메소드/시그널 매핑을 이용한 SDL과 CORBA 시스템의 통합 방법

백 의 현[†] · 허 재 두[††] · 이 형 호[†††]

요 약

본 논문은 인터넷으로 연결된 내장형 시스템을 효율적으로 개발하기 위하여 메소드 호출과 시그널을 매핑하는 방법을 이용하여 SDL과 CORBA 시스템을 통합하는 모델을 제시하였다. SDL과 CORBA를 기반으로 하는 두개의 시스템이 통신하기 위해서는 SDL 시스템의 통신 프로토콜과 CORBA 통신 프로토콜간의 변환 인터페이스가 필수적으로 요구된다. 본 논문에서는 두 시스템 간의 통신 인터페이스 및 통신 프로토콜의 변환을 위하여 IDL을 사용하였으며 IDL 컴파일러에서 프로토콜 연동을 위한 인터페이스를 자동 생성하도록 하였다. 본 모델을 이용할 경우 SDL로 작성된 서버 시스템의 하부에 미들웨어를 채용할 수 있어 서버 시스템의 실행환경 및 위치에 관계 없이 분산 시스템 상에서 서비스가 가능해진다.

Integration Mechanism of SDL and CORBA System using Method/Signal Mapping Rules

EuiHyun Paik[†] · Jae Doo Huh[††] · Hyeong Ho Lee[†††]

ABSTRACT

This paper presents the model that integrates an SDL system and a CORBA system using mapping rules between method and signal for developing embedded systems connected with internet. In order to support communication between the two different systems (SDL and CORBA), it is essential to secure the conversion interface between SDL communication protocol and CORBA communication protocol. In this paper, IDL is adopted for the communication interface and the conversion of the communication protocol between the two systems, and the IDL compiler automatically generates the interface for protocol interoperability. The proposed model adopts middleware on the subpart of the SDL based legacy system, and hence, supports the service on the distributed system, regardless of the environment and location of the server system.

키워드 : 에스디엘(SDL), 코바(CORBA), 분산시스템 통합(Distributed system Integration)

1. 서 론

CORBA(Common Object Request Broker Architecture)는 이기종 분산 시스템 환경에서 응용 소프트웨어 개발에 편리한 기능들을 제공하므로 분산 응용 프로그램 개발에 널리 사용되고 있다. 그러나 현재까지는 일반 범용 컴퓨터를 플랫폼으로 하여 C, C++, Java 등과 같은 구현 언어를 이용하여 응용 소프트웨어를 개발하고 통신이 필요한 부분에 CORBA를 적용하는 수준이다.

소프트웨어의 개발 방법이 구현 중심으로부터 명세와 설계 중심으로 변화함에 따라 분석과 실행이 가능한 명세와 설계를 통하여 프로그램 코드의 구현없이 개발하는 방법이 다양하게 연구되고 있으며[1-4], 이에 따라 CORBA가 제공하는 분산 시스템간 효율적인 메시지 통신방법을 시스템 설계 단계에서 수용하기 위한 CORBA와 SDL을 통합 노력이 여러 방면에서 진행중이다[5, 6].

CORBA와 SDL을 통합하기 위한 접근 방법은 일반적으로 SDL 지향적 접근 방법과 CORBA 지향적 접근 방법이 있다. SDL 지향적 접근 방법은 먼저 SDL시스템을 정의하고 정의된 SDL 다이어그램으로부터 응용 프로그램과 통신을 위한 CORBA IDL을 같이 생성한다. 이 방법은 SDL로 작성된 응용 시스템들의 하부 구조는 CORBA의 지원을 받아 실행되지만 개념적으로는 SDL로 통합되므로 시스템 구성시 CORBA를 고려할 필요가 없다. CORBA 지향적 접근 방법에서 SDL은 C++, Smalltalk들 처럼 주어진 IDL(Interface Definition Language) 인터페이스의 동작을 정의하기 위한 일종의 구현 언어로서의 기능을 제공한다. 이 방법에서는 시스템간의 인터

† 정 회 원 : 한국전자통신연구원 네트워크연구소 책임연구원
†† 정 회 원 : 한국전자통신연구원 네트워크연구소 네트워크팀장, 책임연구원
††† 종신회원 : 한국전자통신연구원 네트워크연구소 액세스기술 연구부장, 책임연구원
논문접수 : 2002년 9월 28일, 심사완료 : 2002년 12월 11일

