2002년 한국정보처리학회 춘계학술발표논문집 제9권 제1호

EJB 서버 시스템의 네미밍 관리 시스템 설계

김성훈, 장철수, 노명찬, 서범수, 함호상, 김중배
한국전자통신연구원
전자거래연구부 모바일응용서버연구팀
e-mail : {saint, jangcs, mcroh, bsseo, swjung}@eti.re.kr

A Design of Naming and Directory Management for EJB Container

SungHoon Kim, ChoulSoo Jang, MyungChan Roh, BeomSu Seo, SeungWoog Jung
Mobile Application Server Research Team,
Electronic Commerce Department,
Electronics and Telecommunications Research Institute

요 약

EJB 는 분산 트랜잭션 기반의 엔터프라이즈 어플리케이션을 위한 컴포넌트 컴퓨팅을 위한 아키텍처이다. EJB 는 J2EE 플랫폼을 위한 서버 측 컴포넌트 아키텍처이며, J2EE 플랫폼의 재사용 가능하고 J2EE 기술 중에서 다중 플랫폼 또는 다중 서버에 이식 가능한 비즈니스 로직을 표현하는 코어 기술이다. EJB 기술을 사용하게 되면 응용 로직 개발자들은 시스템 영역의 서비스들 즉, 트랜잭션, 보안, 생명주기, 쓰레딩, 영속성_등의 기능을 직접 구현하지 않고 손쉽게 응용 로직을 개발할 수 있다. 본 논문에서는 J2EE 의 핵심 서버인 EJB 서버 시스템의 설계 사상과 네이밍 및 디렉토리 관리 모듈에 대한 구체적인 설계 방안을 제시한다.

1. 서론

최근 인터넷 전자상거래, 기업의 인트라넷 정보시스템의 웹 환경으로 전환, 인터넷 포털 시스템의 활성화에 따라서 대용량 비즈니스 업무처리와 업무 로직의 재사용 및 분해 조립이 가능하고, 개발 및 유지 보수가 편리한 분산 컴포넌트 기반의 응용 서버 시스템의 중요성이 크게 부각되고 있다. 또한 기업간의 시스템 호환을 위해서 표준화된 개발 API 와 이를 통한 비즈니스 응용의 개발을 추구하고 있다.

만약 상기와 같은 응용서버 시스템 없이 복잡한 기업한 응용을 개발한다면, 개발자들은 데이터베이스 관련 로직과 더불어 EIS 시스템들에 대한 분산 트랜잭션 관리, 사용자에 대한 인증 및 접근 권한 검사, 로직 객체의 인스턴스 관리 상태 관리와 같은 역할을 담당하는 미들웨어 부분을 직접 코딩해야 할 것이다. 또한 구축된 시스템을 다른 시스템에 이식하거나, 또 다른 시스템을 작성하고자 할 때는 매번 상기와 같은 역할을 하는 미들웨어 부분을 직접 코딩해야 할 것이다. 응용서버 시스템은 이와 같이 개발자들에 의해 공통적으로 필요한 미들웨어 부분과 비즈니스 로직을 개발, 배포, 구동하기 위한 표준 분산 컴포넌트 아키텍처(EJB)를 구현한 컨테이너 부분을 포함하고 있어서 개발자들이 미들웨어에 대한 부분을 직접 코딩하지 않고 단순히 비즈니스 로직만 코딩함으로써 개발 생산성을 높일 수 있는 시스템이다. 또한 표준의 분산 컴포넌트 아키텍처에 의해 개발된 비즈니스 로직은 EJB 를 지원하는 어떠한 벤더의 응용서버 시스템에서도 구동이 가능함으로 로직의 재사용성을 높일 수 있다[1].

