The Journal of Korean
Institute of CALS/EC
Vol. 5, No. 1, June 2000

기업통합을 위한 데이터베이스 브로커 개발

신혜균*, 김정선*, 우훈식**

Development of Database Broker for Enterprise Integration

Hyekyun Shin, Jungsun Kim, Hoon-Shik Woo

Abstract

Enterprise integration and virtual enterprise are practical visions of industrial information implementations in the information society. In these environments, the distributed and heterogeneous data sources should be exchanged and shared in effective and integrated way. However, the distributed and heterogeneous data sources are managed by independent and heterogeneous computer systems, thus system users and developers are faced difficulties in implementing enterprise integration environments. In this study, we designed and developed a database broker system utilizing a routing broker method to provide transparent location access mechanisms for the distributed and heterogeneous data sources. The proposed mechanism is designed to act as a middle tier between clients and multiple servers, and adopts Java, CORBA, and JDBC as its state-of-the-art techniques.

* 한양대학교 전자계산학과
** 대전대학교 정보시스템공학과

1. 서론

전자상거래 (CALS/EC)의 궁극적인 목표는 컴퓨터와 인터넷을 이용하여 기업 내와 기업 간의 모든 비즈니스 프로세스를 전자적으로 처리하는 것으로, 구현 형태는 기업통합/가상기업(Enterprise Integration/Virtual Enterprise)이 된다[4,15,16]. 이러한 기업통합/가상기업 환경 하에서는 이질적이고 분산된 데이터 소스가 효율적이고 통합적인 방법으로 거래 당사자간 교환되고 공유되어야 한다[1,2,3,18]. 그러나 이질 분산형 데이터 소스는 서로 다른 인터페이스를 사용하기 때문에 이를 사용하는 일반 사용자나 기업통합 환경을 구축하고자 하는 시스템 개발자에게는 많은 부담이 된다[4]. 따라서, 분산된 이질 데이터 소스에 대한 일관성 있고 위치 투명성 있는 액세스 메커니즘은 시스템 개발자를 포함한 모든 사용자에게 매우 중요한 도구라고 할 수 있다.

인터넷과 같은 네트워크 기술이 발전함에 따라 이질적이고 분산된 데이터 소스의 환경은 클라이언트/서버 아키텍처로써 구성되는 것이 일반적이다. 현재 일반화된 클라이언트/서버 아키텍처는 2 계층과 3 계층 방식이 존재한다[6]. LAN-기반 클라이언트/서버 환경에서는 클라이언트와 서버가 밀접하게 결합되어 서로를 잘 알고 있어야 하는 2 계층 아키텍처를 가지며, 이러한 아키텍처에서는 응용 분야의 로직이 클라이언트의 인터페이스나 서버 쪽에서, 또는 양쪽 모두에서 구현되어야 하기 때문에 결과적으로 팻(fat) 클라이언트나 팻 서버의 구현이 불가피하다[6]. 또한 수많은 클라이언트와 수많은 서버들이 인터넷에 묶여 공존하는 상황에서 2 계층 아키텍처는 유연성, 규모 확장성, 유지 보수의 측면에서 볼 때 적절하지 못한 단점을 가지고 있다. 이러한 구조는 분산된 이질적 데이터 소스에 대한 효율적이고 규모 확장성 있는 액세스를 제공하기가 어렵다. 3 계층 아키텍처는 클라이언트와 서버, 그리고 응용 프로그램을 완전히 분리하는 것으로, 클라이언트와 서버의 상호 연동을 위한 미들웨어를 중간에 위치시키는 것이다. 이러한 미들웨어에는 통신 미들웨어, 데이터베이스 미들웨어, OLTP (On Line Transaction Processing) 미들웨어, 객체 미들웨어등이 속하며 CORBA, DCOM 등이 현재 주로 사용되고 있다[6].

본 연구에서는 LAN 상에 분산된 이질의 데이터베이스에 대한 일관되고 위치 투명성 있는 액세스를 제공하기 위해 라우팅 브로커링 기법[5]을 채택한 데이터베이스 브로커의 설계 및 구현을 제시한다. 데이터베이스 브로커는 규모 확장성, 유연성, 그리고 효율성을 고려해 라우팅 브로커링을 바탕으로 3 계층 아키텍처로 구성되고[6], 분산 객체 컴퓨팅의 표준인 CORBA 를 미들웨어로 사용하며, JDBC[8]를 활용한다. 데이터베이스 브로커의 구현을 위한 개발 방법론으로 객체지향 분석/설계/구현으로 이어지는 객체지향 개발방법론을 채택하였고, 효과적 모델링을 위하여 OMG 표준의 모델링 언어인 UML(Unified Modeling Language)을 사용하며 CASE 도구로 Rational ROSE 를 사용한다. 본 연구에서 제안한 데이터베이스 브로커의 주요 특징은 다음과 같다.

- 브로커링 기법 채택: 복수의 클라이언트와 복수의 데이터베이스 서버를 연동시키기 위한 대표적인 브로커링 기법의 하나인 라우팅 브로커링 기법을 사용하여 환경 변화에 대한 적응력과 규모 확장성이 용이한 데이터베이스 브로커 시스템을 개발하도록 한다.
- 3 계층 시스템 아키텍처: 사용자 인터페이스, 응용프로그램 로직, 서버가 완전하게 분리된 3 계층 클라이언트/서버 아키텍처로 구성하여, 성능이 우수하고 유지 보수가 쉬우며, 인터넷 클라이언트/서버로의 확장이 용이하도록 한다.
- CORBA 미들웨어: 클라이언트와 데이터베이스 브로커, 데이터베이스 브로커와 데이터베이스 서버 사이의 통신을 위한 미들웨어로서 분산 객체 표준인 CORBA 를 사용하여 LAN 과 인터넷에서 유연하게 동작할 수 있는 데이터베이스 브로커 시스템을 구축한다. 객체지향 언어인 JAVA[10]를 사용하여 이동 컴퓨팅 기능을 최대한 활용하고, 통신 미들웨어로서 CORBA 의 다양한 기능들 (naming service, trade service, persistent service, security service, callback 기능 등)을 이용하여 분산 객체지향 시스템을 위한 두 개의 최신 기술을 적극 활용한다.