페이스를 IDL로 정의하고 IDL 컴파일러를 이용하여 SDL용 스터브/스켈레톤으로 변환한 후 SDL 스터브/스켈레톤에 SDL 시스템과의 행위를 정의하기 위한 각종 정보들을 추가하고 각종 실행 도구(C/C++ 코드 생성기)를 사용하여 실행 가능한 SDL 응용을 생성한다. 이 경우 두 시스템(CORBA와 SDL)은 서로 통신하는 방법이 상이하기 때문에 통합하기 위해서는 통신 프로토콜을 조정하는 인터페이스가 필요하다.

본 논문에서는 다양한 하드웨어 플랫폼상에서 클라이언트 서버 모델로 실행되는 분산 소프트웨어를 상위 설계 및 명세 언어인 SDL(Specification and Description Language)로 개발하고 분산 소프트웨어간 통신 부분은 CORBA를 적용하여 개발할 수 있도록 하기 위하여 CORBA 지향적 접근방법을 이용하여 두 시스템을 통합하였다. SDL과 CORBA 연동을 위한 인터페이스 코드를 자동생성하기 위하여 SDL이 사용하는 모든 외부 시그널에 대응하는 함수를 IDL 파일에 기술하였으며, IDL 컴파일러가 작성된 IDL 화일을 이용하여 SDL 시그널이 CORBA 메소드로 매핑되는 코드를 생성하도록 하였다. 이를 위하여 IDL 컴파일러를 수정하였다.

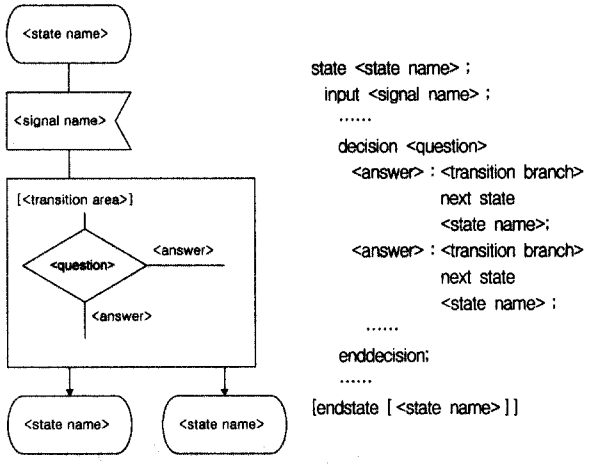
논문에서 제안한 방법을 이용하여 분산 시스템을 개발할 경우 다음과 같은 장점이 있다. 첫째로, 분산 소프트웨어 개발시 상위 명세 및 설계 언어인 SDL를 이용하여 개발할 수 있어 소프트웨어를 설계 단계부터 시험을 해 볼 수 있다. 둘째로, 클라이언트 서버간 통신 부분에 CORBA를 이용할 수 있어 분산 처리를 위한 별도의 프로그래밍이 필요 없으며, 클라이언트는 자신이 요구한 기능를 제공하는 서버의 정보를 알 필요가 없어 서버와 독립적으로 클라이언트를 개발할 수 있다. 따라서 본 방법을 이용할 경우 분산 소프트웨어 개발 비용 감축 및 생산성 향상에 효과가 있다.

2. SDL과 CORBA 개요

2.1 SDL

SDL (Specification and Description Language)은 통신 시스템의 개발 시간과 절감과 시스템의 질적 향상을 위해 ITU-T (International Telecommunication Union Telecommunication Standardization Sector)가 권고한 시스템의 명세(specification)와 기술(description) 언어이다[7]. 이 언어는 유용성이 입증되어 1970년대 중반부터 통신 산업 뿐만이 아니라 교환기 시스템과 같은 실시간 수행을 필요로 하는 모든 영역에서 널리 쓰이고 있다.

