분산객체 시스템을 위한 관계형 데이터베이스 연동도구의 개발

박 우 창†

요 약

분산 데이터베이스 시스템은 여러 곳에 이질적인 데이터 소스가 존재하는 정보환경에서 데이터가 처리되는 시스템이다. OMG에 의하여 제시된 CORBA는 이 기종의 분산 환경 하에서 플랫폼에 대해 독립성을 가지고 응용프로그램들을 통합하여 상호 연동할 수 있는 표준으로 제시되었다. 본 논문은 이러한 환경에서 CORBA 객체를 통해 인터페이스 방식에 의한 코드 생성을 통하여 분산된 관계형 데이터베이스의 상호작동을 돕는 소프트웨어 도구를 개발한다. 본 논문에서 개발된 도구는 관계형 데이터베이스의 스키마를 관리하고, 스키마에 해당하는 테이블에 대한 CORBA IDL 인터페이스를 생성하며, C++ stub을 생성하는 기능을 제공한다. 또 개발된 도구를 이용한 소프트웨어 생성 과정을 설명하고, 어댑터 방식과의 비교, 구현언어에 따른 비교 등 인터페이스 방식에 관한 몇 가지 성능을 실험하고 평가하였다. 개발된 도구는 관계형 데이터베이스 상호 연동에 필요한 코드를 제공함으로써 분산 정보검색 시스템, 데이터 웨어하우징 등의 응용에 사용할 수 있다.

Development of a CASE Tool on Relational Databases for Distributed Object Systems

U-Chang Park†

ABSTRACT

In distributed data processing environments, heterogeneous data sources should corporate each other. The CORBA standards proposed by OMG solve this problem by the integration of platform independent applications. In this paper, we design and implement a CASE tool that help linking CORBA applications to relational databases by interface method. This CASE tool manipulates the database schema and generates IDL and C++ stubs corresponding to the schema. We explain the code generation process and evaluate the data access performance of the interface method compared to adapter method and various implementation languages. The implemented tool can be used for developing applications that need relational database corporation environments such as distributed information systems or data warehousing, etc.

1. 서 론

네트웍과 인터넷 기술의 발달로 데이터베이스 시스템이 중앙집중형에서 분산형으로 바뀌어감에 따라 여러 곳에 데이터소스가 존재하는 분산된 정보 환경을 상호 연동(interoperable)하기 위한 도구와 시스템의 개발이 활발히 연구되고 있다. 1980년대부터 비지니스 응용의 요구에 따라, 지역 자치성과 이질 데이터베이스의 통합 개념에 입각한 여러 형태의 클라이언트/서버(Client/Server) 시스템이 사용되었다. 현재 클라이언트/서버 시스템으로는 SQL Database 서버, Transaction Processing, Groupware, 분산객체 개념 등이 사용되고 있

※ 본 연구는 '99학년도 덕성여자대학교 연구비 지원으로 이루어졌음.
† 정 회 원 : 덕성여자대학교 전산학과 교수
논문접수 : 1999년 9월 18일, 심사완료 : 1999년 12월 9일

으나 서로 다른 플랫폼, 프로토콜, 데이터베이스 등이 혼재하여 시스템의 구축시 시스템 선택과 상호연동성을 야기한다.

이 분산환경의 상호연동의 핵심은 이미 컴퓨터 시스템에 기본기술로 자리잡은 객체지향 기술이다. 분산 객체 모델의 표준인 CORBA[1]를 이용하여 관계형 데이터베이스를 CORBA와 연동시키고 확장된 SQL 언어나 CORBA의 Query Service를 구현함으로써 클라이언트가 서버의 관계형 데이터베이스를 객체모델에 의하여 접근이 가능하도록 하는 연구가 진행되고 있다. 현재 구축된 데이터베이스의 90퍼센트 이상은 레거시(legacy) 시스템이라 부르는 관계형 데이터베이스로, 상용제품에 의하여 각각 독립적으로 구축되어 있어서 기존의 기술로는 상호연동이 어렵다. 이러한 많은 비중을 차지하는 관계형 데이터베이스시스템의 상호 연동은 통합된 정보 환경을 구축하는 핵심이다[2, 3].

본 연구는 이러한 흐름에서 CORBA 환경에 관계형 데이터베이스를 연동시키는 도구를 개발한다. CORBA 환경에서 데이터베이스 연동은 몇 가지 방법이 있으나 본 연동 도구는 CORBA 객체의 메소드가 데이터베이스 API를 포장하는 인터페이스 방식에 의한 연동 코드를 생성한다. 코드를 생성하는 기능 외에 데이터베이스 선택 기능, 데이터베이스를 관리하는 기능, CORBA 객체로 변환할 테이블 선택 기능 등을 포함한다.