2. 기존 방법의 문제점

인터넷에서 복수의 클라이언트와 복수의 동질 또는 이질적 데이터베이스 서버들로 구성된 클라이언트/서버 환경의 구축은 주로 CGI (Common Gateway Interface)를 사용한 2 계층 또는 3 계층의 아키텍처 형태로 구현되었다[6,7]. 그러나 CGI 방법은 HTML form 의 제한된 배치 제어와 연속된 실행 상태의 결여로 단순한 형태의 대화형 서비스만을 제공할 수 밖에 없으며, 유연성(flexibility), 유지 보수성(maintainability), 서버 구성(server configuration), 사용자 인터페이스, 응답성(responsiveness)의 측면에서 다음과 같은 문제점이 지적되었다[6].

- 유연성: 사용자 인터페이스와 원격 연산에 대한 정의가 HTML 폼(form)에 밀접하게 결합되어 있다. 사용자는 사용자 인터페이스의 관리와 원격 연산의 호출 이외에 다른 작업을 수행할 수 없다. 원격 연산의 파라미터로서 단지 문자열만을 사용할 수 밖에 없고 구조체 등을 사용할 수 없다.
- 유지 보수성: HTML 폼을 수정하는 경우, 해당되는 CGI 프로그램 또한 적절하게 수정해야 한다.
- 서버 구성: 웹 서버는 원격 연산에 대한 요구가 발생할 때마다 새로운 CGI 프로그램을 실행시켜야 하므로 성능이 저하될 수 있다.
- 사용자 인터페이스: 사용자 인터페이스 컴포넌트의 설계가 제한적이다.
- 응답성: 원격 연산을 호출할 때마다 새로운 CGI 프로그램을 생성해야 하고, 그때마다 필요한 자원을 할당 받아야 한다. 게다가 클라이언트가 단일 쓰레드로 수행되어야 하기 때문에 성능이 저하될 수 있다.

CGI 사용에서 문제점으로 지적된 유

연성과 성능, 그리고 규모 확장성 등의 문제를 해결하기 위해서는 새로운 기술이 적용되어야 하며 그 대표적인 예로 Java 와 CORBA 의 연동 기술을 들 수 있다. 이식성 있고, 동적이며, 멀티쓰레딩을 내장한 Java 는 대화형과 지능형의 클라이언트 사용자 인터페이스 구축에 매우 적합하고, 설치 및 유지 보수가 용이하다. CORBA 는 OMG 에 의하여 표준화된 ORB 로서 분산 객체 인프라스트럭처를 제공하는 미들웨어이며, 분산 객체 컴퓨팅을 위한 제반 서비스와 어플리케이션 프레임워크를 제공한다. ORB 로서 CORBA 이외에도 DCOM[11]을 사용할 수 있지만 현재의 DCOM 은 단지 Microsoft 플랫폼인 Windows 환경에서만 수행되므로 유연성이 취약하다.

이렇게 인터넷에 연결된 복수의 이질적인 데이터베이스 환경에서는 데이터의 효과적인 분산 방법, 여러 사이트에 분산 저장된 데이터에 대한 투명성 있는 접근 방법 등이 해결해야 할 문제라고 할 수 있다.

3. 브로커링 기법

복수의 클라이언트와 복수의 서버를 연동시키기 위한 대표적 기술로 브로커링 기법을 들 수 있다. 브로커링 기법은 복수의 클라이언트와 복수의 서버로 구성되는 분산 환경 하에서 브로커라고 불리는 특수한 서버를 클라이언트와 서버 사이에 위치시키므로써 이 브로커로 하여금 중간층의 역할을 담당하도록 하는 브로커 중개 모델을 사용한다. 브로커 중개 모델은 소개 브로커(introduction broker)와 라우팅 브로커(routing broker) 라는 두 가지 유형의 브로커 모델을 제공한다.

소개 브로커와 라우팅 브로커의 주요 차이점은 클라이언트와 서버 사이의 통신이 어떻게 이루어지는 가에 있다. 소개 브로커의 경우, 브로커에 의하여 클라이언트와 서버가 소개되고 나면 클라이언트와 서버는 브로커의 개입 없이 직접 통신을 하게 된다. 소개 브로커링 기법을 사용하는 경우 브로커의 기능이 그다지 복잡하지 않다는 장점이 있지만, 클라이언트가 모든 서버에 대한 자세한 내용 (예, 액세스 방법)을 알아야 하기 때문에 팻 클라이언트의 구현이 불가피하고, 유연성이 떨어지며, 규모 확장성이 결여 된다는 단점이 있다. 이에 반해 라우팅 브로커링 기법은 사용자 인터페이스, 응용 프로그램의 로직 또는 프로세스, 그리고 데이터베이스가 완전히 분리된 3 계층 클라이언트/서버 아키텍처를 갖는다. 클라이언트와 서버는 오직 브로커와의 통신만을 수행하며 서로 간에 직접 통신은 하지 않는다. 응용 프로그램의 로직이 라우팅 브로커에 포함되므로 라우팅 브로커의 기능이 다소 복잡하게 되지만, 클라이언트와 서버는 서로에 대하여 전혀 인지할 필요가 없기 때문에 시스템 환경 변화에 대한 적응이 유연하고, 견고하며, 규모 확장성이 용이하다는 장점이 있다.

라우팅 브로커링 방법에서 클라이언트는 서버의 서비스를 받기 위하여 브로커에게 메시지를 전송한다. 그러면 라우팅 브로커는 요구된 서비스를 처리할 수 있는 서버를 선택하고, 선택된 서버에게 클라이언트의 요구 사항을 전송한다. 이때 모든 통신은

브로커가 취급하게 되며, 클라이언트와 서버 사이의 직접 통신은 발생하지 않는다. 클라이언트와 서버를 분리하는 개념은 라우팅 브로커 방법에 대한 타당성을 강력하게 뒷받침하여 준다. 서버와 클라이언트는 단지 브로커에 대한 정보만을 가지고 있으면 되고, 시스템 상의 다른 클라이언트와 서버의 존재는 알 필요가 없다. 많은 수의 클라이언트와 서버가 존재하는 시스템의 경우 라우팅 브로커 기법은 유용하게 쓰인다. 서버에 관한 정보를 중앙 집중 관리함으로써 관련된 정보를 쉽게 처리할 수 있다. 중앙 집중 관리는 클라이언트와 서버간에 부하 균형 작업을 수행함으로써 최적화된 성능을 얻을 수 있도록 하여 준다.

라우팅 브로커링 방법을 사용하는 데이터베이스 서버 모델에서는 데이터베이스에 사용할 목적으로 특화된 라우팅 브로커 즉, 데이터베이스 브로커를 사용한다. 이 모델은 클라이언트로 하여금 다양한 유형의 데이터베이스와 각각에 대한 투명성있는 액세스 방법을 제공할 필요가 있는 프레임워크에 유용하다.