(그림 1)과 같이 SDL은 시스템의 명세와 기술을 위해 흐름도 형태의 그래픽 다이어그램으로 전체 시스템을 표현하는 SDL/GR (Graphical Representation)과 프로그래밍 언어의 형태로 SDL/GR에 상응하는 의미를 갖는 SDL/PR로 표현된다. SDL/GR은 시스템의 구조와 행위를 명확하게 보여주고, 제어와 자료의 흐름을 시각화 하므로 유한 상태 기계에 의해 효과적으로 표현되는 통신 시스템의 명세와 설계에 적합하고, SDL/PR은 시스템의 체계적인 문서화나 다른 이기종 CASE 도구 간의 정보 교환에 적합하다.



(그림 1) SDL GR과 PR

따라서, SDL을 통한 시스템의 명세는 명세 및 기술 문서로부터 소프트웨어의 정형적인 검증과 확인을 가능하게 하고, 이를 바탕으로 구현 프로그램의 자동 생성을 가능하게 한다. 또한, 시스템의 행위가 통신을 수행하는 확장된 유한 상태 기계(Communicating Extended Finite State Machine)형태로 기술되므로 프로세스 간의 상호 작용이 중점이 되는 통신 및 실시간 분산 시스템을 설계하고 시험하는데 적합하다.이러한 이유로 실제적인 산업 분야에서는 SDL을 지원하는 도구들이 개발되어 사용되고 있다.

2.2 CORBA

CORBA는 800개 이상의 산업체가 참여하여 구현이 아닌 인터페이스를 정의하는 규격으로 주요 특징으로는 객체 위치 투명성, 연결 및 메모리 관리, 매개 변수 (언)마셜링, 이벤트 및 요구의 (역)다중화, 오류 처리 및 고장 감내, 객체/서버의 활성화, 병행성 및 보안 기능을 제공한다. 따라서 CORBA는 언어, 운영체제, 하드웨어 및 네트워크 프로토콜의 이질적인 의존성으로부터 독립적인 응용의 구현할 수 있는 인터페이스를 제공한다[1, 2]. CORBA 표준에서는 시스템 개발을 위한 다음과 같이 4가지의 인터페이스를 정의하고 있다.

2.2.1 IDL (Interface Description Language)

구현 객체에 대한 인터페이스는 프로그램 언어에 따라서 매핑을 서로 달리하는데, IDL 컴파일러는 객체에 대한 인터페이스를 표현하기 위하여 프로그래밍 언어에 독립적이며, 생성된 코드는 구현 객체에 대한 서비스를 투명하게 제공할 수 있는 수단을 제공한다.

2.2.2 ORB (Object Request Broker)

ORB는 동일 시스템 또는 원격 시스템에 위치한 서버측 메

소드를 클라이언트가 마치 자신의 메소드를 호출하는 것처럼 자연스럽게 호출하여 사용할 수 있는 방법을 제공한다. 클라이언트의 메소드 호출을 수신하면 해당 메소드를 제공하는 서버측 구현 코드를 호출한 후 결과를 클라이언트로 넘겨주는 중개자 역할을 한다. 이때 클라이언트는 서버측 구현 객체의 위치정보 없이도 객체 호출이 가능하다. 이와 같이 ORB는 이질적인 시스템에 구현된 구현 객체에 대한 서비스를 제공할 수 있는 위치 투명성과, 프로그래밍 언어 독립성 및 다중의 구현 시스템을 지원 할 수 있는 상호 운용성을 제공한다.

2.2.3 POA(Portable Object Adaptor)

객체 어댑터는 객체 서비스 중개자와 구현 객체사이의 매개체 역할을 담당하며, 구현 객체의 활성화, 서비스 요구 및 객체에 대한 상태 정보를 유지 하면서 서비스를 제어한다.

2.2.4 IOP(Interoperability ORB Protocol)

일반적으로 ORB간 통신 프로토콜로는 TCP/IP 프로토콜 기반으로 일반적인 상호 연동 규격을 정의한 GIOP(General Inter - ORB Protocol)와 IIOP(Internet Inter - ORB Protocol)을 사용하여 통신하게 되는데, OMG에서는 전달 계층 프로토콜로 특정 프로토콜을 정의하고 있지 않다. GIOP는 통신은 서비스 요구측에서 자신의 하드웨어 구조 정보, 매개변수 값을 포함하는 정보를 마셜링/언마셜링 하기 위한 규격을 정의한다.

