통신 시스템을 위한 공유 메모리 기반 CORBA 연동 프로토콜 설계 및 구현

장 종 현[†] · 이 동 길[†] · 최 완[†] · 한 치 문[††] · 장 익 현[†††]

요 약

최근 분산컴퓨팅 시스템은 서로 이질적인 시스템간 상호 연동성 문제를 해결하기 위한 새로운 시스템 아키텍처를 제시하고 있다. 본 논문에서는 기존 CORBA 연동 프로토콜의 성능 및 프로세스간 통신 기법의 분석을 통하여 통신 시스템용 소프트웨어에 적합한 공유 메모리 기반의 CORBA 연동 프로토콜 모델을 제시한다. 동일 호스트상에서 메시지 전달 오버헤드를 최소화하기 위해, 제안된 모델을 통신 시스템 개발에 표준화된 CORBA 메커니즘을 적용함으로써 목적 통신 시스템의 하드웨어 구조에 독립적이며 융통성 및 확장성 있는 소프트웨어 구조의 개발이 가능하고 소프트웨어의 생산성과 품질 및 재사용성을 향상시킬 수 있다.

Design and Implementation of CORBA Inter-ORB Protocol Based on Shared Memory for Communication Systems

Jonghyun Jang[†] · DongGill Lee[†] · Wan Choi[†]
ChiMoon Han[††] · IkHyeon Jang[†††]

ABSTRACT

Distributed systems present new system architecture for solving interoperability problem in heterogeneous system. This paper presents CORBA Inter-ORB protocol model based on shared memory to support communication software through analysis of existing CORBA IIOP protocol performance and Inter-Process Communication techniques. In the same host environment, proposed model applied standard CORBA mechanism to minimize message transfer overhead can develop software independently to hardware architecture of target communication system. This communication software that has flexibility and extensibility can improve productivity, quality and reusability of software.

키워드 : 코바(CORBA), 인터넷 연동 프로토콜(IIOP), 미들웨어(Middleware), 통신 시스템(Communication System)

1. 서 론

오늘날 네트워크 구조는 인터넷의 급격한 발전과 서비스 이용자의 증가로 새로운 서비스 요구에 즉시 대응할 수 있는 형태로 발전하고 있다. 따라서 네트워크 시스템 구조는 표준 인터페이스를 이용하여 이기종간 상호 연동을 통해 새로운 서비스를 제공할 수 있는 미들웨어 기반의 개방형 구조를 지향하고 있다.

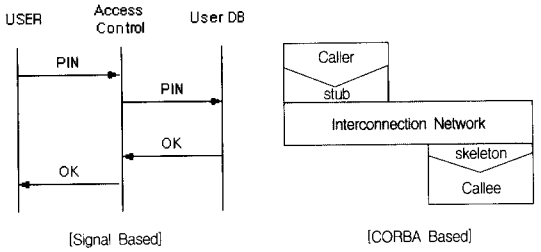
CORBA는 하드웨어와 소프트웨어에 독립적인 객체지향 언어를 기반으로 하는 소프트웨어 서비스 제공 목적으로 OMG (Object Management Group)에 의해 표준 인터페이스 규격을 정의하고 있다. OMG는 1991년에 시작되었으며, 현재 800개 이상의 산업체가 참여하고 있다. 현재 개발된 다른 유형의 미들웨어로는 SUN Microsystems의 Java 언어 기반 RMI (Remote Method Invocations), Microsoft의 DCOM(Distributed Component Object Model) 및 DEC의 RPC 등이 있다. 또한 CORBA를 이용한 분산 플랫폼은 일반적인 응용에서부터 실시간 통신 시스템 및 내장형 시스템으로 확장되어 사용되고 있다.

소프트웨어 생산성 및 재사용성에 대한 중요성이 점차 증가하면서 통신 소프트웨어 분야에서도 운영체제와 응용 프로그램 사이의 분산 처리를 지원하는 CORBA를 기반하는 소프트웨어 개발이 연구되고 있다. 현재 통신 시스템용 소프트웨어 구조가 표준 인터페이스 기반의 개방형 구조로 진화함에 따라 소프트웨어 개발 기간의 단축이 시스템 경쟁력 확보의 관건이 되고 있다.

(그림 1)에서 보는 바와 같이 시그널 기반 소프트웨어 개발 구조는 운영체제 수준에서 지원되는 프로세스간 통신 기법(Inter-Process Communication)을 통하여 분산 객체를 접근함으로써 운영체제, 프로그래밍 언어 및 메시지 전달 프로토콜에 의존하는 구조적이 제약이 있는 반면 CORBA 기반

† 정 회 원 : 한국전자통신연구원
†† 정 회 원 : 한국외국어대학교 전자공학과 교수
††† 종신회원 : 동국대학교 정보통신공학과 교수
논문접수 : 2003년 2월 7일, 심사완료 : 2003년 5월 22일

개발 방법론은 운영체제 및 프로그래밍 언어에 독립적이며, 구현 객체의 위치 투명성을 플랫폼 수준에서 제공하여 소프트웨어 개발의 유연성과 확장성을 확보할 수 있다.



(그림 1) 통신 시스템 소프트웨어 개발 구조

또한, CORBA 기반 소프트웨어 개발 방법론을 통신 시스템용 소프트웨어 개발에 접목함으로써 소프트웨어 개발 기간의 단축과 시스템 구성의 유연성을 확보하여 시스템 유지 관리에 효율성을 향상시킬 수 있다.

