정보학연구 제5권 제1호 2002년 3월(pp.29-37)
The Journal of Information Technology, Vol.5, No.1

분산 객체의 호환을 위한 객체 번역 시스템의 설계 및 구현

김식*

요 약

분산 프로그래밍은 분산된 통신에 대한 언어 지원에 의해서 크게 단순화될 수 있다. 많은 웹 브라우저는 현재 분산 객체의 많은 형태를 제공하고 있으며 분산 객체의 형태와 개수는 계속 흥미롭고 혁신적인 방법으로 바뀌고 있다. 분산 객체 모델의 전형적인 모델인, CORBA IDL과 Java RMI는 분산처리 환경에 대하여 서로 다른 접근 방법을 제공하고 있다. CORBA의 접근방법은 Java RMI에 의해 생성된 어플리케이션을 제공하지 않지만 다중 프로그래밍 언어를 지원한다. RMI와 CORBA사이의 객체 번역 시스템은 두개의 다른 분산 프로그래밍 환경에서 분산 객체의 정보처리 상호운용을 고려하여 디자인되어 구현되었다. 이 제안된 시스템은 분산 객체에서의 중요한 속성인 바인딩, 상속성, 다형성, 객체 패싱과 콜백을 고려하였다. 우리가 제안한 번역 시스템은 Windows/NT(version4.0)와 Java Development Kit(version 1.1.6)을 사용하여 구현되었다.

Ⅰ. 서론

분산 처리 환경에서 프로그래머들은 원격 시스템 사이의 통신을 위한 수단으로서 흔히 소켓(socket)을 사용해 왔다. 소켓은 TCP/IP와 같은 네트웍 프로토콜의 입/출력 인터페이스로서의 역할을 담당한다. 그러나, 소켓을 이용한 프로그래밍은 통신 프로토콜에 대한 충분한 이해 및 통신 상에서 발생하는 수많은 오버헤드에 대한 직접적인 관리를 요구하므로 프로그래머들에게 상당한 부담이 되어왔다. 따라서, 이를 극복하기 위한 분산 프로그래밍 언어에 대한 많은 연구가 진행 중에 있다[1,2,3,4,5,6].

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

CORBA(Common Object Request Broker Architecture)[2]와 자바 RMI(Remote Method Invocation)[7]는 대표적인 분산 객체 모델로써 분산 처리를 위한 서로 다른 접근 방법을 제공한다. CORBA는 원격 시스템에 존재하는 객체의 메쏘드를 마치 로컬 객체의 메쏘드 처럼 호출하게 하는 하부 구조를 제공한다. IDL (Interface Definition Language)를 제안함으로서 분산 객체 접근을 위한 인터페이스의 정의 단계와 구현 단계를 분리하였고 그 결과 이미 개발

* 세명대학교 정보통신학과 교수

된 컴포넌트나 다른 언어와의 통합(mapping)이 용이해 짐으로써 대규모 응용 프로그램의 구축 시 적합한 접근 방식이다.

자바 RMI 메커니즘은 원격 객체에 정의된 메쏘드를 호출하는 다른 접근 방법을 제공한다. RMI의 기본 아이디어는 객체를 생성할 때 내부에 정의된 메쏘드를 다른 자바 가상 머신에서 호출할 수 있도록 허용하는 것으로서, 이러한 접근 방법은 자바 객체를 위한 원격 함수 호출(RPC) 메커니즘을 제공한다. RMI 메카니즘은 자바로 구현된 소규모 응용 프로그램의 개발에 적합한 방식이다.

CORBA는 기존의 컴포넌트와의 통합을 지원하는 시스템 통합 기술이지만, RMI와의 통합은 불가능하다. 따라서, CORBA와 자바 RMI 환경 하에서 개발된 분산 객체간의 상호 운영성(interoperability)에 대한 필요성이 제기되고 있으며, 이를 지원하기 위한 프로토타입(prototype)들이 제안되고 있다[8,9]. 본 논문에서는 접근 방식이 다르고 상호 호환성이 결여된 주요한 두 프로그래밍 환경인 RMI와 CORBA에서 생성된 객체사이의 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.

이 논문의 구성은 서론에 이어 2장에서는 번역 시스템 설계시 고려해야 분산 객체의 기본 특성에 대해 알아보고, 3장에서는 제안된 시스템의 구성에 대해서 설명하기로 한다. 4장에서는 제안된 방법으로 구현된 번역 시스템을 실행 화면 중심으로 살펴보고, 5장에서 향후 연구 과제를 제시하고 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, 네트웍 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 추

출하기 위한 코드를 첨가하여 다형성을 간접적으로 지원해야 한다.