3. SDL과 CORBA 시스템 통합 방안

3.1 통합 규칙

SDL 시스템은 외부 프로세스와 시그널로 통신하는 반면 CORBA 기반 시스템은 함수 또는 메소드 호출로 통신하기 때문에 두 시스템은 기본적으로 통신이 불가능하다. 따라서 두 시스템이 서로 통신하기 위해서는 서로 다른 통신 프로토콜을 연동시키는 작업이 필요하다. 본 논문에서는 두 시스템 간에 통신을 위하여 다음과 같은 규칙을 사용하였다.

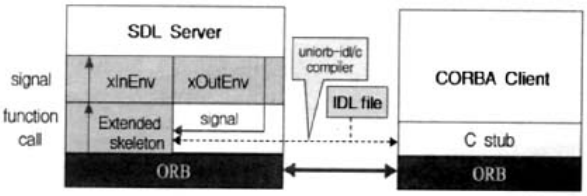
규칙 1 : 클라이언트와 서버 시스템간의 통신매체로 CORBA를 사용한다.

규칙 2 : SDL 다이아그램 내에 선언된 모든 외부 시그널 및 메시지 정보를 이용하여 시그널에 대응하는 함수 및 데이터 구조를 IDL 파일에 기술한다. 이때 SDL 다이아그램 내에서 "외부시그널 수신/실행문 처리[/외부시그널 송신]"등 일련의 실행 쓰래드가 동일한 실행 플로우상에 있는 경우는 두개의 외부 시그널을 하나의 함수로 매핑시킨다.

규칙 3 : 규칙 2에서 외부시그널 송신(리턴 시그널)이 없는 실행 쓰래드와 대응하는 함수는 oneway 함수로 표시하여 리턴 시그널이 있는 경우와 구분한다.

규칙 4 : IDL에서 생성된 스켈레톤 파일에서 메소드/함수 호출로 넘어오는 메시지를 시그널에 실어 SDL 시스템으로 전송하고 응답을 수신하여 ORB로 리턴하도록 IDL 컴파일러를 확장한다. 단 oneway 함수의 경우에는 응답 수신 없이 바로 리턴하도록 한다.

클라이언트 시스템에서 서버 시스템으로 메시지를 송신하기 위하여 IDL에 정의된 함수를 호출하면 IDL 컴파일러에서 생성된 스터브을 통하여 ORB로 전달되고, ORB는 서버 시스템에 링크된 스켈레톤 파일의 해당 함수를 호출한다. 호출된 함수는 SDL 시그널을 하나 생성하고 수신한 파라메터 정보를 SDL 시그널에 실어서 서버 시스템으로 보낸 후 응답 시그널을 기다린다. 응답 시그널이 도착하면 시그널에 실려 있는 응답 메시지를 함수의 리턴 값으로 변환하여 ORB로 넘겨준다. 응답 시그널이 없는 경우는 메시지 전달을 성공하였다는 메시지를 ORB로 리턴한다. (그림 2)는 본 논문에서 제안한 CORBA와 SDL 시스템의 통합 모델 구조도이다.



(그림 2) CORBA와 SDL 시스템 통합 모델 구조도

3.2 SDL과 CORBA 연동 방법

CORBA기반 시스템과 SDL 기반 시스템을 통신을 위하여 SDL 개발도구로부터 혹은 프로그래머에 의해 작성된 인터페이스 정의 파일을 IDL 컴파일러를 이용하여 생성된 스터브 코드를 이용하여 클라이언트 프로그램을 생성한다. 그러나 서버측에 대하여 SDL 정합 기능이 포함된 스켈레톤 코드와 SDL 도구에 의해 자동 생성된 코드 및 CORBA 정합 및 블록간 통신 기능을 제공하기 위한 입출력제어 및 SDL 커널 정합 기능을 제공하는 함수를 포함하여 실행 파일을 생성한다. CORBA와 SDL 시스템간 연동 흐름은 (그림 3)과 같으며, 단계별 동작은 다음과 같다.

단계 1 : SDL 시스템을 실행시키면 시스템 초기화 및 CORBA 연동을 위한 객체 서비스 중개자를 초기화하고 CORBA 객체와 POA를 식별할 수 있는 정보 등을 포함하는 IOR(Interoperable Object Reference) 값을 생성하고, 클라이언트로부터 요구를 기다린다.

단계 2 : SDL 시스템은 원격 시스템과의 시그널 기반 통신을 위한 입력 채널을 구성한다. CORBA 연동에서는 이용되지 않는다.

