2000 년 한국정보처리학회 춘계 학술발표논문집 제 6 권 제 2 호

분산환경에서 클래스 정의 언어를 이용한 병행객체 WRAPPER 의 구현

이창현*, 박윤용**, 이경오**, 임동선***, 정부금***
*선문대학교 정보통신공학과
**선문대학교 컴퓨터정보학부
***전자통신연구원 실시간 OS 팀
e-mail : wolf74@chollian.net

Implementation of Active Object use CDL in Distributed System

Chang-Hyun Lee*, Youn-Yong Park**, Kyung-Oh Lee**
Dong-Sun Lim***, Bu-Geum Jung***
*Dept. Information & Communication, Sun-Moon University
**Dept. Divion of Information & Computer Science, Sun-Moon University
***Real-Time OS Team, ETRI

요 약

본 논문에서는 다양한 분산객체 환경에서 CORBA(Common Object Request Broker Architecture)의 IDL(Interface Definition Language)과 유사한 기능을 수행하는 클래스정의언어(CDL : Class Definition Language)와 병행객체 WRAPPER 에 관하여 설명하였다. 병행객체는 기존의 객체를 독립적으로 CPU 의 스케줄을 받아 병행적으로 실행하게 하는 객체이다. CDL 은 병행객체를 생성하는 클래스를 정의하는 언어이다. 병행객체 WRAPPER 는 CDL 의 형태로 기술한 병행클래스에 대응하는 실행 가능한 코드를 생성한다. 본 논문에서는 CDL 과 병행객체 WRAPPER 를 이용하여 병행객체의 멤버함수 호출을 일반객체의 멤버함수 호출방식과 같게 하고, 병행객체의 생성/소멸 등의 사용에 투명성이 보장되도록 하였다.

1. 서론

병행객체란 객체지향 언어에서 사용되는 객체로서 UNIX 의 스레드(thread)와 같이 독립적으로 CPU 의 스케줄을 받아 병행적으로 실행되는 단위를 의미한다.

병행객체의 구현을 위해 사용할 수 있는 언어로는 전자통신연구원의 OO-CHILL 이 있으며, 이러한 객체를 보다 쉽게 구현하여 사용자가 클라이언트 또는 서버 프로그램을 작성하 는데 도움이 되도록 하기 위해 CDL 을 이용한 병행객 체 WRAPPER 를 구현해 보았다. CDL 이란 CORBA 기반의 응용프로그램들의 개발을 위해 지원되는 IDL 과 같은 역할을 하는 것으로, 병행클래스를 정의한 후, 병행객체 WRAPPER 를 사용하여 구현된 객체의 코드를 얻고, 사용자가 객체간의 송수신 방법이나 구조를 몰라도 사용 가능하게 하여 병행객체에 대한 투명 성을 제공한다.

CDL 은 멤버데이터와 멤버함수를 포함하고 있는 병행클래스를 정의하기위해 이용하는 언어로써 매우 간결하고 사용하기 쉽다. 병행객체 WRAPPER 는 CDL 을 이용하여 정의한 병행클래스를 실행 가능한 코드로 매핑하여, 사용자는 멤버함수만을 구현하게 한다. 병행객체간의 멤버함수 호출시에는 간단한 표기법으로 기술하면 병행객체 WRAPPER 가 이를 실제 실행코드로 변환하게 하여 사용자가 프로그램 작성에 어려움을 느끼지 않도록 하게 하였다.

2 장에서는 병행객체와 관련된 RPC 의 rpcgen 과 CORBA 의 IDL 에 대한 조사를 하였고, 3 장에서는 병행객체가 실행되는 환경에 대해 기술하였다. 4 장에서는 CDL 의 사용법과 병행객체 WRAPPER 를 이용한 병행객체 코드의 생성과정과 멤버함수의 호출과정에 대해 기술하였다.

2. 관련 연구

원격 프로시저 호출(Remote Procedure Call)은 기존의 지역 프로시저를 호출하는 형식으로 원격 프로세스간의 통신이 이루어진다. RPC 는 프로그래머들이 저수준 네트워크 구조의 상세한 부분을 알지 못해도 쉽게 강력한 네트워크 응용프로그램을 작성할 수 있는 투명성을 제공한다. 이러한 방식으로는 Sun RPC, Xerox Couier, Apollo RPC 가 있다. 이중에서 Sun RPC 는 rpcgen 을 이용하여 정의파일로부터 서버(server)측의 스터브 (stub) 파일과 클라이언트(client)측의 스터브 파일을 생성한다. RPC 의 정의파일은 병행클래스정의 파일에 해당하며, rpcgen 은 병행객체 WRAPPER 에 해당한다.