이러한 데이터베이스 서버에서는 다음과 같은 프로세스를 진행한다:

- 데이터베이스 서버 모델에서 클라이언트는 데이터베이스 브로커에게 데이터를 요구한다.
- 데이터베이스 브로커는 요구된 정보를 가지고 있는 데이터베이스가 인식할 수 있는 형태로 변환시킨 요구 사항을 전달한다.
- 클라이언트가 요구한 정보에 따라, 데이터베이스 브로커는 관계형 데이터베이스에 SQL 문장을 전달할 필요가 있거나, 원격 디렉토리를 액세스할 필요가 있다.
- 데이터베이스 브로커는 데이터베이스로부터 검색된 정보를 받아서 클라이언트에게 전달한다. 이때 클라이언트는 정보의 근원지를 인지할 필요는 없다.

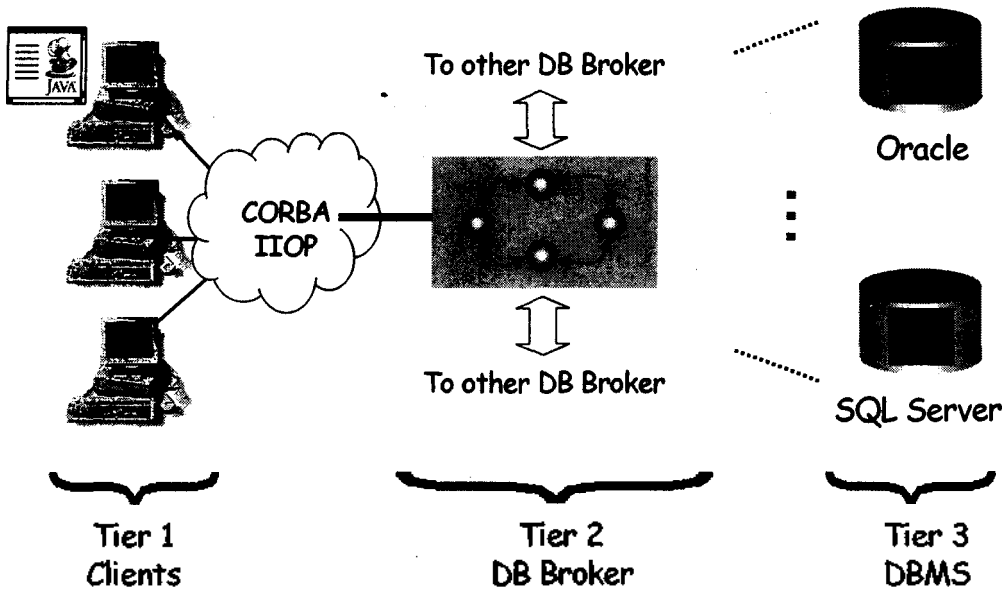
4. 데이터베이스 브로커의 설계

4.1 설계 범위

본 논문에서 설계된 데이터베이스 브로커는 관리하고자 하는 분산된 데이터베이스에 대한 등록을 요구하고 이렇게 등록된 독립적인 데이터베이스의 풀(pool)을 유지하면서 일관성 있는 인터페이스를 통해 사용자가 요구한 데이터 소스에 대한 위치 투명성 있고 효율적인 액세스 메커니즘을 제공한다. 현재는 LAN을 기반으로 하였지만 향후 규모 확장성에 대한 가능성을 고려하여 설계하도록 한다.

4.2 데이터베이스 브로커의 계층 아키텍처

데이터베이스 브로커 시스템의 구조는 [그림 1]에서 보는 바와 같이 3 계층 방식으로서 클라이언트-데이터베이스 브로커-DBMS 로 이루어진다.



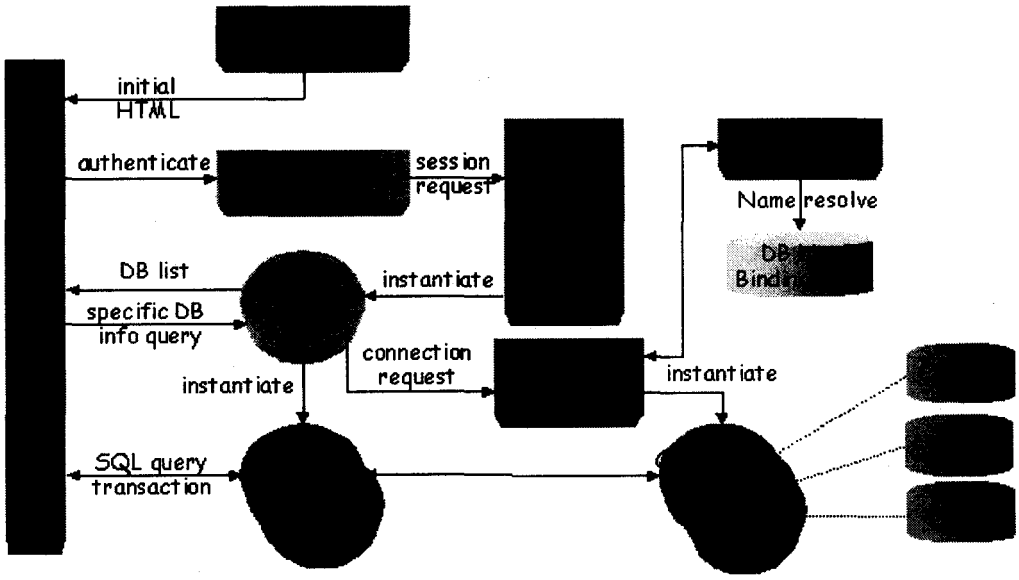
[그림 1] 데이터베이스 브로커의 3 계층 아키텍처

이와 같이 이질적 데이터베이스를 중간층인 데이터베이스 브로커로 연결을 한다면 클라이언트와 서버는 서로를 모르더라도 데이터베이스 브로커를 통해 서로 통신을 할 수가 있다. 클라이언트는 html 문서에 포함된 자바 애플릿의 형태로 구현되며 자바가 지원하는 모든 플랫폼에서 실행될 수 있다.