RMI는 원격 메쏘드 호출시 파라메터나 반환 값으로서 객체가 사용될 경우 값에 의한 전달방법을 지원하지만, CORBA는 참조에 의한 전달방법만 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 메쏘드를 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.

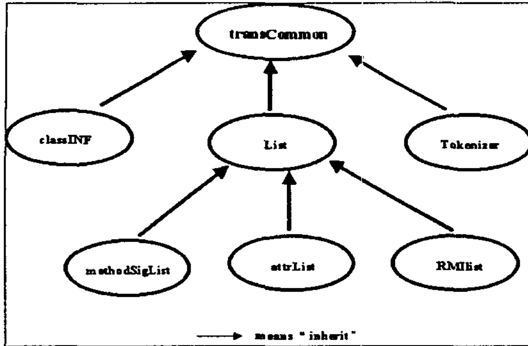
Ⅲ. 객체 번역 시스템의 설계

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.

3.1 번역을 위한 자료구조

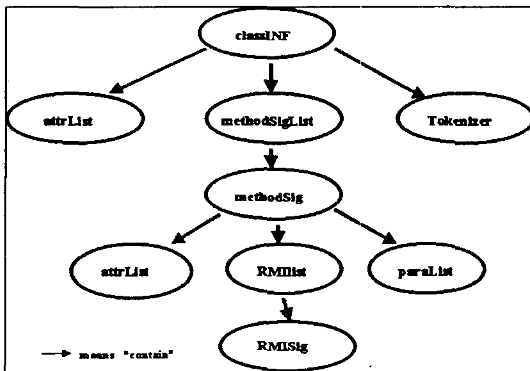
RMI 메카니즘에 의해 생성된 객체를 CORBA 객체로 번역하기 위해서는 해당 클래스에 대한 다양한 정보, 즉, 클래스 이름, 정의된 메쏘드 명세, 애트리 뷰트 명세, 상속 관계 여부, 원격 메쏘드 호출 관련 정보 등의 사항들이 필수적으로 요구된다. 이를 위해, 제안된 시스템에서는 번역의 첫 단계인 번역 정보 구축 단계에서 번역 대상 클래스에 대한 소스 차원의 부분 파싱을 통해 필요한 정보를 수집, 계층화된 클래스 구조로 유지한 뒤 이를 번역 클래스 생성 단계에서 이용하도록 하였다. 본 시스템에서 제안한 클래스 및 그 상속관계 계층도는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

TransCommon은 각 클래스에서 공통적으로 사용되는 상수값 및 메쏘드 들을 정의한다. methodSig는 메쏘드에 관한 정보를 정의하며 메쏘드 이름, 반환 데이터의 형, 전달되는 파라메터, 지역 변수, 원격 메쏘드 호출 등에 관한 상세한 정보들이 저장된다. methodSigList는 클래스 내에 정의된 모든 methodSig에 대한 목록을 저장하며, 이에 대한 효율적인 접근 및 관리를 위한 다양한 메쏘드들을 포함한다. attrList는 특정 클래스 내에 정의된 클래스 애트리뷰트에 대한 정보, 즉 애트리뷰트의 이름, 형 등이 저장되며, paraList는 메쏘드로 전달된 파라메터에 관한 정보들을 유지하며 methodSig 내부에서 인스턴스화 되어 사용된다. RMISig는 클래스 내부에 존재하는 RMI signature 정보를 정의한다. 원격 객체의 특정 메쏘드를 호출하는 시점에서 나타나는 원격 객체의 이름, 메쏘드 명, 전달되는 파라메터 정보 등이 저장된다. RMIList는 클래스 내부에 존재하는 모든 RMISig에 대한 목록을 유지, 관리한다.



[Fig.1] The Inheritance Hierarchy of the Classes for Translation System.