분산객체 환경에서 애플리케이션을 개발하기 위해 CORBA 기반의 애플리케이션 개발의 위해 먼저 IDL을 사용하여 객체의 인터페이스(interface)를 작성하게 된다. 이 IDL은 프로그래밍 언어가 아니기 때문에 실제 객체가 구현된 것이 아니라 인터페이스만을 정의한 것이다. 실제 구현을 위해서는 C++, JAVA와 같은 언어로 구현된 객체가 필요하다. 이를 위해 IDL 컴파일러를 사용하여 IDL파일을 컴파일하여 C++, 또는 JAVA 언어로 매핑된 파일을 생성하게 된다. 이 매핑된 파일들을 사용하여 분산 객체들을 구현하게 된다.

IDL 인터페이스는 속성(Attribute)과 연산(Operation)을 정의하고, 각 연산의 매개변수들을 정의한다. 즉 객체들간의 인터페이스를 정의하는 특징을 갖는다. 인터페이스에서 정의된 속성은 구현객체에서 사용되어질 수 있으며, 분산 환경에서 클라이언트가 이 인터페이스에 접근하여 변수의 값을 읽고 쓸 수 있다. IDL 연산은 객체의 기능에 액세스하기 위해 사용하는 함수의 형식을 지원한다. IDL 연산은 IDL 데이터 타입의 매개변수와 반환값을 가질 수 있다.

IDL에서 지원하는 데이터 타입에는 integer(short, unsigned short, long, unsigned long, long long, unsigned long long), float, double,char, string, any 형등의 기본 타입과, Enum ,Union, Struct의 구조체 타입, 배열등 다양한 데이터 타입을 지원한다.

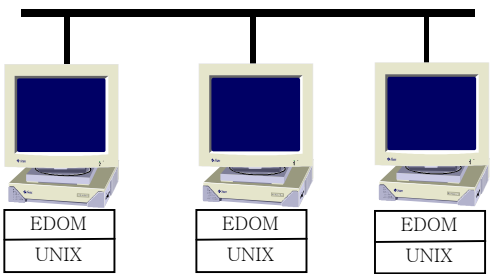
본 논문에서 제안하는 클래스정의언어 CDL(Class Definition Language)도 역시 IDL과 유사한 방식으로 구현되었다. 사용자는 클래스정의언어에 따라 병행클래스를 정의하고, 병행객체 wrapper를 이용하면 C언어로 작성된 코드를 얻을 수 있다. 사용자는 이 때 생성된 코드 중에 멤버함수의 기능을 기술하여 병행객체에 대한 구현을 완성할 수 있다.

CDL 역시 멤버데이터와 멤버함수를 정의하 고, 각 멤버함수의 매개변수들을 정의한다. CDL 역시 IDL과 마찬가지로 각 객체들의 클래스를 정의한다. IDL의 속성과 병행클래스의 멤버데이터, IDL의 연산과 CDL의 멤버함수는 같은 역할을 수행한다.

IDL 인터페이스의 속성과 CDL 클래스의 멤버데이터는 구현객체의 변수에 대응한다. 인터페이스에서 정의된 속성은 구현객체에서 사용되어질 수 있으며, 분산 환경에서 클라이언트가 이 인터페이스에 접근하여 변수의 값을 읽고 쓸 수 있다. CDL에서는 integer, float, char형의 배열을 지원한다. CDL은 현재 개발 진행중으로 앞으로 보다 많은 데이터타입을 지원할 계획이다.

3. 병행객체 실행환경

CDL을 이용하여 기술한 병행클래스를 WRAPPER를 이용하여 병행객체 코드를 생성하고 병행객체 사이의 멤버함수 호출을 실행하기 위한 코드를 생성해준다. 병행객체를 사용하기 위해서는 유닉스 기반의 시스템에 EDOM(Etri Distributed Object Manager)을 설치하여야 한다. EDOM은 아래의 그림과 같은 분산환경의 UNIX 시스템에서 미들웨어 방식으로 구현되어, 커널의 종류에 무관하게 병행객체들을 관리해주는 병행객체 실행 시스템이다.



(그림. 1) 병행객체 실행을 위한 환경

EDOM 에서 제공하는 주요 기능은 다음과 같다.