CORBA 객체와 데이터베이스의 연동 방법은 인터페이스에 의한 방법외에 어댑터에 의한 방식 등이 있고, CORBA 특성상 구현언어의 선택에도 C++외 자바 언어가 많이 사용되고 있다. 본 논문에서는 구현자가 이러한 방법상에서 겪는 선택의 문제에 도움이 되도록 방법과 언어에 대하여 클라이언트 코드의 데이터베이스 접근 성능의 측면에서 비교하여 검토하였다.

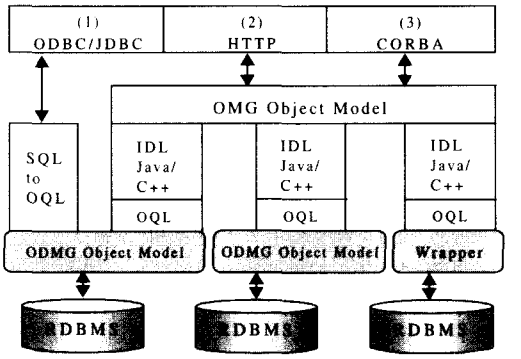
본 논문의 구성은 다음과 같다. 2장에서는 개발될 도구의 시스템 요구사항과 설계 및 구현에 관한 사항을 논한다. 3장에서는 개발된 도구의 작동 과정을 설명하고 사용 예를 보인다. 4장에서는 연동 방식의 비교와 구현 언어의 선택에 관한 실험 결과를 설명하였다. 5장에서는 결론과 앞으로의 연구에 대하여 설명한다.

2. 연동도구 시스템 설계와 구현

2.1 연동도구 시스템의 연구 배경

객체 개념에 의한 CORBA 모델은 클라이언트/서버 타입의 이질적인 데이터베이스를 통합할 수 있는 대안으로 제기된 바 있다[4, 5]. 기존의 방법이나 CORBA를 이용한 환경에서 클라이언트 응용이 데이터베이스 서버에 객체를 접근할 수 있는 환경은 (그림 1)에 보이는 것처럼 다음과 같은 방법들이 있다[6].

(1) ODBC/JDBC에 의한 SQL의 사용과 SQL의 OQL로의 변환
(2) HTTP/HTML에 의한 embedded 형태의 OQL 사용
(3) CORBA환경의 SQL 언어나 COS Query Service를 통한 OQL 사용



(그림 1) 객체에 의한 데이터베이스 클라이언트/서버 환경

위의 방법들 중 (3)은 객체 개념에 의한 객체 메시지의 호출로 분산자원의 물리적 투명성 확보와 확장성, 표준에 의한 플랫폼 투명성 등을 확보할 수 있는 방법이다. 본 연구는 객체 개념을 활용할 수 있도록 기존의 관계형 데이터베이스를 대표하는 Oracle, Informix 시스템과 ORB 제품을 대표하는 Iona의 Orbix[7]에 관계형 데이터베이스와 CORBA를 연동시키는 프로그래밍 코드를 생성하는 프로그램 자동화 도구를 개발하고자 한다.

관계형 데이터베이스를 CORBA 환경에 연동시키는 기존의 연구로 Black&White[8]사의 DB/Enable은 여러 상용 관계형 데이터베이스를 CORBA 객체 환경의 서버에서 접근할 수 있는 어댑터를 제공하는 미들웨어이다. 동적인 SQL 문법이나 저장된 프로시져를 객체에 포장시킴으로써 릴레이션을 접근한다. 이 제품은 일단 관계형 데이터베이스를 CORBA 응용 구축시 사용할 수 있는 도구로 적합하나 객체포장 개념이 아니다.

ObjectDriver[9]는 관계형 데이터베이스를 위한 객체 래퍼(wrapper)이다. ObjectDriver는 ODMG-Compliant 개방형 래퍼로 관계형 데이터베이스 재활용을 목적으로 하고 있다. 관계형 데이터베이스를 데이터베이스 연산(selection, insertion, deletion, update)을 통하여 객체 데이터베이스 처럼 접근한다. 데이터베이스 질의는 ODMG 질의어인 OQL을 통하여 C++와 Java 바인딩이 가능하다.

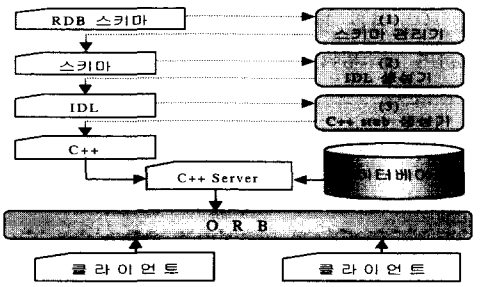
2.2 연동도구의 기능과 세부 설계

본 도구는 CORBA 기반의 분산 객체 환경에 레거시(legacy) 시스템인 관계형 데이터베이스와 연동되는 프로그래밍 환경을 개발하는 것에 목적을 둔다. 개발된 프로그래밍 환경에서는 관계형 데이터베이스의 릴레이션을 CORBA 객체로 연동시키는 프로그래밍 과정을 자동화시킬 수 있다. 본 연동 도구의 기능은 다음과 같다.