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



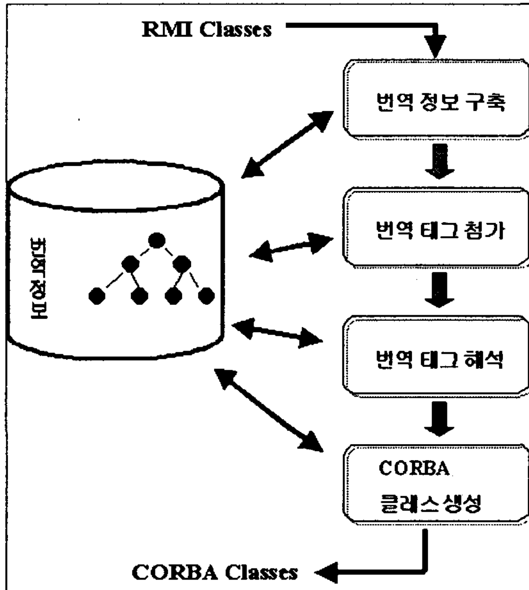
[Fig. 2] The Containment Hierarchy for the Classes for Translation System.

3.2 번역 시스템의 구성

본 논문에서 제안한 객체 번역 시스템은 크게 4부분으로 구성된다. Fig. 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 Fig. 1, 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 형태로서 첨가된다. 마지막 단계인 CORBA 클래스 생성 단계에서는 IDL 및 번역이 완료된 최종 클래스의 생성이 이루어 진다.

3.2.1 번역 정보 구축 단계

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



[Fig. 3] A structure of translation system.

```
Algorithm createClassINF
{
 classINF classinfo = new ClassINF();
 //split source file to tokens
 createTokens(fileName);
 while(hasMoreTokens())
 {
  tokens = readLine(); // get tokens of a line
  switch(getSigType(tokens))
  {
   case Attr_Signature : classinfo.addAttrList
     (tokens);
   case Method_Signature : classinfo.add
     MethodList(tokens);
   case RMI_Signature : classinfo.addRMIList
     (tokens);
  }
 }
}
```

3.2.2 번역 태그 첨가 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 첨가한다. 번역 태그는 번역 정보와 관련 메쏘드를 정의한 클레스를 의미한다. 첨가된 번역 태그에는 번역 코드 생성 단계에서 사용될 생성 정보가 저장되며, 이 정보들은 태그마다 고유한 의미를 지닌다. 태그 첨가 과정을 알고리즘으로 간략화하면 다음과 같다.

```
Algorithm insertTransTag
{
 Vector tokens = getTokenVector();
 while(hasMoreLines)
 {
  ltokens = tokens.readLN();
  switch(getSigType(ltokens))
  {
   case BindRemoteObj_Sig :
        insertTag(tagBindRemoteObj);
   case PolymorphicObj_Sig :
        insertTag(tagPolymorphicObj);
   case RemoteMethodCall_Sig :
        if (ObjectPassByCopy_Sig)
           insertTag(tagSndObjByCopy);
        else if (ObjectPassByReference_Sig)
           insertTag(tagSndObjByRef);
   case inviteCallback_Sig :
        insertTag(tagInviteCallback);
   case acceptCallback_Sig :
        insertTag(acceptCallback);
  }
 }
}
```

3.2.3 태그 해석 및 번역 코드 생성 단계

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

```
public class tagBindRemoteObj implements
  transTAG
{
 classINF cinf; // class information
 String RMI_Server; //registered RMI server
 String OWserver; // OrbixWeb server
 Vector codes=null;
 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) {...} //
   generate translated codes
}
```

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

3.2.4 CORBA 클레스 생성 단계

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

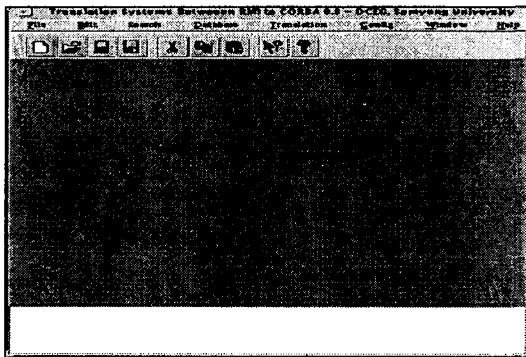
먼저, IDL 생성 부분에서는 분산 객체들을 IDL에 등록하는 작업 및 분산 객체들 사이에 존재하는 상속 관계를 IDL에 유지하는 처리가 이루어진다. 분산 객체 사이의 상속 관계를 IDL에 등록할 때, 상속 클레스와 피상속 클레스 간의 순서관계도 고려하여 처리해야 한다. 한편, OMG가 제안한 표준 CORBA IDL은 부모-자식 클레스 간의 메쏘드 중복을 허용하지 않으므로 상속성이 존재하는 분산 객체들 사이에 중복 메쏘드가 존재하는 경우, 자식 클레스 내부에 중복 메쏘드와 동일한 동작을 하지만 이름이 다른 메쏘드를 하나 더 추가한다. 추가된 메쏘드의 이름은 번역 정보 생성시 할당된 클레스 ID와 연관되어 결정되며 IDL에 등록 가능하다.