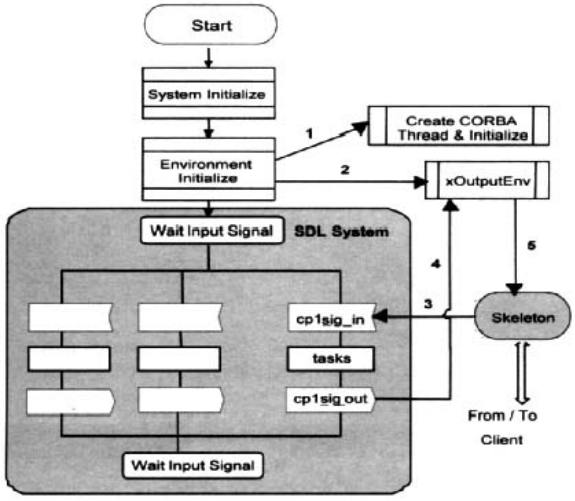
단계 3 : SDL 시스템은 CORBA 연동을 위하여 스켈레톤으로 응답 신호를 전달하기 위한 기능을 초기화 한다.

단계 4 : 클라이언트로부터 요구가 들어오면, 원격 함수 호출(또는 시그널)을 생성하고 연산의 매개인자를 할당

한 후 SDL 시스템으로 전송하고, 응답이 필요한 경우 응답(reply)을 기다린다.

단계 5 : SDL 시스템은 원하는 연산을 실행하고, 원격 시그널이 있는 경우 출력 환경 함수를 호출하고, 시그널 대기 상태로 천이한다.

단계 6 : 스켈레톤에서는 외부 출력 환경 함수로부터 응답 시그널을 수신하면, 응답 정보를 마셜링해서 클라이언트에게 전달한다.



(그림 3) CORBA SDL 연동 흐름도

3.3 스켈레톤 코드 확장 및 생성

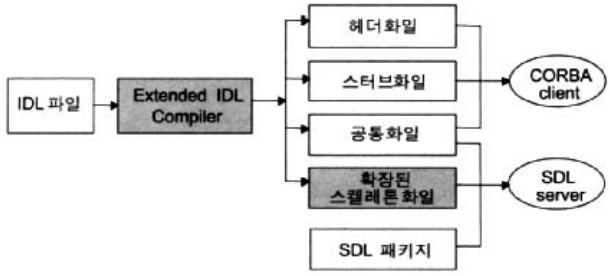
CORBA 시스템에서는 객체 서비스 중개자가 상호 연동 프로토콜(IIOP)을 통하여 클라이언트로부터 메소드 호출을 받으면 스켈레톤에서 구현 객체를 찾아 메소드 호출을 하고 결과 값을 클라이언트에게 반환한다. 그러나 본 모델에서는 클라이언트로부터 메소드 호출을 받으면 스켈레톤 코드는 메소드를 SDL 시그널로 변환시키고 호출 메소드에 포함된 매개변수 값들을 시그널의 매개 변수에 각각 매핑하고 SDL 시스템에 시그널을 전송한다. 클라이언트 측에서 응답이 없는 메소드를 호출한 경우는 시그널 전송 후 바로 리턴한다. 그러나 응답을 요구하는 양방향 메소드 호출인 경우 SDL 시스템의 출력 환경 함수로부터 응답 데이터 수신을 확인하는 신호를 대기한다. 아래 프로그램은 스켈레톤까지 전달되어 온 CORBA 함수 호출을 시그널을 이용하여 SDL 시스템으로 시그널을 전달하는 코드이다.

```
/* parameter setting */
& (((yPDP_senddata_in ) (yOutputSignal )) → Param 2.c )))
   = (* ((uint 32* ) _UNIORB_curptr ) ;
/* Send to SDL System */
SDL_Output (yOutputSignal, (xIdNode * ) 0 ) ;
/* Waiting for reply */
status = pthread_cond_wait (&senddata_wait, & senddata_
        wait.mutex ) ;
/* Return to Client */
```

