코바 컴포넌트의 구현

금 영 욱†

◈ 목 차 ◈

1. 서 론
2. 코바 컴포넌트 정의
3. 코바 구현 컴포넌트
4. 컴포넌트 패키징과 배치
5. 컨테이너 프로그래밍 모델
6. 코바 컴포넌트 모델과 EJB
7. 결 론

1. 서 론

코바 컴포넌트[1]는 코바 3[2,3,4]의 대표적인 기능으로 코바 2의 IIOP(Interoperable Inter ORB Protocol)이래 코바[5]의 가장 혁신적인 발전이다. 현재 대표적인 컴포넌트 기술인 EJB(Enterprise Java Beans)[6,7]는 Java와 같은 특정 언어의 제약을 받으며 COM/DCOM[8,9]은 특정 운영체제인 윈도우의 제약이 있으나 코바 컴포넌트는 이런 제약을 받지 않으므로 현재의 이질 분산 컴퓨팅 환경에 가장 적합하며 또한 EJB와의 상호운용성으로 인해 앞으로 컴포넌트의 활성화에 기대되는 바가 크다.

코바 컴포넌트 모델은 서버 측 객체 모델에 초점을 맞추어 개발되었다. 코바 컴포넌트 명세는 자바의 EJB(Enterprise Java Beans) 모델에 기반을 두었으며 EJB 1.0의 명세를 포함하고 있다.

코바 컴포넌트 모델의 특징을 살펴보면 다음과 같다.

- 코바 컴포넌트는 기존의 코바 모델을 확장한 것이다. IDL을 확장하였으며 CORBA 2.3, transaction 1.1, security 1.2, notification 1.1에 기반을 두었다.
- CIDL(Component Implementation Definition Language)을 도입하여 컴포넌트 구현의 구조와 상태를 표현하는데 사용한다. CIDL의 많은 부분은 IDL과 유사하다. 현재 CIDL의 컴포넌트 상태를 표현하는 부분은 완성이 되었으나 구조를 정의하는 부분은 아직 완성되지 않은 상태이다.
- 컴포넌트를 패키징하고 배치하는데 XML을 사용한다. 패키징과 배치에 사용하는 XML은 Marimba와 MicroSoft 사가 W3C에 제안한 OSD(Open Software Description)를 확장하였다.
- 코바 컴포넌트의 런타임 환경에 시스템 서비스를 제공하기 위해 컨테이너 모델을 정의하였다. 컨테이너 모델은 트랜잭션이나 보안등과 같은 기능을 자동으로 제공하기 때문에 프로그래머는 순수한 애플리케이션 개발에만 집중할 수 있다.

이 논문에서 코바 컴포넌트의 새로운 기능들을 소개하고 코바 컴포넌트를 구현하는 방법을 제시한다. 2장부터 4장까지 코바 컴포넌트 구현을 다루는 부분으로 2장은 코바 컴포넌트 정의, 3장은 코바 구현 컴포넌트, 4장은 컴포넌트 패키지와 배

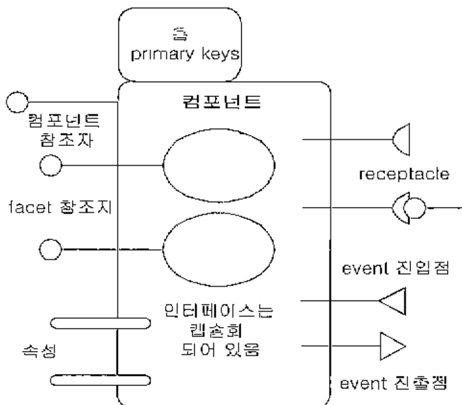
† 정회원 : 성결대학교 컴퓨터학부 교수

치이다. 5장에서 컨테이너 프로그래밍 모델과 6장에서 코바 컴포넌트 모델과 EJB의 상호운용성을 논하고 마지막 7장에서 결론을 내린다.

