분산환경에서 컴포넌트 애플리케이션 서버의 설계 및 구현

○
권 기현*, 최 형진**
* 동원대학 인터넷정보과, ** 강원대학교 전자계산학과
E-mail:kweon@tongwon.ac.kr

A Design and Implementation of Component Application Server for Distributed Environment

Ki-Hyeon Kweon*, Hyung-Jin Choi**
* Dept of Internet Information & Retrieval, Tongwon College
** Dept of Computer Science, Kangwon Natl. Univ.

요 약

다계층 분산처리 구조는 정보를 시기 적절하게 분배해 주고 유지 보수비용을 최소로 줄여줄 수 있는 장점이 있다. 다계층 클라이언트/서버는 현존하는 데이터베이스, 툴, 컴포넌트 안에서 분산 컴퓨팅 구조로 통합시킬 수 있고, 네트워크 부하를 줄일 수 있으며, 많은 서버 상에서 작업을 효율적으로 분배해 줄 수 있는 로드 밸런싱을 조절하고 제어 할 수 있게 한다. 본 논문에서는 컴포넌트 기반 분산처리에 대한 이론적 고찰을 통하여 분산환경에서 효율적으로 어플리케이션을 개발할 수 있는 방안을 제시하고 분산 3-계층 환경에서 컴포넌트를 서비스하는 애플리케이션 서버를 설계하고 구현한다.

Ⅰ. 서 론

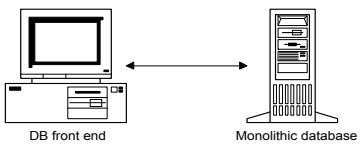
다계층 분산처리 구조는 정보를 시기 적절하게 분배해 주고 유지 보수비용을 최소로 줄여줄 수 있는 장점이 있다. 현재의 LAN환경, 2계층의 클라이언트/서버, 또는 메인프레임까지 대부분의 시스템에 적용될 수 있다. 다계층 클라이언트/서버는 현존하는 데이터베이스, 툴, 컴포넌트 안에서 분산 컴퓨팅 구조로 통합시킬 수 있고, 네트워크 부하를 줄일 수 있으며, 많은 서버 상에서 작업을 효율적으로 분배해 줄 수 있는 로드 밸런싱을 조절하고 제어 할 수 있게 한다.

본 논문에서는 컴포넌트 기반 분산처리에 대한 이론적 고찰을 통하여 분산환경에서 효율적으로 어플리케이션을 개발할 수 있는 방안을 제시하고 분산 3-계층 환경에서 컴포넌트를 서비스하는 애플리케이션 서버를 설계하고 구현한다.

Ⅱ. 관련 연구

2.1 2-계층의 장점과 한계

2계층 클라이언트/서버 시스템에서 애플리케이션 로직은 클라이언트상의 사용자 인터페이스 안이나 데이터베이스 안에 위치하는 구조이다. 2-계층의 장점은 구조가 단순하고 소규모 애플리케이션을 개발하는데 적합하다. 그러나, 2-계층의 한계는 서버측의 데이터 조직에 대해 알아야 하고 확장성이 결여되며 대규모 애플리케이션에 적용하기 어렵다.



(그림 2.1) 2-계층 구조

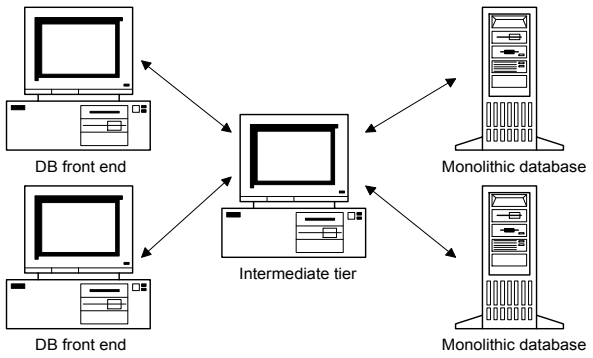
2.2 3-계층 클라이언트 서버

3계층에서 클라이언트에서는 GUI를 제공하고 원격 서비스나 메소드를 통해 서버와 상호 작용한다. 애플리케이션 로직은 중간 계층에 나타난다. 3계층에서 중간 계층의 비즈니스 프로세스들은 사용자 인터페이스 및 데이터베이스와 분리해 관리하고 전개할 수 있다.