EJB 는 이러한 기업형 응용 시스템 개발자들의 요구 사항을 충족하기 위하여 SUN 사에 의해 추진되는 산업계 표준 분산 컴포넌트 아키텍처이다. 기업형 비즈니스 응용을 작성하는 개발자들에게 빈

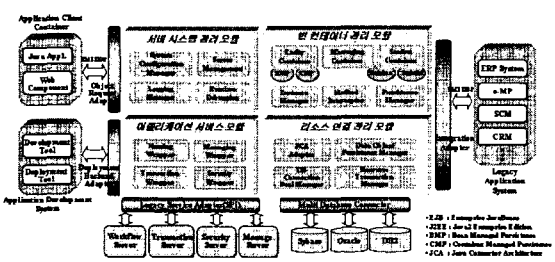
객체의 생명주기 관리, 인증 및 접근 권한 검사, 분산 트랜잭션 처리, 사용자 및 객체의 상태관리, 멀티 쓰레드, 컨넥션 풀링, 영속성 관리 등과 같은 복잡한 하부 미들웨어의 코딩없이 손쉽게 개발할 수 있도록 함으로써 개발 생산성과 비즈니스 업무의 호환성을 높일 수 있다[2].

EJB 기술을 적용하고 사용하기 위해서는 표준의 규약을 준수하는 컨테이너에 대한 서버의 구현이 필수적이다. 본 논문에서는 EJB 컴포넌트를 배포, 구동, 관리, 미들웨어 연결등의 기능을 수행하는 컨테이너 서버에 대한 전반적인 설계 사상과 특히, 컴포넌트 호출의 상호 운용성을 제공하기 위하여 CORBA 기반의 네이밍 및 디렉토리 관리 시스템에 대한 설계에 대한 설명한다. 본 논문의 구성은 다음과 같다. 2 장에서 EJB 표준을 준수하는 EJB 서버 시스템의 구조에 대해 설명하고, 3 장에서 EJB 표준을 만족하기 위하여 제공해야 하는 네이밍 관리 모듈의 요구 사항에 설명한다. 4 장에서는 네이밍 및 디렉토리 관리자의 구조 및 기능을 포함하는 설계 내용에 대해 설명하고, 5 장에서 결론 및 향후 연구 과제를 알아본다.

2. EJB 서버 시스템

현재 구현된 EJB2.0 서버의 설계상의 주요 특징을 살펴보면 다음과 같다. 첫 번째는 프로토콜 중립적인 구조로써 RMI, CORBA 와 같은 다양한 형태의 통신 프로토콜의 지원이 가능함으로써, 클라이언트의 디바이스 성격에 따라 다양한 응용 플랫폼에 적용될 수 있다. 두 번째는 새로운 EJB 빈을 배포하기 위해 시스템 재시작이 필요 없는 동적 자동 배포 기술 제공함으로써 응용의 배포와 관리의 생산성을 높일 수 있다. 세 번째는 사용자가 임의로 로직을 정의할 수 있는 인터셉터를 통해 시스템 기능 확장이 용이함으로써 서버 로직의 커스터마이징이 용이하다. 네 번째는 서버에서 제공하는 미들웨어 연동 서비스는 특정 레거시 미들웨어 시스템을 대상으로 구현하지 않고, 다양한 외부 서비스 등을 사용할 수 있도록 SPI 를 제공함으로써 레거시 미들웨어 통합이 용이하도록 설계되었다.

<그림 1>은 설계된 구현된 EJB 서버 시스템의 전체 구성도이다.



