CORBA 기반의 통신시스템용 미들웨어 설계와 구현

[0]조인성*, 장종현**, 한치문*
한국외국어대학교 전자정보공학부*, 한국전자통신연구원**
([0]ischo, cmhan)@hufs.ac.kr, jangjh@etri.re.kr

Design and Implementation of CORBA based middleware for Communication Networking System

[0]In-Sung Cho*, Jong-Hyun Jang**, ChiMoon Han*
Department of Electronics and Information Engineering*: HUFS
Electronics and Telecommunications Research Institute**

요 약

본 논문에서는 효율적인 통신 시스템용 미들웨어 개발을 위한 3가지 방법을 나타낸다. 첫째, 동일 호스트상에서 메시지 전달 오버헤드를 최적화하기 위해 공유 메모리 기반의 연동 프로토콜을 제시한다. 둘째, 미들웨어 플랫폼의 신뢰성을 향상시키기 위해, 구현 객체 데이터베이스를 이용한 네이밍 서비스 서버 구현 방법을 제시한다. 셋째, 시스템이 운용중인 상태에서 새로운 서비스 모듈의 추가나 기존 모듈의 기능 변경이 필요한 경우 서비스를 중단하지 않고 컴퍼넌트를 실행 중에 동적으로 재구성 하는 방안을 제시한다. 이를 통해 통신 시스템용 미들웨어의 성능이 향상됨을 나타냈다.

1. 서 론

오늘날 인터넷의 급격한 발전과 이용자의 증가로 인해 네트워크의 구조는 새로운 서비스 요구에 대응 할 수 있는 형태로 발전해 나가고 있다. 이를 위해 이기종간의 상호연동을 통해 새로운 서비스를 제공할 수 있는 미들웨어 기반의 네트워크 구조를 지향하고 있다. 통신시스템용 미들웨어 코바는 실시간 처리, 고신뢰성, 고성능의 특성화된 기능을 요구한다.

본 논문에서는 이러한 기능을 만족시키기 위해 공유 메모리 기반 연동 프로토콜, 고신뢰성 네이밍 서비스와 동적 재구성 기능을 추가하여 효율적인 미들웨어 플랫폼 구현에 목적이 있다.

2. 공유 메모리 기반 인터넷 연동 프로토콜 설계

인터넷 연동 프로토콜(IIOP)은 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 IIOP 프로토콜은 그림1과 같이 시스템 외부와 통신을 하기 위해서 TCP/IP를 기반으로 하고, 시스템 내부의 객체간 통신을 위해서는 유닉스 도메인 소켓을 이용한다.

그러나 통신 시스템용 미들웨어 플랫폼의 실시간, 고성능의 요구사항을 만족시키기 위해 시스템 내부 객체간 통신을 위한 공유 메모리 기반의 IIOP 프로토콜 구현이 필수적이다. 즉, 동일 호스트상에서 공유 메모리를 사용하여 GIOP 메시지를 전달하게 되면 메시지 전송 오버헤드를 줄일 수 있다. 따라서 미들웨어 플랫폼의 성능을 향상시킬 수 있는 공유 메모리 기반의 인터넷 연동 프로토콜을 이용한 최적화 방법을 도출 할 수 있다.

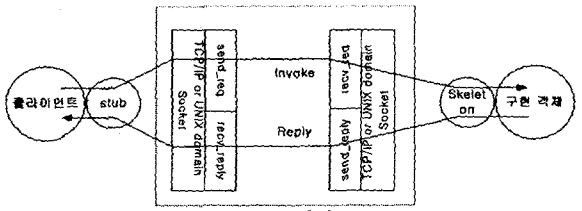

그림 1. TCP/IP 기반 IIOP 구조

공유 메모리를 이용하여 메시지를 전달하기 위한 자료구조는 다음과 같고 클라이언트의 식별, 메시지 정보, 방향 서비스 요청과 응답 여부를 구분한다.

```
typedef struct msgSlot {
  sem_t  mutex;          /*read signal for client */
  sem_t  oneway_mutex;/*read oneway signal for client*/
  int    clientId; char  data[MSGSIZE];
  int    response_expected;} MsgSlot;
```

또한, 서버측에서 메시지 통신을 위해 생성되는 공유 메모리 내의 자료구조는 다음과 같다.

```
struct shmstruct{/*struct stored in shared memory*/
sem_t  mutexforAllocate; /*semaphore for message
                          slot allocation*/
sem_t  mutexforQueue; /*semaphore for queue */
sem_t  mutexforEmpty;
int  skeepalive;  /* Server Alive Check*/
int  svr_reason;  /* Server Dead Cause*/
int  nConnect;    /* number of current onnection*/
MsgSlot msg[MAX_CONN];/*the actual messages*/};
```

공유 메모리 기반의 인터넷 연동 프로토콜의 동작 구조는 그림2와 같고 다음과 같이 동작한다.