비즈니스 로직은 이제 고유의 분리된 계층을 가지고 하나 이상의 서버 상에서 실행할 수 있게된다.

3-계층(계층)의 장점은 다음과 같다.

·대규모 애플리케이션의 요구에 적합하다.
·대부분의 코드가 서버 상에서 실행된다.
·네트웍 트래픽을 최소화한다.
·클라이언트는 비즈니스 로직만을 호출하면 된다.
·데이터베이스를 클라이언트에 노출하지 않는다.



(그림 2.2) 3-계층 구조

2.3 컴포넌트 기반 구조

1) 컴포넌트 기반 아키텍처의 혜택

컴포넌트 기반 애플리케이션으로 미들 계층을 설계하면 다음과 같은 이점을 얻을 수 있다.

·적은 스텝으로 대형 애플리케이션을 개발 가능하다.
·애플리케이션은 컴포넌트들을 재사용할 수 있다.
·적은 스텝으로 대형 애플리케이션을 개발 가능하다.
·쉽고 안전하게 데이터와 기능에 접근할 수 있다.
·애플리케이션은 기존 컴포넌트들과 결합할 수 있다.
·컴포넌트 환경은 노화하지 않는다.

2) 서버 컴포넌트 유형

컴포넌트에는 상태 없는(stateless) 컴포넌트와 상태 있는(stateful) 컴포넌트의 두 가지가 있다.

·상태 없는 컴포넌트

상태 없는 컴포넌트가 호출되면 어떤 인스턴스가 필요한지 결정하고 이것을 데이터베이스에서 검색한 후에 데이터베이스를 갱신한다.

·상태 있는 컴포넌트

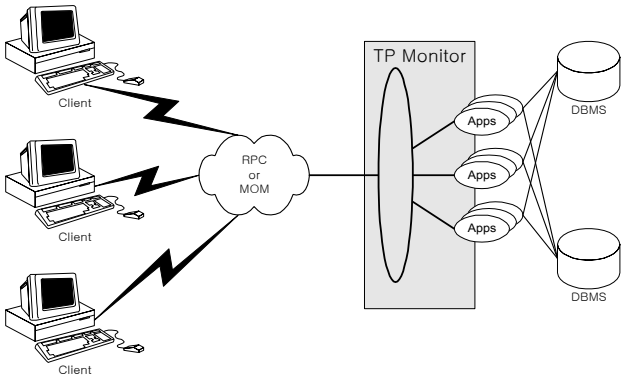
상태가 있는 컴포넌트는 클라이언트들이 유일한 객체 구분자를 사용하는 특정 객체 서비스를 요청한다. 중간 소프트웨어는 특정 개체 요청을 전달한다. 만일, 이미 메모리 내에 있지 않다면, 인프라스트럭처는 객체의 상태와 방법을 찾고 로드해야 한다.

2.4 애플리케이션 서버

1) 애플리케이션 서버의 개요

애플리케이션 서버는 3계층 클라이언트/서버 시스템에서 핵심적 역할을 수행하며 중간계층 프로세스들을 실행하는 프레임웍(framework)을 제공한다.

·복잡한 애플리케이션을 '서비스(services)'라고 부리는 코드 조각으로 분할함으로써 이를 수행한다.
·트랜잭션 사용으로 전체적 조화 속에서 소프트웨어 컴포넌트를 처리한다.
·트랜잭션을 관리, 시스템 교차 경로지정, 실행로드 밸런싱, 실패후 재시작 등을 수행하여 전체적 시스템 성능을 개선한다.
·사용되는 시스템이나 자원관리기에 구애받지 않고 분산 애플리케이션의 모든 면을 감독할 수 있다.



