통신시스템을 위한 공유메모리 기반 ORB 연동 프로토콜의 설계

Design of Shared Memory-based Inter-ORB Protocol for Communication Systems

정익선*, 조영식**, Ilhyeon Jang(Ihjang@dongguk.ac.kr), Youngsuk Cho(ycho@dongguk.ac.kr)**

요약

통신시스템 소프트웨어는 대단히 크고 복잡하기 때문에, 소프트웨어 재사용성, 하드웨어 투명성 등을 기반으로 구축할 수 있는 구조를 요구하고 있다. 이런 요구사항을 만족시키기 위하여, 기존 CORBA IIOP의 성능과 통신방식에 대한 분석을 통해 통신시스템에 적합한 공유메모리 기반의 CORBA 연동 프로토콜을 설계하였다. 실제된 프로토콜은 동일한 인터페이스를 지원하며 통일 시스템 관점에서의 비상이 유사 오버헤드를 최소화시킨다. 다른 프로토콜과 비교해보면 새로운 프로토콜은 약 5% - 28%의 성능차이를 보여주고 있다. 따라서 본 논문에 제시된 프로토콜은 통신시스템을 위한 CORBA 기반의 연동 프로토콜 소프트웨어 개발을 위해 사용될 수 있을 것이다.

Abstract

Since communication systems software is very large and complex, it requires component based architecture for software reusability, hardware transparency, high performance, and easy software reconstruction in different applications. In order to meet these requirements, we analyze performance and inter-process communication techniques of existing CORBA IIOP, and designed a shared memory-based CORBA inter-ORB protocol that would best fit for communication systems software. The designed protocol supports the same interface and can minimize the message transfer overhead in the same host environment. The test results of our protocol compared with other protocols show that the performance is increased by about 15% - 200%. We are thus assured that our protocol can be used in developing CORBA-based component software architecture for communication systems.

** keyword : CORBA, IIOP, Shared Memory

1. Introduction

Due to rapidly developing internet services and growing number of service users, today’s network structure must be able to respond immediately to new service demands. Consequently, network system architecture is becoming middleware based open architecture that can provide new services through inter-networking among heterogeneous equipments by using standard interfaces.

The OMG (Object Management Group) defined the standard interface specifications of CORBA which
allow clients to provide software services for distributed components regardless of component locations, programming languages, OS platforms, communication protocols, and hardware[1]. ESTABLISHED IN 1981, OMG now has more than 500 members, and CORBA extended its application scope from ordinary applications to real-time systems and built-in systems[2].

As the productivity and reusability of software have become more important, CORBA-based software supporting distributed processes between OS and applications software has been researched and developed in the field of communication systems software, too.

In the area of communication systems software, there are two reasons for increased requirements for CORBA. First, there is a requirement for the aspects of internal structure of communication systems. The communication systems software is not only very large and complex but also depends heavily on the hardware of target system. In order to enhance its productivity and quality, component architecture is required for software reuse and the transparency of hardware architecture is needed for an easy reconstruction of software for different applications. Although CORBA can be used as a software bus to meet these requirements, it is impractical to use conventional CORBA as a software bus for communication systems because of its low performance.

Second, there is a requirement for open architecture of communication systems. As communication systems software architecture has been evolving into open architecture based on standard interfaces, reducing the time required for software development has become a key factor to system competitiveness.

The signal-based communication systems software has an inexcusable structural weakness by depending on OS, programming language and message transfer protocol for its development because it approaches distributed objects through inter-process communication supported by the operating system. In contrast, CORBA-based development methodology is independent of OS and programming languages. Therefore, CORBA-based development methodology ensures flexibility and scalability in software development. Moreover, when the CORBA-based software development methodology is applied to the development of software for communication systems, it is possible to reduce the time for software development, to flexibly configure systems, and as a result, to improve efficiency in the system maintenance. Also, CORBA can be adapted to a new trend of an open architecture with a paradigm shift of the methodology to the signal-based communication systems software development to the CORBA-based software development.

