

論文99-36C-2-4

RMI와 CORBA 환경하의 객체 번역 시스템의 설계 및 구현

(Design and Implementation of Translation System between RMI to CORBA)

玄 武 庸 *, 金 滉 **, 李 相 潤 ***

(Mu Yong Hyun, Shik Kim, and Sang Yun Lee)

요 약

CORBA IDL과 RMI는 대표적인 분산 객체 모델로서 분산 처리를 위한 서로 다른 접근 방식을 제공한다. CORBA 규약은 모든 프로그래밍 환경을 지원하는 범용성을 목표로 하고 있으나 또 다른 분산 프로그래밍 환경인 RMI과는 상호 호환성이 결여되어 있다. 본 논문에서는 상호 호환성이 결여된 두 분산 환경하에서 생성된 객체 사이의 상호 연동성(interoperability)을 지원하기 위한 한 방안으로서 객체 번역 시스템을 구현하였다. 제안된 시스템은 분산 객체의 중요한 특성인 바인딩, 상속성, 다형성, 객체의 전달, 콜백을 고려하여 설계되었다. 시스템의 유용성을 검증하기 위하여 하나의 분산 응용 프로그램을 제안하였고 이를 RMI 환경하에서 개발한 뒤 CORBA 버전으로 번역을 시도하였다. 또한, 시스템에 의해 자동 생성된 프로그램과 CORBA IDL의 지원에 의해 구현된 프로그램과의 성능을 상속성, 객체의 전달, 다형성 중심으로 비교 및 분석하였다. 실험 결과, 제안된 번역 시스템에 의한 프로그램이 분산 환경의 지원을 받아 직접 구현된 프로그램과 대등한 성능을 발휘함을 확인 하였다.

Abstract

CORBA IDL and RMI, the representative of distributed object model, support different approaches for distributed computing environments. CORBA approach does not support the application generated by RMI, even though it supports multiple programming languages. Object translation system between RMI to CORBA is designed and implemented for interoperability of distributed objects on the two different distributed programming environments. Suggested system is considered binding, inheritance, polymorphism, object passing, and callback which are key properties on the distributed objects. To validate the usefulness of the proposed system, we adopt a model for distributed objects and implement it as an application on the RMI environment. And, we translated it to an application on the CORBA environment. We observed the performance of translated application in the respect of inheritance, object passing, polymorphism across the machine boundary. Experiments showed that performance of distributed objects, those generated by translation system and those implemented on the CORBA environment, is equivalent.

* 正會員, 大元工科大學 電子計算科
(Dept. of Computer Science, Daewon Technical
College)

** 正會員, 世明大學校 情報通信學科
(Dept. of Telecommunication System, Semyung

Univ.)

*** 正會員, 大元工科大學 電算情報處理科
(Dept. of Information Processing, Daewon Technical
College)

接受日字: 1998年9月21日, 수정완료일: 1999年2월5일

I. 서론

분산 객체 컴퓨팅은 인터넷의 활성화와 더불어 각광 받고 있는 분야 중의 하나이며 많은 응용 소프트웨어들이 분산 객체 기술을 이용한 컴포넌트 형태로 개발되고 있다^[1,2,3,4,5]. 이와 관련되어, 최근의 두드러진 성과중의 하나는 웹 브라우저 관련 분야에서의 분산 객체 기술의 지원이다^[6,7]. 웹 브라우저는 분산 객체로 구성된 프로그램을 전송 받아서 실행시킨 후 그 결과를 웹 페이지 내부에 출력한다. 현재 대부분의 웹 브라우저들은 다양한 언어로 구성된 분산 객체들을 제공하고 있다.

CORBA(Common Object Request Broker Architecture)^[2]와 RMI(Remote Method Invocation)^[8]는 대표적인 분산 객체 모델로서 분산 처리를 위한 서로 다른 접근 방법을 제공한다. CORBA는 원격 시스템에 존재하는 객체의 메소드를 마치 로컬 객체의 메소드 처럼 호출하게 하는 하부 구조를 제공한다. IDL(Interface Definition Language)를 제안함으로써 분산 객체 접근을 위한 인터페이스의 정의 단계와 구현 단계를 분리하였고 그 결과 이미 개발된 컴포넌트나 다른 언어와의 통합(mapping)이 용이해짐으로써 대규모 응용 프로그램의 구축시 적합한 접근 방식이다.

RMI 메커니즘은 원격 객체에 정의된 메소드를 호출하는 다른 접근 방법을 제공한다. RMI의 기본 아이디어는 객체를 생성할 때 내부에 정의된 메소드를 다른 자바 가상 머신에서 호출할 수 있도록 허용하는 것으로서, 이러한 접근 방법은 자바 객체를 위한 원격 함수 호출(RPC) 메커니즘을 제공한다. RMI 메커니즘은 원격 메소드 호출을 위하여 공통적으로 Proxy stub과 Skeleton stub을 제공함으로써 통신을 담당하는 하부구조의 투명성을 보장한다^[1]. RMI 메커니즘은 자바로 구현된 소규모 응용프로그램의 개발에 적합한 방식이다.