1. 관계형 데이터베이스의 스키마 관리 기능
   서버에 연결하여 서버의 데이터베이스 스키마를 연동도구의 비주얼한 인터페이스를 통해서 판독하고, 새로운 릴레이션을 생성하고, 기존의 릴레이션을 삭제하는 작업을 수행한다.
2. 릴레이션 클래스를 위한 IDL 인터페이스 생성기
   데이터베이스에서 데이터 조작에 필요한 연산은 데이터의 검색, 삽입, 삭제에 한정된다. 이를 CORBA 서비스 형태로 제공하기 위하여, 사용자가 지정하는 릴레이션에 대해 데이터를 조작하는 연산을 포함하는 IDL 인터페이스를 생성한다.
3. CORBA용 서버개발을 위한 C++ 용 stub 생성기
   서버 프로그램의 개발을 위해서는 C++ 언어 코딩이 필요하다. 이 C++ 언어로 코딩을 하는 stub을 2번의 IDL 인터페이스와 더불어 생성을 한다. 이 stub에서는 인터페이스에 해당되는 CORBA 연동코드와 데이터베이스 API를 생성시킨다.

관계형 데이터베이스와 CORBA 연동을 위한 도구의 세부 기능에 대한 설계는 다음과 같다((그림 2) 참조).

1. 관계형 데이터베이스용 스키마 관리 도구
   기존의 서버 관계형 데이터베이스에 연결하여 SQL 스키마 스크립트를 자동화 도구내로 전달하며, 구문을 분석, 판독하여 사용자에게 CORBA 객체로 변환



(그림 2) 관계형 데이터베이스와 CORBA 연동을 위한 도구의 기능

될 스키마를 선택할 수 있는 기능을 제공한다. 기존의 관계형 데이터베이스 관리도구를 쓸 수도 있지만 다음 단계인 IDL 생성기를 위하여 간단한 형태의 관리도구를 개발한다.

2. IDL 생성기
   관계형 데이터베이스 스키마로부터 IDL(Interface Definition Language)을 자동 생성하는 도구이다. IDL은 CORBA 기반 응용프로그램 개발시 객체 인터페이스를 정의하는 언어이다. 관계형 데이터베이스 스키마로부터 사용자가 선택한 객체를 CORBA 객체화하여 외부 인터페이스에서 접근할 수 있도록 하기 위해서는 릴레이션의 스키마에 해당되는 객체 인터페이스를 정의해야 한다. CORBA 객체로 변환될 릴레이션들에서, 객체의 상태 값은 튜플의 값을 갖고 객체의 메소드는 릴레이션에 관한 연산으로 구성된다. 연구의 배경에서도 설명한 바와 같이 객체가 데이터베이스에 저장된 객체일 경우 이 객체에 대한 인터페이스는 주로 객체와 관련된 데이터의 접근, 조작 연산이 주를 이룬다. 그러므로 객체의 접근, 조작에 관한 IDL 인터페이스는 한정된 문법의 형태를 갖게 된다.
   예를 들어 아래와 같이 Book이라는 스키마를 갖는 객체가 그 속성으로 Author와 Title을 가진다고 정의하자. 스키마 Book을 위한 IDL 인터페이스는 다음과 같은 연산들을 포함한다고 볼 수 있다.

```
//릴레이션 Book의 스키마 Book
//(Author,Title)
    객체 Book
    객체 Book에 관한 서버 객체
     { 첨가, 삭제, 변경 연산, 질의 연산 }
```

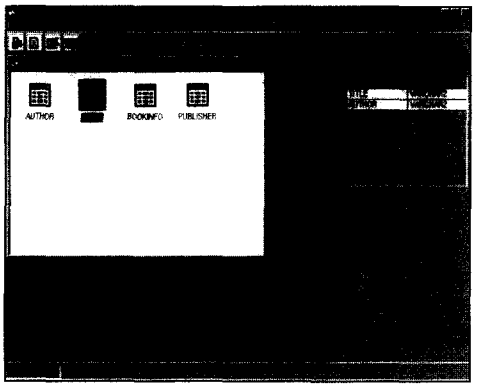
여기서 튜플 검색, 튜플 삭제, 튜플 첨가와 같은 연산은 모든 데이터베이스 스키마 객체가 기본적으로 가지고 있어야 할 요소들이다.