In this paper, we focus on the first reason that is a requirement for the aspects of the internal structure of communication systems. In order to design a high-speed CORBA, we propose a shared memory-based inter-ORB protocol for distributed processing between software blocks running on the same host, which will ensure the development of a high performance CORBA platform for communication systems.

This paper is organized as follows: Chapter 2 analyzes the specifications and internal structure of CORBA. Chapter 3 describes the design of inter-ORB protocol based on shared memory; Chapter 4 analyzes the performance of the proposed model. Finally, we discuss concluding remarks in chapter 5.

II. CORBA Specification and Internal Overview

CORBA specifications define interoperability protocol to support inter-networking among object request
CORBA specifications define the following four interfaces for systems development[1][2]:

1) IDL (Interface Definition Language)
   Interfaces for implementation objects show different mappings depending on the programming language.

2) ORB (Object Request Broker)
   ORB is a software system that allows clients and servers to communicate without knowledge of each other's location.

3) POA (Portable Object Adaptor)
   POA is a software system that provides a framework for implementing CORBA objects.

4) IOP (Interoperability ORB Protocol)
   IOP is a protocol that allows ORBs to communicate with each other.

Fig. 1. Key components of CORBA reference model
2. IIOP Specifications

IIOP uses CDR (Common Data Representation) grammar to marshal the messages written by CORBA IDL. Then IIOP transfers the standard GICP formatted messages to the destination CORBA through TCP/IP. IIOP operations are defined in the OMG Specifications 2.0, and they have following basic functions[1][2].

1) Object location service

IIOP defines location service for objects in the client/server model. When a client transmits LocateRequest message for access to objects, the server responds with LocateReply message containing information on the relevant objects and lets the client know their location.

2) Connection management

The client configures and manages its connection according to the server address information extracted from the inter-operable object reference created by the server. As single-thread-based CORBA applies connection oriented transfer protocol to service requests and response messages, the sequence of response messages is guaranteed. In contrast, multi-thread-based CORBA shows asynchronous operations without guaranteeing the sequence of response messages. Disconnecting operation may be activated by both the client and the server. After receiving disconnect message, IIOP objects will activate disconnection and cancel allocation of allocate resources on the basis of connection identifiers.

3) Message format

Seven types of messages are defined as shown in Table 1. Every message includes 12-byte necessary message header information. Additional header information may be added depending on the types of messages.

<table>
<thead>
<tr>
<th>Types</th>
<th>Direction</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Client&gt;Server</td>
<td>2</td>
<td>Service request</td>
</tr>
<tr>
<td>Reply</td>
<td>Client&lt;Server</td>
<td>1</td>
<td>Service response</td>
</tr>
<tr>
<td>ConcurReq</td>
<td>Client&lt;Server</td>
<td>2</td>
<td>Service continuation request</td>
</tr>
<tr>
<td>LocateReq</td>
<td>Client&lt;Server</td>
<td>3</td>
<td>Object location request</td>
</tr>
<tr>
<td>LocateRes</td>
<td>Client&lt;Server</td>
<td>4</td>
<td>Object location response</td>
</tr>
<tr>
<td>CloseCon</td>
<td>Client&lt;Server</td>
<td>5</td>
<td>Connection closing</td>
</tr>
<tr>
<td>MessageErr</td>
<td>Client&lt;Server</td>
<td>6</td>
<td>Message error</td>
</tr>
</tbody>
</table>

4) Message transfer

Generally, IIOP supports TCP/IP-based transfer protocols. These protocols are often developed by applying high-speed transfer protocols like ATM depending on message characteristics, data size, or application fields.

3. Analysis of Methods for Improving CORBA Performance

IIOP based generic CORBA platform is applicable only to the network management like TMN since it does not meet the high performance requirements of communication systems. Consequently, various studies have been conducted to improve CORBA performance. The studies focus on the three following themes[3][4].