따라서, 본 본문에서는 개방형 추세에 부응하기 위하여 시그널기반의 통신 시스템용 소프트웨어 개발 방법에서 객체 지향 개념의 CORBA 기반의 소프트웨어 개발 방법의 새로운 패러다임을 채택한다. 이를 위하여 통신 시스템용 고성능 CORBA 플랫폼을 개발하기 위한 방안으로 동일 호스트상에서 실행되는 응용 블록간 복사 처리를 위한 공유 메모리 기반 연동 프로토콜 모델을 제안한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 CORBA 분야의 규격 및 기존 연구 내용을 분석하고, 제 3장에서는 공유 메모리 기반 연동 프로토콜 모델의 설계 및 구현에 대하여 기술하고, 제 4장에서는 제안한 모델의 성능 분석을 통한 문제점을 도출하고, 제 5장에서 결론을 맺는다.

2. 관련 연구 분석

CORBA 규격에서는 객체중개자간 상호 연동을 지원하기 위하여 연동 프로토콜을 정의하고 있는데, 이것은 객체중개자간의 객체서비스의 상호 교환에 공통의 메시지 형식과 통신 프로토콜을 정의하여 서비스를 투명하게 제공할 수 있도록 한다. OMG에서 정의한 CORBA에 대한 주요 기능과 성능 향상을 위한 관련 연구의 분석하고, 일반 CORBA의 구성 요소에 대한 성능 분석을 통하여 성능 향상 방법을 안출하고자 한다.

2.1 CORBA 규격

일반적으로 CORBA는 구현이 아닌 인터페이스를 정의하는 규격으로, 주요 특징으로는 객체 위치 투명성, 연결 및 메모리 관리, 매개 변수 (디)마셜링, 이벤트 및 요구의 (역)다중화, 오류 처리 및 고장 감내, 객체/서버의 활성화, 병행성 및 보안 기능을 제공한다. CORBA는 언어, 운영체제, 하드웨어 및 네트워크 프로토콜의 이질적인 의존성으로부터 독립적인 응용을 구현할 수 있는 인터페이스를 제공한다[1].

CORBA 표준에서는 시스템 개발을 위한 다음 4가지 인터페이스를 정의하고 있다.

- IDL(Interface Definition Language)

구현 객체에 대한 인터페이스는 프로그래밍 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 제공하여야 한다.

- ORB(Object Request Broker)

CORBA를 이용한 클라이언트 프로그램은 객체 서비스 중개자를 통하여 동일 시스템 또는 원격 시스템에 위치한 서버측 구현 코드를 투명하게 요구할 수 있는데, 객체 서비스 중개자는 클라이언트의 서비스 요구를 가로채어, 객체 서비스를 제공하게 될 객체 어댑터를 찾아서 매개 변수 값을 포함하여 해당 메쏘드를 호출한 후 반환 값이 필요한 경우 결과 값을 반환 받는 절차를 수행한다. 이때 클라이언트는 서비스 구현 객체의 위치, 운영체제 및 하드웨어 및 프로그래밍 언어에 대한 정보는 알 필요 없이 서비스를 요구하게 된다. 이와 같이 객체 서비스 중개자는 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.

- POA(Portable Object Adaptor)

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

- 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, 매개 변수 값을 포함하는 정보를 마셜링/디마셜링 하기 위한 규격을 정의한다.

2.2 IIOP(Internet Inter-ORB Protocol) 프로토콜

IIOP 프로토콜은 CORBA IDL 언어로 작성된 메시지를 CDR(Common Data Representation) 문법을 이용하여 마셜링한 후 TCP/IP 전송 프로토콜을 통하여 표준 GIOP 포맷의 메시지를 상대방 CORBA로 전달하는 기능을 담당한다. IIOP 동작은 OMG 규격 2.0에 정의되어 있으며 기본 기능은 다음과 같다.

2.2.1 객체 위치 서비스

IIOP 프로토콜은 클라이언트/서버 모델에서 객체의 위치 서비스를 정의한다. 즉, 클라이언트가 객체 접근을 위하여 *LocateRequest* 메시지를 전달하면, 서버는 해당 객체에 대한 객체 정보를 포함하는 *LocateReply* 메시지로 응답하여 객체의 위치를 알려주게 된다.

2.2.2 연결 관리

클라이언트는 서버가 생성한 객체 참조값(Interoperable Object Reference)으로부터 추출된 서버의 주소정보를 기반으로 연결을 설정하고 관리하게 된다. 단일 쓰레드 기반의 CORBA에서 서비스 요구 및 응답 메시지는 연결 서비스 기반의 전송 프로토콜을 적용하므로 순서가 보장되지만, 멀티 쓰레드 기반 CORBA에서는 응답 메시지의 순서는 보장되지 않는 비동기 모드로 동작하게 된다. 연결 해제는 클라이언트/서버 양측에서 기동할 수 있는데, 연결해제 메시지를 수신한 IIOP 객체는 연결 식별자 기반으로 연결과 할당된 자원을 해제하게 된다.

2.2.3 메시지 형식

<표 1>와 같이 7개의 메시지 타입이 정의되어 있으며, 모든 메시지에는 12바이트의 필수 메시지 헤더 정보가 포함되며 메시지 타입에 따라 서비스 관련 부가 헤더 정보가 추가될 수 있다.