CORBA는 기존의 컴포넌트와의 통합을 지원하는 시스템 통합 기술이지만, RMI와의 통합은 불가능하다. 따라서, CORBA와 RMI 환경하에서 개발된 분산 객체간의 상호 운영성(interoperability)에 대한 필요성이 제기되고 있으며, 이를 지원하기 위한 프로토타입(prototype)들이 제안되고 있다^[9,10]. 본 논문에서는 접근 방식이 다르고 상호 호환성이 결여된 주요한 두

프로그래밍 환경인 RMI와 CORBA에서 생성된 객체 사이의 상호 운영성을 지원하기 위한 모델로서, 두 환경에서 생성된 객체사이의 자동 번역 시스템을 구현하였다. 번역 정보 저장을 위한 자료구조를 계층화된 클래스 구조로서 제안하였고, 번역 상의 중요한 쟁점들인 바인딩, 상속성, 다형성, 객체의 전달 등에 관해 논한 뒤 쟁점들과 관련된 번역 전략을 제시하였다.

제안된 시스템의 유용성을 검증하기 위하여 하나의 분산 응용 프로그램인 WB_DAO(WhiteBoard based on Distributed Active Objects)를 채택하였고 이를 RMI 환경하에서 개발한 뒤 번역 시스템을 이용하여 CORBA 버전으로의 자동 번역을 시도하였다. WB_DAO는 네트워크로 연결된 여러명의 사용자들이 하나의 드로잉 영역을 공유하면서 이 영역에 가해지는 모든 상태 변화를 각각의 사용자들이 유지할 수 있도록 하는 분산 화이트 보드 시스템이다. 번역에 의해 자동 생성된 시스템의 성능을 평가하기 위해서 번역 시스템에 의한 WB_DAO와 CORBA 환경의 지원에 의해 구현된 WB_DAO와의 성능을 상속성, 객체의 전달, 다형성 중심으로 비교하는 실험을 실시하였다.

이 논문의 구성은 서론에 이어 2장에서는 번역 시스템 설계시 고려해야 할 분산 객체의 기본 특성에 대해 알아보고, 3장에서는 제안된 시스템의 구성에 대해서 설명하기로 한다. 4장에서는 제안된 방법의 유용성 및 성능 검증에 관한 실험 결과를 제시하고, 5장에서 결론을 맺는다.

II. 분산 객체의 주요 특성

분산 객체간의 번역 시스템의 구현 시 고려해야 할 특성으로서 원격 객체 바인딩, 상속성, 객체의 전달, 다형성, 콜백이 있다. 분산 환경에서의 바인딩이란 원격 호출에 관련된 객체사이에 메시지 전달을 위한 연결 설정 과정을 의미한다. 바인딩 과정을 통해서 원격 호출에 관련된 객체들은 상대방의 주소 및 메시지 전달을 위해 사용할 프로토콜 등을 결정한다. 바인딩의 시점과 방법은 응용 프로그램의 성능과 융통성에 상당한 영향을 미친다. 본 논문에서 기반으로 하고 있는 두 분산환경에서는 응용 프로그램의 융통성에 중점을 둔 동적 바인딩을 채택하고 있다. 명명 서비스(Name Service)에 의한 바인딩은 RMI에서 지원되지 않으므로 고려 대상에서 제외하기로 한다.

상속성은 객체지향 프로그래밍의 중요 특성 중의 하나이다. RMI는 분산 객체를 지원하기 위해서 표준 자바 언어를 확장한 형태이므로, 네트워크 간의 상속성을 지원하는 반면, CORBA는 IDL에 등록된 상속성만 지원한다. 따라서, 번역에 의해 생성된 CORBA 객체 사이의 상속성이 성립되기 위해서는 번역 단계에서 상속 관계를 IDL에 등록하는 절차가 필요하며, IDL에 등록된 인터페이스의 구현 클래스 생성 시에도 그 상속 관계는 유지되어야 한다.

다형성은 상속관계 내의 다른 클래스들의 객체들이 동일한 멤버 함수 호출에 대해 다르게 반응하는 특성을 말한다. RMI는 분산 객체의 다형성을 지원하지만 CORBA는 다형성을 직적으로 지원하지 않는다. 따라서, 다형성을 가지는 분산 RMI 객체에 대한 번역시 원격 객체를 전달받은 수신측에서 객체의 실제 클래스 형을 추출하기 위한 코드를 첨가하여 다형성을 간접적으로 지원해야 한다.