3. CORBA 응용 서버개발을 위한 C++용 stub 생성기
C++용 stub 생성기는 앞에서 생성된 IDL을 관계형 데이터베이스 상품에 맞추어 Embedded SQL 형태의 C++ 서버 프로그램을 생성한다. 이 과정은 먼저 기존의 ORB 제품에서 제공된 IDL 컴파일러를 이용하여 기본적인 C++ stub을 생성하고 여기에 데이터베이스 접근 메소드들에 대한 C++ 코드를 구현하는 것이다. 이때 Embedded SQL 명령어를 C++ 코드에 사용하기 때문에 상업용 데이터베이스 문법에 맞게 생성해야 한다. 본 연구는 Informix와 Oracle 두 제품에 대한 코드생성을 목표로 하였다.

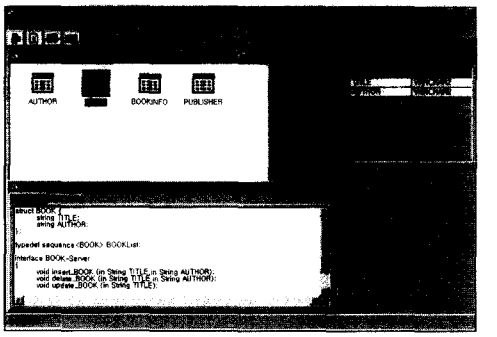
3. 연동도구 작동과정과 사용 예

3.1 연동도구의 작동 구조

자동화 도구는 비주얼 베이직 언어를 이용하여 윈도우즈 98 시스템 상에서 실행된다. 도구는 파일 관리 기능, 서버의 데이터베이스 타입에 따라 데이터베이스를 선택하는 데이터베이스 선택 기능, 서버의 데이터베이스에 연결하는 연결 기능, 서버 데이터베이스에 테이블을 생성하는 기능, 관계형 데이터베이스의 테이블에 해당되는 CORBA 객체 연동 IDL 생성 기능, C++용 코드 생성 기능 등을 포함한다. C++용 코드는 Sun Sparc 시스템의 C++ 컴파일러를 목적 코드로 하여 생성이 되며, 데이터베이스는 상용 데이터베이스인 Oracle, Informix와 연결이 되도록 코드를 생성한다. 연동 도구



(그림 3) 연동도구의 서버 연결 후 스키마 호출화면



(그림 4) 연동도구의 테이블 Book에 대한 IDL 코드 생성화면

의 데이터베이스 연결 후의 화면과 코드 생성 화면은 다음과 같다((그림 3), (그림 4)).

3.2 연동도구에 의한 코드생성 예

연동도구의 핵심 기능은 C++용 코드의 생성이다. 코드 생성의 예를 들어보기로 하자. 서버의 데이터베이스에서 읽어들인 테이블 Book의 스키마가 다음과 같다고 하자.

```
Book(Title VARCHAR(20), Author VARCHAR(20))
```

연동도구의 코드생성 부분에서 IDL 생성 과정을 거치면 앞 절에서 본 것과 같은 Book 테이블에 대해 연동될 수 있는 인터페이스 객체의 IDL이 생성된다. 즉, 여기서는 데이터를 삽입하고, 삭제하고, 변경하고, SQL 문을 통해서 질의할 수 있는 서버 개체와 메소드가 생성된다. 테이블 Book에 대한 IDL이 생성된 결과는 다음과 같다.

```
struct Book {
   char Title[20];
   char Author[20];
}
typedef sequence<Book> BookList;
Interface Book-Server // Book의 서버 객체
{
   void insert_Book (in String Title, in String Author);
   void delete_Book (in String Title, in String Author);
   void update_Book (in String Title);
   BookList query_Book(in String Query);
}
```

생성된 IDL을 구현할 수 있는 C++ 코드도 코드 생성 과정을 거치면 다음과 같이 Dynamic SQL기법을 사용하여 수행되도록 생성이 된다. 인터페이스 Book_Server의 매소드 query_Book()의 코드 생성의 예를 들면 다음과 같다.

```
BookList query_Book(in String Query,
                    CORBA::Environment &IT_env)
{ Book *buf = BookList::allocbuf(n);
  int i=0;
  struct { char title[20]; char author[20]} book;
  EXEC SQL PREPARE s FROM Query;
  EXEC SQL DECLARE c FOR s;
  EXEC SQL OPEN c;
  for(;;)
  { EXEC SQL FETCH c INTO book;
    strcpy(buf[i].title, book.title);
    strcpy(buf[i].author, book.author);
  }
  EXEC SQL close c;
  return new BookList(i, i, buf, 1);
}
```

구현된 서버 코드를 이용하는 클라이언트의 코드는 다음과 같다. Dynamic SQL 기법을 이용하여 개발자는 query_Book()의 인자를 Book 테이블을 검색하는 SQL 문을 사용한다. 클라이언트 프로그램이 서버의 데이터베이스 접근을 위하여 호출하는 코드는 다음과 같다.

```
        ...
try {
  val=mylib->query_Book("select * from book");
        ...
}
```

