2000년 한국정보처리학회 추계 학술논문발표집 제7권 제2호

코바 객체 모델의 확장을 위한 코바 컴포넌트 모델(CCM)의 활용

차상현
동국대학교 정보통신공학과

e-mail: shcha@lily.dongguk.ac.kr

Using CORBA Component Model(CCM) for Extending Traditional CORBA Object Model

Sanghyun Ch'a
Dept of Information and Communication Engineering,
Dongguk University

요약

분산 이기종 환경에서 다양한 어플리케이션을 통합할 수 있는 코바의 기존 객체 모델의 단점을 극복하고 이를 확장하기 위한 CCM(CORBA Component Model) 구조의 장점 및 주요 특징과 서비스를 정의한다. 이를 위해 클라이언트와 컴포넌트 개발자의 관점에서 컴포넌트의 형성과정과 생성된 컴포넌트의 종류, 컴포넌트의 실행을 활성화하는 컨테이너의 구조를 살펴보고 CCM명세서가 CCM 프레임워크의 실행을 단순화시키기 위한 ORB확장방안에 대해서 알아본다.

1. 서론

코바와 같은 많은 분산 객체 미들웨어의 출현은 복잡한 어플리케이션의 개발과 유지를 어렵게 만들고 있다. 코바[2]는 객체의 위치, 프로그래밍언어, 운영체제 플랫폼, 통신 프로토콜, 하드웨어에 관계없이 분산 객체상의 기능을 클라이언트가 호출하는 것을 가능하게 한다.[3] 코바 객체 모델은 객체들이 서로 상호작용 할 수 있도록 하고 객체가 소프트웨어 버스를 통해 원격이나 로컬에 존재하는 객체 상의 기능을 호출하는 것을 가능하게 한다. OMG(Object Management Group)는 컴포넌트의 재사용을 위해서 명명이나 트레이딩과 같은 common서비스에 접근하기 위한 코바 객체 서비스를 구체화한다. 그러나, 코바 초기 버전에서는 여러 가지 객체의 부적절함이 발생하였고 이를 해결하기 위하여 코바 버전3.0 명세서에서 CCM(CORBA Component Model)을 채택하였다. 본 논문에서는 기존 코바 버전의 여러 가지 객체 구현 단점을 알아본 후 이를 극복하기 위한 방안인 CCM모델의 주요 특징과 제공서비스 및 CCM 아키텍처를 사용하여 얻을 수 있는 장점과 관련 기술을 알아본다.

2. 기존 코바 객체의 한계

코바 버전2.3[2]까지의 객체는 다음과 같은 점을 구현하는 것이 어려웠다. 첫째, 객체의 기능들을 확장하는 것이 어렵다는 것이다. 전통적인 코바 객체 모델에서 객체는 상속에 의해 확장된다. 둘째, 객체 실행을 위한 표준방식이 부재하다는 것이다. 코바 명세서는 서버 프로세스에서 객체 실행을 위한 표준 방식을 정의하지 않는다. 따라서 객체들은 서로 의존하기 때문에, 대규모 분산 시스템에서 임시 객체들은 복잡해질 수 있다. 셋째, 코바 서버를 위한 일반적인 프로그래밍 기능 지원이 부족하다는 것이다. 코바 명세서는 서버를 지원하기 위한 풍부한 기능들을 가지고 있으나 많은 어플리케이션에서 그들의 제한된 부분만을 사용한다는 것이다. 따라서, 일반 어플리케이션을 실행하기 위해서는 자동적으로 코바 코드를 생성해주는 툴을 통해 필요한 기능들을 지원하는 것이 필요하다. 넷째, 코바 명세서는 런타임동안 어떤 객체 서비스가 유용한지를 정의하지 않는다. 다섯째, 객체의 생명 주기 처리에 표준이 없다는 것이다. 객체의 생명 주기 서비스를 정의하는 것은 매우 번거로운 절차이고, 자동으로 실행 가능해야 하는 데, 초기 코바 명세서는 이런 자동 실행의 특징이 부족하다. 이러한 코바의 부적절함을 해결하기