(그림 2.3) 3-계층 애플리케이션 서버

2) 애플리케이션 서버의 작동원리

클라이언트/서버 애플리케이션의 구축, 실행, 관리를 돕는 사전구축된 프레임웍을 제공하는 것으로 생각할 수 있다. 서버 측에서 자원 관리기를 캡슐화하는 서비스라고 불리는 모듈식, 재사용 가능한 프로시져를 제작할 수 있게 한다. 자원 관리기는 데이터베이스 관리기 연속 큐 또는 트랜잭션 파일 시스템 같은 공유된 자원을 관리하는 모든 소프트웨어 조각이다.

애플리케이션 서버는 단지 좀더 서비스를 추가함으로써 고도로 복잡한 애플리케이션을 만들 수 있게 한다. 애플리케이션 서버는 관련 없는 서비스가 ACID 조화 내에서 함께 작동할 것을 보장한다. ACID는 극소성(Atomicity), 일관성(Consistency), 격리(Isolation), 내구성(Durability)을 상징한다.

애플리케이션 서버는 또한 자원 관리기를 혼용할 수 있게 해주므로 하나의 자원관리기로 시작할 수 있고 그 다음 기존 함수 호출들에 대한 투자를 보호하면서 다른 자원관리기로 옮길 수 있다. 모든 서비스는 애플리케이션 서버로 관리된 재사용 가능한 프로시져의 풀에 합류한다. 달리 말하면, 애플리케이션 서버는 이질적 서버 자원을 기존 애플리케이션 아키텍쳐를 변경하지 않으면서 추가할 수 있도록 해준다는 것이다.

Ⅲ. 컴포넌트 애플리케이션 서버의 설계

3.1 애플리케이션 서버의 기능

웹 어플리케이션 서버는 다음과 같은 중요한 기능을 처리한다.

① 데이터베이스 연결 관리 기능
② 트랜잭션 관리 기능
③ 부하 관리 기능
④ 고장방지 관리 기능
⑤ 세션 및 상태 정보 관리 기능

3.2 애플리케이션의 요구사항

본 연구에서는 자바 언어의 특성 중에서 “java is a network and distributed language"라는 특성을 이용하여 분산 3-계층 환경에서 컴포넌트를 서비스하는 애플리케이션 서버를 설계한다. 적용 사례는 소프트웨어는 주식 거래를 대행하는 StockBroker로서 클라이언트를 자바 애플리케이션으로 하고 중간계층을 컴포넌트를 서비스하는 애플리케이션 서버로 하는 3-계층로 구현하였다.

1) 구조 요구사항

분산환경에서 3-계층의 형태로 구성된다.

·클라이언트 인터페이스는 컴포넌트로 작성하여 사용자 인터페이스의 뷰(view)와 모델을 분리한다.
·클라이언트와 애플리케이션 서버의 통신은 최소화하여 경량(thin) 클라이언트로 구성된다.
·클라이언트의 비즈니스 로직의 오류를 조기에 발견하여 서버의 부담을 최소화한다.
·비즈니스 로직은 컴포넌트로 작성하여 애플리케이션 서버에 의해 제공된다.
·클라이언트에 대한 로드 밸런싱을 처리한다.
·클라이언트에 대응하는 서버가 서비스를 중단한 경우에도 연결 복구(fail over) 기능을 수행한다.