4. 연동 도구의 평가

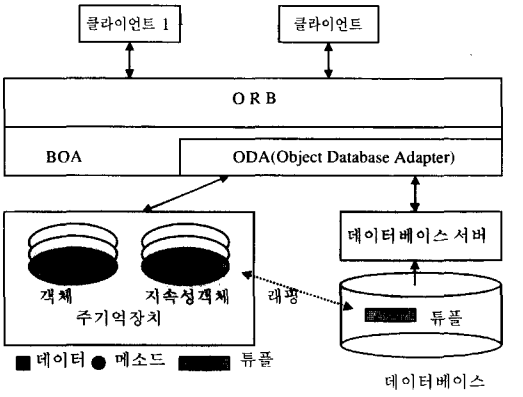
4.1 연동 방법의 비교

분산 환경에서 데이터베이스 접근 방법은 ODBC를 이용한 방법, CGI, Java의 RMI, CORBA, TP Monitor 등 여러 가지가 있다. 이중 분산 객체 개념을 이용한 CORBA에서의 데이터베이스 접근 방법은 플랫폼에 독립적으로 상호연동을 지원하는 장점을 가진다.

CORBA 구조에서 데이터베이스 접근은 다음의 세 가지 방법으로 생각해 볼 수 있다. 첫째는 CORBA 표준에서 제안된 지속성서비스(Persistence Service)[5]를 이용하는 방법, 둘째는 CORBA 객체 데이터베이스 어댑터를 이용한 어댑터 방식에 의한 데이터베이스 접근[10], 셋째는 본 논문에서 사용한 데이터베이스 인터페이스 방식에 의한 접근이다. 각각에 대해서 살펴보기로 하자.

지속성 서비스는 CORBA 구조에서 서비스의 형태로 제공된다. CORBA 구조는 ORB, 기본 서비스, 응용 서비스 형태로 구성된다. 이 중 기본 서비스로는 네이밍(Naming) 서비스, 라이프사이클(Life Cycle) 서비스, 이벤트(Event) 서비스 등이 있으며 지속성 서비스는 이러한 서비스 중의 하나이다[5]. 지속성 서비스는 데이터 조작과 접근 연산을 클라이언트에서 작동시키며, 데이터베이스의 관리는 CORBA의 서비스 형태로 결합되어 있다.

어댑터 방식에 의한 데이터베이스 접근은 CORBA 객체와 데이터베이스의 연동을 통하여 CORBA 객체가 데이터베이스 객체의 대리자 역할을 하도록 하여 구현하는 방법이다. 즉, CORBA 객체의 조작을 통하여 해당되는 데이터베이스를 조작, 접근하는 방식이다. 클라이언트는 데이터베이스 접근에 투명한 방식이다. 즉 클라이언트는 조작하는 객체가 지속성을 갖는지에 관여할 필요가 없다. 어댑터 방식을 그림으로 비교하면 다음과 같다(그림 5).



(그림 5) 객체 데이터베이스 어댑터에 의한 데이터베이스 연동

인터페이스 방식은 CORBA 객체를 통하여 데이터베이스에 접근하는 부분에 있어서 어댑터 방식과 비슷하나 CORBA 객체가 데이터베이스 객체의 대리자 역

할을 하는 것은 아니고, CORBA 객체의 메소드를 통하여 응용 프로그램 인터페이스(API, Application Programming Interface) 방식으로 접근한다. 인터페이스 방식을 (그림 5)의 어댑터 방식과 비교하면 ODA가 없이 CORBA 객체의 메소드를 통해서 데이터베이스를 접근하는 것이다.

지속성 서비스는 CORBA에 명세 되어 있기는 하지만 아직 구현된 사례가 없고, 어댑터 방식은 상용 제품[11]으로 구현된 것이 있으나 주로 객체지향 데이터베이스를 중심으로 이루어지고 있다.

본 연구에서는 비교 분석이 가능한 어댑터 방식과 인터페이스 방식의 두 가지 구현 방법에 대하여 데이터베이스 접근 성능을 비교하였다. 클라이언트에서 1개의 튜플을 읽어오는 작업을 1000회 반복하여 얻은 시간을 측정하였다. 데이터베이스는 Oracle을 사용하였고, 클라이언트와 서버의 코드는 C++를 사용하였으며, Unix 운영체제의 한 개의 머신에 데이터베이스와 서버, 클라이언트 프로그램이 존재하는 환경이다. 성능 측정의 결과는 <표 1>과 같다. 두 방식이 데이터베이스를 접근하는 코드는 비슷하나, 어댑터 방식은 CORBA 객체를 어댑터에 의하여 관리함으로써 활성화 시간을 빨리할 수 있다고 결론을 내릴 수 있다.