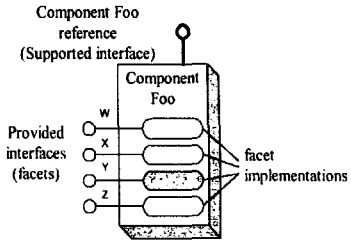
위해서 OMG는 코바 버전 3.0명세서에서 코바 컴포넌트 모델(CCM)을 채택했다[5]. CCM은 어플리케이션 프로그래머가 표준환경에서 지속성, 보안, 트랜잭션, 이벤트 서비스 등의 코바 서비스를 통해 결합하는 컴포넌트들을 실행하는 서비스에 의해 전통적인 코바 객체 모델을 확장한다. CCM표준은 서버를 통해 소프트웨어의 재활용을 가능하게 할뿐만 아니라, 코바 어플리케이션의 동적 구현을 위한 유연성을 제공한다. 어플리케이션 분야에서 코바 수용의 증가는 대용량의 클라이언트/서버 어플리케이션의 구현을 위해 유용하다. CCM 제공자에 의해 제공된 실행 메커니즘은 컴포넌트의 실행을 주관하는 컴포넌트 서버가 관련된 DLL(Dynamic-Linked Library)을 로딩하는 것에 의해 컴포넌트를 실행하기 위해 사용한다. 그러므로 컴포넌트들은 컴포넌트 서버 상에서 실행되고 클라이언트 요청을 처리하는 데 유용하다. 따라서, 본 논문에서는 코바 개발자와 클라이언트의 관점에서 CCM의 컴포넌트를 묘사하고 CCM을 지원하기 위해 필요한 코바 ORB의 요건이 무엇인지 알아본다.

3. 컴포넌트에 대한 클라이언트의 관점

다음은 컴포넌트의 정의와 클라이언트의 관점으로부터 컴포넌트를 이용하는 방법을 나타낸다.

```
정의:
interface A,B;                    //선언
component Foo supports A,B  //equivalent interface와
{                                 //supported interface의 정의
   provides W,X,Y,Z;              //provided interfaces(Facets)
   .....                          //다른 컴포넌트의 정의
   };
```

그림1에서 보듯이 클라이언트 측면에 있어서 컴포넌트 Foo의 인스턴스에 대한 참조는 인터페이스 Foo의 인스턴스에 대한 코바 객체 참조와 같다. 컴포넌트를 인식하지 않는 클라이언트들은 컴포넌트 인스턴스를 동일시하는 컴포넌트의 equivalent interface인 참조를 통해 객체를 호출한다. 컴포넌트의 equivalent interface는 컴포넌트의 supported interface라 불리는 다른 인터페이스로부터 상속받을 수 있다. 그러나 객체들은 하나의 실행 실체를 다중 인터페이스에 연결할 수 없기 때문에 상속을 통해 코바 객체를 확장하기 힘들다. 그래서 CCM은 컴포넌트에 Provided interface라고 하는 Facets를 추가하여 컴포넌트로 하여금 서로 관련성이 없는 인터페이스를 지원하도록 한다. Navigation interface는 모든 equivalent interface를 상속하는 CORBA::CCMO



[그림1] 코바 컴포넌트의 클라이언트 측면

bject인터페이스에서 정의되는데 클라이언트들은 컴포넌트가 제공하는 모든 Facets를 나열하기 위해서 Navigation interface에서 제공하는 기능을 사용한다. Facet을 참조하는 클라이언트는 컴포넌트의 equivalent interface에 대한 참조를 얻기 위해서 다음의 get_component() 기능을 사용할 수 있다. CCM객체 모델에서 Facets의 추가는 컴포넌트의 재사용을 향상시킨다.