〈표 1〉 IIOP 메시지 타입

메시지 타입	방향	메시지 값	설명
Request	Client ⇨ Server	0	서비스 요청
Reply	Client ⇦ Server	1	서비스 응답
CancelRequest	Client ⇨ Server	2	서비스 취소 요청
LocateRequest	Client ⇨ Server	3	객체 위치 요청
LocateReply	Client ⇦ Server	4	객체 위치 응답
CloseConnection	Client ⇦ Server	5	연결 종료
MessageError	Client ⇔ Server	6	메시지 오류

2.2.4 메시지 전송

IIOP에서 지원할 수 있는 전송 프로토콜로는 일반적으로 TCP/IP를 기반으로 하며, 전달되는 메시지의 특성, 데이터 양 또는 적용 분야에 따라 ATM과 같은 고속의 전송 프로토콜을 적용하여 개발되기도 한다.

2.3 CORBA 성능 향상 방안 분석

IIOP 기반의 범용 CORBA 플랫폼을 통신 시스템과 같은 고성능을 요구하는 시스템에 적용하기 위해서는 성능을 만족시키지 못하여 TMN과 같은 망관리 분야에 국한되어 적용되고 있다. 따라서, CORBA의 성능을 향상시키기 위한 다양한 연구가 진행되고 있는데, 그 방안을 요약하면 크게 세가지로 나누어 볼 수 있다[2, 3].

① 데이터 타입 변환을 위한 (디)마셜링 기법의 최적화

데이터 타입 변환과정에서 발생하는 빈번한 메모리 복사는 CORBA의 성능 저하를 야기 시킨다. 하지만, 입출력 벡터와 같은 방식으로 최적화를 할 수 있으나 통신 시스템에 적용할 수 있는 성능을 만족시키지는 못한다.

② 데이터 및 헤더 정보에 대한 전달 정보의 압축

GIOP 헤더 정보 중에서 magic 정보 등을 생략할 수 있으나 CORBA 규격을 위배하게 되며, 헤더 정보를 제외한 데이터 압축 방식은 데이터의 크기 및 시스템의 부하 상태에 따라 성능이 크게 변화함을 알 수 있다.

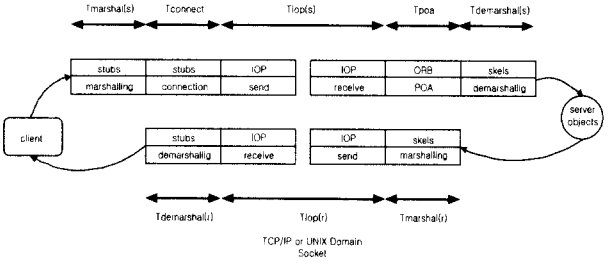
③ 메시지 전달 시간의 최적화

전체 처리 시간의 약 70%에 달하는 메시지 전달 시간을 최적화하기 위한 방안으로 ATM과 같은 고속 전달망을 활용할 수 있으나, 동일시스템에서의 공유 메모리 또는 메시지 큐와 같은 메커니즘을 적용할 수 있다.

본 논문에서는 기존 CORBA 플랫폼에 대한 구성 요소 및 전달 방식에 따른 성능 분석을 통하여 최상의 성능 향상 방법을 찾고자 한다.

2.4 CORBA 연동 프로토콜 성능 분석

IIOP 프로토콜은 TCP/IP 기반의 네트워크 상에서 여러 개의 객체 중개자(ORB)간의 통신을 위해 표준 메시지 형식과 공통 데이터 표현형의 세트로 정의된 GIOP 메시지를 교환하는 프로토콜을 의미한다. 일반적인 CORBA 통신 모델을 살펴보면 (그림 2)과 같이 두 컴포넌트간 통신을 위하여 TCP/IP 또는 유닉스 도메인 소켓을 주로 이용한다.



(그림 2) CORBA 통신 모델

통신 시스템용 고성능 플랫폼 설계를 위하여 썬 마이크로시스템즈사의 Solaris2.7 운영체제와 UltraSparc 167MHz, 192MB 바이트의 주 기억장치를 가진 시험환경에서 기본 CORBA 플랫폼의 성능 분석을 통하여 최적화할 수 있는 요소를 찾고자 한다.

따라서, 클라이언트에서 서버로의 CORBA 서비스 요청이 항상 일정한 패킷 전달능력을 유지한다고 가정할 때, CORBA의 성능은 (디)마셜링, 객체 활성화 및 메시지 전달 시간과 같은 속성에 의하여 결정되게 된다.

본 논문에서는 3개의 *in*, *out*, *inout* 매개변수와 동일한

타입의 값을 반환하는 서비스에 대한 성능 분석을 통하여 기존 모델에서 성능을 향상시킬 수 있는 기준을 도출하고자 한다.

다양한 데이터 타입에 대하여 성능은 <표 2>과 같은 결과를 얻을 수 있었다.

〈표 2〉 CORBA 구성요소 별 실행시간 비교

(단위 : 초)

구 분	*Ttrans*	*Tpoa*	*Tmarshal*	*Ttotal*	*(Ttrans/Ttotal) ×100*
Character	*0.007552*	*0.000067*	*0.000207*	*0.007826*	*94%*
Long	*0.003467*	*0.000078*	*0.000201*	*0.003747*	*93%*
Float	*0.003776*	*0.000066*	*0.000203*	*0.004045*	*88%*
Double	*0.005549*	*0.000067*	*0.000207*	*0.005823*	*91%*
String	*0.005071*	*0.000063*	*0.000249*	*0.005384*	*90%*
Sequence	*0.003208*	*0.000064*	*0.000264*	*0.003536*	*84%*

여기서, 마셜링 시간 *Tmarshal*는

$$Tmarshal = \sum (Tmarshal(s) + Tdemarshal(s) + Tmarshal(r) + Tdearshal(r)) \quad (1)$$

서버측의 객체중개자가 사용하는 시간 *Tpoa*는 요청된 서비스를 해석하고, 구현 객체 저장소에서 해당 객체를 찾아서 서비스를 기동시키는데 소요되는 시간을 말한다.