〈표 1〉 어댑터 방식과 인터페이스 방식의 데이터베이스 접근 속도

	접근 시간(milisecond)
어댑터 방식	6035
인터페이스 방식	7019

4.2 연동 언어의 선택에 따른 처리 속도의 비교

CORBA는 인터페이스 정의와 구현을 분리하는 특징을 갖고 있어서 구현자가 응용을 여러 언어로 구현하여 한 서버 혹은 여러 서버에 분산시킬 수 있다. 데이터베이스가 포함된 응용에도 이러한 기법이 이용될 수 있으므로 본 연구에서는 인터페이스 방식에 대하여 구현 언어에 따라 데이터베이스 접근 속도를 측정하였다. 현재 가장 많이 쓰이는 구현 언어로는 C++와 Java가 있다. 본 연구는 CORBA와 데이터베이스가 결합된 응용에서 서버와 클라이언트의 언어로 C++와 Java를 선택할 때 가능한 다양한 구현 조합에 대하여 데이터베이스 접근 시간을 측정하여 성능을 비교하였다.

4.2.1 실험 개요 및 환경

서버와 클라이언트의 머신으로 Sun UltraSparc 시스템을 사용하였다. 데이터베이스는 Oracle을, ORB로는 Iona사의 OrbixMT2.3과 OrbixWeb3.0을 사용하였다[7]. Java와 C++ 컴파일러로 각각 JDK1.1.6[12]과 버전 4.1을 사용하였다. 그리고 Java 서버 구현에 Oracle JDBC thin Driver를 사용하였다. 그리고 Oracle 데이터베이스에 각 레코드 당 14bytes크기의 500개 레코드를 읽어오는 시간을 10번 측정하여 그 평균값을 계산하였다. 측정된 시간은 다음과 같다.

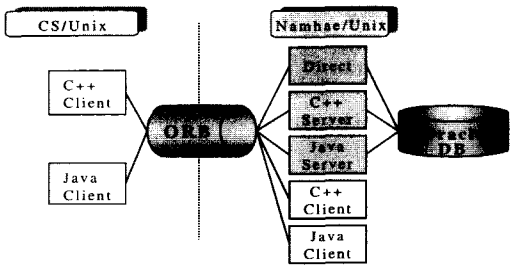
(1) 전체시간(total time) -클라이언트에서 get() 메소드를 실행시키는 전체시간,
(2) 데이터베이스 시간(fetch time) - 서버 쪽에서 순수하게 데이터베이스를 읽어내는 시간,
(3) 서버시간(server time) - 마지막으로 서버 쪽에서 데이터베이스를 읽는 시간을 포함한 전체동작 시간

즉 전체시간(total time)은 클라이언트 시간 + 네트웍 시간 + 서버시간(server time) 이고, 서버시간은 처리시간 + 데이터베이스 시간(fetch time)으로 계산될 수 있다.

4.2.2 실험의 내용

우선 데이터베이스의 테이블을 접근하는데 있어서 릴레이션의 튜플 단위로 서버와 클라이언트 사이를 이동하는 오버헤드를 줄이기 위해 IDL의 sequence타입을 이용하여, 읽어낸 레코드를 sequence 블록으로 묶어서 클라이언트로 한번에 전송하도록 하였다. CORBA의 특징인 구현 언어 독립성을 이용하여 C++와 Java로 서버와 클라이언트를 여러 유형으로 구현 조합하고, 각각의 경우 데이터베이스를 읽어오는 시간을 측정하여 성능을 비교하였다. CORBA 객체 서버는 데이터베이스와 항상 같은 머신에 두었다. 반면 클라이언트는 객체 서버와 함께 같은 머신에서 ORB를 통과하는 경우와 다른 머신에서 ORB를 통해 서버객체와 통신하는 두 가지 방식으로 나누어서 실험하였다. 실험에 사용된 모델의 개념도는 다음과 같다.

그리고 분산객체의 처리를 Java에서 응용한 RMI를 이용해서 CORBA구현의 경우와 같은 방법으로 시간을 측정하였다. 마지막으로 지금까지 실험한 분산객체 처리 방법과 대비하여, C언어를 이용한 CGI방식과 Oracle DBMS에서 제공하는 프리컴파일러(precompiler) ProC를(Direct 방법) 이용한 데이터 엑세스의 시간(direct) 측정결과를 추가하였다.