SDL 시스템으로의 시그널을 전달하기 위하여 *xGetSignal* (*object_name*) 프리미티브를 통하여 해당 객체에 대한 시그널 정보 구조체(*yOutputSignal*)를 할당하고, 클라이언트로부터 전달된 매개 변수 값을 시그널정보 구조체의 각각의 매개변수 값으로 매핑시킨다. 이와 같이 생성된 시그널 정보를 SDL 커널을 통한 시스템과의 통신 프리미티브인 *SDL_Output* (*yOutput Signal*)을 이용하여 전달하고 응답이 필요한 경우 SDL 시스템으로부터 쓰레드 조건 시그널 응답을 기다리게 된다. SDL 시스템은 객체에 대한 동작을 수행한 후 응답이 결과 시그널을 처리할 수 없을 경우 외부 블록과의 통신을 위하여 시그널 출력 환경 함수를 통하여 스켈레톤으로 신호를 전달하게 된다. 여기서 시스템의 부하를 최소화할 수 있는 기법을 적용하였다.

CORBA의 함수 호출을 SDL 시그널 전송으로 변환하여 호출하는 작업을 위하여 스켈레톤 코드를 확장하였으며 확장된 코드 생성을 위하여 IDL 컴파일러를 수정하였다. IDL 컴파일러는 IDL에 정의된 함수에서 SDL 프로세스로 시그널을 송신하고 응답 시그널을 수신하여 클라이언트로 리턴하도록 하였다. 서버 시스템 측면에서 볼 때 클라이언트는 SDL 기반으로 구축되어있는 것처럼 보이고 클라이언트 측면에서 보면 서버가 CORBA 기반으로 구축된 것처럼 보이므로 두 시스템은 자연스럽게 연동된다.

SDL 시스템과 CORBA의 연동을 위하여 SDL 시스템이 사용하는 모든 전역 시그널에 대하여 대응하는 함수를 IDL 파일에 기술한다. IDL 컴파일러는 (그림 4)와 같은 단계로 실행된다.



(그림 4) 확장된 IDL 컴파일러 처리 절차

4. 시험 및 비교 평가

4.1 시 험

본 논문에서 제안한 SDL/CORBA 통합 모델을 시험하기 위하여 서버 시스템은 SDL로 구축하고 클라이언트 시스템은 uniORB 2.1을 이용하여 구성하였다. SDL/CORBA 통합 모델을 시험하기 위하여 (그림 5)에서 보듯이 type.idl이라는 IDL 파일이 제공하는 기본적인 자료 구조에 대한 연산을 수행하는 인터페스이스를 정의하고, 이를 위한 클라이언트/서버 시스템을 구축하였다. SDL/CORBA 통합 모델을 비교 분석

하기 위하여 SDL 및 CORBA 그리고, Socket을 사용하여 다음과 같은 두가지 모델을 시험하였다.

[모델 1] 서버시스템은 SDL시스템으로 생성하고, 클라이언트와 TCP 소켓을 이용

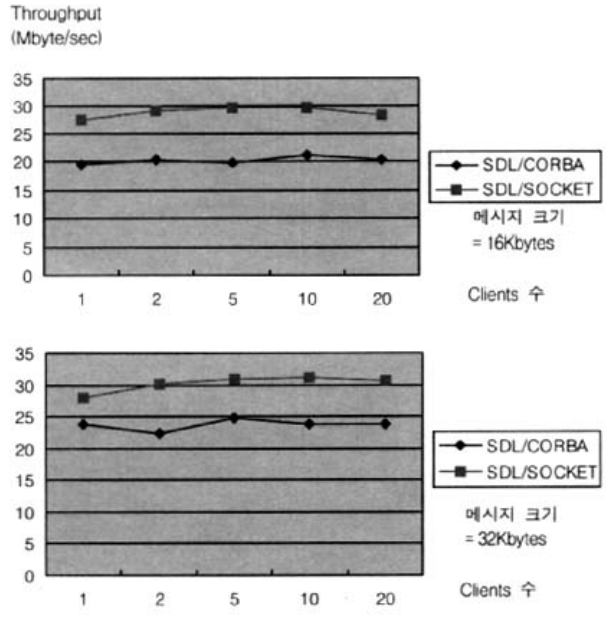
[모델 2] 서버는 SDL을 통하여 구축하고 클라이언트는 uniORB 2.1을 통하여 구축

```
interface test {
        typedef char msgType 1 [4×1024] ;
        typedef char msgType 2 [16×1024] ;
        typedef char msgType 3 [32×1024] ;
        typedef char msgType 4 [64×1024] ;
        long            sendMsg 1 (in msgType 1 param 1) ;
        long            sendMsg 2 (in msgType 2 param 1) ;
        long            sendMsg 3 (in msgType 3 param 1) ;
        long            sendMsg 4 (in msgType 4 param 1) ;
) ;
```