전달 계층을 통하여 클라이언트 측에서 마셜링된 메시지를 서버측으로 전달하고, 결과값을 반환하는데 소요되는 시간 *Ttrans*는

$$Trrans = \sum (Tiop(s) + Tiop(r) + Tconnect) \quad (2)$$

따라서, 하나의 CORBA 서비스를 처리하는데 소요되는 시간 *Ttotal*은

$$Ttotal = \sum (Tmarshal + Tpoa + Ttrans) \quad (3)$$

식 (2)와 식(3)으로부터 연동프로토콜의 전달에 소요되는 비율 *Ratio*는 다음과 같다.

$$Ratio = (Ttrans / Ttotal) \times 100$$

메시지를 연동 프로토콜을 통하여 전달하는데 소요되는 *Ttrans* 시간이 차지하는 비율은 데이터 타입에 따라 약간의 차이가 있으며, 또한 전달되는 매개 변수의 데이터 크기에 따라 변화는 있겠지만 약 80% 이상의 시간이 소요됨을 확인할 수 있었다[5].

다음은 동일 시스템내의 클라이언트/서버 모델에서 프로세스간 통신 방법으로 공유 메모리와 TCP/IP, Unix Domain Socket에 대하여 다른 크기의 데이터에 대하여 1000번씩 전달하는데 소요되는 시간을 분석하였다.

〈표 3〉 통신 모델에 대한 수행시간 비교

(단위 : 초)

구 분	1KB	4KB	16KB	32KB	64KB
TCP/IP	0.0832	0.0862	0.1640	0.3054	0.6106
Unix Domain Socket	0.0310	0.0371	0.1329	0.2618	0.4537
Shared Memory	0.0755	0.0790	0.0784	0.0770	0.0768

<표 3>에서 보듯이 시스템 내부에서 공유 메모리를 이용하는 경우 데이터 크기에 상관없이 거의 동일한 전달 능력을 보이는 반면, TCP/IP와 Unix Domain Socket 프로토콜에서는 데이터 크기에 민감하게 실행시간이 급격히 증가하였다.

3. 공유 메모리 기반 연동 프로토콜 모델 설계 및 구현

통신 시스템의 특성상 분산 하드웨어 장치에 기능이 내장되어 실행되는 특성을 지원하기 위한 CORBA는 실시간 처리, 고신뢰성, 고성능의 특성화된 기능을 요구한다. 일반적으로 분리된 기능 모듈간 통신을 위하여 소켓과 같은 프로세스간 연동 프로토콜을 적용하거나 RPC를 이용한 원격 프로시쥬어 호출 기능을 이용하지만, CORBA 응용의 경우에는 소켓 기반의 IIOP 프로토콜을 표준으로 하고 있다.

본 논문에서는 제 2장의 성능 분석을 통하여 메시지 전달 시간의 최소화를 위한 방안으로 동일 시스템 내에서 공유 메모리 기반의 연동 프로토콜 모델을 제시하고, 해당 모델을 적용하여 통신 시스템에 적용 가능한 고성능 CORBA 플랫폼을 개발하고자 한다. 그리고, 본 논문에 적용된 CORBA는 미국 redhat사의 지원하에 진행되는 GNOME 프로젝트의 하나인 ORBit[6] CORBA를 기본 플랫폼으로 사용하였다.

3.1 공유 메모리 기반 인터넷 연동 프로토콜 모델 구조

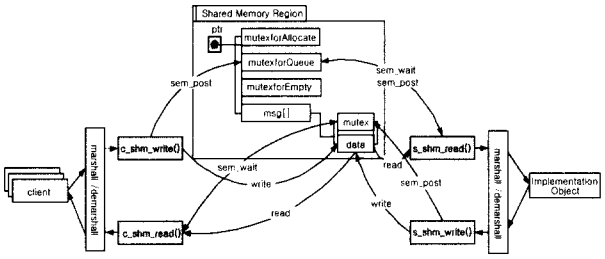
현재 유닉스 기반 운영체제에서 이용할 수 있는 공유메모리 기법은 System V 계열과 Posix 계열이 있다. System V 계열 공유 메모리 기법은 제한적인 공유 메모리의 크기를 변경하기 위해서는 커널을 재구축해야 하는 문제점이 있으나, POSIX 계열은 이러한 제약이 없으므로 본 모델 구현에 적합하다.

공유 메모리 기반의 인터넷 연동 프로토콜의 모델을 구현하기 위해서는 다음과 같은 조건을 만족하여야 한다.

- IPC 통신을 위하여 서버측 IIOP에서 공유 메모리를 할당
- 각각의 클라이언트에서 공유 메모리를 공유해야 하므로 상대주소를 사용
- 다중 프로세스의 동기화 기능을 제공

이 조건을 만족하기 위하여 (그림 3)에서 보는 바와 같

이 서버측 IIOP에서 공유 메모리 영역을 할당하고 동기화를 위한 세마포어 변수들의 초기화 절차를 수행한다. 클라이언트측 에서는 생성된 IIOP 메시지를 공유 메모리의 메시지 영역에 삽입한 후 세마포어 값을 변경하여 서버측에서 처리할 수 있는 방식으로 처리하게 된다. 자세한 동작 알고리즘 및 상태 천이는 다음 절에서 소개한다.



