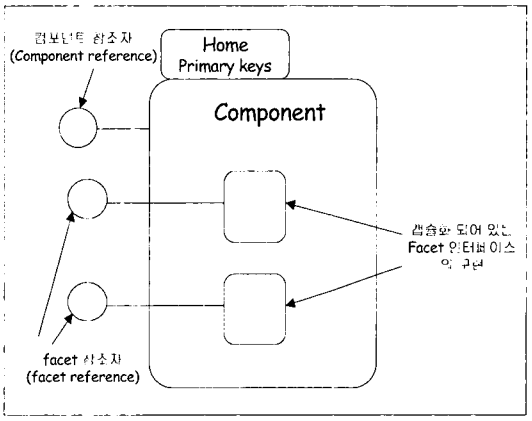
CCM은 CORBA3.0에서 컴포넌트에 대해 정의한 모델이며, 다음과 같은 특징을 가진다[2].

- CORBA 컴포넌트는 기존의 CORBA 모델을 확장한 것이다. IDL을 확장하여, 기존에 없던 component, local, import 문이 새로 추가되었다.
- CIDL(Component Implementation Definition Language)을 도입하였다. CIDL의 많은 부분은 IDL과 유사하다.

컴포넌트를 지원하는 ORB에 의해 CIDL이 번역되어 컴포넌트 스켈리톤(skeleton)이 만들어지면, 컴포넌트 개발자는 컴포넌트 스켈리톤을 확장하여 구현 컴포넌트를 만들 수 있다.

컴포넌트는 CORBA의 새로운 메타 타입으로 객체 메타 타입을 확장한 것이다. 컴포넌트 타입은 IDL component문으로 정의하며 인터페이스 저장소에 저장된다. CORBA 컴포넌트는 상속받은 인터페이스와 자신이 정의한 인터페이스를 지원하며, 자신의 내부 표현과 구현을 캡슐화하고 있어 클라이언트가 볼 수 없다.

3.2 CCM 구조와 구성요소



(그림 1) CORBA 컴포넌트 구조

CORBA 컴포넌트는 (그림 1)과 같은 구조를 가지며, 다음과 같은 요소로 구성된다.

- 컴포넌트 참조자 : CORBA 객체 참조자와 동일한 개념으로 컴포넌트 전체를 대표하며 컴포넌트를 참조할 때 필요하다
- 인터페이스 : 컴포넌트에 구현된 인터페이스로 캡슐화되어 있으며 외부에서 facet 참조자를 사용하여 이 인터페이스를 호출한다.
- 포트 : 컴포넌트와 외부를 연결하는데 사용하며, Facet, Receptacle, Source, Sink, attribute와 같은 5개 종류의 포트를 제공한다.
- 홈 : 선택적인 요소로 컴포넌트를 관리한다.

3.2.1 홈(home)

컴포넌트를 정의할 때 선택적으로 홈을 정의할 수 있다. 홈은 컴포넌트의 인스턴스를 관리하는 인터페이스를 제공한다. 한 개의 홈은 한 개의 컴포넌트형만 관리할 수 있다. 그러나 여러 개의 홈이 동일한 컴포넌트형을 관리할 수 있다. 홈은 단일 상속을 지원한다.

홈에 primary key를 정의할 수 있는데 이 값으로 홈이 관리하는 각각의 컴포넌트 인스턴스를 구별한다. 클라이언트가 어떤 primary key 값을 홈에게 알려주면 홈은 이 primary key에 대응하는 컴포넌트 인스턴스의 객체 참조자를 반환한다. 예를 들면 어떤 애플리케이션에서 primary key 값으로 고객번호를 할당할 수 있으며 이때 홈은 특정한 고객번호가 입력되면 이 고객에 해당하는 컴포넌트의 객체 참조자를 반환한다. 전에는 CORBA 객체를 사용하려면 반드시 CORBA 객체 참조자를 알아야 했으나 클라이언트는 primary key를 사용하여 컴포넌트의 객체 참조자를 얻을 수 있다.