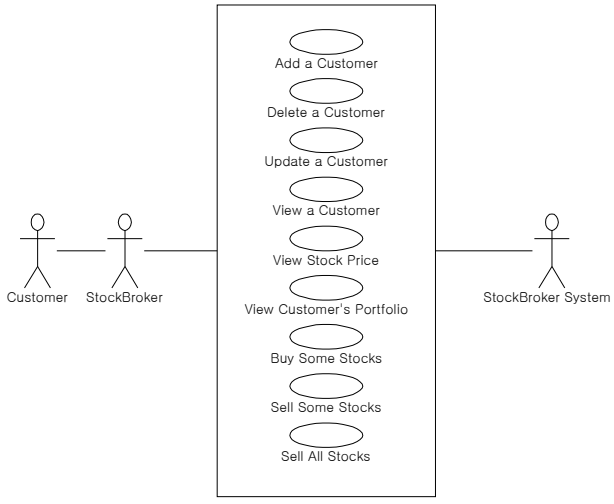
2) 서비스 유형 추출

다음의 서비스 유형은 컴포넌트로 제공되어 애플리케이션 서버에 의해 클라이언트로 서비스된다.

·새로운 고객을 추가할 수 있다.
·기존의 고객을 삭제할 수 있다.
·기존의 고객의 정보를 갱신할 수 있다.
·한 사람의 고객 정보를 볼 수 있다.
·모든 고객 정보를 볼 수 있다.
·한 고객이 보유한 주식을 볼 수 있다.
·고객은 주식을 매수할 수 있다.
·보유한 주식을 일부나 모두를 매도할 수 있다.
·증권사의 모든 주식 정보를 볼 수 있다.
·한 주식의 가격을 볼 수 있다.

3.3 UML을 이용한 애플리케이션 서버 모델링

1) Use Case 다이어그램



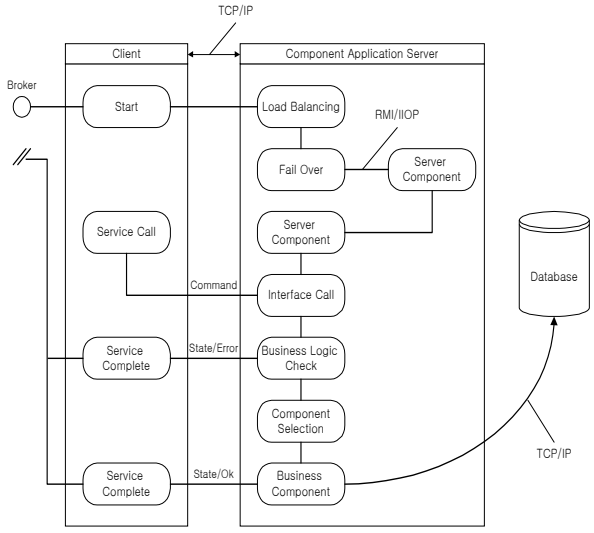
(그림 3.1) Use Case 다이어그램

제안한 StockBroker에서 클라이언트와 애플리케이션 서버 사이의 기능에 대한 Use Case 다이어그램을 보여준다.

2) Use Navigation 다이어그램

클라이언트와 애플리케이션 서버의 동작은 다음과 같은 순서에 의해 처리된다.

클라이언트와 애플리케이션 서버의 Use Navigation 다이어그램은 다음과 같다.

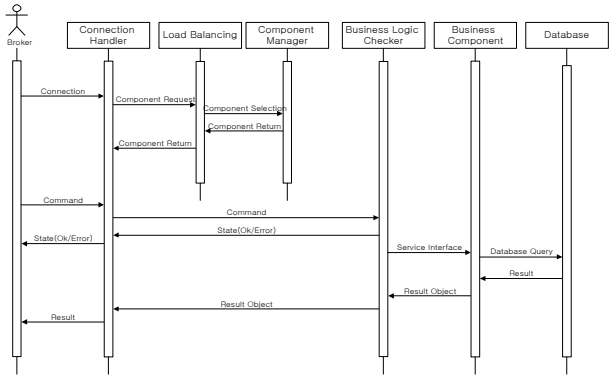


(그림 3.2) Use Navigation 다이어그램

·클라이언트에서 Broker는 주식 거래를 시작한다.
·애플리케이션 서버는 부하 균등 및 연결 복구를 처리한다.
·요청된 서버 컴포넌트를 선택하여 서비스 객체를 얻는다.
·클라이언트는 명령을 보내 인터페이스를 요청한다.
·애플리케이션 서버는 서비스 요청에 대한 비즈니스 로직을 검사한다.
·애플리케이션 서버는 클라이언트의 서비스 요청에 부응하는 비즈니스 컴포넌트를 선택한다.
·비즈니스 컴포넌트는 데이터베이스와 연동하며 결과와 처리 상태를 클라이언트에 리턴 한다.