1) 서버측 연동 프로토콜에서 공유 메모리 영역을 할당하고 필요한 자료 구조에 대해 초기화 작업을 수행한 후, 클라이언트로부터 메시지 수신을 위해 mutexforQueue 세마포어를 검사한다.

2) 클라이언트는 통신을 위해 고유의 슬롯을 할당 받고, 주어진 공유 메모리상의 데이터 영역에 메시지를 복사한 후 mutexforQueue 세마포어 카운터를 증가시킨다.

3) 서버측 프로토콜은 mutexforQueue 세마포어 값이 증가되어 있으면 데이터 영역으로부터 해당 메시지를 읽어 객체중개자로 전달하고, mutexforQueue 세마포어카운터를 증가시킨다.

4) 서버측 연동 프로토콜은 응답이 있는 경우 응답 메시지를 해당 클라이언트 슬롯의 데이터 영역에 복사하고, 해당 슬롯에 대한 세마포어 값을 증가시킨다.

5) 클라이언트측 연동 프로토콜은 자신의 데이터에 대한 세마포어가 증가되면, 해당 메시지를 읽어 객체 중개자로 전달한다.

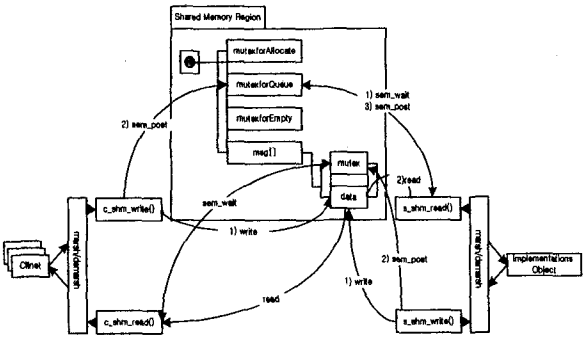

그림 2. 공유메모리 기반 IIOP 구조도

공유 메모리 기반 연동 프로토콜 구현에서 발생할 수 있는 문제점은 다음과 같다. 첫째, 서버측 연동 프로토콜 이 공유 메모리 영역을 생성할 때 까지 클라이언트의 공유 메모리 접근을 방지하여야 한다. 둘째, 서버측 연동 프로토콜이 응답을 반환할 때까지 새로운 서비스를 시작하지 않아야 한다. 셋째, 객체 중개자가 구현객체를 수행하고 있는 도중에 실행 중지가 발생하게 되면, 공유 메모리에 구현 객체의 비활성화 정보를 설정하여 세마포어의 오류를 방지하여야 한다.

이와 같은 문제점을 해결하기 위해 서버측 연동 프로토콜은 구현객체가 실행되는 동안 사용자에 의한 실행 중지를 방지하여야 하고, 프로그램 오류에 의해 중지가 발생되는 경우는 세마포어 정보를 초기화하여 오류를 방지하여야 한다.

3. 고신뢰성 네이밍 서비스 설계

네이밍 서비스는 이름을 이용하여 코바 객체가 다른 객체를 찾을 수 있도록 지원하는 서비스로 하나의 객체에 여러 개의 논리적인 이름을 연관시킬 수 있도록 해 준다. 먼저 이름과 객체를 연결짓는 것을 바인딩이라 하는데, 어떤 객체를 바인딩하면 바인딩 세트를 저장하는 컨테이너 객체인 네이밍 컨텍스트내에 이름과 객체에 대한 정보를 그래프 형태로 구성하게 된다. 클라이언트가 이름을 매개변수로 해석을 요구하면 네이밍 서버는 이 그래프를 탐색함으로써 복합 이름(compound name)을 이용하는 특정 객체를 찾아 객체 정보를 반환하게 된다[1].

다중의 클라이언트가 하나의 제어 보드에 의해 동작하는 구현 객체로 접근할 경우 네이밍 서비스를 이용한다. 이때 일반적으로 네이밍 서비스가 동작하는 보드의 하드웨어 불안 또는 사용자에 의해 재시동하는 경우 서버 객체를 재 실행하여야 서비스를 요청할 수 있다.