사용자가 컴포넌트의 primary key를 정의하면 primary key 값을 가진 컴포넌트를 생성하는 create, 컴포넌트를 찾아주는 find, 컴포넌트를 없애는 destroy 오퍼레이션이 추가로 생성된다. 이런 오퍼레이션 외에 사용자가 원하는 다른 오퍼레이션도 홈에 구현할 수 있다.

3.2.2 포트(port)

컴포넌트는 내부가 캡슐화 되어 있으므로 사용자나 애플리케이션이 접속할 수 있는 다양한 기능을 컴포넌트 표면에 제공하는데 이를 일반적으로 포트라고 부른다. 컴포넌트 모델은 5가지 종류의 포트를 제공한다.

1) Facet

컴포넌트는 캡슐화된 여러 개의 IDL 인터페이스를 포함하고 있다. 이 인터페이스를 외부에서 사용할 수 있도록 컴포넌트가 제공하는 객체 참조자를 facet이라 부른다. 클라이언트는 facet을 사용하여 컴포넌트의 인터페이스를 호출할 수 있다.

컴포넌트는 구현될 때 한 개의 인터페이스로 표현되는데 이 인터페이스를 "컴포넌트와 동등한 인터페이스"라고 칭하는데 이를 컴포넌트 인터페이스라 부른다. 컴포넌트 참조자는 컴포넌트 인터페이스의 참조자를 제공한다.

facet은 다음과 같이 정의한다.

provides<인터페이스 형> <facet 이름>

인터페이스형은 Object이거나 컴포넌트가 아닌 인터페이스를 의미한다. facet 이름은 facet을 의미한다. 위의 IDL 정의는 컴포넌트 인터페이스에서 다음과 같은 오퍼레이션으로 변환된다.

<인터페이스 형> provides_<facet 이름> ();

따라서 클라이언트는 컴포넌트 인터페이스에 있는 provides_<facet 이름>() 오퍼레이션을 호출함으로써 위에 정의된 객체 참조자 facet을 얻을 수 있다.

2) Receptacle

Receptacle은 외부의 에이전트가 제공하는 객체 참조자를 컴포넌트가 사용할 수 있게 하는 연결점(connection point)이다. 외부 에이전트가 receptacle에 객체 참조자를 제공하면 컴포넌트는 이를 사용하여 그 객체를 호출할 수 있다.

3) 이벤트 진출점(source)

CORBA 컴포넌트 모델은 publish/subscribe 이벤트 모델을 지원한다. CORBA 컴포넌트 이벤트 모델은 OMG notification 서비스와 호환될 수 있도록 설계되었다. 그러나 컴포넌트 이벤트 모델을 구현하는데 이 서비스들을 반드시 사용하지 않아도 된다. 이벤트 진출점은 CORBA 컴포넌트가 특정한 타입의 이벤트를 생성하여 관련된 이벤트 소비자나 이벤트 채널에 보내는 연결점이다.

4) 이벤트 진입점(sink)

이벤트 진입점은 컴포넌트가 특정한 타입의 이벤트를 받아들이는 연결점이다. 이벤트 진입점은 타입이 이벤트 소비자인 특별한 목적의 facet이다.

5) 속성(attribute)

검색 함수(accessor)나 설정 함수(mutator)를 이용하여 얻거나 설정할 수 있는 값이다. 속성은 컴포넌트의 구성을 위한 것이 주목적이다.

3.3 확장된 IDL

IDL CORBA 컴포넌트를 위해 새로운 문장들을 지원한다. 이를 간략히 설명하면 다음과 같다.

1) component

IDL component는 컴포넌트를 정의하는데 사용한다. 이 component 문안에 컴포넌트가 지원하는 인터페이스와 포트 타입을 정의할 수 있다.

2) local

IDL local은 새로운 CORBA 타임인 로컬 인터페이스 타입을 정의하는데 사용한다. 로컬 인터페이스는 한 지역에 국한된 객체를 위한 인터페이스이다. 문법은 CORBA 객체를 정의하는 방법과 비슷하다.

3) import

IDL import는 IDL 외부에 정의된 이름을 IDL 내에서 참조하고자 할 때 사용한다.