RMI는 원격 메소드 호출시 파라미터나 반환값으로서 객체가 사용될 경우 값에 의한 전달방법을 지원하지만, CORBA는 참조에 의한 전달 방법만 지원한다. 따라서, 값에 의한 전달방법을 사용한 객체의 번역시 전송 받은 객체에 대한 복사본을 생성하는 추가적인 코드가 필요하다. 제안된 시스템에서는 표준 JDK에서 기본적으로 제공되는 객체 직렬화(object serialization) 기법을 이용하여 객체의 상태를 전달하는 추가적인 코드를 생성한다.

콜백은 서버가 클라이언트의 메소드를 호출하는 메카니즘을 의미하며 응용 프로그램을 클라이언트/서버 모델로 구현할 때 필수적으로 사용된다. RMI 환경에서는 클라이언트 객체의 등록 및 콜백을 위한 프락시 생성 루틴들을 API 형태로 제공한다. CORBA 환경에서는 클라이언트 객체에 대한 참조를 서버측으로 전송한 뒤 이를 통해 클라이언트 객체에 접근하도록 하고 있으며 콜백을 처리하기 위한 쓰레드를 추가적으로 실행시켜야 한다.

III. 객체 번역 시스템의 설계

본 시스템은 번역의 초기 단계에서 번역 정보들을 저장하기 위한 클래스들을 생성하여 이후의 모든 단계에서 동적으로 관리한다. 이 장에서는 제안된 번역 정보 저장을 위한 클래스들의 종류 및 그 역할을 클래스

계층도를 중심으로 알아본 뒤 번역의 전 과정을 단계별로 상세하게 설명하기로 한다.

1. 번역을 위한 자료구조

RMI 메카니즘에 의해 생성된 객체를 CORBA 객체로 번역하기 위해서는 해당 클래스에 대한 다양한 정보, 즉, 클래스 이름, 정의된 메소드 명세, 애트리뷰트 명세, 상속 관계 여부, 원격 메소드 호출 관련 정보 등의 사항들이 필수적으로 요구된다. 이를 위해, 제안된 시스템에서는 번역의 첫 단계인 번역 정보 구축 단계에서 번역 대상 클래스에 대한 소스 차원의 부분 파싱을 통해 필요한 정보를 수집, 계층화된 클래스 구조로 유지한 뒤 이를 번역 클래스 생성 단계에서 이용하도록 하였다. 본 시스템에서 제안한 클래스 및 그 상속관계 계층도는 그림 1과 같다.

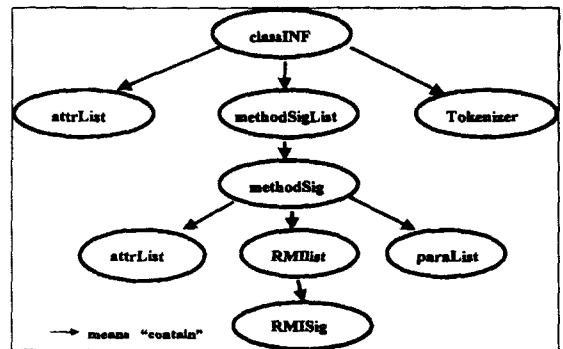


그림 1. 번역 정보 클래스 간의 상속 관계 계층도
Fig. 1. The Inheritance Hierarchy of the Classes for Translation System.

*transCommon*은 각 클래스에서 공통적으로 사용되는 상수값 및 메소드 들을 정의한다. *methodSig*는 메소드에 관한 정보를 정의하며 메소드 이름, 반환 데이터의 형, 전달되는 파라미터, 지역 변수, 원격 메소드 호출 등에 관한 상세한 정보들이 저장된다. *methodSigList*는 클래스 내에 정의된 모든 *methodSig*에 대한 목록을 저장하며, 이에 대한 효율적인 접근 및 관리를 위한 다양한 메소드들을 포함한다. *attrList*는 특정 클래스 내에 정의된 클래스 애트리뷰트에 대한 정보, 즉 애트리뷰트의 이름, 형 등이 저장되며, *paraList*는 메소드로 전달된 파라미터에 관한 정보들을 유지하며 *methodSig* 내부에서 인스턴스화 되어 사용된다. *RMSig*는 클래스 내부에 존재하는 RMI signature 정보를 정의한다. 원격 객체

의 특정 메소드를 호출하는 시점에서 나타나는 원격 객체의 이름, 메소드 명, 전달되는 파라미터 정보 등이 저장된다. *RMIList*는 클래스 내부에 존재하는 모든 *RMISig*에 대한 목록을 유지, 관리한다.

정확한 번역을 위해서는 대상 클래스의 소스 코드를 토큰 단위로 분리한 뒤 분석하는 과정이 필요한데, *Tokenizer*는 이를 위한 애트리뷰트와 메소드들을 포함한다. *classINF*는 위에서 언급된 모든 번역 정보들을 일괄적으로 관리하기 위해서 고안된 클래스로서, 번역 대상 클래스에 대한 모든 번역 정보들을 유지, 관리하는 컨테이너(Container)의 역할을 담당한다. 그림 1에 나타난 클래스간의 포함 관계 계층도는 그림 2와 같다.