2. 코바 컴포넌트의 정의

2.1 코바 컴포넌트의 구조

코바 컴포넌트[1,2]는 코바의 새로운 메타 타입으로 객체 메타 타입을 확장한 것이다. 컴포넌트 타입은 IDL component문으로 정의하며 인터페이스 저장소에 저장된다. 컴포넌트는 객체 참조자로 표현되는 컴포넌트 참조자를 갖는다.



(그림 1) 코바 컴포넌트 구조

코바 컴포넌트는 단일 상속만을 지원하며 최상위 컴포넌트는 CCMObject이다. 코바 컴포넌트는 상속받은 인터페이스와 자신이 정의한 인터페이스를 지원한다. 코바 컴포넌트는 다음과 같은 요소로 구성된다. (그림 1)이 코바 컴포넌트의 구조를 보여 주고 있다.

- 컴포넌트 참조자 : 코바 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 그 외부를 연결하는데 사용한다.
- 홈 : 선택적인 요소로 컴포넌트를 관리한다.

2.2 홈(home)

컴포넌트를 정의할 때 선택적으로 홈을 정의할 수 있다. 홈은 컴포넌트의 인스턴스를 관리하는 인터페이스를 제공한다. 홈은 primary key 값을 가질 수 있는데 이 값으로 홈이 관리하는 컴포넌트 인터페이스를 구별한다.

2.3 포트(port)

컴포넌트는 내부가 캡슐화 되어 있으므로 사용자나 에플리케이션이 접속할 수 있는 다양한 기능을 컴포넌트 표면에 제공하는 데 이를 일반적으로 포트(port)라고 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 facet이라고 부른다. 클라이언트는 facet을 사용하여 컴포넌트의 인터페이스를 호출할 수 있다.

클라이언트는 컴포넌트 참조자를 이용하여 facet 및 다른 포트를 찾아 갈 수 있다. 반대로 클라이언트가 임의의 facet에서 컴포넌트 인터페이스로 찾아 갈 수 있다.

2) Receptacle

Receptacle은 외부의 에이전트가 객체 참조자를 이 곳에 제공하면 컴포넌트가 그 객체를 호출할 수 있게 하는 연결점(connection point)이다.

3) 이벤트 진출점(source)

이벤트 진출점은 코바 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)

속성은 검색 함수(accessor)나 설정 함수(mutator)를 이용하여 얻거나 설정할 수 있는 값이다. 속성을 사용하여 컴포넌트를 구성한다.

3.4 코바 컴포넌트 정의

새로 추가된 IDL을 사용하여 리스트 1에 컴포넌트 A와 B를 정의한다. 컴포넌트 A는 인터페이스 I를 지원하며 I의 facet은 Iport로 정의하였다.

리스트 1. 코바 컴포넌트의 정의

```
module M {
  interface I {
        void op();
  };
  component A supports I {
        provides I Iport;
        uses J Jport;
  };
  home AHome manages A {};
};
module N {
  interface J {
        void opJ();
  };
  component B supports J {
        provides J Jport;
  };
  home BHome manages B {};
};
```

A가 사용하는 외부의 인터페이스는 J이며 Jport로 호출할 수 있는데 이는 컴포넌트 B가 제공한다. 두 컴포넌트는 홈을 가지고 있다. 다음 장에서 이 IDL을 사용하여 컴포넌트를 구현한다.

3. 코바 구현 컴포넌트

IDL을 사용하여 컴포넌트를 정의하면 다음 단계로 CIDL을 사용하여 구현 컴포넌트의 구조와 상태를 정의한다. CIDL 컴파일러가 CIDL 정의를 컴파일하면 스켈리톤이 생성된다. 컴포넌트 프로그래머가 스켈리톤을 확장하여 완전한 컴포넌트를 구현한다.

3.1 컴포넌트 컴포지션

Composition은 CIDL로 정의하는 메타형으로 IDL의 component에 정의한 컴포넌트를 구현한다. Composition 정의에서 제일 중요한 두 가지 요소는 다음과 같다.