1) Optimization of (de)mashaling techniques for a data type conversion

A frequent memory copy during a data type conversion degrades CORBA performances. I/O vectors can be used to optimize CORBA performances. However, this method can not satisfy performance requirements for communication systems.
2) Data compression

It is possible to omit version information or 1 byte magic information from GOIP header. However, this violates CORBA specifications. Compressing data without the header leads to a large fluctuation in performance depending on data sizes and system loads.

3) Optimization of message transfer time

High-speed transfer networks like ATM can be used to optimize message transfer time which takes about 70% of the whole processing time. However, a shared memory in the same system, or mechanisms like message queues can be applied.

In the following sections, we will analyze the performances of the CORBA platform depending on its constituents and the transfer methods in order to find best ways to improve its performances.

4. Analysis of CORBA inter-ORB Protocol Performance

GOIP is used to exchange GOIP messages defined as a set of standard message format combined with common data expression types, in order to ensure the communication among several ORBs in the TCP/IP network. General CORBA communication model uses primarily TCP/IP or UNIX domain socket to ensure communications between two components, as shown in [Fig. 1].

![Fig. 2: CORBA communication model](image)

This paper will find optimizing elements by analyzing the performances of the basic CORBA platform. The test environment for this analysis comprises Solaris 2.7, UltraSparc 16/7MHz, and 128MB memory. The objective is to design a high-performance platform for communication systems.

Assuming that client-to-server CORBA service request maintains the packet transfer capability at a constant level, the CORBA performances can be decided by the attributes like the (de)marshaling, the object activation and the message transfer time.

| Table 2: Comparison of execution time by CORBA constituent (unit: ms) |
|-----------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
|                      | Traceh | Tracea | Trandal | Trandh | Ratio  |
| Chordle             | 0.1550 | 0.0067 | 0.0204 | 0.1628 | 94 %   |
| Long               | 0.3407 | 0.0078 | 0.0206 | 0.3512 | 95 %   |
| Fixed               | 0.3773 | 0.0086 | 0.0204 | 0.3464 | 88 %   |
| Dovole             | 0.5844 | 0.0077 | 0.0204 | 0.5829 | 91 %   |
| Sliding            | 0.5081 | 0.0065 | 0.0206 | 0.5080 | 90 %   |
| Sequence         | 0.3203 | 0.0064 | 0.0206 | 0.3203 | 90 %   |

We will present criteria for improving performances of the conventional models by providing performance analysis for a service which has three parameter types, in, out and inout, and returns values of the same type.

[Table 2] shows performance analysis results regarding various data types.

\[ \text{Transh} = \sum \text{Tranal} + \text{Trandh} + \text{Trandh(r)} + \text{Trandal(r)} \]  

(1)

Where, Transh is parameter marshalling time.

\[ \text{Trans} = \sum \text{Tras} + \text{Tras(r)} + \text{Trans(r)} \]  

(2)

Trans is the time spent transferring messages marshaled on the client side to the server side through transfer layer and returning the result values.
Therefore, Total (time needed to process one COFBA service) is:

$$\text{Total} = \Sigma (T_{\text{Req}} + T_{\text{Trans}} + T_{\text{Pca}} + T_{\text{Trans}})$$

(3)

TTptime used by ORB on the server side is the
time required to interpret requested services, to find
the relevant objects in the implementation object
repository and to operate services.

The ratio of Ttrans needed to transfer messages
through inter-ORB protocol showed a slight difference
depending on the data types,

$$\text{Ratio} = \frac{(T_{\text{trans}})}{(\text{Total time})} \times 100$$

Though there may be variations depending on the
data size of transferred parameters, more than 80% of
the whole time was spent transferring messages, as
shown in [6].

Table 3 shows the average time spent transferring
data of different sizes one thousand times by using a
shared memory, TCP/IP and UNIX domain socket, as
ways of communications in the model of client/server
in the same system.