(그림 3) 공유 메모리 기반 IIOP 모델

공유 메모리를 이용하여 GIOP 메시지를 전달하기 위한 자료구조는 다음과 같이 클라이언트의 식별, 메시지 정보 및 방향 서비스 요청과 응답 여부를 구분하기 위한 자료구조는 다음과 같다.

```
typedef struct GIOPMsgSlot {
    sem_t   mutex ;              /* read signal for client */
    int     clientID ;           /* Client Identifier */
    char    data[MSGSIZE] ;      /* GIOP Message */
    int     response_expected ;  /* Check oneway / both-way */
} MsgSlot ;
```

또한, 서버측에서 메시지 통신을 위하여 생성되는 공유 메모리 내에 유지되는 자료구조는 다음과 같다.

```
struct ShmIIOPMsg {                    /* struct stored in shared
                                          memory */
    sem_t       mutexforAllocate ;     /* semaphore for slot
                                          allocation */
    sem_t       mutexforQueue ;        /* semaphore for queue
                                          control */
    sem_t       mutexforEmpty ;        /* semaphore for queue
                                          control */
    int         reason ;               /* Server failed reason */
    GIOPMsgSlot msg[MAX_CONN] ;        /* Actual GIOPmessages */
} ;
```

ShmIIOPMsg 구조체는 서버와 클라이언트 IIOP 프로토콜간의 통신을 위한 공유메모리에 유지되며, *mutexforAllocate*는 *GIOPMsgSlot*를 할당할 때 클라이언트간 충돌을 방지하기 위하여 사용되는 세마포어이다. *mutexforQueue*와 *mutexforEmpty* 세마포어는 클라이언트와 서버간의 삽입된 메시지를 상대방이 읽을 수 있도록 제어하기 위하여 사용된다.

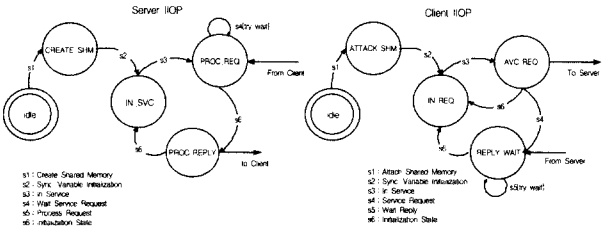
그리고, 클라이언트에서는 *reason* 값을 검사하여 서버 객체의 활성화 여부를 판단하고, 비정상적인 종료시 할당된 자원을 회수하고 초기화함으로써 동기화 문제를 해결하는데 사용된다.

또한, 공유 메모리 기반 IIOP 모델을 지원하기 위해서는 IOR 구조를 확장하여야 하는데, *IOP_TAG_UNIORB_SHAREDMEMORY*의 새로운 프로파일에 대한 타입을 정의하고 서버가 생성한 공유 메모리의 경로 정보를 포함하도록 (그림 4)와 같이 확장하였다.

Object ID	Profile Count	Profile Type	Shared Memory Name Path	Object Key

(그림 4) 공유 메모리 기반 IIOP 모델의 IOR 구조

3.3 상태 기계(State Machine) 및 동작 알고리즘 분석



(그림 5) IIOP 프로토콜 상태 천이도

(그림 5)에서 보는 바와 같이 서버측 상태 천이는 초기의 IDLE 상태에서 특정 크기의 공유 메모리를 생성하기 위한 CREATE_SHM로 천이하고, 만약 이미 존재하면 기존 공유 메모리를 삭제하고 새로운 공유 메모리를 생성하고 IN_SVC 상태로 천이하게 된다. 다중의 클라이언트 프로세스로부터 서비스 요청에 대한 다중 프로세스 동기화를 위한 세마포어 변수들을 초기화하는 IN_SVC 상태에서 클라이언트로부터 서비스 요구를 대기하는 PROC_REQ로 천이하게 된다. 클라이언트로부터 메시지 삽입을 확인하는 세마포어 값이 변경되면 해당 서비스를 처리한 후 응답을 보내기 위한 PROC_REPLY에서 응답 메시지를 전달한 후 새로운 서비스 요청을 처리하기 위한 IN_SVC 상태로 천이하게 된다.

클라이언트 상태 천이는 초기 IDLE 상태에서 서버측에서 생성된 공유 메모리를 공유하기 위하여 ATTACH_SHM 상태로 천이한 후, 동기화 변수값을 초기화하기 위한 IN_SVC 상태로 천이하기 된다. 여기서 다중 클라이언트간 충돌을 방지하기 위해 정의된 메시지 슬롯을 할당 받게 되면 서비스 요청을 위한 메시지를 생성한 후 서버측으로 전달할 수 있는 SVC_REQ 상태가 된다. 만약 서버로부터 응답이 요구되는 요청인 경우 REPLAY_WAIT 상태로 천이하여 응답을 대기하고, 응답 처리 후 IN_SVC 상태로 천이하게 된다. 그러나 단방향 서비스 요청인 경우 서비스를 시작할 수 있는 IN_SVC 상태로 천이하는 과정을 반복하게 된다.