클라이언트와 데이터베이스 사이에 존재하는 데이터베이스 브로커의 주된 기능은 크게 두 가지이다.첫째, 다수의 사용자를 효율적으로 관리하는 세션 관리기이다. 이 세션 관리기는 각 사용자에 대한 접속을 전담하는 객체를 생성하며, 사용자가 접속을 끊거나 일정시간 사용자의 응답이 없으면 이 전담객체를 삭제하는 기능을 수행한다. 클라이언트와 데이터베이스 브로커 사이의 통신은 ORB 간의 통신 메커니즘인 IIOP(Inter Internet ORB Protocol)[12]를 사용한다. 이 IIOP 를 사용하여 다른 데이터베이스 브로커와 연결하여 WAN 기반의 데이터베이스 브로커로 확장할 수 있다. 둘째는 다수의 데이터베이스와 연결하고 직접적인 상호작용을 하는 연결 관리기이다. 이 연결 관리기는 데이터베이스에게 질의를 보내고 그 결과를 사용자 전담 객체에게 보낸다. 데이터베이스 브로커는 분산된 이질의 데이터베이스와의 연결을 위해 JDBC 를 사용한다.

4.3 데이터베이스 브로커 컴포넌트

[그림 2]는 데이터베이스 브로커의 구성도로 각각의 컴포넌트와 컴포넌트 간의 관계를 나타내며, 특히 데이터베이스 브로커, 클라이언트, 그리고 데이터베이스와 상호 연동 관계를 보여준다.



[그림 2] 데이터베이스 브로커 구성도

각각의 컴포넌트를 살펴 보면 다음과 같다.

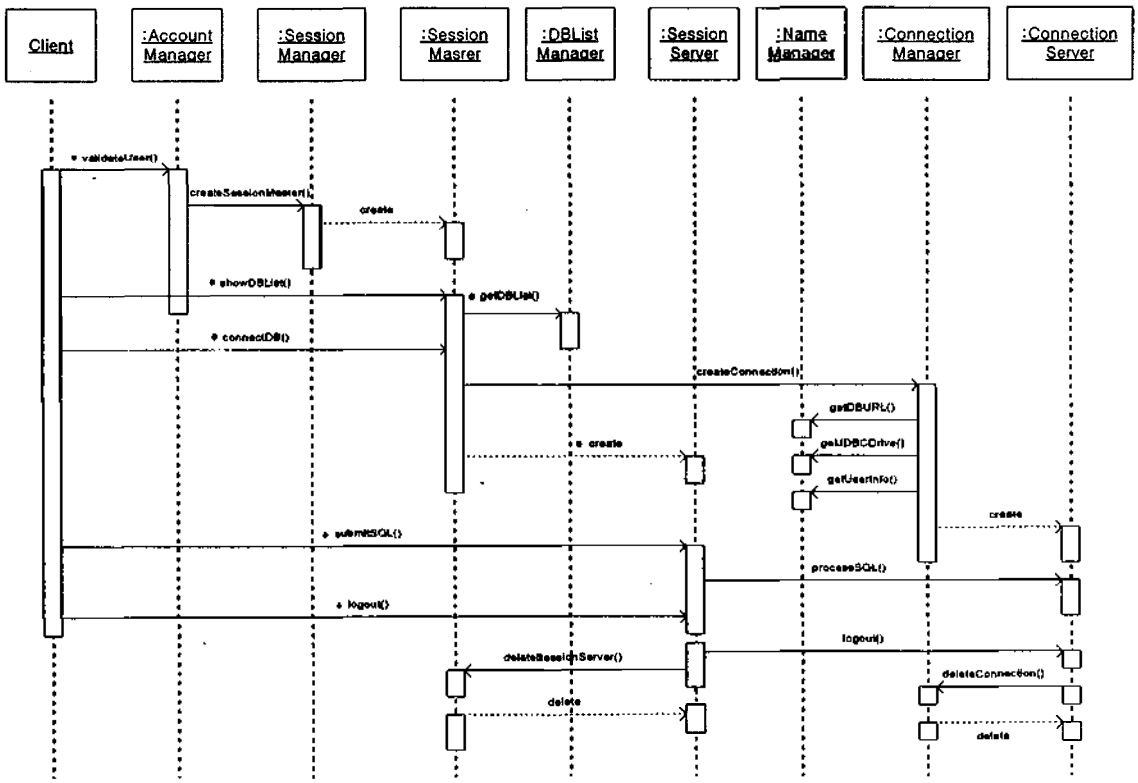
- **AccountManager**: 데이터베이스 브로커의 사용자 인증을 위한 컴포넌트이다. 일단 사용자가 인증을 받으면 AccountManager 는 SessionManager 에게 현재 사용자의 세션을 담당할 SessionMaster 의 생성을 요구한다.
- **SessionManager**: 각 사용자들에 대한 SessionMaster 의 생성과 유지, 삭제를 담당한다. 좀비(jombie) 프로세스를 방지하기 위해 SessionMaster 의 사용을 주기적으로 감시한다.
- **SessionMaster**: SessionManager 에 의해서 생성되며 각 사용자에 대하여 1:1 연결을 유지한다. 현재 사용자의 세션을 담당한다. 사용자가 새로운 데이터베이스의 연결을 요구하면 SessionServer 를 생성하고 ConnectionManager 에게 ConnectionServer 의 생성을 요구한다. 그리고 주기적으로 SessionServer 의 사용 유무를 검사해 일정 시간동안 응답이 없으면 이에 관련된 SessionServer 와 ConnectionServer 를 제거 한다.
- **SessionServer**: SessionMaster 는 하나의 사용자를 전담하는 객체이고, SessionServer 는 하나의 사용자가 사용할 특정 데이터베이스에 대한 연결을 전담하는 객체이다. 데이터베이스와 연결된 ConnectionServer 와 1:1 연결을 유지하며 사용자로부터 SQL 문장을 받으면 ConnectionServer 에게 처리를 요구한

다.

- **NameManager**: 데이터베이스 브로커가 액세스할 수 있는 데이터베이스의 바인딩 정보(위치정보, JDBC 드라이버 정보 등)를 저장하고 있다. SessionMaster 의 요청에 의해 정보가 제공된다.
- **ConnectionManager**: NameManager 로부터 사용자가 선택한 데이터베이스의 위치정보, JDBC 드라이버 정보를 알아내 데이터베이스와의 연결을 전담할 ConnectionServer 와 생성, 유지 및 제거를 담당한다.
- **ConnectionServer**: DBMS 와의 직접적인 연결 통로로서 SessionServer 와 1:1 로 상호 작용을 한다. SessionServer 의 SQL 트랜잭션을 처리하여 그 결과를 돌려준다. SQL 문장의 처리는 각 데이터베이스에 해당하는 JDBC 드라이버를 사용하여 처리한다.

4.4 동적 모델

[그림 3]은 각 컴포넌트의 상호 작용을 나타낸 시나리오에 대한 시퀀스 다이어그램[9,17]이다.