<그림 1> E504 EJB 컨테이너 구조

<그림 1>의 EJB 시스템의 각 모듈 중에서, 먼저 시스템에서 제공하는 사용자 정의 인터셉터 기능은 다양한 외부 미들웨어(즉, 보안, 트랜잭션, 메시징서버)를 필요에 따라 시스템 관리자가 교체하거나 대체할 수 있도록 함으로써 시스템의 유연성을 제공한다. 데이터베이스 리소스 관리 측면에서는 EJB 빈에서 사용되는 데이터베이스에 대한 리소스 접근 타입을 분석하여 미등록 자원에 대한 처리를 효율적으로 처리하도록 하고 있다. 또한, 분산 트랜잭션의 효율적 관리와 성능 향상을 위해 로컬 트랜잭션의 최적화 기술을 제공한다. 성능 향상의 또 다른 이슈는 EJB 인터페이스의 구현 코드 단순화와 클라이언트의 빈 메소드 호출에 대해 모든 메소드 호출을 컨테이너에 전달하지 않고, 전처리 단계를 두어 컨테이너 전 단계에서 응답할 수 있는 요청을 처리함으로써 컨테이너의 부하를 줄이고 요청 처리 시간을 단축함으로써 향상된 성능을 제공한다.

유사 기술로 오픈 프로젝트로 진행 중인 OpenEJB, JBoss, SUN 의 J2EE 와 구조와 기능적인 비교를 하면 다음과 같다. 먼저, OpenEJB 의 경우에는 사용자가 서버 시스템의 로직을 정의할 수 있는 인터셉터의 개념이 없으며, 미 등록된 리소스 자원에 대한 처리를 지원하지 않는다. 무엇보다도 하나의 컨테이너가 여러 개의 빈들을 관리함으로써 시스템의 부하가 크고 소스가 복잡하다는 단점을 가지고 있다. JBoss 의 경우에는 JMX 를 이용하여 EJB 서버를 관리하며 인터셉터의 개념을 사용하고 있으나 사용자 정의 인터셉터의 개념은 지원하지 않고 있다. 초기 분석시에는 컨넥터 아키텍처를 사용하지 않았으나 최근 이 기술을 적용하여 재작성 중이다. 빈과 컨테이너의 관계를 1:1 로 구성하여 시스템 유지 보수 및 확장이 용이한 편이며, 내부적으로 트랜잭션 기능을 구현하고 있고, EJB 1.1 을 기준으로 하고 있다. J2EE 의 경우에는 EJB 2.0 에서 제안한 모든 기능을 충실히 구현하고 있는 것으로 판단되나 시스템의 성능을 고려하지 않고 참조 구현만을 고려하고 있어서 소스 코드가 복잡하고 성능이 떨어지는 단점을 지니고 있어 상업적인 용도로 사용이 불가능하다[3].

본 논문의 EJB 서버는 컨테이너가 실행하여야 할 중요한 기능들을 효율적으로 개발하기 위해서 오픈 소스로 공개되어 있는 EJB 컨테이너중 하나인 JBoss 의 인터셉터의 개념을 확장하여 수용하고 있으며, 인터셉터의 개념을 통해서 컨테이너의 핵심적 기능을 분리하여 클라이언트의 메소드 호출을 여러 인터셉터를 거치면서 순차적으로 처리하도록 설계되었다. 또한, 사용자 정의 인터셉터를 추가할 수 있는 구조를 제공하여 컨테이너의 기능을 쉽게 확장하여 커스터마이징할 수 있도록 하고 있다.

3. 네이밍 및 디렉토리 관리자

JNDI 는 JNDI 는 썬의 J2EE 플랫폼의 일부분으로써 자바 프로그램들을 DNS, LDAP 및 NDS 등과 같은 네이밍/디렉토리 서비스에 연결하여 클라이언트의 다른 객체로 하여금 이름만으로 객체의 레퍼런스를 접근할 수 있도록 하는 썬 마이크로시스템즈의 API 이다. 응

용 로직은 JNDI API 를 이용하여 작성되며, 디렉터리 드라이버는 JNDI SPI (Service Provider Interface)를 구현하여 작성된다[4].

네이밍 관리자의 기능 중에서 객체 및 레퍼런스 바인딩 기능에 관한 요구사항은 다음과 같다.