3) Sequence 다이어그램

애플리케이션 서버의 객체 상호작용을 표시하였다.

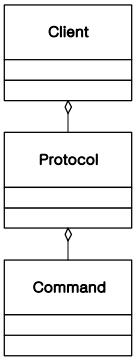


(그림 3.3) Sequence 다이어그램

4) Class 다이어그램

① 클라이언트의 Class 다이어그램

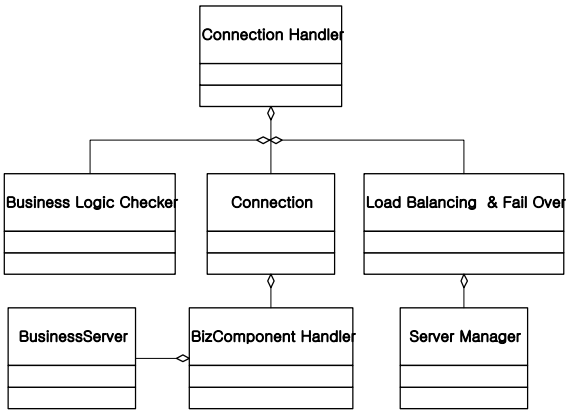
클라이언트가 애플리케이션 서버에 접속하고 사용자의 서비스 요청을 위한 클래스 다이어그램이다.



(그림 3.4) 클라이언트 Class 다이어그램

② 애플리케이션 서버의 Class 다이어그램

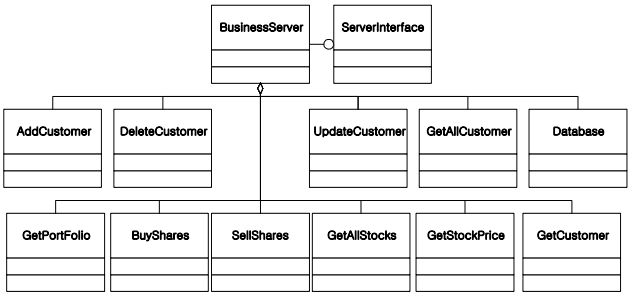
클라이언트의 접속을 처리하고, 부하 균등 및 안전 복구처리 및 비즈니스 서버의 서비스 컴포넌트를 요청하기 위해 필요한 클래스 다이어그램을 보여준다.



(그림 3.5) 애플리케이션 서버의 Class 다이어그램

3) 비즈니스 서버의 Class 다이어그램

애플리케이션 서버의 비즈니스 인터페이스 요청에 대해 각 서비스 컴포넌트를 핸들링하고 데이터베이스를 처리하기 위한 클래스 다이어그램을 보여준다.



(그림 3.6) 비즈니스 서버의 Class 다이어그램

Ⅳ. 컴포넌트 애플리케이션 서버의 구현

4.1 StockBroker의 구현 환경

1) 클라이언트

StockBroker의 클라이언트는 자바 애플릿으로 수행하도록 개발하였다. 클라이언트와 애플리케이션 서버의 동작은 TCP/IP 상위에서 소켓의 객체 입출력에 의해 상호 연동 한다.

2) 애플리케이션 서버

StockBroker의 중간 계층인 애플리케이션 서버는 클라이언트의 연결 설정, 서버 객체에 대한 부하 균등(load balancing)이나 클라이언트에 대응하는 서버가 서비스를 중단한 경우에도 연결 복구(fail over) 기능을 수행하며 비즈니스 컴포넌트를 관리하고 서비스하는 기능을 수행한다. 애플리케이션 서버와 서버 컴포넌트의 동작은 RMI/IIOP에 의해 서버 객체 서비스를 수행한다. 서버 객체는 비즈니스 로직을 컴포넌트화하여 준비된 비즈니스 컴포넌트를 관리하고 서비스한다.

