A Study on the IDL Compiler using the Marshal Buffer Management

Dong Hyun Kim *

* Dept. of Computer Information, Suncheon Cheongam College.

Abstract

The development of distributed application in the standardized CORBA(Common Object Request Broker Architecture) environments reduces the developing time and maintaining cost of the systems. Because of these advantages, the development of application is being progressed in the several fields using the CORBA environments.

The programmers in the CORBA environments usually develop the application programs using the CORBA IDL(Interface Definition Language). The IDL files are compiled by IDL compiler and translated into the stubs and skeleton codes which are mapped onto particular target language. The stubs produced by IDL compilers processes the marshaling a data into message buffer. Before a stub can marshal a data into its message buffer, the stub must ensure that the buffer has at least enough free space to contain the encoded representation of the data. But, the stubs produced by typical IDL compilers check the amount of free buffer space before every atomic data is marshaled, and if necessary, expand the message buffer. These repeated tests are wasteful and incidence of overheads, especially if the marshal buffer space must be continually expanded. Thus, the performance of the application program may be poor.

In this paper, we suggest the way that the stub code is maintain the enough free space before marshaling the data into message buffer. This methods were analyzes the overall storage requirements of every message that will be exchanged between client and server. For these analysis, in the Front End of compiler has maintain the information that the storage requirements and alignment constraints for data types. Thus, stub code is optimized and the performance of application program is increased.

I. Introduction

An IDL compiler accepts the IDL files represe-
nting the services that object of server support to the clients and generates the stub and skeleton code that is mapped onto particular programming language. The stub codes generat-
ed by IDL compiler marshal data into message buffer and the buffer must have at least enough free space to contain the encoded representation of the data. But, the stubs produced by typical IDL compilers check the amount of free buffer space before every atomic data is marshaled, and if necessary, expand the message buffer. These repeated tests are wasteful and incidence of overheads.[1][[3][4][5][7][8]

In this paper, we propose and design the mechanism to solve the above problem. With this mechanism, stub code maintains the enough free space before marshalling the data into message buffer. Therefore the performance of the stub code is increased.

The second section of this paper is the overview of the CORBA and an IDL compiler and the third is specification to implement optimized IDL compiler.

II. The CORBA and An IDL Compiler

2.1 CORBA

The CORBA of OMG supports the interopera-bility between objects and transparency of request and reply in the distributed environment. In Fig 1., the client is the program that requests the method in the implementation object and wish to return the result value. The implemen-tation object is the program that implements the attributes and operations of the object that clients require.

The principle of client and implementation object is following. First, the client sends the service request to ORB. The ORB searches the implementation object to process the service request and direct it to implementation object. The corresponding implementation object serves the request using the operation and returns the result value through the ORB.

The CORBA consists of the followings.

◉ ORB

The ORB sends the request to implementation object and reply to client. Generally, the ORB supports the transparency of object location, object implementation, object execution state, and communication mechanism.

◉ IDL (Interface Definition Language)

The interface of object specifies the operation and types which is supported by object. In the CORBA, the interface of object is specified by IDL that is independent of programming language. The OMG IDL is not programming language, but declarative language. Thus, the implementation of object is implemented by different language.

◉ Client Stub and Skeleton

The client stub is the mechanism in which the client invokes the static request and send to object implementation. The skeleton is the mechanism that the static request is sent to object implementation. The client stub and skeleton is generated by IDL compiler using the OMG IDL interface definition.

◉ Dynamic Invocation

The CORBA system supports the generation of request through the dynamic invocation interface using the interface repository which stores interface information during the run-time.[1][2][3][7][9][11]

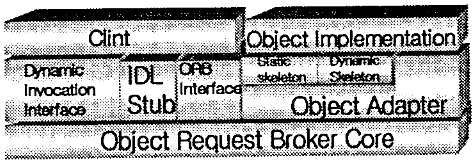


Fig 1. structure of CORBA

2.2 IDL Compiler

An IDL compiler is generally different from general-purpose compiler that generates the machine code. An IDL compiler is software that reads the IDL files representing the service that object of server supports to the clients and translates an IDL file into the stubs and skeleton codes which are mapped onto particular programming language.

Shown as Fig 2., an IDL compiler reads the IDL file as input and generates the client, object implementation, stub, and skeleton code as output which are mapped onto particular programming language. Finally, the programs executed in ORB were generated by particular programming language compiler with stub code, client program and skeleton code, and object implementation program. Especially, each of the stub and skeleton code can separately be generated with different programming language in CORBA. Thus, the previously implemented program is compatible with program implemented in other programming language.[3][5][6]