[그림 3] 시퀀스 다이어그램

이 다이어그램은 사용자가 데이터베이스 브로커에 연결하여 원하는 데이터베이스를 액세스하는 과정을 보여준다. 데이터베이스 브로커는 클라이언트로부터 요청받은 작업을 다음과 같은 순서로 데이터베이스와 연결하여 처리한다.

① 클라이언트는 데이터베이스 브로커의 URL 을 통해 데이터베이스 브로커 시스템에 접속한다. 이때 브로커를 액세스할 수 있는 애플릿이 HTML 문서에 포함되어 클라이언트에게 보내진다. 이 애플릿은 데이터베이스 브로커를 사용할 수 있는 사용자 이름과 비밀번호를 입력하는 인터페이스이다.

② 이 애플릿은 사용자가 입력한 사용자 이름과 비밀번호로 AccountManager 에게 인증을 요구한다. 인증이 되면 AccountManager 는 SessionManager 에게 인증된 사용자를 담당할 SessionMaster 의 생성을 요구한다. 이때 사용자에게 데이터베이스를 선택할 수 있는 다이얼로그 박스가 보여진다.

③ 클라이언트는 이 다이얼로그 박스를 통하여 SessionMaster 에게 접근 가능한 데이터베이스 목록을 요구한다.

④ SessionMaster 는 NameManager 로부터 데이터베이스 목록을 획득하여 클라이언트에게 보여준다. 선택 박스 컴포넌트에 연결 가능한 데이터베이스가 보여진다.

⑤ 클라이언트는 다이얼로그 박스를 통하여 원하는 데이터베이스를 선택한다.

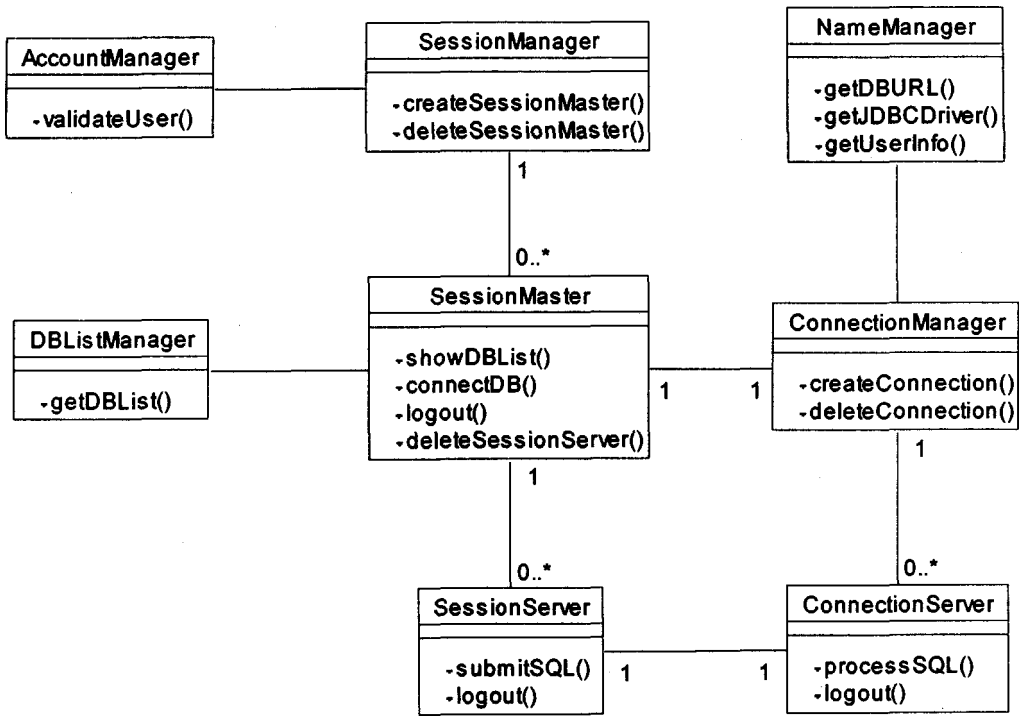
⑥ 선택된 데이터베이스에 대한 정보가 적절하면 SessionMaster 는 SessionServer 를 생성하고 ConnectionManager 에게 ConnectionServer 의 생성을 요청한다. 그러면 SessionServer 와 ConnectionServer 는 1:1 로 연결된다. 그리고 클라이언트에게는 SQL 다이얼로그 박스가 보여진다.

⑦ 클라이언트는 원하는 SQL 문장을 SessionServer 와 ConnectionServer 를 통해 결과를 얻어낸다. 만약, 다른 데이터베이스와의 연결을 원한다면 SessionMaster 가 담당하는 메인 메뉴에서 또 하나의 데이터베이스를 선택하면 그에 해당하는 SessionServer 와 ConnectionServer 가 생성 될 것이다.

⑧ 모든 작업이 끝나면 SessionMaster 는 해당 SessionServer 와 ConnectionServer 를 모두 삭제한다. 그리고 SessionManager 에게 자신의 삭제를 요구한다. SessionMaster 가 삭제되면 데이터베이스 브로커의 데이터베이스 액세스 과정은 종료가 된다.

4.5 정적 모델

[그림 4]는 [그림 2]의 구성도에서 기술된 각 콤포넌트의 클래스 다이어그램[9,17]을 나타낸 것이다.



[그림 4] 클래스 다이어그램

이 다이어그램에서 중요한 세 가지 관계를 살펴보면 다음과 같다:

첫째, SessionManager 와 SessionMaster 는 일대다의 관계를 가지고 있다. SessionManager 는 사용자를 전담하는 SessionMaster 의 생성, 삭제를 관리한다. 사용자가 데이터베이스 브로커에 로그인을 하면 SessionManager 는 SessionMaseter 를 생성하고 사용자가 로그아웃을 하면 SessionMaster 를 삭제한다. 따라서 하나의 데이터베이스 브로커는 다수의 사용자를 관리할 수 있게 된다.

둘째, SessionMaster 와 SessionServer 의 관계도 마찬가지로 일대다의 관계로, 사용자가 다수의 데이터베이스를 액세스 할 수 있음을 나타낸다. 사용자를 전담하는 SessionMaster 는 데이터베이스 접속을 위한 SessionServer 를 생성, 삭제를 관리한다. 한 명의 사용자가 데이터베이스에 연결을 할 때마다 5.2.3 절에서 설명하는 SQLDialog 객체와 상호작용을 하는 SessionServer 가 생성되게 되고 데이터베이스 접속을 끝낼 때 SessionServer 가 삭제된다.