1) Home executor : Home executor는 IDL로 정의한 컴포넌트의 홈을 구현한다.
2) Component executor : Component executor는 IDL로 정의한 컴포넌트를 구현한다. Executor를 세분하여 segment를 정의할 수 있는데 segement는 독립적으로 영속적 상태를 가지며 또한 개별적으로 활성화될 수 있다.

3.2 구현 컴포넌트

리스트 2는 리스트 1에 있는 컴포넌트 A에 대한 컴포지션 정의를 하였다(컴포넌트 B는 생략). CIDL에서 IDL 문을 참조하려면 import 문을 사용한다. 리스트 2에서 composition의 이름은 AImpl이다. 생성된 home executor의 이름은 AHomeImpl이고 component executor의 이름은 ASessionImpl이다. ASessionImpl은 컴포넌트 A를 구현한 것이다. IDL에서 정의한 컴포넌트 A와 이에 대응하는

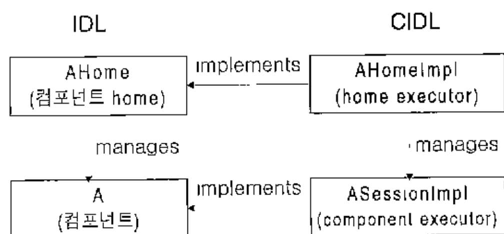
CIDL의 composition AImpl 사이의 연결은 컴포넌트의 A의 홈인 AHome을 정의하는 implements 문에서 M:AHome로 지정된다. (그림 2)에서 이들 정의간에 관계를 표시하였다.

리스트 2. 코바 구현 컴포넌트의 정의

```
// IDL
module M {
// 생략 : 리스트 1 참조
// CIDL
import ::M;
module CompM {
  composition session AImpl {
    home executor AHomeImpl {
      implements M:AHome;
      manages ASessionImpl;
    };  }; };
```

(그림 3)에 있는 것처럼 CIDL 컴파일러가 위의 CIDL을 컴파일하면 다음 3가지가 생성된다.

- component executor ASessionImpl에 대한 스켈리톤
- home executor AHomeImpl에 대한 완전한 구현



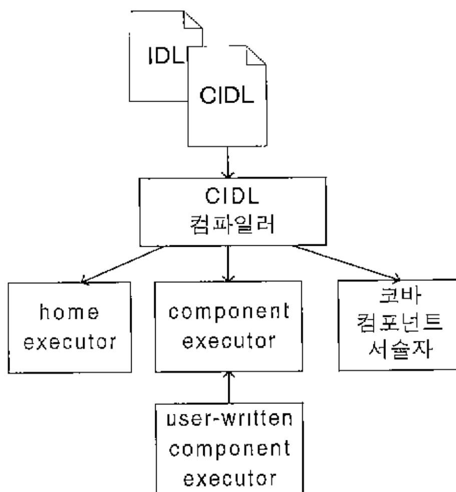
(그림 2) IDL과 CIDL 정의의 관계

- 코바 컴포넌트 서술자

리스트 3은 리스트 1을 IDL 컴파일러가 번역하여 생성한 스켈리톤이다. 간단하게 하기 위해 리스트 1의 receptacle 부분은 생략하였다. 컴포넌트 A에 대응하는 동등한 인터페이스 AOperation이 생성되었고 홈을 위한 인터페이스들이 생성되었다.

리스트 3. IDL에서 생성된 스켈리톤

```
// IDL에서 생성
package M;
import org.omg.Components.*;
public interface IOperations {
 public void op();
}
public interface AOperations
   extends CCMObjectOperations {
   M.I provide_Iport();
}
public interface AHomeExplicitOperations
   extends CCMHomeOperations { }
public interface AHomeImplicitOperations
   extends KeylessCCMHomeOperations {
   A create();
}
public interface AHomeOperations extends
    AHomeExplicitOperations, AHomeImplicit
    Operations {}
```



(그림 3) 구현 컴포넌트 생성

리스트 4는 CIDL 정의를 컴파일하여 생성된 스켈리톤 component executor이다.