<table>
<thead>
<tr>
<th>Method</th>
<th>1KB</th>
<th>4KB</th>
<th>8KB</th>
<th>32KB</th>
<th>64KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
<td>0.062</td>
<td>0.082</td>
<td>0.160</td>
<td>0.210</td>
<td>0.272</td>
</tr>
<tr>
<td>UNIX Domain Socket</td>
<td>0.030</td>
<td>0.070</td>
<td>0.160</td>
<td>0.230</td>
<td>0.320</td>
</tr>
<tr>
<td>Shared Memory</td>
<td>0.038</td>
<td>0.076</td>
<td>0.174</td>
<td>0.270</td>
<td>0.370</td>
</tr>
</tbody>
</table>

As shown in Table 3, the transfer capability of a
shared memory model remains almost the same
although the increase of data size is steep. In contrast,
the transfer time of TCP/IP and UNIX domain socket
increases rapidly as the data size increases.

III. Shared Memory-based Inter-ORB Protocol

To support the characteristics of the communication
system that the system functions are distributed in
distributed hardware, COFBA has to support
specialized functions like real-time processing, high
reliability and high performance. In general,
communications among separate function modules are
performed through inter-process inter-networking
protocols like sockets, or remote procedure calls. In
contrast, COFBA applications use socket-based IOP
as a standard.

Based on the performance analysis in chapter 2, we
will propose a shared memory-based IOP as a way of
minimizing message transfer time, and apply the
proposed protocol to the development of a
high-performance COFBA platform applicable to
communication systems. We use OPEN CORBA[6]
as a basic platform, which is one of GNOME projects
performed with the support of Fedora.

1. Architecture of Shared Memory-based IOP

Shared memory techniques used in UNIX-based
operating systems include System V and POSIX
model. System V shared memory techniques require
manual rebuilding in order to change the size of the
shared memory. In contrast, POSIX does not require
manual rebuilding. Therefore, we applied POSIX model
to implement the proposed protocol.

Following requirements must be met to implement a
shared memory-based IOP:

- A server side IOP must allocate a shared memory
  for inter-process communication.
- Relative addresses should be used to share a
  shared memory by clients.
- A multi-process synchronization must be provided.
Following is the data structure maintained inside the shared memory generated for communications on the server side.

```c
struct SmlICPMsg {
    struct {
        int msgId; /* message ID */
        time_t time; /* time stamp */
        struct msgHdr hdr; /* message header */
        short type; /* message type */
        char data[MSG_SZ]; /* message data */
    } msg;
    int reason; /* reason for failure */
    int state; /* state of communication */
};
```

The SmlICPMsg structure is maintained in the shared memory for communications between the server IOP and the client IOP. A semaphore used to prevent inter-client conflicts in allocating SmlICPMsgs, mutexForQueue and mutexForEmpty, are used to control the inserted messages between clients and servers so that other parties can read the messages.

A client inspects the value of reason to determine whether server objects are activated, recovers the allocated resources when the server shows an abnormal stop, and initializes the resources to solve synchronization problems.

Supporting a shared memory-based IOP model requires an expansion of IOR structures, as shown in Table 4. A new profile type for IOP_TAO_UNORD_SHARDED_MEMORY is defined and the path information of the shared memory created by a server is included.

Table 4. IOR structure of shared memory-based IOP model

<table>
<thead>
<tr>
<th>Object ID</th>
<th>Profile Count</th>
<th>Profile Type</th>
<th>Shared Memory Name Path</th>
<th>Object Key</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Analysis of State Transition and Operation Algorithms

Server IOP

CREATE_SHM

S1: Create Shared Memory
S2: Sync. Variable Initialization
S3: In Service
S4: Wait Service Request
S5: Process Request
S6: Initialization State

IN_SVC

From client

PROC_REQ

S1 (try.wait)

To client

S2

PROC_REPLY

S3

Fig. 4. IOP protocol state transition

As shown in [Fig. 4], a server side shifts from initial IDLE to CREATE_SHM to create a specific size of the shared memory. If a shared memory already exists, a server side shifts to IN_SVC by deleting the existing shared memory and creating a new shared memory. A server side shifts from IN_SVC, where semaphore variables for a multi-process synchronization of service requests from multi-client processes are initialized, to PROC_REQ where service requests from clients are waiting. When semaphore values, which confirm messages, are inserted from clients, the server side shifts first to PROC_REPLY to process relevant services and send responses, and next to IN_SVC to process new service requests after response messages are transmitted.