동시성 서비스와 부하 조절 기능을 위해 클라이언트와의 연결에 대해서는 멀티 쓰레드(multi-thread)를 사용하여 쓰레드 풀(pool)에 의해 접속을 관리한다.

3) 데이터베이스

서버 컴포넌트와 데이터베이스는 JDBC(Java Database Connectivity)에 의해 연결하여 하나의 클라이언트에 대해 하나 이상의 데이터베이스의 맵핑(mapping)이 가능하도록 한다.

4.2. 컴포넌트 구성

1) 서버 컴포넌트

서버 컴포넌트는는 RMI/IIOP에 의해 제작되어 클라이언트의 요청에 응답하는 구조이다.

2) 비즈니스 컴포넌트

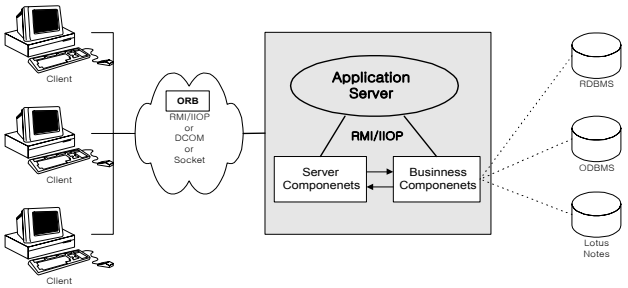
비즈니스 컴포넌트는 서버 컴포넌트의 인터페이스 호출에 의해 로드되어 특정 비즈니스 로직을 처리하는데 사용되고 데이터베이스에 대한 질의 및 결과를 서버 컴포넌트에게 넘기는 역할을 한다.

제공되는 비즈니스 컴포넌트는 다음과 같다.

① AddCustomer ; 고객을 데이터베이스에 추가
② BuyShares : 지정한 주식의 매수 주문 처리
③ DeleteCustomer : 고객의 삭제 처리
④ GetAllCustomers : 모든 고객의 정보 출력
⑤ GetAllStocks : 모든 주식의 정보 출력
⑥ GetCustomer : 한 고객의 정보 출력
⑦ GetPortfolio : 고객이 보유한 주식 정보 출력
⑧ GetStockPrice : 주식의 현재가 출력
⑨ SellShares : 주식의 임의 보유량을 매도
⑩ UpdateCustomer : 고객의 정보를 갱신

4.3 애플리케이션 서버의 구조 및 흐름

1) 애플리케이션 서버의 구조



(그림 4.7) 애플리케이션 서버의 구조

제안한 컴포넌트 애플리케이션 서버의 구조는 (그림 4.9)와 같이 서버 컴포넌트와 비즈니스 컴포넌트를 분리하여 서버 컴포넌트의 인터페이스 호출에 의해 비즈니스 컴포넌트가 요청되는 구조이며 이때 RMI/IIOP 프로토콜에 의해 컴포넌트간에 연동 하게 된다. 또한, 클라이언트와 애플리케이션 서버 사이에도 투명한 네트웍 연결을 통해 서비스를 수행한다. 구조적으로는 사용자 인터페이스 부와 애플리케이션 서

버 부분 그리고 데이터베이스 부분으로 구분된 분산 3-계층 구조를 가진다.

2) 동시성(concurrent) 서비스

클라이언트의 동시성(concurrent) 서비스를 위해 애플리케이션 서버는 쓰레드(thread) 풀을 이용하여 클라이언트에 대해 멀티 쓰레드 서비스를 수행한다.