리스트 4. component executor 스켈리톤

```
// CIDL 정의에서 생성
package compM;
import M;
import org.omg.Components.*;
abstract public class ASessionImpl
  implements AOperations, SessionComponent,
  ExecutorSegmentBase
{
protected ASessionImpl() {
// generated implementation ...
}
abstract public IOperations
  _get_facet_Iport();
}
```

리스트 5는 CIDL 정의를 컴파일하여 생성된 완전하게 구현된 home executor이다.

리스트 5. home executor

```
// CIDL 정의에서 생성
package compM;
import M;
import org.omg.Components.*;
public class AHomeImpl
  implements M.AHomeOperations,
  ExecutorSegmentBase, CCMHome
{
..........
CCMObject create_component()
{  return create(); }
void remove_component(CCMObject comp)
{  ........ }
A create()
{ ...........}
................ }
```

리스트 6은 컴포넌트 개발자가 구현한 component executor이다. executor의 facet과 제공되는 인터페이스의 오퍼레이션 op()를 이 클래스에서 구현한다.

리스트 6. 구현된 component executor

```
import M.*;
import compM.*;
public class myAImpl extends AImpl
  implements IOperations {
public myAImpl() {
  super();
}
public void op () {
 // 프로그래머가 구현
}
public IOperations _get_facet_Iport() {
   return (IOperations) this;
 }
}
```

4. 컴포넌트 패키징과 배치

구현 컴포넌트는 한 개 또는 여러 개가 모여 패키징되며 패키징된 것은 실제 시스템에 설치하는 과정이 필요하다.

4.1 컴포넌트 패키지

한 개의 구현 컴포넌트와 컴포넌트 서술자는 코바 컴포넌트 제품이 제공하는 패키징 툴을 사용하여 컴포넌트 아카이브 파일(Component Archive File)로 패키징된다. 패키징된 파일의 확장자는 car로 직접 배치되거나 또는 컴포넌트 어셈블리에 사용될 수 있다.

4.1.1 코바 컴포넌트 서술자

구현 컴포넌트는 바이너리 파일이므로 컴포넌

트 사용자나 컴포넌트를 관리하는 툴이 그 내용을 쉽게 알 수 없다. 따라서 컴포넌트는 코바 컴포넌트 서술자를 사용하여 표현된다. 코바 컴포넌트 서술자는 컴포넌트가 지원하는 인터페이스, 상속받은 컴포넌트, 포트들에 관하여 서술한다. 코바 컴포넌트 서술자의 componentfeatures 원소는 상속받은 컴포넌트, 지원하는 인터페이스, uses와 provides로 서술된 인터페이스, emits/publishes와 consumes로 서술된 포트를 표시한다. 이런 정보를 사용하여 뒤에 설명하는 컴포넌트 어셈블리 서술자에서 컴포넌트들을 연결할 수 있다. 상속받은 컴포넌트는 별도의 componentfeatures 원소를 가지며 inheritscomponent로 참조한다.

각 인터페이스는 interface 원소로 표시되며 인터페이스의 repository ID로 참조된다. 인터페이스의 상속은 inheritsinterface 원소로 서술된다.

또한 컴포넌트를 배치하는 시점에 코바 컴포넌트 서술자는 컴포넌트가 설치될 컨테이너의 유형과 컨테이너에게 컴포넌트의 정보를 제공한다. 예를 들면 componentkind 원소는 session, service, process, entity를 서술하여 생성될 컨테이너 종류를 표시한다. 코바 컴포넌트 서술자의 확장자는 ccd인데 CORBA Component Descriptor를 의미한다.