- actobj_classdef() : 병행클래스 생성 프리미티브

- actobj_start() : 병행객체 생성 프리미티브
- actobj_exit() : 병행객체 종료 프리미티브
- actobj_send() : 병행객체의 멤버함수 호출 프리미티브
  proxy_SEND() : * 원격 능동객체의 메소드호출을 처리하는 송신측의 서브루틴
- actobj_rcv() : 병행객체의 멤버함수 수신 프리미티브
- actobj_reply(): 수신병행객체가 메소드실행 후 복귀값을 전하기 위해 사용하는 프리미티브
- actobj_dbind() : 생성된 능동객체의 멤버데이터를 저장하기 위한 포인터를 할당
- actobj_edom_init() : EOM 내의 자료구조들을 초기화하는 프리미티브
- edom_malloc() : EDOM 내에 할당된 히프 메모리에서 주어진 크기만큼의 포인터를 복귀하는 프리미티브
- edom_calloc() : EDOM 내에 할당된 히프 메모리에서 주어진 크기만큼의 포인터를 0 으로초기화하여서 복귀하는 프리미티브

병행객체의 멤버함수 호출은 일반객체의 멤버함수호출과 동일한 구조로 사용할 수 있도록 함으로써 사용자의 편의성을 보장하였고, 지역멤버함수호출과 원격멤버함수 호출인 경우에도 동일한 구조로 프로그램들이 프로그램 할 수 있게 함으로써 사용자 들에게 완벽한 지역투명성(location transparency)를 제공하도록 구현하였다. 또한, 복귀값이 있는 멤버 함수의 호출인 경우에도 사용자들은 일반 멤버함수 호출과 동일한 방식으로 사용하도록 함으로써 EDOM 을 사용하여 프로그램을 개발하는 소프트웨어의 모듈화(modular)를 가능하게 하였다.

병행객체의 멤버함수호출은 일반객체의 멤버함수 호출과는 다르게 스택을 이용하여 함수의 실 인수 값을 전달하지 못한다. 따라서 병행객체의 멤버함수에 사용되는 실 인수의 값들을 문자열 형태의 값으로 변환하여 메시지형으로 전달하여야 한다. 이 과정을 마샬링(marshalling)이라고 하고, 문자열형태의 값으로 전달된 메시지를 멤버함수의 실 인수로 풀어주는 기능을 언마샬링(unmarshalling) 이라고 한다. 이러한 작업을 wrapper를 이용하여 실행하도록 하여 사용자에 대한 편의성을 보장한다.

4. 병행객체 WRAPPER 의 구현

병행객체 WRAPPER 는 입력받은 병행클래스정의 파일을 병행객체 WRAPPER 를 이용하여 병행객체 코드를 생성한다. 병행객체의 구현(implementation)은 사용자가 프로토타입(prototype)으로 정의된 멤버함수를 작성하면 된다.

4.1 병행클래스 정의 언어(CDL)

사용자가 병행객체를 구현하기 위해서는 먼저 병행클래스를 정의하여야 한다. 병행클래스를 정의 하기 위해서는 방법으로는 CDL 을 이용하는 방법이 있다. CDL 은 병행클래스를 정의하기 위해 만든 언어로서 아래와 같은 형식을 취하고 있다. 병행클래스 정의파일의 이름은 클래스 이름과 같아야 하고, 파일의 확장자는 ".a" 를 사용있다.

클래스 정의 언어는 다음과 같은 규칙으로 작성해야 한다. 클래스 정의를 위해서는 먼저 클래스를 선언해야 하고, 선언된 클래스의 멤버데이터를 기술하고, 멤버함수를 선언한다.

```
ACTCLASS  class-name {
    DATA:
        /* 멤버 데이터 선언*/
      type   VAR_NAME1;
             type   VAR_NAME2;
    FUNCTION:
        /* 함수 선언          */
        function_name1(arg1, arg2);
        function_name2(arg3, arg4,...);
} number;
```

(그림. 2) CDL 의 일반적인 형식

클래스 선언은 "ACTCLASS" 라는 예약어 다음에 클래스 이름을 쓴다.

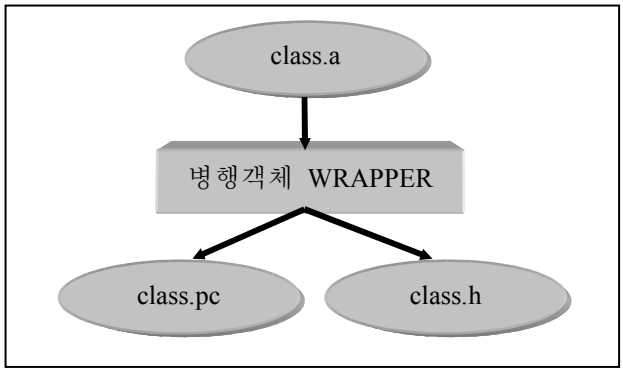
멤버데이터 선언은 대문자 "DATA:" 라는 예약어로 시작을 한다. 변수 이름은 대문자로 쓰고, 한 라인에 한 변수씩만 선언한다.