2. 번역 시스템의 구성

본 논문에서 제안한 객체 번역 시스템은 크게 4부분으로 구성된다. 그림 3은 제안된 번역 시스템의 구성도이다. 첫 번째 단계인 번역 정보 구축 단계는 번역에 필요한 정보를 생성하는 단계이다. 입력된 RMI 객체 소스들을 토큰 단위로 분리하여 분석한 뒤 획득된 정보를 그림1, 2와 같은 클래스 계층 구조로 번역 데이터베이스에 저장한다. 두 번째 단계인 번역 태그 첨가 단계에서는 구축된 번역 정보를 이용, RMI 객체들을 분석한 뒤 적절한 번역 정보를 담은 태그를 추가한다. 세 번째 단계인 태그 해석 단계는 첨가된 번역 태그를 분석하여 번역 코드를 생성하는 단계이다. 이 단계에서 생성된 번역 코드들은 후처리 과정을 거쳐 적절한 위치에 소스 코드 형태로 첨가된다. 마지막 단계인 CORBA 클래스 생성 단계에서는 IDL 및 번역이 완료된 최종 클래스의 생성이 이루어 진다.

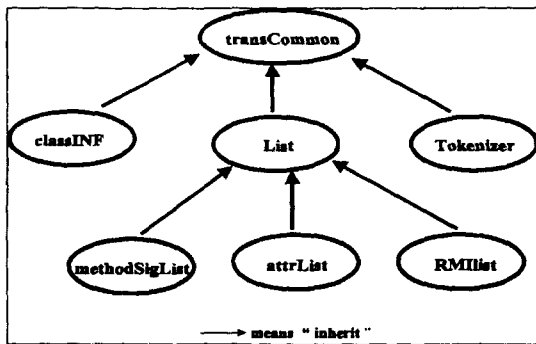


그림 2. 번역 정보 클래스 간의 포함 관계 계층도
Fig. 2. The Containment Hierarchy of the Classes for Translation System.

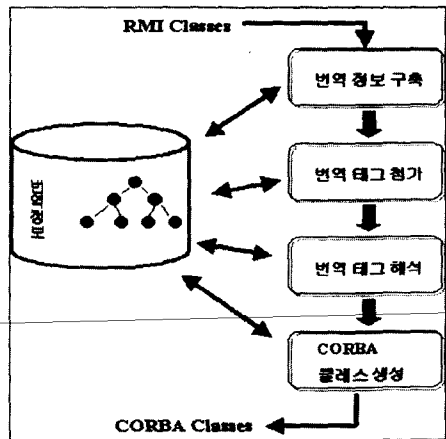


그림 3. 시스템 구성도
Fig. 3. A Structure of Translation System.

1) 번역 정보 구축 단계

이 단계에서는 입력된 RMI 객체 소스들을 토큰 단위로 분리한 뒤, 부분 파싱을 통해서 메소드 명세 리스트, 클래스 애트리뷰트 리스트, RMI 정보 리스트등을 추출, *classINF*에 저장한다. 또한, 생성된 *classINF*를 이용하여 원격 메소드 호출시에 전달되는 파라미터가 사용자 정의 객체인 경우, 이 객체의 상속 관계 및 재정의 메소드(override method)의 목록을 추출하여 *classINF*에 등록시킨다. 이 정보는 번역 태그 첨가 단계에서 분산 객체의 상속성 및 다형성의 처리에 사용된다. 이 과정을 알고리즘으로 간략화 하면 다음과 같다.

Algorithm createClassINF

```

{
    classINF classinfo = new ClassINF();
    createTokens(RMI_class); // RMI 클래스 소스를 토큰 단위로 분리
    while(hasMoreTokens())
    {
        tokens = readLine(); // 한 라인의 토큰들을 얻음
        switch(getSigType(tokens))
        {
            case Attr_Signature : classinfo.addAttrList(tokens); //애트리뷰트 정보 등록
            case Method_Signature : classinfo.addMethodList(tokens); //메소드 정보 등록
            case RMI_Signature : classinfo.addRMIList(tokens); //RMI 정보 등록
        }
    }
}
    
```

2) 번역 태그 추가 단계

이 단계에서는 구축된 번역 정보를 이용, 입력된 RMI 객체 소스에 대한 라인 단위의 부분 파싱을 통해 제안된 분산 객체 특성과의 관련성 여부를 조사한 뒤 적절한 번역 태그를 추가한다. 번역 태그는 번역 정보와 관련 메소드를 정의한 클래스를 의미한다. 추가된 번역 태그에는 번역 코드 생성 단계에서 사용될 생성 정보가 저장되며, 이 정보들은 태그마다 고유한 의미를 지닌다. 태그 추가 과정을 알고리즘으로 간략화하면 다음과 같다.