4.1.2 코바 컴포넌트 서술자의 예

리스트 7은 리스트 2로 만든 코바 컴포넌트 A에 대한 컴포넌트 어셈블리 서술자이다. 컴포넌트 A의 홈, 포트, 인터페이스에 대한 정보를 포함하고 있다. uses 태그에서 컴포넌트 B가 제공하는 인터페이스 J를 서술하였다. 서술자에 있는 각 IDL의 요소(예를 들어 홈 AHome)들은 유일하게 구별하기 위해 repository ID를 사용하였다.

리스트 7. 코바 컴포넌트 서술자

```
<!DOCTYPE corbacomponent SYSTEM "corba-
  component.dtd">
<corbacomponent>
  <corbaversion> 3.0 </corbaversion>
  <componentrepid repid="IDL:M/A:1.0" />
  <homerepid repid="IDL:M/AHome:1.0" />
  <componentkind>
    <entity>
      <servant lifetime="process" />
    </entity>
  </componentkind>
  <componentfeatures name="AHome"
     repid="IDL:M/AHome:1.0">
  <ports>
    <provides
      providesname="Iport"
      repid="IDL:M:A:I:1.0"
      facettag="1">
    </provides>
    <uses
      usesname="Jport"
      repid="IDL:N:B:J:1.0" />
  </ports>
  </componentfeatures>
  <interface name="I" repid="IDL:M:A:I:1.0"/>
  <interface name="J" repid="IDL:N:B:J:1.0"/>
</corbacomponent>
```

4.2 코바 어셈블리 패키지

위에 설명한 한 개의 컴포넌트만으로 이루어진 컴포넌트 패키지를 사용하여 애플리케이션을 구성할 수도 있고 여러 컴포넌트가 모여서 한 개의 애플리케이션을 구성할 수도 있다. 상호 연관된 컴포넌트와 컴포넌트 홈의 집합을 컴포넌트 어셈블리라고 한다. 컴포넌트 어셈블리를 표현하는데 컴포넌트 어셈블리 서술자가 사용된다.

컴포넌트 어셈블리 패키지는 컴포넌트 어셈블리 서술자, 컴포넌트 패키지와 프로퍼티 파일의 집합이다. 이 파일들은 아카이브 파일에 함께 수록되거나 또는 분산되기도 한다. 분산될 때에는 서술자가 관련된 파일의 링크를 가지고 있다. 컴포넌트 어셈블리 아카이브 파일의 확장자는 aar을 갖는다.

4.2.1 컴포넌트 어셈블리 서술자

컴포넌트 어셈블리 서술자를 사용하여 컴포넌트 간에 연결을 표시한다. 활성화된 컴포넌트들은 provides 인터페이스와 uses 인터페이스로, 또는 emits/publishes 이벤트와 consumes 이벤트로 서로 연결된다. 예를 들어 어떤 컴포넌트가 특정한 인터페이스를 제공하고(provides)하고 다른 컴포넌트가 이 인터페이스를 사용하면(uses) 제공된 인터페이스의 참조자를 사용하는 컴포넌트에 넘기면 두 개의 컴포넌트가 연결된다. 동일한 방법으로 이벤트를 생성하는(emis/publishes) 컴포넌트와 이를 소비하는 (consumes)하는 컴포넌트간에 연결할 수 있다.

또한 컴포넌트 어셈블리를 사용하여 한 프로세스에 함께 설치해야 하는 컴포넌트들 또는 한 호스트에 함께 설치해야 하는 컴포넌트들을 지정할 수 있다.

컴포넌트 어셈블리 서술자의 확장자는 cad인데 Component Assembly Descriptor를 나타낸다. ccd 또는 cad에 있는 컴포넌트가 컴포넌트 어셈블리 서술자의 입력으로 사용된다.

4.2.2 컴포넌트 어셈블리 서술자의 예

리스트 8은 컴포넌트 어셈블리 서술자의 예를 보여 준다. 리스트 1에 정의한 컴포넌트 A와 컴포넌트 B가 합하여 한 개의 컴포넌트 어셈블리 패키지를 구성한다. componentfile에서 각 컴포넌트를 표시한다. connectinterface 원소에서 컴포넌트 B가 제공하는 인터페이스 J를 컴포넌트 A가 사용하는 것을 보여주고 있다.