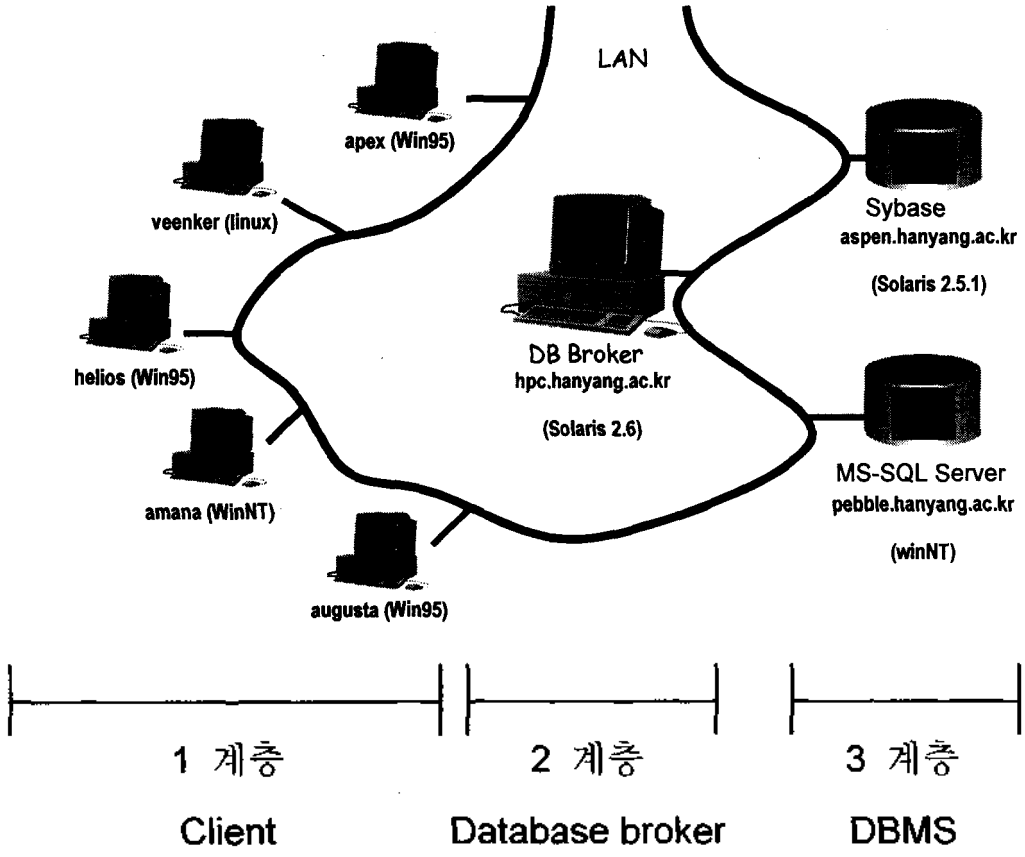
세째, SessionServer 와 ConnectionServer 와는 일대일 관계로 되어있다. 그러므로 하나의 데이터베이스를 액세스하기 위해서 SessionServer 객체와 ConnectionServer 객체가 한 쌍으로 생성되어야 한다. 클라이언트

는 데이터베이스 액세스를 요구하면 SessionServer, ConnectionServer, 데이터베이스의 순서로 이를 처리한다.

5. 데이터베이스 브로커의 구현

전장의 설계를 바탕으로 데이터베이스 브로커 시스템을 구현하기 위하여 [그림 5]와 같은 테스트베드를 구축하였다.

[그림 5]는 LAN 상에 클라이언트, 데이터베이스 브로커, 데이터베이스 서버의 3 계층 아키텍처로 구성된 테스트베드로 제 1 계층은 각 사용자가 기존에 설치한 서로 다른 운영체제와 웹브라우저를 가지고 있는 클라이언트이다. 웹브라우저는 운영체제에 상관없이 데이터베이스 브로커에 접근할 수 있도록 자바언어를 지원하는 넷스케이프나 익스플로러를 사용한다. 제 2 계층는 본 연구에서 설계된 데이터베이스 브로커가 구현된다. 구현언어는 자바이고, 클라이언트와의 통신은 CORBA 를 통해서 이루어진다.



[그림 5] 데이터베이스 브로커 테스트베드

제 3 계층는 이기종의 데이터베이스 서버가 위치한다. 위와 같이 3 계층으로 구성함으로써, 클라이언트는 데이터베이스 서버를 모르더라도 데이터베이스 브로커를 통한 위치 투명성과 일관된 액세스가 제공 된다.

5.1 IDL 의 작성

IDL 파일은 클라이언트가 데이터베이스 브로커 객체를 호출할 수 있는 객체를 정의해 놓은 인터페이스 정의 언어이다 [13,14]. 이 IDL 파일을 idl2java 컴파일러로 컴파일을 하면 데이터베이스 브로커 객체와 클라이언트가 통신할 스켈레톤 코드와 스텁 코드가 자동으로 생성된다. 스켈레톤 코드는 데이터베이스 브로커 객체를 구현할 때 사용하고 스텁 코드는 클라이언트를 구현할 때 사용한다.

```
Module DBbroker{
        // 데이터베이스 목록과 SQL 문장, 결과를 저장할 스트링 배열의 선언
        typedef sequence<string> strarray;

        // AccountManager 객체 정의
        interface AccountManager {
                // 사용자 인증 메소드
                boolean validateUser(in string username, in string password);
        };

        // SessionMaster 객체 정의
        interface SessionMaster {
                // 데이터베이스 목록을 가져오는 메소드
                strarray showDBList();
                // 선택된 데이터베이스와의 연결 메소드
                boolean connectDB(in string strSession);
                // 데이터베이스 브로커와의 연결을 끊는 메소드
                boolean logout(in string strSession);

        };

        // SessionServer 객체 정의
        interface SessionServer {
                // 데이터베이스와의 연결을 닫는 메소드
                boolean logout();
                // SQL 문장의 처리 메소드
                strarray submitSQL(in string strSQL);
        };
};
```

[리스트 1] DBbroker.idl

5.2 사용자 인터페이스

사용자 인터페이스는 Password.java, MainMenu.java 그리고 SQLDialog.java 로 구성된다. Password 객체는 HTML 문서에 포함될 자바 애플릿으로 구현된다. MainMenu 객체와 SQLDialog 객체는 Swing 의 JFrame 객체로부터 상속 받아 구현되었다.