Algorithm insertTransTag

```

{
  while(hasMoreLines)
  {
    tokens = readLine(); // 한 라인의 토큰들을 얻음
    switch(getSigType(tokens))
    {
      case BindRemoteObj_Sig : //원격 객체의 바인딩 관련 태그 생성 및 추가
        insertTag(tagBindRemoteObj);
      case PolymorphicObj_Sig : //다형성 관련 태그 생성 및 추가
        insertTag(tagPolymorphicObj);
      case RemoteMethodCall_Sig : // 전달 방법에 따른 분산 객체의 처리
        if (ObjectPassByCopy_Sig) insertTag(tag SndObjByCopy);
        else if (ObjectPassByReference_Sig) insert Tag(tagSndObjByRef);
      case inviteCallback_Sig : // 콜백 관련 태그 생성 및 추가 (클라이언트 측)
        insertTag(tagInviteCallback);
      case acceptCallback_Sig : // 콜백 관련 태그 생성 및 추가 (서버 측)
        insertTag(acceptCallback);
    }
  }
}

```

3) 태그 해석 및 번역 코드 생성 단계

이전 단계에서 추가된 번역 태그에 대한 분석을 통해 새로운 번역 코드들을 생성하는 단계이다. 번역 태그에는 이미 필요한 번역 정보 및 이를 이용하여 번역 코드를 생성하는 메소드들이 정의되어 있다. 다음은 제안된 태그들 중 원격 객체의 바인딩에 관련된 태그의 클래스 구조이다.

```

public class tagBindRemoteObj implements
transTAG
{
  classINF cinf; // 클래스 정보
  String RMI_Server; // 등록된 RMI 서버명
  String OWserver; // OrbixWeb 서버명
  Vector codes; //생성된 클래스 코드
  public tagBindRemoteObj(classINF cinf,
    String RMI_Server,
    String ow_server)
  {
    this.cinf = cinf;
    this.RMI_Server = RMI_Server;
    this.OWserver = ow_server;
  }
  ...
  public void translate(int cur_pos) {...} // 번역 코드를 생성
}

```

토큰 단위로 분리된 RMI 객체를 분석하는 과정에서 번역 태그가 발견되면, 태그 내부에 정의된 번역 코드 생성 루틴을 호출함으로써 새로운 번역 코드를 생성한다. 생성된 번역 코드들은 적절한 후처리 단계를 거친 후 소스 코드 형태로 추가된다.

4) CORBA 클래스 생성 단계

지금까지 생성된 번역 정보를 바탕으로 CORBA 클래스를 생성하는 단계이다. 이 단계는 CORBA IDL 생성 부분과 IDL에 등록된 인터페이스에 대한 구현 클래스 생성 부분으로 크게 나눌수 있다.

먼저, IDL 생성 부분에서는 분산 객체들을 IDL에 등록하는 작업 및 분산 객체들 사이에 존재하는 상속 관계를 IDL에 유지하는 처리가 이루어진다. 분산 객체 사이의 상속 관계를 IDL에 등록할 때, 상속 클래스와 피상속 클래스 간의 순서관계도 고려하여 처리해야 한다. 한편, OMG가 제안한 표준 CORBA IDL은 부모-자식 클래스 간의 메소드 재정의의 허용하지 않으므로 상속성이 존재하는 분산 객체들 사이에 재정의의 메소드가 존재하는 경우, 자식 클래스 내부에 재정의의 메소드와 동일한 동작을 하지만 이름이 다른 메소드를 하나 더 추가한다. 추가된 메소드의 이름은 번역 정보 생성시 할당된 클래스 ID와 연관되어 결정되며 IDL에 등록 가능하다.

IDL에 등록된 인터페이스에 대한 구현 클래스 생성 단계에서는 인터페이스 내부에서 정의된 애트리뷰트들에 대한 접근 메소드의 생성 및 번역 단계에서 추가된 애트리뷰트에 대한 처리가 이루어진다. 이 단계에서 생성된 모든 CORBA 클래스들은 특정 디렉토리에 패

키지 형태로 저장된다.

IV. 실험 및 고찰

이 장에서는 제안된 번역 시스템의 동작을 검증하기 위해서 RMI 환경하에서 개발된 하나의 응용 프로그램을 CORBA 환경하에서 동작하도록 자동 번역을 시도하였다. 또한, 번역 시스템에 의해서 생성된 응용 프로그램과 분산 환경의 지원에 의해 직접 구현된 응용 프로그램과의 성능을 3장에서 언급된 분산 객체의 주요 특성 중심으로 비교 및 평가하였다. 실험은 100Mbps 이더넷에 연결되어 있고 Window NT (version 4.0)가 탑재된 pentium MMX(200 MHz) 시스템들 사이에서 실시되었으며, 실험을 위한 분산 프로그래밍 환경으로는 HORB 와 OrbixWeb (version 3.0)을 채택하였다. HORB^[11]는 RMI 메커니즘을 채택한 분산 프로그래밍 환경의 일종이며, OrbixWeb^[12]은 IONA사의 분산 프로그래밍 환경인 Orbix의 자바 버전으로서, CORBA 규약을 지원한다.