리스트 8. 컴포넌트 어셈블리 서술자

```
<!DOCTYPE componentassembly SYSTEM
    "componentassembly.dtd">
<componentassembly id="ZZZ123">
  <description>Example assembly"</description>
  <componentfiles>
    <componentfile id="A">
      <fileinarchive name="ca.car"/>
    </componentfile>
    <componentfile id="B">
      <fileinarchive name="cb.car"/>
    </componentfile>
  </componentfiles>
  <partitioning>
    <homeplacement id="AaHome">
      <componentfileref idref="A"/>
      <componentinstantiation id="Aa"/>
    </homeplacement>
    <homeplacement id="BbHome">
      <componentfileref idref="B"/>
      <componentinstantiation id="Bb"/>
    </homeplacement>
  </partitioning>
  <connections>
    <connectinterface>
      <usesport>
        <usesidentifier>Jport</usesidentifier>
        <componentinstantiationref idref="Aa"/>
      </usesport>
      <providesport>

<providesidentifier>Jport</providesidentifier>
          <componentinstantiationref idref="Bb"/>
      </providesport>
    </connectinterface>
  </connections>
</componentassembly>
```

4.3 컴포넌트 배치

컴포넌트, 컴포넌트 홈과 컴포넌트 어셈블리는 네트워크에 있는 대상 호스트에 배치 툴을 사용하여 배치된다. 배치의 목적은 논리적 컴포넌트의 토폴로지를 물리적 컴퓨팅 환경에 설치하고 연결하는 것이다. 컴포넌트 패키지나 컴포넌트 어셈블리 패키지를 사용하여 배치를 한다.

배치의 기본적인 과정은 다음과 같다.

1) 배치 툴을 사용하여 컴포넌트가 설치될 대

상 객체를 결정한다. 컴포넌트는 한 개 또는 여러 개의 컴포넌트와 함께 배치될 수 있다. 한 프로세스에 또는 한 호스트에 여러 개의 컴포넌트를 함께 설치할 수 있다.

2) 구현 컴포넌트를 컴포넌트 인스턴스가 배치될 플랫폼에 설치한다.
3) 특정한 호스트에 있는 컴포넌트와 컴포넌트 홈을 활성화한다.
4) 컴포넌트를 컴포넌트 어셈블리 서술자의 connect 문에 있는 것처럼 연결한다.

어셈블리 파일이 아닌 독립적인 컴포넌트 파일이 배치될 때는 (4)번 과정이 필요 없다.

5. 컨테이너 프로그래밍 모델

컨테이너는 코바 컴포넌트 구현을 위한 서버의 런타임 환경으로 코바 애플리케이션을 쉽게 만들도록 하는 API 프레임워크를 제공한다. 컨테이너 프로그래밍 모델은 다음과 같은 4개의 요소로 구성된다. (그림 4)는 컨테이너 프로그래밍 모델의 요소와 다른 코바 요소와의 관계를 보여 준다.

5.1 외부형(external types)

외부형은 컴포넌트의 클라이언트가 사용할 수 있는 인터페이스이다. 홈(home) 인터페이스와 애플리케이션 인터페이스의 두 가지 외부형이 존재한다. 리스트 1에 정의한 AHome이 홈 인터페이스이고 인터페이스 I가 애플리케이션 인터페이스이다. 홈 인터페이스를 사용하여 클라이언트는 컴포넌트가 구현하는 애플리케이션 인터페이스의 객체 참조자를 얻을 수 있다.