공유 메모리 기반 연동 프로토콜 모델을 구현하는 과정에서 발생할 수 있는 다음과 같은 문제점을 확인할 수 있

었다. 첫째, 서버측 연동 프로토콜이 공유 메모리 영역을 생성할 때까지 클라이언트의 공유 메모리 접근을 방지하여야 한다. 둘째, 서버측 연동 프로토콜이 응답을 반환할 때까지 새로운 서비스를 시작하지 않아야 한다. 셋째, 객체중개자가 구현 객체를 수행하고 있는 도중에 실행중지가 발생하게 되면, 공유 메모리에 구현 객체의 비활성화 정보를 전달하여 세마포어의 오류를 방지하여야 한다.

이와 같은 문제점을 해결하기 위하여 서버측 연동 프로토콜은 구현 객체가 실행되는 동안 클라이언트 프로그램이 사용자에 의한 실행 중지 및 프로그램 오류에 의해 서비스 중지가 발생되는 경우 클라이언트측에서 제공하는 *keepalive* 값을 검사하여 세마포어 정보를 초기화하는 기법으로 문제점을 해결하였다.

3.3.1 서버측 IIOP 프로토콜 알고리즘

```
// Server Side IIOP
Declaration variables
SharedMemoryBuffer {
    Locks MUTEX_LOCK [M] ;
    msg [SIZE] GIOPMessage ;
} ;
SharedPtr *Ptr ;

Initialization
        Ptr = MemoryMap (NULL, fd, flags) ;
        Semaphore Initialize (Lock[M])
Action
1 : Make Server Shared Memory Buffer Create
2 : While loop
3 : do
4 :     Semaphore Trywait (Lock)
5 :     Wait for Service Request from Client
6 :     Read GIOPMessage from SharedBuffer
7 :     Semaphore Post (Lock)
8 :     Decode GIOPMessage
9 :     Handle Incoming Message
10 :    if Reply_required then
11 :        Send Reply GIOPMessage to Client
12 :        Semaphore Post (Lock)
13 :    fi
14 : done
```

① 서버측 연동 프로토콜에서 공유 메모리 영역을 할당하고 클라이언트와 메시지 교환을 위한 자료 구조를 생성하고 초기화를 수행한다.
② 클라이언트로부터 서비스 요구를 기다린다.
③ 공유 메모리 버퍼에서 GIOP 메시지를 읽는다.
④ 클라이언트가 새로운 서비스 요청을 하지 못하도록 동기화를 위한 세마포어 변수를 변경한다.
⑤ GIOP 메시지를 해석하고 요구 서비스를 처리한다.
⑥ 응답이 필요한 경우 응답 메시지를 생성하여 공유 버퍼에 삽입하고 클라이언트가 읽을 수 있도록 세마포어 값을 변경한다.
⑦ ②~⑥ 절차를 반복 수행한다.

3.3.2 클라이언트측 IIOP 프로토콜 알고리즘

```
// Client Side IIOP
Declaration variables
SharedMemoryBuffer {
    Lock MUTEX_LOCK [M] ;
    Channel Integer [N] ;
    msg [SIZE] GIOPMessage ;
    keelAlicve Integer ;
    reply_expected Integer ;
} ;
SharedPtr *Ptr ;
Initialization
        Ptr = MemoryMap (NULL, fd, flags) ;
Action
1 : Make Client Shared Memory Buffer Create
2 : Channel Allocation for Client Process
3 :
4 : Construct GIOPMessage
5 : Semaphore TryWait (Lock)
6 : Write GIOPMessage to SharedBuffer
7 : Semaphore post (Lock)
8 : Semaphore wait (Lock)
9 : if oneway is not set then
10 :    Wait for ReplyMessage from Server
11 :    Handle ReplyMessage
12 : fi
13 : Channel Close
```

① 서버측 IIOP 프로토콜에서 생성한 공유 메모리를 자신의 프로세스에 부착하고 자료 구조를 생성한다.
② 클라이언트간 충돌을 방지하기 위한 고유의 채널을 할당 받는다.
③ 서비스 요청을 위한 GIOP 메시지를 생성한다.
④ 생성된 메시지를 공유 메모리 버퍼의 데이터 영역에 삽입하기 위한 세마포어를 획득한다.
⑤ 공유 메모리상의 데이터 영역에 메시지를 복사한 후 세마포어 카운터를 변경시킨다.
⑥ 만약 단방향 서비스 요청인 경우 해당 채널을 해제하고 서비스를 종료한다.
⑦ 응답이 필요한 경우 서버로부터 응답 메시지를 대기한다.
⑧ 응답 메시지가 수신되면 해당 메시지를 처리한다.
⑨ 서비스가 완료되면 해당 채널을 해제한다.

4. 성능 분석

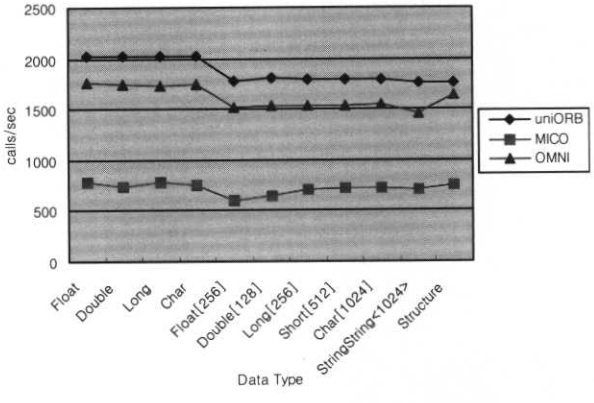
성능 분석은 널(null) 함수로 작성된 서버 프로그램에 대하여 OMG 규격에서 정의한 데이터 타입을 이용하였다. 시스템 구성은 서버와 4개의 클라이언트로 구성하였으며, 4개의 클라이언트에서 동시에 서비스를 요청한 후, 결과를 반환받는 형태로 CORBA 처리율을 비교하였다. 성능 분석 시스템 환경은 썬 마이크로 시스템즈사의 솔라리스 버전 2.7 운영체제와 UltraSparc 167MHz, 192MB바이트의 주 기억장치의 시스템을 사용하였다.