(그림 5) type.idl

먼저 [모델 1]은 (그림 5)의 type.idl 파일에 정의된 인터페이스 메소드들을 한 쌍의 시그널 셋으로 정의 한 후 서버는 SDL을 통하여 기술한 후 Tau의 C-advance 자동 코드 생성을 통하여 작성하고, SDL과 외부로의 통신 시그널들을 기반으로 UDP 소켓을 이용하는 클라이언트 프로그램을 작성하는데, 각각의 인터페이스 메소드에 대한 추상화를 하였다. [모델 2]는 본 논문에서 제안하는 SDL과 CORBA를 통합하는 방법이다. 인터페이스가 정의된 type.idl 파일을 SDL 시스템과 통합 가능하게 수정된 IDL 컴파일러를 통하여 스터브와 스켈레톤을 작성한 후, 서버는 [모델 1]과 마찬가지로 SDL로 기술 후 Tau를 통하여 자동 생성한 후, 클라이언트는 [모델 1]에서 사용했던 프로그램을 그대로 사용하였다. 이 두가지 모델에 대한 시험은 455MHZ UltraSparc 프로세서와 SunOS 5.7운영체제를 사용하였으며 SDL 시스템의 작성을 위하여 Tau 3.5를 사용하고 ORB로는 uniORB 2.1을 사용하였다. 성능의 비교를 위하여 두가지 실험 모델을 각각 10,000번씩 호출하는 실험을 하였다.

(그림 6)에서 보듯이 기존의 CORBA를 이용한 클라이언트/서버 모델과 SDL/Socket 모델에 비하여 본 논문에서 제안한 SDL/CORBA의 통합 모델은 초당 메시지 전송 능력에 있어서 약 27.7~33.4%의 오버헤드가 있음을 알 수 있다. 하지만, 본 논문에서의 실험은 두 시스템간의 단순한 통신 연산에 걸린 시간 비교이므로 본 논문에서 제안하는 장점으로 보상되어질 수 있다.



(그림 6) 통신 모델의 실행 성능 비교

4.2 비교 평가

본 논문에서 제안한 방법과 Orbix/SDL이 사용한 방법은 모두 CORBA를 통신 매체를 사용한다는 면에서는 동일하다. 그러나 이 두 방법은 <표 1>과 같이 SDL과 CORBA 시스템을 통합하는 방법에 차이가 있다. Orbix/SDL은 IDL을 사용하여 SDL과 CORBA를 통합하는 CORBA 지향적 접근방법을 사용한 반면 본 논문에서 제안한 방법은 SDL 시스템에서 통신 인터페이스 정보를 추출하고 이를 이용하여 SDL과 CORBA를 통합하는 SDL 지향적 접근방법을 사용하였다.

Orbix/SDL은 CORBA를 SDL 시스템의 하부 통신 매체로 사용하기 위하여 IDL-to-C++ 컴파일러와 IDL-to-SDL 컴파일러를 도입하고 이를 이용하여 SDL 시스템에서 변환된 C 프로그램과 Orbix ORB와의 연동을 위한 스터브/스켈레톤 코드와 C++ wrapper를 생성하였다[6]. Orbix/SDL은 이 코드를 통하여 클라이언트의 요구를 SDL 시스템에 매핑시켰다. 반면에 본 논문에서 제안한 모델은 SDL 시스템에서 생

<표 1> Orbix/SDL과 uniORB/SDL 비교

	Orbix/SDL	uniORB/SDL
통합 방법	SDL 수준의 스터브/스켈레톤 사용	메소드/시그널 매핑 규칙 사용
매핑 언어	SDL-C++ 매핑	SDL-C 매핑 및 SDL-C++ 매핑
통합코드 생성	IDL-to-SDL 컴파일러 사용 IDL-to-C++ 컴파일러 사용	IDL-to-C 컴파일러 사용 IDL-to-C++ 컴파일러 사용
통합 ORB	Orbix	uniORB

성된 통신 인터페이스 정보를 입력으로 하고 IDL-to-C 컴파일러/ IDL-to-C++ 컴파일러를 사용하여 클라이언트의 요구를 SDL 시스템에 매핑시킨다. Orbix/SDL은 통합을 위한 구현 언어로 C++만을 지원하지만 본 논문에서는 통합을 위한 구현 언어로 C언어 및 C++ 언어를 모두 지원한다.