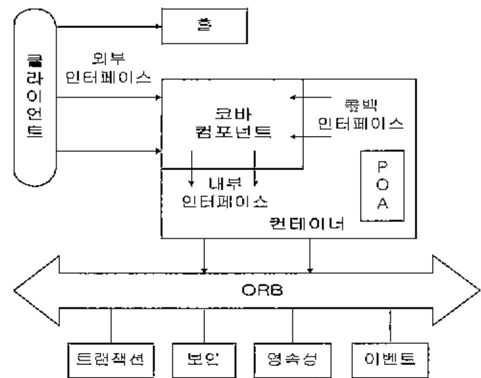
5.2 컨테이너형(container types)

컨테이너형은 컴포넌트와 컴포넌트 컨테이너 사이의 API 프레임워크를 정의한다. 컴포넌트 개발자가 사용할 수 있는 API 프레임워크는 내부의 인터페이스와 콜백 인터페이스로 구성된다. 이 인터페이스들은 지역에 한정된 인터페이스를 서술하는 새로운 IDL의 local 문을 사용하여 정의된다.

컨테이너형의 종류로 일시적(transient) 컨테이너형은 일시적 객체 참조를 제공하는 컴포넌트를 위한 프레임워크를 정의한다. 영속적(persistent) 컨테이너형은 영속적 객체 참조를 제공하는 컴포넌트를 위한 프레임워크를 정의한다.

5.3 컨테이너 구현형(container implementation type)

컨테이너 구현형은 컨테이너와 코바(POA, ORB 코바 서비스 등)와의 교류를 정의한다. 컨테이너 구현형은 POA와 코바 서비스들과 교류 형식을 서술하는 정책들에 의해 조정된다. 이 정책들은 XML을 사용하여 컴포넌트 서술자에 정의되며 컨테이너가 생성될 때 컨테이너 공장에 의해 POA가 생성된다.



(그림 4) 컨테이너 프로그램 모델의 구조

컨테이너 구현형의 3개의 교류 패턴은 다음과 같다.

- stateless : 임의의 ObjectId를 지원하는 POA 서번트와 연합하여 일시적 객체 참조자를 사용한다.
- conversational : 특정한 ObjectId에 할당된 POA

서번트와 연합하여 일시적 객체 참조자를 사용한다.

- durable : 특정한 ObjectId에 할당된 POA 서번트와 연합하여 영속적 객체 참조자를 사용한다.

5.4 컴포넌트 분류(component category)

컴포넌트 분류는 외부형(클라이언트의 뷰), 컨테이너형(서버의 뷰)과 컨테이너 구현형의 구체적인 조합으로 코바 컴포넌트 기술로 구현하는 애플리케이션을 구현한다.

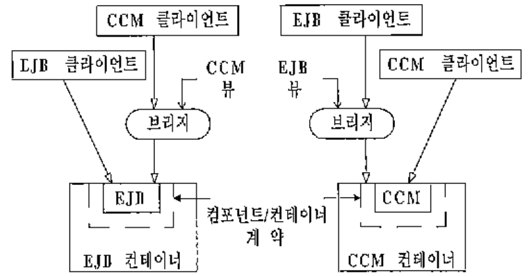
〈표 1〉 컴포넌트 분류의 정의

컴포넌트 분류	컨테이너 구현 형	컨테이너 형	Primary Kcy	EJB 빈
서비스	stateless	일시적	없음	-
세션	conver -sational	일시적	없음	세션
프로세스	durable	영속적	없음	-
엔터티	durable	영속적	있음	엔터티

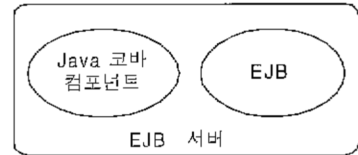
6. 코바 컴포넌트 모델과 EJB

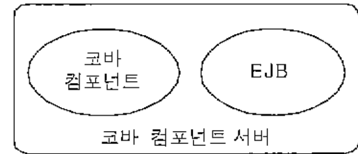
코바 컴포넌트 모델은 자바의 EJB(Enterprise Java Beans)[6,7] 1.0에 기반을 두어 확장을 하였으므로 코바 컴포넌트와 EJB는 여러 가지 상호 호환성을 제공한다. 이를 살펴보면 다음과 같다.