5.2.1 Password 객체

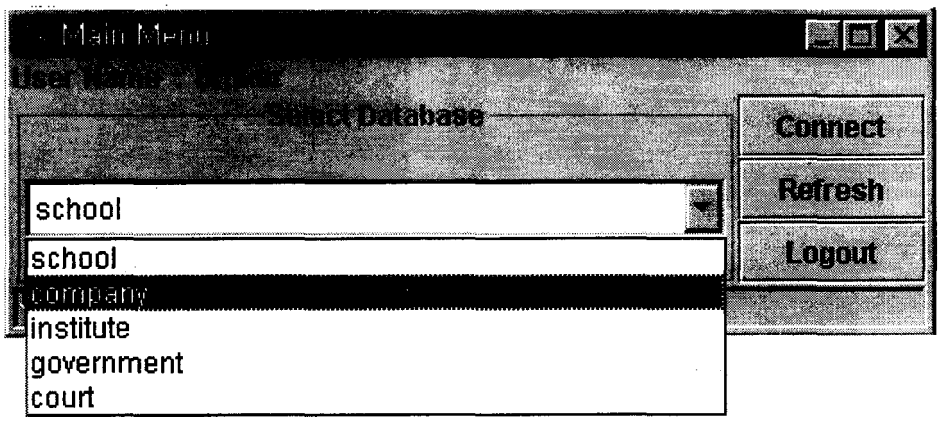
사용자가 데이터베이스를 액세스하기 위해서는 데이터베이스 브로커의 URL 로 접근을 하여 Password 애플릿이 포함된 HTML 문서를 가져와야 한다.

사용자 이름과 비밀번호를 입력 받는 Password 애플릿은 데이터베이스 브로커의 사용자 인증을 위한 객체와의 통신위해 AccountManager 와 바인딩을 한다. AccountManager 와 바인딩이 되면, 사용자 인증을 위한 validateUser() 메소드를 호출하여 인증이 되면 데이터베이스 선택을 위한 메인 메뉴를 보인다.

5.2.2 MainMenu 객체

사용자 이름과 비밀번호가 인증이 되었으면 사용자는 원하는 데이터베이스를 선택해야 한다. 이 선택과 연결을 위한 MainMenu 는 [그림 6]과 같다. 이 MainMenu 객체는 데이터베이스 브로커 SessionMaster 와 연결되어 데이터베이스 목록을 보여주고, 사용자가 선택한 데이터베이스와 연결을 할 수 있도록 해준다.

메인 메뉴는 데이터베이스 목록을 보여주는 Choice 컴포넌트와, 버튼 3 개로 구성이 된다. 각 버튼의 기능은 다음 [표 1]과 같다.



[그림 6] 데이터베이스를 선택할 수 있는 메인 메뉴

[표 1] 메인 메뉴 컴포넌트

Button	설 명	엔터티	메소드
Connect	선택된 데이터베이스와의 연결을 한다.	SessionMaster	connectDB()
Refresh	데이터베이스 목록을 갱신한다.	SessionMaster	showDBList()
Logout	데이터베이스 브로커의 사용을 끝낸다. 만약 다수의 데이터베이스와 이미 연결이 되어있다면 열려있는 SQLDialog 객체를 모두 삭제한다.	SessionMaster	logout()

5.2.3 SQLDialog 객체

SQLDialog 객체는 메인 메뉴를 통해 선택된 데이터베이스를 액세스하기 위한 사용자 인터페이스이다. 컴포넌트의 구성은 SQL 문장의 입력과 결과에 대한 TextArea 2개와 SQL 문장의 실행 버튼, 데이터베이스에 연결을 끊는 Logout 버튼으로 구성된다. SQLDialog 는 데이터베이스 브로커의 SessionServer 객체와 상호작용을 한다. Go 버튼과 Close 버튼이 클릭되면 SessionServer 의 submitSQL()과 logout()의 메소드를 호출한다.

5.3 세션 관리

데이터베이스 브로커는 세션 관리를 위해 SessionManager, SessionMaster, SessionServer 의 세가지 객체와 사용자 인증을 위한 AccountManager 로 구성된다. AccountManager 는 데이터베이스 브로커의 사용자에 대한 인증을 하는 객체이다. SessionManager 는 사용자를 담당하는 SessionMaster 의 생성과 삭제를 담당한다. 여기서 생성이란 객체의 생성 뿐만이 아니라 이 생성된 객체가 CORBA 의 구현객체 저장소에 등록됨을 뜻한다. 이 구현객체 저장소에 객체가 등록되어야만 클라이언트로부터 호출을 받을 수 있다. 이러한 객체를 활성화하고 비활성화 하는 역할은 BOA(Basic Object Adapter)가 담당한다. BOA.obj_is_ready() 는 객체를 활성화 하는 것이고 BOA.deactivate_obj()로 객체를 비활성화 시킬 수 있다.

SessionManager 가 다수의 SessionMaster 객체를 담당하려면 SessionMaster 객체를 서로 다른 이름으로서 구분하여 등록하고 이를 필요할 때 마다 비활성화 할 수 있어야 한다. 이 부분의 구현은 java.util 의 HashTable 객체를 사용하여 처리를 하였다. HashTable 의 키값은 SessionMaster 객체의 참조가 저장되고, 데이터베이스 브로커 사용자 이름이 키값에 대한 내용으로 저장이 된다. 이 HashTable 을 사용하여 SessionManager 는 SessionMaster 를 관리한다.

SessionMaster 가 SessionServer 를 관리하는 방법도 위와 같은 방법을 사용한다. HashTable 의 키값은 SessionServer 객체의 참조가 저장되고 키값에 대한 내용은 사용자 이름과 데이터베이스 이름을 더한 스트링이 저장된다. 또한 SessionMaster 는 ConnectionManager 에게 ConnectionServer 의 생성을 요구한다.

5.4 연결 관리

연결관리에 관련된 객체는 데이터베이스 연결을 위한 정보를 제공해주는 NameManager 객체와 ConnectionManager 객체, ConnectionServer 객체이다.

5.4.1 NameManager 객체

NameManager 는 [표 2]와 같이 데이터베이스 브로커에 등록된 데이터베이스들에 대한 위치정보, JDBC 드라이버 정보 그리고 데이터베이스 사용자 이름과 비밀번호에 대한 정보가 저장되어 있다.

NameManager 객체의 각 메소드(getURL, getUserInfo, getJDBCDriver)는 데이터베이스 이름을 인자로 받아 해당 정보를 리턴하도록 구현하였다.

5.4.2 ConnectionManager 객체

ConnectionManager 는 NameManager 로부터 사용자가 선택한 데이터베이스들에 대한 위치정보, JDBC 드라이버 정보를 알아내 ConnectionServer 를 생성한다. 생성한 ConnectionServer 의 참조를 SessionMaster 에게 리턴하여 SessionServer 가 삭제될때 같이 삭제되도록 하였다.