3.1 컴포넌트의 생명주기 처리

컴포넌트 생명주기 처리를 표준화하기 위해 CCM은 home이라고 불리는 새로운 키워드를 소개했으며 keyed와 keyless로 구분한다. Keyless homes는 컴포넌트의 새로운 인터페이스를 만드는 factory 기능[4]을 지원하고, 반면에 keyed homes는 finder기능을 지원하는 데 이는 클라이언트들이 그들에 의해 제공되는 키를 사용하는 지속적인 컴포넌트 인스턴스로 대응되도록 하는 그리고 특정한 컴포넌트 인터페이스가 더 이상 필요 없을 때, 클라이언트는 home 인터페이스상의 remove_component() 기능을 호출하여 더 이상 필요 없는 인터페이스를 제거한다.

3.2 컴포넌트 운영 시나리오

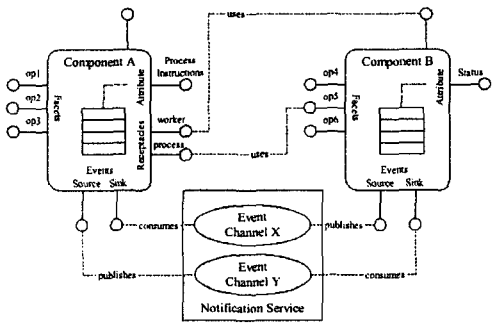
컴포넌트를 사용하기 위해 클라이언트들이 home 인터페이스에 접근하는 부트스트랩 메커니즘을 데이터베이스에 저장한다. 데이터베이스에 접근하기 위해서 CCM은 코바의 상호작용을 위한 명명 서비스[1]와 유사한 HomeFinder 인터페이스를 제공하는데, 이 HomeFinder 인터페이스에 대한 객체 참조를 얻기 위해서 표준 코바 부트스트랩핑 API인 resolve_initial_reference("HomeFinder")를 사용한다. 그리고, 클라이언트는 컴포넌트의 home 인터페이스에 대한 참조를 얻기 위해 컴포넌트 home에 대해 디렉토리 서비스를 실행하고 home 인터페이스에 대한 참조가 얻어지면 최종 컴포넌트 참조를 찾기 위

한 factory 기능을 호출한다.

4. 컴포넌트 개발자의 관점

4.1 컴포넌트

컴포넌트 프로그래밍 모델은 상속보다는 혼합에 의해 컴포넌트들을 재사용 한다. 프로그래밍 모델을 지원하기 위해서 CCM 컴포넌트는 관련성이 없는 인터페이스나 인터페이스 네비게이션을 제공할 수도 있다. 이러한 환경은 개발자들에게는 컴포넌트들을 확장하게 하고, 클라이언트들에게는 컴포넌트들의 위치를 파악하고 사용하도록 한다. CCM에서 컴포넌트들은 포트라고 불리는 인터페이스들을 통해서 ORB, 다른 컴포넌트, 클라이언트들에 의해 제공된 서비스와 같은 외부의 객체들과 상호 작용한다. 클라이언트 관점에서 Facets라고 불리는 하나의 포트 메커니즘은 이미 언급했고, 다른 포트 메커니즘들은 다음과 같다. 첫째는 Receptacles이다. 컴포넌트 프로그래밍 모델은 상속보다는 혼합에 의해 기능을 재사용을 하므로 컴포넌트는 다른 컴포넌트에게 몇몇 기능들을 위임할 수 있다. 이 경우 컴포넌트는 다른 컴포넌트의 인스턴스에 대해 객체 참조를 얻어야만 하는 데 CCM은 이러한 참조를 "object connection"이라고 부르고, 이러한 connection의 포트이름을