3.4 컨테이너 구조

컨테이너는 ORB 위에서 제공되는 서버 측의 프레임워크이며, POA(Portable Object Adapter)이고, CORBA 컴포넌트에게 실행 환경을 제공해 주는 CORBA 서비스의 집합으로 구성된다.

컨테이너 구조는 컨테이너 제공자(Container Provider)와 CORBA ORB와의 관계로 표현될 수 있으며, 컨테이너를 POA에 통합하기 위한 구조를 포함한다. 컴포넌트 모듈에서 정의하는 모든 인터페이스들은 컴포넌트 모듈 내에 포함된다.

3.4.1 BOA와 POA

컨테이너 구조에서 POA를 언급하였는데 BOA와 POA에 대해 간단하게 설명한다.

객체 어댑터는 객체를 활성화하는 작업을 담당한다. 물론, 필요에 따라 서버는 다양한 객체 어댑터를 가질 수 있으며, 서버는 이러한 객체 어댑터에 크게 의존하고 있다. 이러한 객체 어댑터의 형태가 많아지는 것을 방지하기 위해 CORBA 2.0 규격 이전에는 BOA(Basic Object Adapter)라는 것을 정의했다. 그런데 ORB를 제작하는 벤더간의 규격 차이 때문에 타 ORB와의 송수신이 안 되는 등의 문제점이 발생했다. 그래서 이러한 문제점을 보완하기 위한 새로운 객체 어댑터 규격이 CORBA 2.3 규격에서 제시되었는데 이것이 바로 POA 규격이다.

POA는 다음과 같은 특징이 있다.

- 프로그래머에게 다른 ORB 제품 사이에서 이동할 수 있는 객체를 구현할 수 있게 한다.
- POA는 여러 서버 주기를 도는 객체를 위한 일관된 서비스를 제공하는 객체 구현을 생성하도록 설계했다.
- 객체의 투명한 활성화를 지원한다.
- 하나의 서번트가 여러 개의 객체 구분자(Object ID) 가질 수 있도록 허용한다.
- 서버에 존재하는 POA의 구분된 멀티 인스턴스를 허용한다.
- 최소한의 프로그램 작업의 노력으로 일시적인 객체를 허용한다.
- POA에 구현된 객체에 대한 정보를 결합하기 위한 확장 메커니즘을 제공한다.
- 프로그래머에게 정적 또는 동적 스켈레톤 클래스를 상속한 객체를 구현 할 수 있도록 한다.

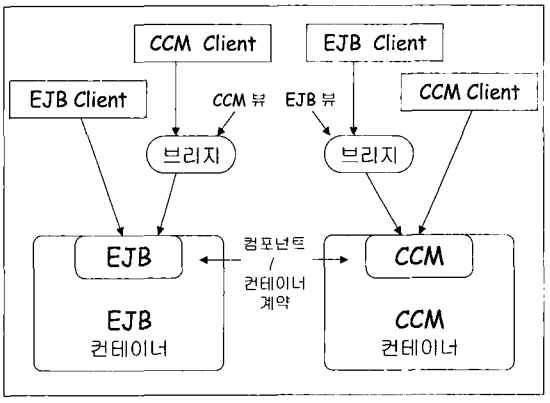
3.4.2 컴포넌트 서버

컴포넌트 서버는 세 가지 종류의 CORBA 컴포넌

트들을 관리하는 기능을 가진 각각의 세 가지 컴포넌트 컨테이너들을 포함하는 프로세스이다. 각각의 컨테이너는 컨테이너 관리자에 의해 관리되는 특수화된 POA이다. 컨테이너 관리자는 CORBA 컴포넌트를 지원하는 프로세스가 서버 운영 체제에서 초기화될 때 실행된다.

CORBA 컴포넌트는 컴포넌트 단위로 컨테이너에 설치되며, 컴포넌트 서술자(Component Descriptor)가 설치된 컴포넌트를 구분한다. 이러한 정의는 컨테이너 관리자에 의해 CORBA 컴포넌트 컨테이너를 생성하기 위한 특정한 POA 정책으로 해석된다. 이러한 정책들은 컨테이너가 컴포넌트에 서비스를 제공할 수 있는 내부적 인터페이스를 구현할 수 있도록 하며, 컴포넌트 제공자가 구현해야 하는 콜백 인터페이스를 호출할 수 있고, 클라이언트에게 외부적 인터페이스를 제공한다.