멤버함수 선언은 대문자 "FUNCTION:" 라는 예약어로 시작한다. 현재 복귀(return)값은 정수, 실수, 문자열을 사용할 수 있다.

4.2 병행객체 코드 생성

병행클래스 정의 파일인 *class-name*.a 를 정의하고 병행객체 wrapper 를 이용하면 아래의 그림과 같이 병행클래스 코드파일인 *class-name*.pc 와 병행클래스 헤더파일인 *class-name*.h 가 생성된다.

(그림. 3) 병행객체 WRAPPER 의 실행 과정 A



4.3 멤버함수 호출 과정

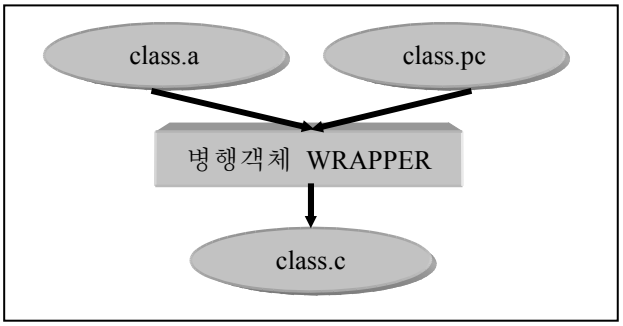
EDOM에서의 멤버함수 호출 방법은 병행객체의 이름을 사용하여 호출하도록 설계하였다. 특히, CORBA 형태의 닷(dot) 표기법을 도입하여 CORBA의 표준사용법과 통일할 수 있도록 구현하였다. C 또는 C++언어에서는 닷 표기법의 멤버함수 호출 문법(syntax)을 제공하고 있지 않기 때문에 선행처리기 (preprocessor)를 구현하여 닷 표기법을 구현하였다.

EDOM에서 멤버함수 호출을 위해 사용되는 닷 표기법은 병행객체가 존재하는 호스트이름과 병행클래스이름 그리고 병행객체 이름과 멤버함수 이름 사이에 닷(.)을 사용하여 기술함으로써 멤버함수를 호출하게 하였다. 이를 위해 사용자들은 아래와 같이

#call *host-name.class-name.object-name.function-name(argument list);*

#call 이라는 구별자를 먼저 기술하여 병행객체 WRAPPER 에게 멤버함수 호출과 관련된 코드를 생성하도록 하였다. 병행객체 WRAPPER 는 정의된 병행클래스와 그림 에서 생성된 .pc 파일을 분석하여 #call 구문을 아래와 같이 변환시켜주는 역할을 한다.

actobj_send("***host-name***", "***class-name***", "***object-name***", ***function-name*** , "value of argument list");



(그림. 4) 병행객체 WRAPPER 의 실행 과정 B

병행객체 WRAPPER 에 의해 생성된 코드들과 사용자가 작성한 코드들을 링크하여 컴파일하면 병행객체를 실행시킬 수 있는 실행파일을 얻는다.

4.4 원격멤버함수 호출

EDOM 은 지역(local) 멤버함수호출과 원격(remote) 멤버함수호출에 관해서 사용자에게 완전한 투명성(transparency)를 제공하고 있다. 따라서 위의 4.3 절에서 기술한 방식대로 코드를 기술하면 된다.

5. 결론

본 논문에서는, 클래스 정의 언어(CDL)를 이용하여 병행클래스를 정의하고, 분산객체 환경에서 사용되는 병행객체는 병행클래스를 바탕으로 구현할 수 있는 병행객체 WRAPPER 를 이용하여 구현되도록 하였다. 또한, 병행객체 간의 메시지 송/수신에 대한 부분도 닷(.) 표기법을 이용하여 호출할 수 있도록 하였다. 송/수신시의 실 인수에 대한 표기는 서브루틴에서의 실 인수 표기법과 동일하게 하여, 분산환경에서 병행객체를 사용하는 프로그램을 작성할 때에 병행객체 WRAPPER 를 이용함으로써 편리하게 투명성이 제공되는 환경에서 병행객체를 사용할 수 있다.

참고문헌

[1] SunSoft, SunOStm, "RPC: Remote Procedure Call":, Request For Comments: 1057, Sun Microsystems, Inc, 1988.

[2] IONA Technologies PLC, "OrbixWeb Programmers' Guide", pp 75 ~ 140, 1997

[3] 왕창종, 이세훈, "Inside CORBA3 프로그래밍", 대림출판사, pp 145 ~ 203, 1999

[4] W.Richard Stevens, "Unix Network programming" International Edition, Prentice Hall International, Inc. pp 692 ~ 719, 1994

[5] 박재현, 코아 코바, 영한출판사, 1998

[6] 원유헌, 백정현, 'UNIX System Programming", 정익사, 1995