본 논문에서는 서버 객체를 재 실행하지 않고 단지 네이밍 서버를 이전의 바인딩 구현 객체 정보를 이용하여 네이밍 그래프를 재구성할 수 있도록 구성하여, 시스템의 신뢰성을 향상시키는 방법을 적용한다.

따라서, 설계된 네이밍 서버에서 구현 객체가 바인딩을 요청하면 네이밍 그래프를 생성하면서 그림3와 같은 절차로 다음과 같은 스트링화된 네이밍 정보 데이터 베이스를 구축한다.

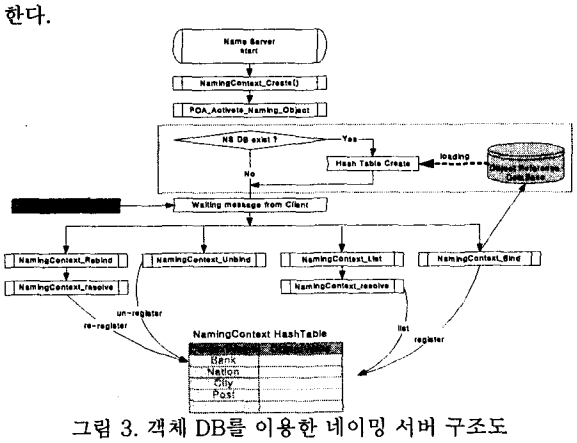

그림 3. 객체 DB를 이용한 네이밍 서버 구조도

여기서 네이밍 서버를 재실행하면 네이밍 정보 데이터베이스의 존재를 확인한 후, 스트링화된 정보를*CORBA _string _to_ object*() 함수를 이용하여 구현객체 정보를 생성하고 *CORBA_NamingContext_bind*()함수를 이용하여 바인딩을 요청한다.

4. 동적 재구성

미들웨어 기반의 통신시스템에 구현된 서비스 객체의 기능향상과 기능변경 시 서비스를 중단하지 않고 제공하기 위하여 고장 감내(Fault-Tolerant)기능이 필요하다.

동적 객체 관리기가 현재 서비스중인 서버의 객체 상태 정보를 모니터링하여 구현 객체에 대한 기능 변경(향상, 추가, 삭제)과 같은 재구성이 필요한 경우 관리기가 객체 상태정보를 변경하면 서버 응용 프로그램은 객체 관리기로부터 전달된 상태변경정보에 따라 그림4와 같이 동적으로 객체를 재구성하게 된다.

코바 기반의 통신시스템에서의 동적 객체 재구성 기능을 구현하기 위한 개발 환경은 클라이언트로부터 코바 객체 중개자(ORB)를 통하여 인터넷 연동 프로토콜(IIOP)을 통하여 전달된 GIOP 형식의 메시지를 서버 시스템에서 분석하여 해당 구현 객체를 활성화 한 후 서비스를 실행하여 그 결과를 반환 받게 된다.

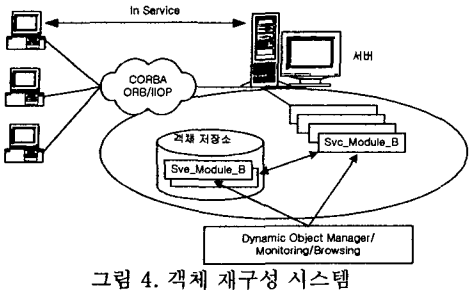

그림 4. 객체 재구성 시스템

구현한 동적 객체 재구성 모듈은 표1과 같이 크게 3가지로 구분할 수 있으며 각 모듈의 기능은 아래와 같다.