3) 서버 컴포넌트의 호출방식

클라이언트의 서비스 요청 개시에 따라 애플리케이션 서버의 서버 컴포넌트를 얻는 방법은 RMI/IIOP에 의해 처리한다. 서버 컴포넌트는 RMI/IIOP에 의해 다음과 같이 호출되어 사용된다.

```
String serverURL = "rmi:///" + database + "/StockBrokerServer";
System.out.println("Looking up " + serverURL);
myDB = (StockBrokerInterface)Naming.lookup(serverURL);
```

(4) 서버 컴포넌트와 비즈니스 컴포넌트의 연동

서버 컴포넌트의 인터페이스 요청에 대해 11개의 비즈니스 컴포넌트를 호출하는 방식 자바 빈(java bean)의 방법을 사용한다.

```
Class aBeanCls = Class.forName("AddCustomer");
ClassLoader cl = aBeanCls.getClassLoader();
AddCustomer ac =
        (AddCustomer)Beans.instantiate(cl, "AddCustomer.ser");
ac.setName(name);
ac.setSSN(ssn);
ac.setAddr(addr);
```

(5) 비즈니스 컴포넌트와 데이터베이스

서버 컴포넌트는 데이터베이스에 대한 쓰레드 풀(thread pool)을 제공하여 동시성 서비스를 지원하며 서버 컴포넌트의 인터페이스 호출에 대해 비즈니스 컴포넌트에 데이터베이스에 대한 연결 객체를 전달한다. 비즈니스 객체는 데이터베이스에 대한 연결 객체를 전달받아 데이터베이스에 대한 질의를 수행한다.

Ⅴ. 결론 및 연구 방향

본 연구에서는 컴포넌트 기반 애플리케이션 서버를 다음과 같이 설계하고 구현하였다. 클라이언트의 연결 설정, 서버 객체에 대한 부하 균등(load balancing)이나 클라이언트에 대응하는 서버가 서비스를 중단한 경우에도 연결 복구(fail over) 기능을 수행하며 비즈니스 컴포넌트를 관리하고 서비스하는 기능을 수행한다. 애플리케이션 서버와 서버 컴포넌트의 동작은 RMI/IIOP에 의해 서버 객체 서비스를 수행한다. 서버 객체는 비즈니스 로직을 컴포넌트화하여 준비된 비즈니스 컴포넌트를 관리하고 서비스한다.

동시성 서비스와 부하 조절 기능을 위해 클라이언트와의 연결에 대해서는 멀티 쓰레드(multi-thread)를 사용하여 쓰레드 풀(pool)에 의해 접속을 관리한다.

StockBroker의 백 엔드(back end)인 데이터베이스는 mSQL2.0 데이터베이스를 사용하여 주식 거래 및 고객의 정보를 저장하고 제공하는 기능을 수행한다.

참 고 문 헌

[1] Ibrahim Cingil, "An Adaptable Workflow System Architecture on the Internet for Electronic Commerce Applications", Univ. of N.Y., 1998

[2] V. Tsaoussidis, "Resource COntrol of Distributed Application in Heterogeneous Environments", IEEE ISCC '98 , 1998

[3] Brian Jepson, "Java Database Programming", John Wiley & Sons,Inc., 1996

[4] Mary Campione, Kathy Walrath, "The Java Tutorial", Addison-Wesily, 1996

[5] Elliotte Rusty Harold, "Java Networking Programming", O'Reilly & Associates, Inc., 1996

[6] Prashant Sridharan, "Advanced Java Networking", Prentice-Hall, Inc., 1997

[7] Orfali, Harkey and Edwards, "The Essential Distributed Objects Survival Guide", Wiley Press, 1996

[8] Jim Waldo, Geoff Wyant, Ann Wolrath and Sam Kenedall, "A Note on Distributed Computing", Sun Microsystems, 1994

[9] "JDBC specification", http://splash.javasoft.com/jdbc

[10] Emerson, Darnovsky, and Bowman, "The Practical SQL Handbook", Addison-Wesley, 1989

[11] "The Java Remote Method Invocation Specification", http://chatsubo.javasoft.com/current/doc/ rmi-spec/rmi-spec.ps

[12] "Frequently Asked Questions, RMI and Object Serialization", http://chatsubo.javasoft.com/current/ faq.html