[그림2] 포트 메커니즘을 통해 상호작용하는 CCM 컴포넌트들

"receptacle"라고 한다. Receptacle은 컴포넌트의 어떤 객체타입을 연결하는 일반적인 방식을 제공한다. 그림2는 CCM 포트 메커니즘에서 어떻게 컴포넌트가 구성되는지를 보여준다. 둘째는 Event sources/sinks이다. 이는 컴포넌트들이 오퍼레이션 호출을 통해서 상호 작용하는 것과 더불어 다른 컴포넌트의 상태 변화를 감시하고 있다가 어떤 상태 변화 이벤트가 일어날 때 반응하는 방식으로 상호작용 할 수 있다는 것이다. 셋째는 Attributes인데 컴포넌트의 형성을 용이하게 하기 위해, CCM명세서는 코바에서 정의된 attribute들을 확장한다. Attribute들은 컴포넌트의 생성 값을 미리 설정하기 위한 생성 툴에 의해 생성될 수 있다. 위에서의 Facets, receptacle, event source/link, attributes등과 같은 포트 메커니즘은 컴포넌트 클라이언트들에 의해 사용될 수 있으며 Facets을 제외하고 나머지 메커니즘은 주로 컴포넌트 형성을 위해 CCM 프레임워크가 사용한다.

4.2 Component home

컴포넌트 개발자는 컴포넌트들에 대해 다른 생명주기처리를 정의할 다중 home interface를 제공한다.

4.3 컴포넌트 형성

개발자들은 컴포넌트가 이벤트를 publish/receive하기 위해 무슨 컴포넌트와 연결되어야만 하고 어떤 채널을 사용해야 하는지 등과 같은 구체적인 형성과정을 나타낼 메커니즘을 필요로 한다. 컴포넌트형성 인터페이스 Components::StandardConfigurator는 컴포넌트 서버가 컴포넌트 형성을 실행하도록 돕는다.

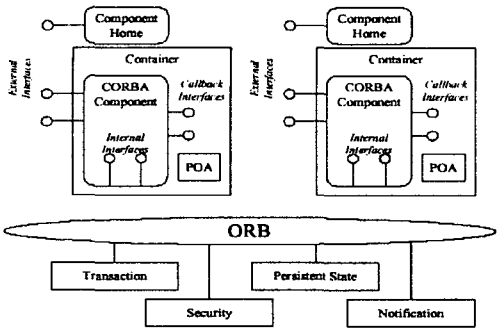
4.4 Component Implementation Framework

코바 CIF는 컴포넌트들의 지속적인(persistent)상태를 처리하고 컴포넌트 실행을 구현하기 위한 프로그래밍 모델을 정의한다. 컴포넌트들은 컴포넌트 인턴페이스가 활성화될 때마다 재 호출 되어야하는 persistent데이터로 컴포넌트 인터페이스와 맵핑될 때 persistent 상태를 가질 수 있다.

4.5 컨테이너(Containers)

컨테이너는 런타임 환경에서 다음과 같은 특징을 제공한다. 첫째, 메인 메모리와 같은 제한된 시스템 자원을 보존하기 위해 컴포넌트 실행을 활성화한다. 둘째,Transaction, Persistence, Security, Notification 서비스와 같은 네 개의 Common service에 대해 adaptation layer를 제공한다. Adaptation layer는 CCM이 처리하는 실질적인 서비스로 클라이언트 요청을 전송하는 것에 의해 클라이언트가 서비스의 위치를 파악하는 수고를 덜어준다. 셋째, 컨테이너와 ORB가 관심 있는 이벤트를 컴포넌트에게 통지하는 콜 백(callbacks)을 위해 adaptation layer를 제공한다. 넷째, 컴포넌트 참조를 만드는 법을 결정하기 위해 POA(Portable Object Adaptor)정책을 처리한다. 그림3은 CCM의 컨테이너 구조를 나타내며 컨테이너가 다루는 모든 인터페이스는 다음과 같이 나누어진다. 첫째, External API는 컴포넌트에 의해 제공되는 인터페이스로 클라이언트는 여기에 바로 접근할 수 있다. 둘째, Container API는 컨테이너가 컴포넌트 상에서 호출할 수 있는 콜 백 인터페이스뿐

만 아니라, 컴포넌트에서 제공하는 서비스에 접근하기 위해 호출할 수 있는 Internal 인터페이스를 포함한다.