(그림 6) 구현 언어에 따른 데이터베이스 서버와 클라이언트 모델

〈표 2〉 구현언어에 따른 데이터베이스 접근 속도 비교 - 서버와 클라이언트를 같은 머신에 둔 경우

서버	클라이언트	total time(ms)	server time(ms)	fetch time(ms)
c++	c++	873.87	857.97	460.42
c++	java	931.70	919.73	503.94
java	java	1887.80	1689.40	1058.56
RMI		2395.60	1823.50	1299.88
CGI		747.84		7.16
Direct		773.93		7.89

〈표 3〉 구현언어에 따른 데이터베이스 접근 속도 비교 - 서버와 클라이언트를 다른 머신에 둔 경우

서버	클라이언트	total time(ms)	server time(ms)	fetch time(ms)
c++	c++	986.76	933.03	505.15
c++	java	936.80	893.21	466.64
java	java	3589.40	1541.26	1021.90
RMI		3399.60	1705.30	1302.16

4.2.3 실험 결과 및 분석

우선 CORBA를 공통적으로 사용하고, 클라이언트와 서버를 같은 머신에서 실행시킨 경우와 다른 머신에서 실행시킨 경우를 살펴보자. 먼저 전체시간(total time)은 모든 경우에서 서버와 클라이언트를 다른 머신에 두었을 때 네트웍 상의 딜레이(delay)로 속도가 느리게 나타났다.

C++ 구현언어

모든 실험결과 중에서 같은 머신에서 서버와 클라이언트를 모두 C++로 실험한 경우에 측정속도가 가장 빠르다. 하지만 서버를 C++로 사용하고 클라이언트를 Java로 사용한 경우에서 보면, 같은 머신에서 측정한 전체시간(total time)과 다른 머신에서 측정한 전체시간(total time)은 크게 차이를 보이지 않는다. 오히려 다른 머신에서 측정한 서버시간(server time)과 데이터베이스시간(fetch time)이 줄어드는 실험결과를 볼 수 있다. 이는 클라이언트의 구현을 Java로 할 경우, 서버 측에서 발생할 수 있는 부하가 클라이언트로 분산됨으로써 서버시간이 단축되기 때문이다. 또 같은 머신을 사용하게 될 경우 자원을 공유로 발생하는 경쟁을 피할 수 있기 때문이다.

즉, 객체들이 지역적으로 분산되어 있는 분산객체의 환경과 Java언어의 뛰어난 이식성과 유연성을 고려한다면, 서버 쪽은 속도 면에서 우수한 C++언어를 사용하고, 클라이언트의 구현언어로는 Java를 사용하는 것이 더욱 효과적일 것으로 기대된다.

Java 구현언어

Java 서버와 Java 클라이언트를 사용하는 경우 전체 처리 시간은 다음과 같이 볼 수 있다. Java 클라이언트에서 수행되는 시간, 클라이언트 프로그램과 서버 프로그램과의 연결 및 서버 프로그램의 수행 시간, 서버 프로그램의 데이터베이스 접근 시간이다. 이 중 C++ 서버를 사용했을 때 보다 시간이 많이 증가한 부분은 클라이언트와 서버 프로그램의 연결 및 서버 프로그램의 수행 시간이다. 그 이유는 Java로 구현된 서버 코드는 많은 클래스를 ORB로부터 상속받기 때문에 사용자가 작성한 코드보다 프로그램의 길이가 커지며, 이 클래스를 인터프리트 하는데 많은 시간이 소요되기 때문이다. 이점은 ORB에 관련된 자바 코드를 컴파일 하든지 아니면 Java언어를 구현언어로 사용할 수 있는 C 혹은 C++ 등 컴파일 언어로 구현한 ORB를 사용함으로써 보완할 수 있다.

CORBA와 RMI비교

양쪽 모두를 Java로 구현하여 측정한 시간을 비교하면 RMI를 사용한 경우보다 CORBA를 사용한 경우에서 더욱 우수한 성능을 보이는 것을 알 수 있다. RMI는 다른 언어의 구성 요소를 가지지 않고 순수하게 Java로만 솔루션을 구축하고자 한다면 최선의 선택이 될 수도 있다[12]. 그러나 이미 앞의 실험 결과에서 보았듯이 Java로 서버와 클라이언트를 모두 구현한 경우에 우수한 성능을 기대기는 어렵다. 그리고 RMI는 사용 가능한 언어가 Java로만 한정된다는 큰 한계를 안고 있다. 그에 비해 CORBA는 매우 다양한 언어를 지원할 수 있기 때문에 유리하다.

5. 결론과 앞으로의 연구

본 논문의 연동 도구는 관계형 데이터베이스를 CORBA 환경에 접근시키는 프로그램 환경을 지원한다. 도구는 서버의 데이터베이스를 관리하고, 데이터베이스 접근을 위한 CORBA IDL과 C++ 용 코드를 생성한다. 많은 분산 응용에서 데이터베이스의 상호연동은 필수이지만 이를 해결하기 위한 CORBA와 관계형데이터베이스의 연동 지원 도구는 많지 않다. 본 도구는 이러한 환경을 지원하기 위한 시도로써, CORBA 객체의 메소드를 통하여 데이터베이스를 접근하는 인터페이스 방식을 사용한 소프트웨어 개발 도구이다.