5. 결 론

본 논문에서는 인터넷으로 연결된 분산 내장형 시스템에서 실행되는 응용 시스템을 효과적으로 개발하기 위하여 시스템 설계 및 구현은 SDL을 이용하고 시스템간의 통신 인터페이스는 미들웨어가 담당하도록 한 분산 병렬 시스템 모델을 제시하였다. SDL 시스템과 CORBA 간의 통신을 가능하게 하기 위하여 SDL과 CORBA 통신 프로토콜을 매핑하는 방법을 사용하였다. 이 방법은 다양한 하드웨어 플랫폼상에서 병렬로 실행되는 통신용 프로그램을 상위 설계 및 명세 언어인 SDL로 개발할 수 있어 시스템 설계 및 구현 단계에 소요되는 비용을 줄일 수 있으며, SDL로 작성된 서버 시스템 하부에 미들웨어를 채용할 수 있어 분산처리를 위한 별도의 프로그래밍 없이 이미 검증된 루틴을 사용하여 소프트웨어 시험에 소요되는 비용을 획기적으로 감축할 수 있다. 본 모델을 이용할 경우 SDL로 작성된 서버 시스템의 하부에 미들웨어를 채용할 수 있어 인터넷으로 연결된 어떠한 노드에서 실행되더라도 클라이언트와 연동이 가능해진다.

참 고 문 헌

[1] Object Management Group, *The Common Object Request Broker : architecture and Specification*, 2.4 ed., Oct., 2000.

[2] Object Management Group, *Realtime CORBA Joint Revised Submission*, OMG Document orbos/99-02-12 ed., Mar., 1999.

[3] K. E. Cheng and L. N. Jackson, "MELBA+ : An SDL Software Engineering Environment," The fourth SDL forum, SDL'89 : The Language at Work, pp.95-103, 1989.

[4] R. R. Singh and J. Serviss, "Code Generation using GEODE : A CASE Study," The ninth SDL Forum, SDL'97 : TIME FOR TESTING-SDL, MSC and Trends, pp.539-550, 1997.

[5] The CORBA Integration, SDT manual, 1997.

[6] Morgan Bjkander, Using SDL to develop CORBA object implementations, Telelogic AB, 1997.

[7] ITU-T.Z.100, CCITT Specification and Description Language (SDL). ITU, 1993.

백 의 현

e-mail : ehpaik@etri.re.kr

1984년 숭실대학교 전자계산학과(공학사)

1987년 숭실대학교 전자계산학과(공학석사)

1997년 숭실대학교 전자계산학과(공학박사)

1987년~현재 한국전자통신연구원 네트워크 연구소 책임연구원

관심분야 : 광가입자망 기술, 통신망 관리, 인터넷 프로토콜, 병렬처리, 미들웨어 등

허 재 두

e-mail : jdhuh@etri.re.kr

1987년 경북대학교 전자공학과(공학사)

1990년 경북대학교 전자공학과(공학석사)

2000년 경북대학교 전자공학과 정보통신공학(공학박사)

1987년~현재 한국전자통신연구원 네트워크 연구소 네트워킹기술팀장, 책임연구원

관심분야 : 광가입자망 전송, 통신망 시그널링, 고속 인터넷 프로토콜, 고속 패킷통신, 홈네트워크, 통신망 관리 등

이 형 호

e-mail : holee@etri.re.kr

1977년 서울대학교 공업교육과 전자전공(공학사)

1979년 한국과학기술원 전기 및 전자공학과(공학석사)

1983년 한국과학기술원 전기 및 전자공학과(공학박사)

1983년~현재 한국전자통신연구원 네트워크연구소 액세스기술 연구부장, 책임연구원

관심분야 : 광 가입자망 기술, 고속 LAN 및 라우터 기술, 인터넷, 패킷통신, BISDN망, ATM 교환, 무선 및 이동 망