- 객체의 바인딩 기능 제공
- 각 컨테이너에 탑재된 빈의 홈 스텁을 코바 네이밍 서비스에 등록
- 각 컨테이너에 탑재된 빈의 홈 스텁을 RMI 레지스트리에 등록
- 데이터베이스 연결 리소스의 등록
- 로컬 컨텍스트에 JMS 서버 측의 타겟과 컨넥션 팩토리 등록
- 글로벌 네이밍 컨텍스트와 로컬 네이밍 컨텍스트에 트랜잭션 관리자의 User Transaction 등록

- 레퍼런스 바인딩 기능 제공
(빈의 환경 변수로 설정된 값을 로컬 네이밍 서비스에 등록)
- 빈 환경 변수 값의 등록(String, Byte, Long, ...) EJB References (Bean Stubs)
- 컨넥션 팩토리의 리소스 레퍼런스 등록 (Database Connection, URL, Mail)
- 리소스 환경 레퍼런스 등록 (JMS 관리 객체)

본 논문의 EJB 서버 시스템의 글로벌 네이밍 서버는 CORBA 의 네이밍 서비스와 RMI 레지스트리를 사용하며, 자체적인 네이밍 서버에 대한 구현은 제공하지 않는다. 이는 서버 시스템의 구조에서 알 수 있듯이 어떠한 외부 서버와도 연동할 수 있는 기능을 제공하기 위함이다. 빈과 클라이언트를 위한 네이밍 서비스 컨텍스트 구현에 대한 요구 사항은 다음과 같다.

- Context 구현 제공
  - 글로벌 네이밍 컨텍스트 (CORBA 및 RMI 객체 바인딩)
  - 로컬 네이밍 컨텍스트 (Serialize 객체와 레퍼런스 객체의 바이딩)
- JNDI API 와 SPI 에 대한 구현 제공
  - EJB 빈에서 로컬 네이밍 컨텍스트에 접근하기 위한 ContextFactory 객체 제공
  - Reference Object 와 Environment Object 를 위한 Context 자료 구조 객체 제공
  - Reference 객체의 URL Reloading 을 위한 ObjectFactory 구현 객체 제공
- Local Context 의 Serialization
  - Local Context 의 영속성을 위한 네이밍 관리자의 기능 제공

JNDI LinkRef 에 의해 Local Context 에 등록되어 있는 각 링크 레퍼런스 객체의 실제 레퍼런스를 얻기 위해 구현되어야 하는 Object Factory 의 구현 요구 사항은 다음과 같다.

- Global Naming Object Factory
  CORBA Naming Context 또는 RMI 레지스트리에 등록되어 있는 리모트 홈 객체
- Local Object Factory
  Local Home Repository 에 등록되어 있는 Local Home 객체
- MailObject Factory
  등록된 메일 세션 객체

기본적으로 EJB 서버 시스템에서 구동되는 각 매니저들(예를 들어, 트랜잭션 관리자, ORB 관리자 등)은 Singleton 객체인 서비스 관리자를 통해 매니저의 레퍼런스를 얻는다. 그러나, 같은 프로세스라 할지라도 Persistence Manager 처럼 별도의 로직으로 구동되는 부분에서는 네이밍 관리자를 통해 매니저를 호출할 수도 있다. 따라서, 시스템 내부에서 로컬 컨텍스트 팩토리를 통해 요청되는 각 매니저의 레퍼런스는 로컬 컨텍스트에서 서비스 관리자를 호출하여 매니저의 레퍼런스를 리턴하는 기능을 제공해야 한다.

- 호출되는 형식 : java:pm/TransactionManager

EJB 표준에 명시된 서버 공용 환경 변수를 로컬 컨텍스트에 등록 및 사용을 지원해야 한다. 이들 환경 변수들은 외부 클라이언트에서 사용되는 값이 아닌 내부 빈 클라이언트에서 사용되는 값이다. 또한 이들 값은 PortableRemoteObject 로 구현되어 있는 값이 아니므로 리모트에서 호출할 수 없는 값이다.