3.5 CORBA 컴포넌트 모델과 EJB



(그림 2) CCM과 EJB의 상호 호출

CCM은 자바의 EJB(Enterprise Java Beans)1.0에 기반을 두고 확장을 하였으므로 CORBA 컴포넌트와 EJB는 다음과 같은 여러 가지 상호 호환성을 제공한다.

- CORBA 컴포넌트 서버에서 수행되는 CORBA 컴포넌트와 EJB 서버에서 수행되는 EJB를 혼합하여 분산 애플리케이션을 개발 할 수 있다.
- CORBA 컴포넌트 서버가 CORBA 컴포넌트와 EJB를 지원한다. 성능, 보안 또는 다른 이유가 있으면 CORBA 컴포넌트와 EJB를 동일한 CORBA 컴포넌트 서버에 배치할 수 있다.
- EJB 형식을 따라 Java로 만들어진 코바 컴포넌트가 EJB 서버에 배치될 수 있다.

4. 결 론

분산 응용 프로그램 구축 시 분산 응용 프로그램의 특성과 요구사항을 분석하여 가장 효율적이고 효과적인 미들웨어 기술을 적용하는 것이 중요하다.

컴포넌트는 OOP 기술에서 문제점으로 나타난 재사용성과 확장성을 보완할 수 있는 개념으로 여러 분야에서 컴포넌트 개념을 적용하는 기술에 대한 연구가 진행 중에 있다.

본 고에서는 다양한 컴포넌트 기반 기술 중에서 CORBA 3.0 스펙에 포함된 CCM 모델의 중요한 기능과 EJB 상호 호환성을 중심으로 설명하였다.

참고문헌

[1] OMG TC Documents orbos/99-02-05, "CORBA Components" March 1, 1999.

[2] Rosemary Rock-Evans, "Middleware the key to Distributed Computing", Ovum, 1995.

[3] OMG, "CORBA Service : Common Object Service Specification", 1999.

[4] E. Nett, M. Gergeleit, and M. Mock, "An Adaptive Approach to Object-Oriented Real-Time Computing", IEEE Workshop Proceedings of ISORC'98 20-22 April, 1998.

[5] Object Management Group, The Common Object Request Broker : Architecture and Specification,

2.2 ed., Fed. 1998.

[6] D. Schmidt, D. Levine, and S. Mungee: "The Design of the TAO Real-Time Object Request Broker", to appear in the Computer Communications Journal, summer, 1997.

저자약력

윤 은 영

1997년 숙명여자대학교 전산학과(이학사)
1999년 숙명여자대학교 전산학과(이학석사)
1999년-현재 숙명여자대학교 컴퓨터과학과 박사과정
관심분야 : 분산 미들웨어 시스템, 컴포넌트 기반 기술, 실시간 시스템

윤 용 익

1983년 동국대학교 통계학과(이학사)
1985년 한국과학기술원 전산학과(공학석사)
1994년 한국과학기술원 전산학과(공학박사)
1998년-현재 숙명여자대학교 정보과학부 교수
관심분야 : 멀티미디어통신, 실시간 처리, 분산미들웨어, 분산 데이터베이스 시스템

윤 석 환

1982년 아주대학교 산업공학과 졸업(공학사)
1984년 건국대학교 산업공학과 (공학석사)
1992년 품질관리 기술사 자격획득
1996년 아주대학교 산업공학과(박사)
1986년-1997년 한국전자통신연구원 책임연구원
1998년-현재 정보통신연구진흥원 책임연구원(융자1팀장)
1998년-현재 한국정보처리학회 이사 겸 학회지 편집위원장
관심분야 : 분산처리, 소프트웨어 공학, 품질보증, 개발체계, ERP
e-mail : yoonsh@iita.re.kr