1. 실험 모델로서의 화이트 보드 시스템

제안된 객체 번역 시스템은 window NT를 운영체제로 하는 pentium II (266MHz) 시스템에서 Java 언어로 구현되었다. 그림 4는 객체 번역 시스템의 실행 화면이다.

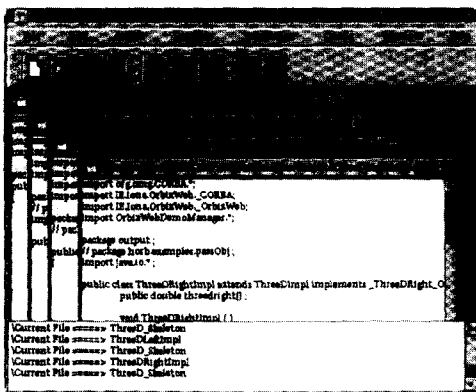


그림 4. 객체 번역 시스템의 실행 화면
Fig. 4. User Interface of Translation System.

화면 중앙 부분의 영역은 번역 단계에서 구축된 번역 정보 및 번역 결과 생성된 클래스들을 보여주기 위한 작업 공간이며, 하단부의 윈도우에는 번역의 진행 상황을 알려주는 다양한 메시지들이 출력된다. 번역

시스템의 주요 기능들은 그림 4의 주 메뉴상의 Database, Translation, Configuration로 구성된다. Database는 번역을 위한 번역 정보 구축(Build) 및 구축된 번역 정보들을 확인(View)하는 기능이 포함된다. Translation은 구축된 번역 정보를 이용하여 실제 번역 작업에 대한 실행(Run) 및 그 결과를 파일로 출력(Save To File)하는 기능을 정의하고 있다. Configuration은 번역 시스템의 환경 설정에 관련된 사용자 인터페이스 기능이다. 환경 설정 파일에 미리 정의된 RMI 소스 디렉토리 명, 패키지명, 원격 객체 바인딩을 위한 서버ID 및 객체ID, 생성될 IDL 파일명 등을 보여주며, 설정값의 변경 및 저장도 가능하다.

제안된 번역 시스템의 정확한 동작을 확인하기 위한 모델로서, 분산 화이트 보드 시스템을 채택하였다. Henry M. Levy^[13]는, 자신의 논문에서, 분산 객체의 기본 특성들을 실험할 수 있는 적당한 모델로서 “화이트보드” 시스템을 제안한 바 있다. 그림 5는 본 시스템에서 번역의 대상으로 채택한 분산 화이트 보드 시스템(WB_DAO)의 구조를 보여주고 있다.

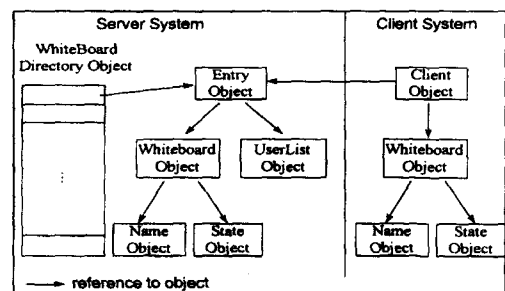


그림 5. WB_DAO 시스템의 구조
Fig. 5. A Structure of WB_DAO.

시스템 자료구조의 모든 단위들은 분산 객체로 구성되었으며, 그림에서 제시된 바와 같이 크게 세가지 모듈로 구성된다. 그림의 왼쪽 부분에 위치한 모듈은 화이트 보드 디렉토리로서 현재 활성화되어 사용중인 모든 화이트 보드의 엔트리들을 유지, 관리하는 서버 역할을 담당한다. 그림의 가운데 위치한 모듈은 활성화된 화이트 보드에 대한 엔트리 객체로서 화이트 보드 객체와 사용자 리스트 객체에 대한 참조를 유지한다. 각각의 화이트 보드 객체들은 화이트 보드의 상태 정보를 유지하는 상태 객체와 이름 정보를 유지하는 이름 객체에 대한 참조를 가진다. 그림의 오른쪽에 위치한 모듈은 화이트 보드의 클라이언트 측 인터페이스를

제공하며, 각 사용자들은 이 클라이언트 객체를 실행시켜 화이트 보드에 접속한다. 클라이언트 객체는 현재 사용중인 화이트 보드 엔트리 객체 및 지역적으로 복사되어 유지되는 화이트보드 객체에 대한 참조를 저장한다. 웹 환경에서 분산 객체로 구현된 WB_DAO 시스템은 인터넷에 연결된 서버 객체와 유기적인 관계를 유지하며 동작한다. 그림 6은 번역 시스템에 의해 생성된 WB_DAO의 클라이언트 측 동작화면이다.