- 서버의 공용 환경 변수
  - HandleDelegate, UserTransaction

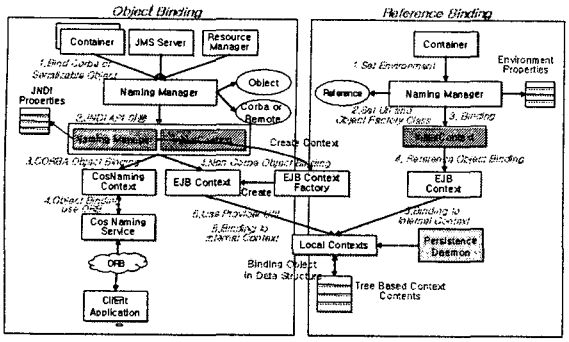
EJB 서버가 비정상적으로 종료된 후 시스템을 다시 시작하게 되면 기존에 서비스하던 내용을 모두 상실하게 되는데 이를 위한 서비스가 네이밍 관리자의 마지막 요구사항인 영속성 서비스이다. 즉, 각 모듈에 대한 통합 지원과 관계없이 네이밍 관리자의 내부 컨텍스트 자료들에 대한 영속성 방법을 지원해야 한다.

4. 네이밍 및 디렉터리 관리자 구조

EJB 서버 시스템에서 네이밍 서비스는 EJB 객체에 대한 위치 투명성을 제공하여 빈 객체가 어떤 서버에 있는지에 관한 정보 없이 빈이 사용될 수 있는 기능을 제공하며, 빈의 내부에서 사용되는 리소스에 대한 레퍼런스를 관리하는 기능을 제공한다. 본 논문의 EJB 서버 시스템에서 구현된 네이밍 및 디렉토리 시스템은 흐름은 <그림 2>에서 보는 바와 같이 구성된다.

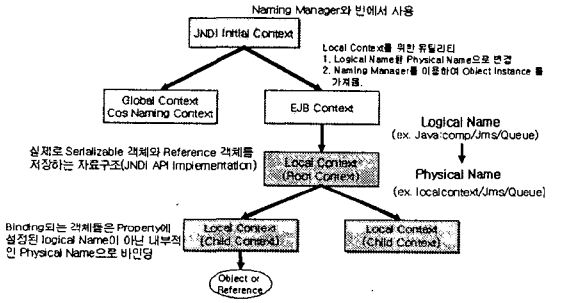
네이밍 관리자의 구성은 크게 시스템 관리 모듈 또는 빈 컨테이너에서 접근하는 네이밍 래퍼 모듈과 CORBA 네이밍 서버에 바인딩된 객체의 레퍼런스와 EJB 빈의 리소스 레퍼런스를 담고있는 로컬 네이밍 관리 모듈로 구성되어 있다. 또한 컨테이너에 탑재된 빈이 내부 네이밍 서비스에 접근할 수 있도록 JNDI SPI 에 대한 구현을 제공하며, 클라이언트 응용에서

접근할 수 있는 API 에 대한 구현을 제공하고 있다.



<그림 2> 네이밍 관리 모듈 처리 흐름도

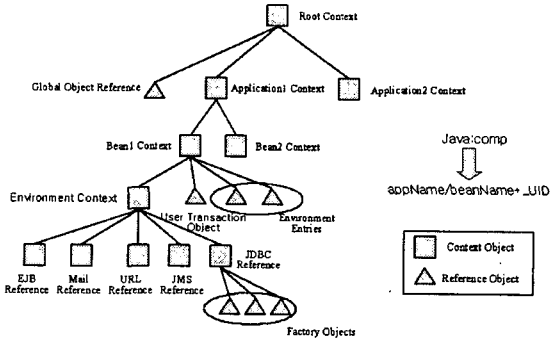
상기 흐름도에서 CosNamingContext 는 RMI Registry 컨텍스트로 대체될 수 있으며, 네이밍 서버 제조업자의 컨텍스트 팩토리를 통해 생성된다. 상기 구조에서 네이밍 관리자에서 사용되는 컨텍스트들의 구조를 좀더 자세히 표현하면 <그림 3>과 같다.