연동 방식의 선택에 있어서 인터페이스 방식은 어댑터 방식보다 비슷한 성능을 내며 어댑터 방식은 개발자가 쉽게 응용을 구축할 수 있는 장점이 있다. 언어의 선택에 있어서는 서버쪽의 구현은 C++가 필수이지만 클라이언트는 자바 언어가 C++ 언어보다 크게 성능이 떨어지지 않음을 알 수 있었다.

객체에 기초한 데이터베이스의 상호연동 기술은 객체지향 데이터베이스뿐 아니라 관계형 데이터베이스 등 서로 다른 데이터 모델과 서로 다른 데이터베이스 제품을 통합하는 기초 기술이 된다. 도구는 이러한 기술을 지원하며 서로 다른 관계형 데이터베이스를 통합해야하는 데이터 웨어하우징, 멀티 데이터베이스의 기초 기술이 되고, 분산된 정보를 관리하는 분산 검색시스템, 원격의료 등에 이용할 수 있다.

앞으로의 연구는 개발된 도구를 실제 응용 구축에 적용을 하며 문제점을 해결해 나가는 것이다. 특히 복잡한 데이터베이스 스키마에 대한 코드 생성, 클라이언트와 서버 사이의 수행 결과의 효율적인 전송, 또 클라이언트가 쉽게 데이터를 접근할 수 있는 질의어의 제공 등을 도구를 통하여 지원하는 문제이다.

참 고 문 헌

[1] R.G.G. Cattell, Object Database Standard : ODMG-93, Morgan Kaufmann, 1993.

[2] Michael L. Brodie, "The Promise of Distributed Computing and the Challenge of Legacy Information Systems," Advances in Object-Oriented Database Systems, NATO ASI series, 1993.

[3] William Kent, "Object-Orientation and Interoperability," Advances in Object-Oriented Database Systems, NATO ASI series, 1993.

[4] Object Management Group, "The Common Object Request Broker : Architecture and Specification," pp.29-38, John Wiley & Sons, Inc, 1991.

[5] Object Management Group, Common Object Request Broker : Architecture and Specification, at http://www.omg.org/corbask.html, 1996.

[6] The OQL Standard Emerges, Byte, March 1998, pp.55-56.

[7] Orbix at http://www.iona.com/.

[8] Black & White web page, http://www.blackwhite. com/.

[9] http://www.inria.fr/cermics/dbteam/ObjectDriver/.

[10] 박우창, "CORBA 객체의 지속성을 위한 관계 데이터베이스용 객체 데이터베이스 어댑터의 구현", 데이터베이스저널, 제3권 2호, 한국 데이타베이스 학회.

[11] Object Design, ObjectStore/DBConnect, Product Brief, Object Design, Inc., Burlington, MA, 1995.

[12] Java Homepage at http://java.sun.com/.

[13] Jian Hu, et al., "CORBA as Infrastructure for Database Interoperability," atttp://www.ri.bbsrc.ac.uk/bioinformatics/corba-multidb-pdcs/corba-multidb.htm.

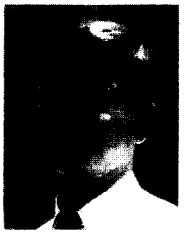
[14] Laks V. S. Lakshmanan, "SchemaSQL - A Language for Interoperability in Relational Multi-database Systems," pp.239-250, VLDB 1996.

[15] Evaggelia Pitoura, Omran Bukhres, and Ahmed Elmagarmid, "Object Orientation in Multidatabase Systems," ACM Computing Survey, Vol.27, No.2, June 1995, pp.141-195.

[16] Uchang Park, "An Algebraic Formulation of Aggregative Closure Query," Theoretical Computer Science, Vol.166, pp.49-62, Elsevier Science, Netherlands.

[17] 박우창외 5인, CORBA 환경을 위한 관계 데이터베이스용 소프트웨어 개발도구", 제12회 정보처리학회 추계 학술발표대회, 1999년 10월.

[18] 박우창외 4인, CORBA 환경에서 데이터베이스 클라이언트 성능평가", 제12회 정보처리학회 추계 학술발표대회, 1999년 10월.



박 우 창

e-mail : ucpark@center.duksung.ac.kr

1982년 서울대학교 계산통계학과 졸업(학사)

1985년 서울대학교 대학원 계산통계학과 전산과학전공 졸업(석사)

1993년 서울대학교 대학원 계산통계학과 전산과학전공 졸업(박사)

1985년~1988년 울산대학교 공과대학 전자계산학과 전임강사

1996년~1996년 미국 LOS ALAMOS 국립연구소 방문연구원

1988년~현재 덕성여자대학교 자연과학대학 전산학과 부교수

관심분야 : 분산 데이터베이스, 객체지향시스템, 인터넷 응용