표 1. 동적 객체 재구성 모듈

구 분	지원 함수
Loading module	Add_module(), Module_open(), Module_symbol(), Module_state()
Removing Module	Del_module(), Module_open(), Module_symbol(), Module_state()
Update Module	Update_module(), module_open(), Module_symbol(), Module_state()

로딩모듈은 Add_module 함수에 의해 지원되며, 매개 변수로 임의의 개수를 포함할 수 있으며, 한번에 하나의 모듈을 로딩할 수 있다. 모듈을 로딩할 때 동일한 모듈의 로딩을 검사하여 이미 로딩되어 있으면 오류메세지를 반환한다. 제거모듈은 Del_module 함수에 의해 지원되며, 제거할 모듈이 현재 사용중인 경우(In-Service)에는 상호 배제(Mutual Excl usion)기능을 이용하여 사용이 완료되었을 때 제거하게 되며, 이때 구현객체의 상태(Out-of-Service)를 변경하여 서비스 요구가 있을 경우 해당 객체가 존재하지 않음을 통지하여야 한다. 갱신모듈은 갱신할 구현객체의 상태를 변경한게 되며 서비스 요구를 중단시킨 후 재구성 객체를 갱신하고, 상태 정보를 사용중(In-Service)로 변경하여 서비스를 재개할 수 있게 한다.

5. 미들웨어 성능 분석

성능 분석은 널(null) 함수로 작성된 서버 프로그램에 대하여 OMG 규격에서 정의한 데이터 타입을 이용하였다. 시스템 구성은 서버와 4개의 클라이언트로 구성하였고, 4개의 클라이언트에서 동시에 서비스를 요청한 후, 결과를 반환 받는 형태로 미들웨어 처리율을 비교하였다.

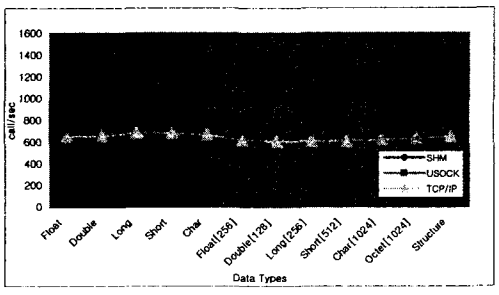

그림 5. 데이터 타입에 따른 처리율 비교

첫번째 성능 측정은 클라이언트에서 다양한 형태의 데이터 타입에 대한 서비스를 1000회 요청하는 시험을 수행하였다. 그림5에서 보면, 공유 메모리 기반의 연동 프로토콜을 이용하는 경우 유닉스 도메인 소켓에 비하여 약 30%, TCP/IP 기반의 인터넷 연동 프로토콜에 비하여 2배의 처리율이 향상됨을 알 수 있다.

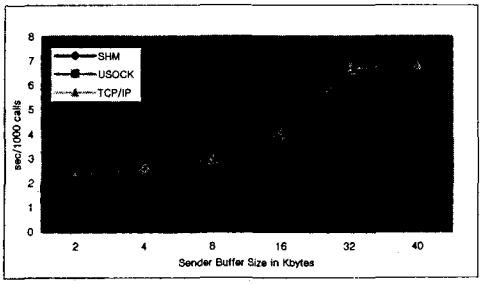

그림 6. 데이터 크기에 따른 처리율 비교

두 번째 성능 분석은 전달되는 문자 타입의 데이터 크기를 변화시키고, 서버측 구현코드에서 전달된 데이터 크기를 비교한 후 결과값을 반환하는 실험을 수행하여 처리율을 비교하였다. 공유메모리 기반의 연동 프로토콜은 데이터 크기가 작으면 성능이 상당히 개선되지만, 데이터 크기가 증가하게 되면 다른 방식에 비해 성능이 크게 개선되지는 않았다. 이러한 이유는 데이터 복사에 요구되는 시간이 미들웨어의 데이터 형식 표현을 변경하는 마샬링/언마샬링 및 객체 중개자의 처리 시간에 비하여 많기 때문이다.

6. 결론

본 논문의 성능 분석 결과를 통해, 공유 메모리 기반의 연동 프로토콜은 시스템의 주기억장치 메모리가 충분하고, 작은 크기의 데이터를 빈번하게 처리하는 응용에 적합함을 알 수 있다. 또한 시스템의 신뢰성을 향상시키기 위해 서버 객체를 재실행하지 않고 네이밍 서버를 이전의 바인딩 정보를 이용하여 네이밍 그래프를 재구성할 수 있도록 하였고, 서비스의 중단 없이 모듈을 동적으로 재구성 할 수 있는 방안을 적용하였다.

향후 uniORB를 내장형 시스템에서 실행되는 통신용 소프트웨어에 적용하기 위해서는 메모리 사용의 최적화, 시스템의 안정화 관련 연구를 수행하여야 한다.

참고 문헌

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

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

[3] Imran Ahmad, Shikharesh Majumdar, Archiving High Performance in CORBA based Systems with limited heterogeneity, 1999.