- 코바 컴포넌트 서버에서 수행되는 코바 컴포넌트와 EJB 서버에서 수행되는 EJB를 혼합하여 분산 애플리케이션을 개발할 수 있다. 클라이언트와 컴포넌트간에 가능한 4개의 조합이 (그림 5)에 표시되어 있다.
- 코바 컴포넌트 서버가 코바 컴포넌트와 EJB를 지원한다. 성능, 보안 등의 이유로 코바 컴포넌트와 EJB를 동일한 코바 컴포넌트 서버에 배치할 수 있다. (그림 6)의 하단부가 이를 보여 준다.



(그림 5) 코바 컴포넌트와 EJB의 상호 호출





(그림 6) 서버의 지원

○ EJB 형식을 따라 Java로 만들어진 코바 컴포넌트가 EJB 서버에 배치될 수 있다. (그림 6)의 상단부가 이를 보여 준다.

7. 결 론

코바 컴포넌트는 코바의 새로운 타입으로, 코바 객체에서 진일보하여 컴포넌트로 애플리케이션을 만드는 소프트웨어 제작의 커다란 변혁을 의미한다. 코바 컴포넌트는 어떤 플랫폼, 어떤 네트워크 프로토콜, 어떤 언어로 쓰였든지 코바 하부 구조를 통하여 서로 결합하여 한 개의 애플리케이션을 구현할 수 있다.

코바 컴포넌트의 구현을 위해 새로운 기능들이 많이 도입되었다. 컴포넌트를 정의하기 위해 IDL에 component 문이 추가되었다. 구현 컴포넌트의 구조와 상태를 정의하기 위해 CIDL을 도입하였

다. 컴포넌트 패키징과 배치를 위해 OSD를 변형한 XML이 사용된다. 컴포넌트가 하위의 플랫폼과 독립적으로 수행되기 위해 컨테이너 모델이 도입되었다.

이 논문에서 코바 컴포넌트의 중요한 기능들을 소개하였고 실제 구현하는 방법을 제시하였으며 코바 컴포넌트와 EJB간의 상호호환성에 대해 논하였다..

참고문헌

[1] CORBA component http://www.omg.org/techprocess/meetings/schedule/ CORBA_Component_Model_RFP.html

[2] 금영욱 장연세, 코바 3 프로그래밍 바이블, 도서출판 그린, 2000

[3] Jon Siegel, "A preview of CORBA 3", IEEE Computer, Vol. 32 No.5, 1999, pp. 114-116.

[4] Jon Siegel, CORBA 3 Fundamentals and Programming, Wiley, 2000

[5] 금영욱 윤민영, "CORBA - 분산 객체 통합 기술", 한국정보과학회 소프트웨어공학회지, 제 12권 제 2호, 1999년 6월, p.5-14.

[6] Enterprise JavaBeans Specification, v1.1 ftp://ftp.java.sun.com/pub/ejb/11final-129822/ejb1_1-spec.pdf

[7] Ed Roman, Mastering Enterprise JavaBeans, Wiley, 1999.

[8] 금영욱 장연세, "CORBA와 DCOM의 통합", 한국정보과학회지, 제 17권 제 7호, 1999년 7월, p.55-64.

[9] DCOM architecture, http://www.microsoft.com/com/wpaper/default.asp#DCOMpapers

금 영 욱

1978년 서울대학교 수학과(이학사)
1992년 서강대학교 정보처리학과 (이학석사)
1997년 서강대학교 전자계산학과 (공학박사)
1978년-1981년 (주)대우 시스템즈 엔지니어
1982년-1993년 한국 및 미국 IBM 시스템즈 엔지니어
1997년-현재 성결대학교 컴퓨터학부 교수
관심분야 : 분산·병렬처리, 분산객체기술, 컴포넌트 기술