<그림 3> 컨텍스트 구조도

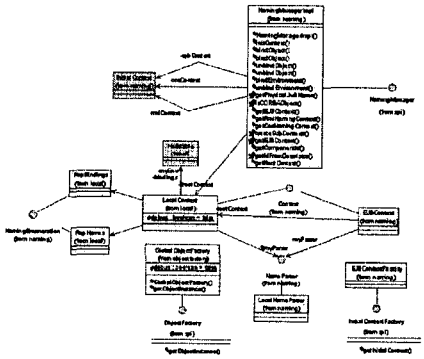
EJB 빈과 내부 시스템에서 사용하는 JNDI API 중 컨텍스트는 EJBContext 이며, 내부적으로 로컬 컨텍스트를 사용하도록 구성되어 있다. 즉, 빈에서 개발자가 InitialContext 를 요청하면 EJBContextFactory 에 의해 EJBContext 가 생성되어 리턴된다. EJBContext 는 시스템 초기화나 런타임시에 내부 시스템이나 빈에서 Serializable 객체나 레퍼런스를 Binding 또는 Lookup 할 때 사용된다. 로컬 컨텍스트에 바인딩되는 객체의 이름은 내부적으로 런타임시에 빈 객체의 트리구조를 파악하기 위하여 실제 이름을 논리적인 이름으로 변경하여 관리한다.

로컬 컨텍스트의 자료구조는 <그림 4>와 같다. 내부의 자료구조는 트리구조의 형태로 관리되며, 각 노드는 컨텍스트 또는 객체를 나타낸다. 최상위 노드는 글로벌 네이밍에 바인딩된 레퍼런스의 링크 또는 컨텍스와 EJB 서버에 배포된 응용 단위가 바인딩되고, 각 응용의 빈 객체를 위한 컨텍스트가 그 다음 단계의 노드에 바인딩되도록 설계되어 있다. 바인딩되는 빈 객체의 레퍼런스 컨텍스트들은 EJB Reference, Mail Reference, URL Reference, JMS Reference 등 이며, User Transaction 이 같은 단계에 바인딩된다. 실제 이름과 논리적인 이름의 관계는 java:comp 로 바인딩되거나 lookup 되는 경우 배로된 응용의 이름, 빈 이름과 바인딩시 랜덤하게 생성된 UID 의 조합으로 생성된다.



<그림 4> 컨텍스트 자료구조

<그림 5>는 상기 요구 사항과 설계 사상을 기반으로 구현된 네이밍 관리자의 클래스 다이어그램이다



<그림 5> 클래스 다이어그램

5. 결론 및 향후과제

본 논문에서는 EJB 표준 스펙의 구현 시스템과 시스템의 모듈 중에서 네이밍 관리자의 요구 사항 및 설계 내용에 관해 설명하였다. 현재 설계된 네이밍 관리자는 실제 객체의 레퍼런스를 바인딩하는 네이밍 서버로써 외부 시스템을 사용하였으며, 내부적으로 빈 또는 서버 시스템에서 네이밍의 기능을 사용할 수 있도록 자료구조와 JNDI API 및 SPI 를 설계하였다.

EJB2.0 규약의 네이밍 기능의 완벽한 지원과 서로 다른 벤더간의 상호 호출을 위한 상호운용성 지원, 성능 향상을 위해 네이밍 서버의 구현 등이 향후 과제로 남아있다.

[참고문헌]

[1] 박재현, "전자상거래용 분산컴포넌트 개발에 관한 연구 최종연구보고서", 에이젠텍, 한국전자통신연구원, 1999

[2] Linda G. Demichiel 외 2 인, "Enterprise Javabeans Specification, Version 2.0", Final Release, Sun Microsystems, 2001

[3] "JBoss - World class J2EE technologies in open source", http://www.jboss.org

[4] "Java Naming and Directory Service(JNDI)", Version 1.2.1, Sun Microsystems, 1999