A client shifts to ATTACH_SHM to share the shared memory created on the server side from initial IDLE, and changes its state to IN_SVC in order to initialize synchronization variables. If message slots, defined in order to prevent conflicts among multi-clients, are provided to a client, the client shifts to SVC_REQ where messages of service requests can be created and transferred to the server side. If responses from the server are requested, the client shifts to REPLY_Wait, waits for responses from the server, processes responses, and changes the state to IN_SVC. However, if the request is a one-way service, the client repeats the transition to IN_SVC where services can be started again.

Following potential problems were found in the implementation of the shared memory-based inter-ORB protocol. First, clients should not be allowed to access the shared memory until the server side inter-ORB protocol creates a shared memory area. Second, new services must not be started until the server side inter-ORB protocol returns responses. Third, if the execution is interrupted while ORB performs the implementation objects, the information on the deactivation of the implementation objects must be transferred to the shared memory to prevent semaphore errors.

To solve these problems, we used a technique with which the server side inter-ORB protocol checks the value of inaccessible and initializes semaphore information. The value is provided by a client side when client programs are interrupted by a user or their services are interrupted by program errors while implementation objects are executed.
2.1 Server Side IICP Algorithm
1) ① A server side inter-ORB protocol allocates shared memory areas, creates data structures to exchange messages with clients, and initializes them.
2) ② Wait for service requests from clients.
3) ③ Read IICP messages in the shared memory buffer.
4) ④ Change semaphore variables for synchronization to prevent clients from making new service requests.
5) ⑤ Interpret IICP messages and process requested services.
6) ⑥-⑦ Create response messages, if necessary, and insert them into shared buffer. Change semaphore values so that clients can read them.
7) Repeat steps ⑥-⑦.

```c
// Server Side IICP
Declaration of variables
SharedMemoryBuffer {
  LockDMM MURB, 10MB
  sem(512) 5MB
}

SharedPtr "Ptr"

Initialization
  Ptr = MemoryMap(NULL, szEvent);
  Semaphore initialize (LockDMM)

Action
  1. Make Server Shared Memory Buffer Create
  2. While loop
  3. do Semaphore TryWrtLock
  4. Wait for Service Request from Client
  5. Read IICPMessage From SharedBuffer
  6. Semaphore PritisLock
  7. Encode IICPMessage
  8. Send IICPMessage
  9. Semaphore PostWrite
  10. If Reply_required then
  11. Send Reply IICPMessage to Client
  12. Semaphore PostLock
  13. End
```

2.2 Client Side IICP Algorithm
1) ① Client side IICP attaches shared memory created by the server side IICP to its process and creates data structures.
2) ② Allocate a channel for client processes to prevent inter-client conflicts.
3) ③ Create IICP messages for service requests.
4) ④ Acquire a semaphore to insert created messages into the data area of a shared memory buffer.
5) ⑤-⑥ Paste messages in the data area of the shared memory and change the semaphore counter.
6) ⑦ Disable the relevant channel and terminate services in case of a one-way service request.
7) ⑧ Wait for response messages from the server if responses are required.
8) ⑨ If a response message is received, process relevant messages.
9) ⑩ Disable the relevant channel when services are provided.

```c
// Client side IICP
Declaration of variables
SharedMemoryBuffer {
  LockDMM MURB, 10MB
  sem(512) 5MB
  semaphore initialize (LockDMM)
  semaphore reply_sended (lock)
}

SharedPtr "Ptr"

Initialization
  Ptr = MemoryMap(NULL, szEvent);
  semaphore initialize (LockDMM)

Action
  1. Make Client Shared Memory Buffer Create
  2. Channel Allocation For IICP Processes
  3. 4. Construct IICPMessage
  5. Semaphore PritisLock
  6. Write IICPMessage to SharedBuffer
  7. Semaphore postLock
  8. Semaphore postLock
  9. If message is not sent then
  10. Wait for ReplyMessage From Server
  11. Handle ReplyMessage
  12. End Channel Close
```
IV. Performance Analysis