[그림3] 코바 컴포넌트 모델의 컨테이너 구조

4.6 컴포넌트 카타고리들

몇몇 어플리케이션들은 컴포넌트 인스턴스를 특정 지속적인 데이터 저장소에 맵핑시키기도 하고, 어떤 어플리케이션의 컴포넌트들은 지속적인 상태를 갖지 않을 수 도 있다. 코바 객체 모델은 객체참조가 일시적인지 지속적인지를 구분하는 코바 사용 모델을 정의한다. CCM 컴포넌트 카타고리는 컨테이너 API 타입과 코바 사용 모델을 결합함으로써 원래 코바 사용 모델을 확장한다. 컴포넌트 개발자는 CIDL(Component Implementation Definition Language)파일에서 컴포넌트 카타고리를 구체화하고 CIDL 컴파일러는 지정된 컴파일러 카타고리에 부합되는 적절한 인터페이스를 생성한다. CCM에 의해 지원되는 컴포넌트 카타고리는 Service, Session, Process, Entity 컴포넌트등이 있다.

4.7 패키징과 전개

대규모 분산 시스템에서 컴포넌트 실행은 다른 실행 언어, 운영체제, 컴파일러를 사용하는 다중서버를 통해 이루어진다. 몇몇 컴포넌트들은 같은 프로세스나 호스트에서 전개되어야 하므로 컴포넌트의 패키징과 전개는 복잡해 질 수 있다.

5. Object Request Broker(ORB)의 확장

CCM명세서는 CCM프레임워크의 실행을 단순화시키기 위해 많은 ORB확장을 추가함으로써 컴포넌트 개발자들이 컴포넌트의 성능을 향상시키고 그들의 작업을 단순화시키도록 한다.

5.1 지역 제한 인터페이스

코바3.0 명세서 이전 버전의 코바는 지역 제한적인 인터페이스를 요구했다. 그 이유는 IDL인터페이스가 임시 프로세스에서만 유효했고, 원격의 클라이언트에게 전송될 수 없었기 때문이다. CCM 또한 IDL에서 지역 제한적인 인터페이스를 지원하기 위해 'local'이라는 키워드를 사용한다.

5.2 인터페이스 저장소에 대한 확장

인터페이스 저장소는 코바에서 실행시간 동안에 애플리케이션이 IDL인터페이스의 정의와 데이터 타입을 발견하도록 하는 표준 메커니즘이다. CCM명세서는 컴포넌트를 지원하고 컴포넌트의 모든 인터페이스에 대한 기능을 제공하기 위해서 인터페이스 저장소를 확장한다.

5.3 IDL언어에 대한 확장

CCM명세서는 분리된 파일들에서 정의된 IDL 타입의 선언들을 가져오기 위한 구조를 가지며 IDL 언어를 확장하고 인터페이스 저장소를 조절하기 위한 표준 메커니즘을 제공한다..

6. 결론

CORBA 객체 모델은 플랫폼과 언어간 표준분산 객체 컴퓨팅 프레임워크 이다. 최근의 CCM의 추가는 코바의 상호작용과 언어의 중립을 유지하면서 EJB(Enterprise Java Beans)로부터 성공적인 컴포넌트 프로그래밍 모델의 통합을 이루었다. 또한, 컨테이너들은 CCM 컴포넌트들에게 훌륭한 품질의 서비스(QoS)를 위한 이상적인 실행 환경과 많은 서비스를 제공하기 위해서 쉽게 확장될 수 있다. 이러한 컨테이너의 확장은 서로 다른 프로그래밍 환경을 중립화시키거나, 컴포넌트 및 컨테이너들과 운영체제를 더욱 더 긴밀히 상호작용 하도록 한다. 따라서, CCM의 이러한 특징이 기존 코바 객체 모델의 성능을 향상시킨다.

참고문헌

[1] Object Management Group(1998), Interoperable Naming Service specification.

[2] Object Management Group(1999a), The Common Object Request Broker : Architecture and Specification, version 2.3

[3] Henning, M. and Vinoski, S(1999), Advanced CORBA Programming with C++, Addison-Wesley Longman, Reading, MA

[4] Gamma,E.et.al.(1994), Design Patterns:Elements of Reusable Object-Oriented software, Addison-Wesley, MA

[5] Object Management Group(1999b), CORBA Component Model Joint Revised submission.