2. 성능 평가 및 분석

WB_DAO 시스템은 사용자의 요구에 실시간적으로 반응하기 위해 현재 공유중인 화이트 보드의 상태 정보에 대한 복사본을 지역적으로 생성한다. 생성된 상태 정보 복사본은 현재 활성화된 그래픽 객체들의 링크드 리스트 구조로서 유지되고 관리된다. 각 클라이언트 측에서는 지역적으로 유지되는 화이트 보드 상태들 사이의 일관성 유지를 위해 화이트 보드의 상태가 변경될 때마다 변화된 상태 정보들을 다른 사용자들에게 통보해야 하며, 이는 링크드 리스트 구조로 유지되고 있는 화이트 보드 상태의 빈번한 전송을 의미한다. 전송된 그래픽 객체들은 분산 객체의 기본 특성에 따라 적절하게 동작한다.

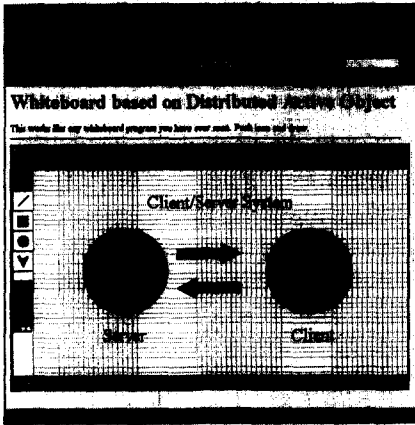


그림 6. 웹 기반 WB_DAO 시스템
Fig. 6. Web-based WB_DAO System.

이 절에서는 번역 시스템에 의해서 생성된 WB_DAO와 분산 환경의 지원에 의해 구현된 WB_DAO의 실행 시간을 비교하는 실험을 행하였다. 실험을 위해서 3장에서 언급된 분산 객체의 주요 특성에 관련된 단위 모듈들을 각각의 방법에 의해 구현한 뒤 실행시간을 측정하였다. 실험 결과는 그림 7, 8, 9와 같다.

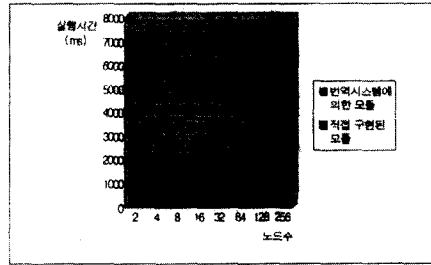


그림 7. 객체의 전달 특성에 의한 비교
Fig. 7. Comparison on the Object Passing by Copy.

그림 7은 객체의 전달 특성을 비교한 실험으로서 전송되는 전 이진 트리의 노드의 개수에 따른 평균 실행 시간을 나타낸다. 그림 8은 분산 객체의 상속성을 비교한 실험으로서 객체의 참조를 원격 시스템으로 전달한 후 부모 객체에 정의된 메소드들을 호출하는 과정을 상속의 깊이를 변경시키면서 측정한 평균 실행 시간을 나타낸다. 그림 9는 다형성이 존재하는 객체의 참조를 원격 시스템으로 전달한 다음 다형성과 연관된 메소드를 일정 횟수 만큼 반복하여 호출하는 과정을 다형성을 가진 인스턴스(polymorphic instance)의 갯수를 변경시키면서 실험한 결과이다. 실험에 의하면 비교된 두 개의 시스템은 거의 대등한 성능을 보여주고 있는데 이는 두 시스템에서 분산 객체를 처리하기 위해 채택한 전략이 동일하기 때문이다.

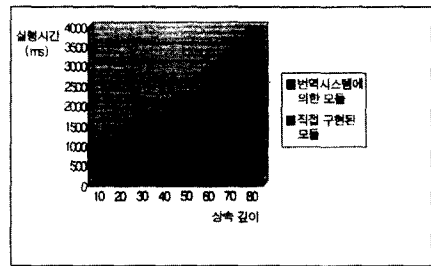


그림 8. 상속성에 의한 비교
Fig. 8. Comparison on the Inheritance.

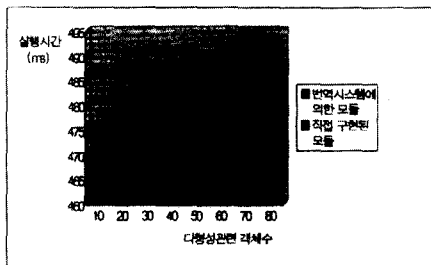


그림 9. 다형성에 의한 비교
Fig. 9. Comparison on the Polymorphism.

V. 결 론

CORBA IDL과 RMI는 대표적인 분산 객체 모델로서 분산 처리를 위한 서로 다른 접근 방식을 제공한다. CORBA 규약은 모든 프로그래밍 환경을 지원하는 범용성을 목표로 하고 있으나 또 다른 분산 프로그래밍 환경인 RMI과는 상호 연관성이 결여되어 있다.