5.4.3 ConnectionServer 객체

생성된 ConnectionServer 는 SessionServer 로부터 SQL 문을 받아 JDBC 드라이버를 통하여 처리한다. 처리된 결과는 SessionServer 를 통해 사용자에게 보여진다. 또한 SessionServer 로부터 로그아웃 요청을 받으면 ConnectionManager 에게 자신의 삭제를 요청한다.

[표 2] NameManager 등록정보

		company
URL	jdbc:ff-sybase://aspen.hanyang.ac.kr:7000/school	jdbc:ff-microsoft://amana.hanyang.ac.kr:1433/company
User	Broker	broker
Password	******	******
JDBC	connect.sybase.SybaseDriver	connect.microsoft.MicrosoftDriver

6. 결론 및 향후 방향

본 연구에서는 기업통합/가상기업 구현을 위한 핵심 기술의 하나로 이기종 분산형 데이터베이스에 대하여 일관성 있고 위치 투명성 있는 액세스 메커니즘을 제공하는 데이터베이스 브로커를 설계하고 구현하였다. 개발된 데이터베이스 브로커는 LAN 상의 동질 또는 이질의 데이터베이스에 대한 일관되고 위치 투명성 있는 액세스를 제공하기 위해 라우팅 브로커링 기법을 채택하였으며 3 계층 아키텍처로 설계함으로써 WAN 상으로 규모확장이 용이하도록 하였다. 개발된 데이터베이스 브로커 시스템은 OMG 표준의 UML 을 이용하여 설계되었으며, CORBA 를 미들웨어로 사용하였고, JAVA 및 JDBC 를 활용하여 인터넷/웹 환경 하에서 사용될 수 있도록 개발하였다.

개발된 데이터베이스 브로커를 통해 분산된 복수의 이질적인 데이터베이스를 관리한다면 사용자는 데이터베이스의 위치와 기종에 상관 없이 기존의 데이터베이스 시스템을 이용하여 정보의 교환과 공유를 효과적으로 할 수 있게 된다. 하지만, 현재 구현된 데이터베이스 브로커는 소규모의 LAN 환경 하에서만 작동되는 프로토타입 수준의 핵심기능만이 구현되어 있다. 본격적인 활용을 위해서는 현재 구현된 데이터베이스 브로커를 바탕으로 고장감내기능, 부하균형기능, 그리고 다른 브로커와의 상호 연동 등의 기능을 추가되어야 한다. 이를 통하여 WAN 상의 이질적 데이터베이스에 대한 액세스가 보다 효과적으로 수행될 수 있으며 궁극적으로는 기업통합/가상기업의 구현을 위한 효과적인 도구로 활용될 수 있을 것이다.

참고 문헌

[1] 김철한, 우훈식, 임동순, 김중인, "객체지향적 엔지니어링 데이터베이스 설계", IE Interface, 대한산업공학회, 제 10 권 제 3 호 pp. 11-21, 1997.

[2] 김철한, 우훈식, 김중인, 임동순, "CALS 를 위한 기능 모델링 방법론", 한국 CALS/EC 학회지, 제 2 권 제 2 호 pp. 89-112, 1997.

[3] 임동순, 김철한, 우훈식, 김중인, "비즈니스 프로세스 모델링 연계 방법론: IDEF0, IDEF3, Petri Net", 한국 CALS/EC 학회지, 제 3 권 제 2 호 pp. 141-159, 1998.

[4] 우훈식, 정석찬, "CORBA 기반의 CALS 통합 데이터베이스 설계", 한국 CALS/EC 학회지, Vol. 2, No. 2, pp. 155-169, 1997.

[5] Randy Otte, Paul Patrick and Mark Roy, Understanding CORBA, Prentice Hall, 1996.

[6] Robert Orfali, Dan Harkey, Jeri Edwards, The Essential Client/Server Survival Guide, Second Edition, John Wiley & Sons, Inc, 1996.

[7] Robert Orfali, Dan Harkey, Client/Server Programming with Java and CORBA, John Wiley & Sons, Inc., 1997.

[8] Art Taylor, JDBC Developer's Resource, Informix Press, 1996.

[9] Hans-Erik Eriksson, UML Toolkit, John Wiley & Sons, Inc, 1998.

[10] Eric Evans and Daniel Rogers, Using Java Applets and CORBA for Multi-User Distributed Applications, IEEE Internet Computing, vol, no. 3, pp. 43-55, 1997.

[11] DR Richard Grimes, Professional DCOM Programming, WROX Press, 1997.

[12] Andreas Vogel, Keith Duddy, Java Programming with CORBA, Object Manage Group, John Wiley & Sons, Inc, 1997.

[13] Thomas J.Mowbray Ron Zahavi, The Essential CORBA: Systems Integration Using Distributed Objects, John Wiley & Sons, Inc, 1995.

[14] Object Manage Group CORBA 2.0 Specification, Available at http://www.omg.org/library/corbserv.htm

[15] Joan Smith, An introduction to CALS: The strategy and the standards, Technology appraisals, 1990.

[16] CALS Information Starter Kit, The Department of National Defence, CALS Office, CANADA.

[17] M. Fowler, K. Scott, UML Distilled, 1997, Addison Wesley.

[18] Cheol-Han Kim, Hoon-Shik Woo, Joong-In Kim, Dong-Soon Yim, Kyung-Huy Lee,"Functional Requirements for CALS Methodology", CALS EXPO 98 USA, 1998, AFEI.

저자소개

신혜균
건국대학교 물리학과 학사
한양대학교 전자계산학과 석사
현재 한양대학교 전자계산학과 박사과정
관심분야: 분산객체컴퓨팅, 병렬/분산처리시스템

김정선
서울대학교 컴퓨터공학과 학사
미국 Iowa State University 전기 및 컴퓨터공학과 석사
미국 Iowa State University 전기 및 컴퓨터공학과 박사
한국전자통신연구원 (ETRI) 선임연구원
현재 한양대학교 전자계산학과 조교수
관심분야: 분산객체컴퓨팅, 컴포넌트 시스템, 병렬/분산처리시스템

우훈식
한양대학교 공과대학 산업공학과 학사
미국 Iowa State University 산업공학과 석사
미국 Iowa State University 산업공학과 박사
한국전자통신연구원 (ETRI) 선임연구원
현재 대전대학교 공과대학 정보시스템공학과 전임강사
관심분야: 전자상거래 (CALS/EC), 공급체인관리(SCM)