기존 범용 및 상용 CORBA 플랫폼은 공유 메모리 기반의 IIOP 프로토콜을 지원하지 않아 비교를 할 수 없으므로, 평

가 대상을 동일 플랫폼의 IIOP 프로토콜간 성능으로 제한하였다.

먼저, 플랫폼의 신뢰성을 위하여 독일에서 공개용으로 개발된 MICO(Version2.3.5)[4] 및 루슨트테크놀러지에서 개발된 omniORB(3.0.4)[5] CORBA를 대상으로 비교하였다.

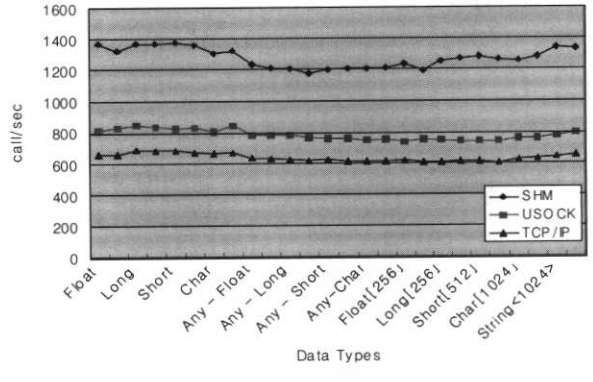
5개의 다중 클라이언트에서 다양한 데이터 타입의 함수[7]를 1000회 호출하여 소요되는 시간을 비교한 결과 (그림 6)와 같이 본 논문에서 제안된 모델을 적용한 CORBA 플랫폼(uniORB)은 C++언어로 구현된 MICO에 비하여 각각의 데이터 타입에 대하여 약 2.5배, omniORB에 비하여 15% 의 성능 향상을 보였다.



(그림 6) 다중 클라이언트에서의 성능 비교

다음은 본 논문에서 제안한 방식의 공유 메모리 기반의 IIOP 모델과 TCP/IP 및 유닉스 도메인 소켓 기반 플랫폼에 대하여 성능을 측정하였다.

첫 번째 성능 측정은 클라이언트에서 다양한 형태의 데이터 타입의 서비스를 1000회 요청하는 시험을 수행하였다. 데이터 유형에 따른 처리율을 (그림 7)에 나타냈다. 그림에서 보면, 공유 메모리 기반의 연동 프로토콜을 이용하는 경우 유닉스 도메인 소켓에 비하여 약 50%, TCP/IP 기반의 인터넷 연동 프로토콜에 비하여 2배의 처리율이 향상됨을 알 수 있다.

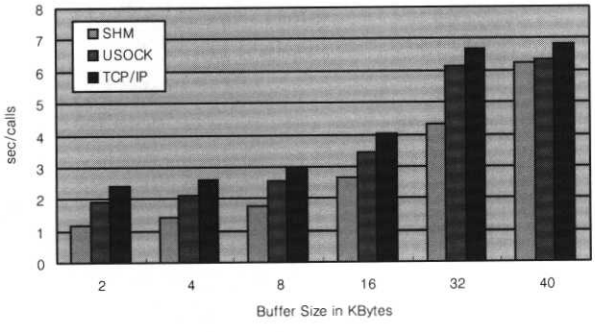


(그림 7) 데이터 타입에 따른 처리율 비교

이와 같은 성능 향상은 데이터 타입에 따른 마셜링/디마셜링 및 객체중개자에서 해당 객체를 찾고 활성화에 소요되는 시간은 동일하지만 TCP/IP 기반 모델에 비하여 커널 내부에서의 데이터 복제 시간에 따른 오버헤더가 발생하지 않으므로 2배 이상의 성능 향상을 보였다.

두 번째 성능 분석은 전달되는 문자 타입의 데이터 크기를 변화시키고, 서버측 구현 코드에서 전달된 데이터 크기를 비교한 후 결과값을 반환하는 실험을 수행하여 처리율을 비교하였다. (그림 8)는 1000회 서비스 요청/응답에 대해 소요되는 시간을 나타냈다.

그림에서 보면, 공유 메모리 기반의 연동 프로토콜이 데이터 크기가 작으면 성능 향상은 상당히 개선되지만, 데이터 크기가 증가하면 다른 방식에 비해 성능 차이는 크게 개선되지 않는다. 이러한 이유는 데이터 복사에 요구되는 시간이 상대적으로 CORBA의 데이터 형식 표현을 변경하는 마셜링/디마셜링 및 객체중개자의 처리 시간에 비하여 많기 때문이다.



(그림 8) 데이터 크기에 따른 처리율 비교

본 논문의 성능 분석 결과를 통해, 공유 메모리 기반의 연동 프로토콜은 시스템의 주 기억장치 메모리가 충분하고, 가능한 작은 크기의 데이터를 빈번하게 처리하는 응용에 적합하다. 또한, 수십 개의 클라이언트가 동시에 하나의 서버에 접근하는 응용에 적용하기 위하여 보다 향상된 공유 메모리 처리 기법이 요구된다.