본 논문에서는 RMI와 CORBA 환경에서 개발된 객체사이의 상호 연관성을 지원하기 위한 모델로서 두 환경에서 생성된 객체 사이의 객체 번역 시스템을 구현하였다. 번역 정보 저장을 위한 자료 구조를 계층화된 클래스 구조로서 제안하였고, 분산 객체 사이의 바인딩, 상속성, 다형성, 객체의 전달, 콜백 등 번역에 관련된 중요한 쟁점들을 논하고 이와 관련된 번역 전략을 제시하였다.

구현된 번역 시스템의 번역 전략이 타당한가를 검증하기 위하여 본 논문에서는 분산 화이트 보드 시스템인 WB_DAO를 RMI 환경하에서 개발한 뒤 CORBA 버전으로의 자동 번역을 시도하는 실험을 실시하였다. 또한, 번역 시스템에 의한 프로그램과 CORBA IDL의 지원에 의해 구현된 프로그램을 분산 객체의 상속성, 객체의 전달, 다형성 중심으로 비교 및 분석하였다. 실험 결과, 제안된 번역 시스템에 의한 프로그램이 분산 환경의 지원을 받아 구현된 프로그램과 대등한 성능을 발휘함을 확인하였다.

끝으로, 본 연구를 수행하는 과정에서 많은 도움을 준 세명대학교 분산 처리 연구실 이도학군 이하 대학원생들에게 감사를 드린다.

참 고 문 헌

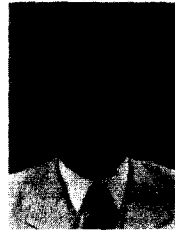
- [1] Andreas Vogel and Keith Duddy, 'Java Programming with CORBA', John Wiley & Sons, 1997.
- [2] John Siegel, 'CORBA Fundamentals and Programming', John Wiley & Sons, 1996.
- [3] Robert Orfali, 'The Essential Distributed Objects Survival Guide', John Wiley & Sons, 1996.
- [4] Robert Orfali and Dan Harkey, 'Client/Server Programming with Java and CORBA', John Wiley & Sons, 1997.
- [5] Ann Wollrath, Jim Waldo, and Roger Riggs, "Java-Centric Distributed Computing", IEEE Micro, vol. 17(2), May, 1997.
- [6] Marc H. Brown and Marc A. Najork, "Distributed Active Objects", Computer Networks and ISDN Systems, vol. 28, pp. 1037-1052, 1996.
- [7] Eric Evans and Daniel Rogers, "Using Java Applets and CORBA for Multiuser Distributed Applications", IEEE Internet Computing, vol. 1(3), May, 1997.
- [8] Sun Microsystems, JDK 1.1 Documentation <URL:http://java.sun.com/>.
- [9] S. Kim, M. Hyun and S. Lee, "Translation Issues for Distributed Active Objects based on RMI and CORBA", proceeding of ICT '98, vol 3, pp. 204-208, June, 1998.
- [10] Trusted consulting, Inc., RMI on CORBA Documentation, <URL:http://www.distributedobjects.com/>, 1997.
- [11] Hirano Satoshi, HORB User's guide <URL:http://www.etl.go.jp/openlab/horb>, March, 1996.
- [12] IONA Technologies, 'OrbixWeb programming guide', IONA Technologies Ltd., 1996.
- [13] Henry M. Levy and Ewan D. Tempero, "Modules, Objects and Distributed Programming: Issues in RPC and Remote Object Invocation", Software-Practice and Experience, vol. 21(1), pp. 77-90, January, 1991.

저 자 소 개



玄 武 庸(正會員)

1993년 경북대학교 컴퓨터 공학과 졸업(공학사). 1995년 경북대학교 대학원 컴퓨터공학과 졸업(공학석사). 1998년 ~ 현재 경북대학교 대학원 컴퓨터 공학과 박사과정. 1995년 ~ 1996년 대원공과대학 전자계산과 전임강사. 1997년 ~ 현재 대원공과대학 전자계산과 조교수. 주관심분야는 네트워크 컴퓨팅, 분산처리, 분산언어, 분산운영체제



金 湜(正會員)

1979년 경북대학교 전자공학과(전산전공) 졸업(공학사). 1979년 ~ 1988년 국방과학연구소 선임연구원. 1990년 Texas A&M 컴퓨터 공학과 졸업(공학석사). 1995년 경북대학교 대학원 컴퓨터 공학과 박사 수료. 1993년 ~ 현재 세명대학교 전산정보학부 교수. 주관심분야는 네트워크 컴퓨팅, 분산처리, 분산언어, 분산운영체제



李 相 潤(正會員)

1993년 경북대학교 컴퓨터 공학과 졸업(공학사). 1995년 경북대학교 대학원 컴퓨터 공학과 졸업(공학석사). 1999년 현재 경북대학교 대학원 컴퓨터 공학과 박사과정. 1997년 ~ 현재 대원공과대학 전산정보처리과 전임강사. 주관심분야는 분산 처리, 분산 언어, 멀티미디어