In order to conduct a performance analysis, we use data types defined in the OMG specifications for server programs written by null function. The test system consists of a server and four clients. We compared COFREA throughput by requesting services to all four clients simultaneously and by receiving results from them. Since the conventional generic and commercial COFREA platforms do not support a shared memory-based inter-ORB protocol, the comparison of its performance was conducted on the same platform of Solaris 2.7, UltraSparc 137MHz and 152MB memory.

First, omniORB(3.0.1)[16] developed by Lucence Technology, and MICO(2.3.5)[17] developed in Germany for open releases were compared to ensure the reliability of platforms. In comparing the time spent to call functions one thousand times employing various data types of five clients[18], the performance of the COFREA based on our protocol (uniORB) achieved a 250% improvement over MICO implemented by C++, and a 15% improvement over omniORB for each data type, as shown in [Fig. 5].

![Fig. 5. Comparison of performance in multi-client](image)

The first test measured time spent on a client calls for one thousand times of services of various data types. The results by data types are shown in [Fig. 6]. This figure shows that the shared memory-based inter-ORB protocol improves its throughputs by 50% higher than the UNIX domain socket, and by 200% higher than the TCP/IP based protocol.

We can conclude that there is no distinction between marshalling/demarshalling time for all data types and the time required for the ORB to find and activate relevant objects in every method. However, our protocol does not incur any overhead by not copying data inside the kernel, as contrast with TCP/IP-based protocol.

The second test is aimed at comparing throughputs by data sizes. A client changes the data sizes of the character types to be transformed. The server side implementation codes compare the size of the data received and return its result values. [Fig. 7] shows the time spent on service requests/responses of one thousand times.

![Fig. 7. Comparison of throughout by data size](image)
Shared memory-based inter-ORB protocol shows more improved throughout when data size is small. The reason for this is when data size is large, the time required to transfer the data is relatively longer than the time required to process ORB and to marshal/ demarshal.

The performance analysis in this paper shows that the shared memory-based inter-ORB protocol is better fit for the application that has enough system memory and frequently processes small-sized data.

V. Conclusion

The shared memory-based inter-ORB protocol proposed in this paper is designed to develop a higher-performance CORBA for communication systems. The designed protocol supports the same interface and can minimize the message transfer overhead in the same host environment.

Through performance tests with various data types, we can see that our protocol achieved a maximum 258% improvement over MICO, and 15% improvement over omniORB. Our protocol allows to design component architecture of communication systems software with high performance.

To practically apply the CORBA platform based on our protocol to communications software, further research is required on the optimization of memory usage, on development of services like notification, and on the shared memory processing techniques to allow two of clients to access one server simultaneously.

장 앵 련(Keun Jang)
경희원
- 1964년 2월 : 서울대학교 계산동계학과 (이학사)
- 1968년 6월 : 한국과학기술원 전산학과 (공학석사)
- 1968년 6월 : 한국과학기술원 전산학과 (공학박사)
- 1968년 3월 ~ 1999년 2월 : (주)대이온 책임연구원
- 1999년 3월 ~ 현재 : 동국대학교 정보통신공학과 교수
  <관심분야> : 컴퓨터통신, 분산시스템, 인터넷응용

조 영 석(Youngsoo Cho)
경희원
- 1978년 : 사강대학교 총학과 (문학석사)
- 1988년 : Louisiana State University (정보학석사)
- 1984년 : Louisiana State University (정보학박사)
- 1980년 ~ 1984년 : 한국미디어(주), Systems Analyst
- 1980년 ~ 1984년 : Louisiana State University, Computer Analyst
- 1984년 ~ 현재 : 동국대학교 컴퓨터·멀티미디어학과 교수
  <관심분야> : 소프트웨어 개발용 소프트웨어 개발 방법론, 설계테마, 자료저작 방법론, 정보검색