5. 결 론

본 논문에서 제시한 공유 메모리 기반 연동프로토콜은 기존 CORBA 제품의 성능으로는 만족할 수 없는 통신 시스템용 소프트웨어 개발에 적용하기 위한 고성능 CORBA 플랫폼을 개발하기 위하여 설계되었다. 본 논문에서 제시한 모델은 다양한 데이터 타입에 대한 시험 및 성능을 분석한 결과 기존의 TCP/IP 프로토콜 기반의 인터넷 연동 프로토콜에 비하여 약 2배 이상의 성능 향상을 나타내었다. 통신 시스템용 소프트웨어 개발을 시그널 기반에서 CORBA 표준 인터페이스를 기반으로 하는 새로운 패러다임의 통신 시

스템용 소프트웨어 개발 방법을 제시할 수 있게 되었다.

결론적으로, 제안된 모델 기반의 CORBA 플랫폼은 표준 인터페이스를 이용한 시스템의 설계를 통해 확장성 및 재사용성의 향상된 시스템을 개발함으로써 시스템의 경쟁력을 확보할 수 있을 것으로 기대된다.

향후 내장형 시스템에서 실행되는 통신용 소프트웨어에 적용하기 위해서는 메모리 사용의 최적화, 데이터 표현 형식의 최적화를 통한 성능 향상과 QoS 지원, 통지 서비스 등의 다양한 서비스 개발 및 시스템의 안정화 관련 연구를 수행하여야 한다.

참 고 문 헌

[1] OMG, The Common Object Broker : Architecture and Specification, Revision 2.3, OMG Document Formal/98-12-01, June, 1999.

[2] Imran Ahmad, Shikharesh Majumdar, "Achieving High Performance in CORBA based Systems with Limited Heterogeneity," Object-Oriented Real-Time Distributed Computing (ISORC-2001. Proceedings), 2001.

[3] Aniruddha S. Gokhale and Douglas C. Schmidt, "Optimizing a CORBA Internet Inter-ORB Protocol (IIOP) Engine for Minimal Footprint Embedded Multimedia Systems," IEEE Journal on Selected Areas in Communications, Vol.17, No.9, Sep., 1999.

[4] Puder and K. Romer, MICO-User and Reference Manual, Technical Report TR-98-031, International Computer Science Institute, Berkeley.

[5] Sai-Lai Ro, David Riddoch and Duncan Grisby, "The omniORB version 3.0 User's Guide, "AT&T Laboratories Cambridge," May, 2000.

[6] GNOME/ORBit, http:/orbit-resouce.sourceforge.net.

[7] Petr Tůma, Adam Buble, "Technical Report on Open CORBA Benchmarking," http://nenya.ms.mff.cuni.cz/~bench/package/Benchmarking-Techreport-200101.pdf.

장 종 현

e-mail : jangjh@etri.re.kr

1988년 경북대학교 전자공학과(공학사)

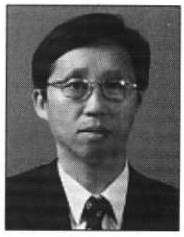
2000년 충남대학교 전자공학과(공학석사)

현재 한국외국어대학교 전자공학과 박사과정

1988년~1994년 대우통신(주) 종합연구소

1994년~현재 한국전자통신연구원 교환 전송기술연구소 선임연구원

관심분야 : 네트워크 보안, 이동통신, 미들웨어 등



이 동 길

e-mail : dglee@etri.re.kr

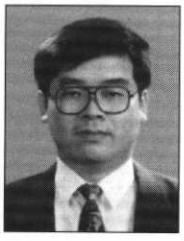
1983년 경북대학교 전자공학과(공학사)

1985년 한국과학기술원 전산학석사

1994년 한국과학기술원 전산학박사

1985년~현재 한국전자통신연구원 책임연구원

관심분야 : 컴파일러 구성론, 프로그래밍 언어론, 미들웨어 등



최 완

e-mail : wchoi@etri.re.kr

1981년 경북대학교 전자공학과(전자계산전공)(공학사)

1983년 KAIST 전산학과(공학석사)

1983년~1985년 KAIST 전산학과 조교

1988년 정보처리(전자계산기조직응용) 기술사

2000년 SPICE 심사원

1985년~현재 한국전자통신연구원 책연, 팀장

관심분야 : 실시간 시스템 OS/DBMS/Middleware, 통신프로토콜, S/W 공학

한 치 문

e-mail : cmhan@hufs.ac.kr

1977년 경북대학교 전자공학과(공학사)

1983년 연세대학교 대학원 전자공학과 (공학석사)

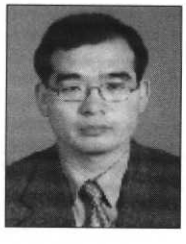
1990년 The University of Tokyo(동경대) 전자정보공학과(공학박사)

1977년~1983년 한국과학기술연구원(KIST) 연구원

1983년~1997년 한국전자통신연구원(ETRI) 선연, 책연 교환기술연구단 계통연구부장 역임

1997년~현재 한국외국어대학교 전자공학과 교수

관심분야 : ATM 통신망 및 교환, IMT-2000 네트워크, 네트워크 보안 등



장 익 현

e-mail : ihjang@dongguk.ac.kr

1984년 서울대학교 계산통계학과(이학사)

1986년 한국과학기술원 전산학석사

1998년 한국과학기술원 전산학박사

1986년~1999년 (주)데이콤 책연

1999년~현재 동국대학교 정보통신공학과 조교수

관심분야 : 컴퓨터통신, 컴퓨터네트워크, 분산 시스템, 병렬처리 시스템 등