IDL에 등록된 인터페이스에 대한 구현 클레스 생성 단계에서는 인터페이스 내부에서 정의된 애트리뷰트들에 대한 접근 메쏘드의 생성 및 번역 단계에서 추가된 애트리 뷰트에 대한 처리가 이루어진다. 이 단계에서 생성된 모든 CORBA 클레스들은 특정 디렉토리에 패키지 형태로 저장된다.

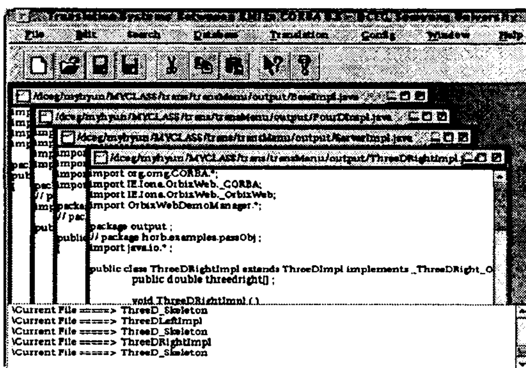
Ⅳ. 구현

제안된 객체 번역 시스템은 window NT를 운영체제로 하는 pentium II (266MHz) 시스템에서 Java 언어로 구현되었다. Fig. 4는 객체 번

역 시스템의 실행 초기 화면이다. 화면 중앙 부분의 영역은 번역 단계에서 구축된 번역 정보 및 번역 결과 생성된 클레스들을 보여주기 위한 작업 공간이며, 하단부의 윈도우에는 번역의 진행 상황을 알려주는 다양한 메시지들이 출력된다. 번역 시스템의 주요 기능들은 Fig. 4의 주 메뉴상의 Database, Translation, Configuration로 구성된다. Database는 번역을 위한 번역 정보 구축(Build) 및 구축된 번역 정보들을 확인(View)하는 기능이 포함된다. Translation는 구축된 번역 정보를 이용하여 실제 번역 작업에



〔 Fig. 4 〕 User interface of translation system.



〔 Fig. 5 〕 Excution Results of Translation System.

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. Fig. 5는 번역 결과 생성된 클레스들을 확인하는 화면이다.

V. 결론

CORBA IDL과 자바 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 자바 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 구*현하였다. 번역 정보 저장을 위한 자료 구조를 계층화된 클레스 구조로서 제안하였고, 분산 객체 사이의 바인딩, 상속성, 다형성, 객체의 전달, 콜백 등 번역에 관련된 중요한 쟁점들을 논하고 이와 관련된 번역 전략을 제시하였다.

향후 연구 과제로서 제안된 시스템으로 번역된 분산 객체의 유용성을 검증하기 위한 일련의 실험이 요구된다. 본 연구팀은 번역 시스템에 의해 자동 생성된 시스템과 분산 환경의 지원을 받아 직접 구현된 시스템과의 실행 시간 비교를 통해 제안된 시스템의 성능을 검증하는 실험을

계속하고 있다.

참고문헌

[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] Hirano Satoshi, HORB User's guide <URL:http://www.etl.go.jp/openlab/horb>, March, 1996.

[6] IONA Technologies, "OrbixWeb programming guide", IONA Technologies Ltd., 1996.

[7] Sun Microsystems, JDK 1.1 Documentation <URL:http://java.sun.com/>.

[8] S. Kim, M. Hyun and S. Lee, "Translation Issues for Distributed Active Objects based on RMI and CORBA", proceeding of ICT '98, Vol3, pp.204-208, June, 1998.

[9] Trusted consulting, Inc., RMI on CORBA Documentation, <URL:http://www.DistributedObjects.com/>.

On Design and Implementation of Distributed Objects Translation System for Inter-Operability

Shik Kim*

Abstract

Distributed programming can be greatly simplified by language support for distributed communication. Many web browsers now offer some form of distributed objects and the number and types of them are growing daily in interesting and innovative ways CORBA IDL and Java RMI, the representative of distributed object model, support different approaches for distributed computing environments. CORBA approach does not support the application generated by java 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. Translation System we suggested is implemented on the Window/NT(version 4.0) with Java Development Kit(version 1.1.6).

* Dept. of Information Communication, Semyung University