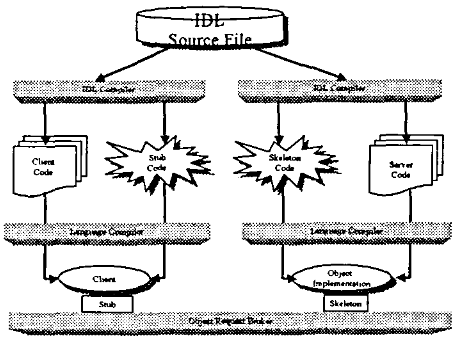
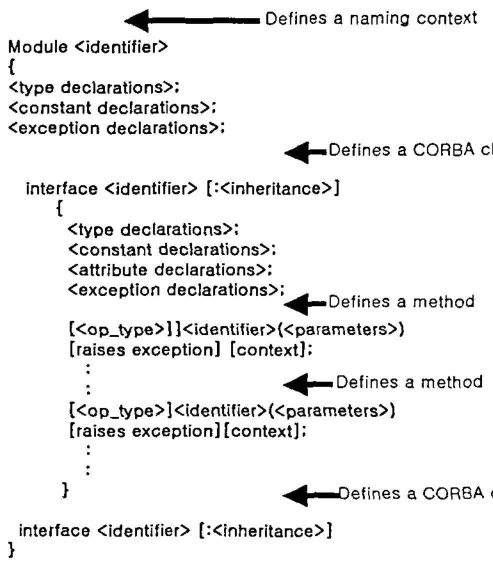


Fig 2. Role of IDL compiler

The Fig 3. shows the major component of CORBA IDL.



```
                         Defines a naming context
Module <identifier>
{
<type declarations>;
<constant declarations>;
<exception declarations>;
                                   Defines a CORBA class
  interface <identifier> [:<inheritance>]
     {
      <type declarations>;
      <constant declarations>;
      <attribute declarations>;
      <exception declarations>;
                                   Defines a method
      [<op_type>]]<identifier>(<parameters>)
      [raises exception] [context];
        :
        :                          Defines a method
      [<op_type>]<identifier>(<parameters>)
      [raises exception][context];
        :
        :
     }                             Defines a CORBA class
  interface <identifier> [:<inheritance>]
}
```

Fig 3. Structure of IDL file

An IDL specifies the ability, representation type, character related with interface of object. At this point of view, the CORBA interface is collection of operations, attributes, data types and can be inherited from other interfaces.[4][5][7][8][9]

The functions of components are the followings.

- module : offers name scope.
- interface : supports a multiple inheritance.
- operation : is an identifiable entity that denotes a service that can be requested.
- data type : basic types- short, long, unsigned long, unsigned short, float, double, char, boolean, octet, etc
  constructed types- enum, string, struct, array, union, sequence, any, etc

III. Method of increasing performance in the previous compiler and Propose

An IDL compiler is generate the two stub routines and other codes mapped onto C++ code. One is skeleton code that appears skeleton class for object implementation in the C++ mapping. Other is client stub that mapped onto proxy class. Two stub is contains routine for reciprocal action among objects implementation. Thus, the distributed application is produced using class generated by IDL compiler.

This paper is implemented using omniORB 2.7. An IDL compiler, omniidl is consists front end using CFE and back end take charge of C++ mapping. In the omniORB 2.7, the client is gain the reference about object for calling operation of object. The reference is gain by translate the stringified representation generated by identical ORB or the result of object operation. The ORB is generate the proxy local representation about the object in case of the object is remote object in the other address space. Side of clients, the proxy object is identifies with object implementation. When the client is request the operation to proxy object, the ORB is transport the invocation to remote object. The CORBA is not stipulate how to transport the request. But, The CORBA is constraints support the IIOP that is up-level of TCP/IP.

In Fig 4., shows the progress of process, when the client is request the operation in the remote address.

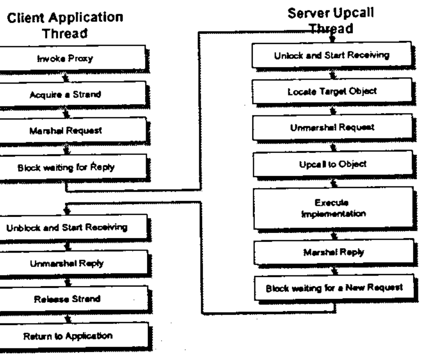


Fig 4. Progress of process to remote request

The client is essentially through the marshal data into message, when the client request the remote operation.

Before a stub can marshal a data into its message buffer, the stub must ensure that the buffer has at least enough free space to contain the encoded representation of the data. But, the stubs produced by typical IDL compilers check the amount of free buffer space before every atomic data is marshaled, and if necessary, expand the message buffer. These repeated tests are wasteful and incidence of overheads. Therefore, we have gain the performance of application by efficient buffer management. we suggest the way that support the marshaling buffer management through the following progress. First, we analyzes the overall storage requirements of every message that will be exchanged between client and server. For these analysis, in the Front End of compiler has maintain the information that need for the storage requirements and alignment constraints to every data types. The value of alignment is need for GIOP protocol, because of message is take relative value based on start address about every type. For example, the case of real type has take a size of 8 byte and alignment method is multiple of 8. Besides, the value is need for distinguish the fixed or variable include in each message. This is distinguished three types. First, the variable is fixed size. The second, variable and take maximum value. Final, not variable and not take maximum value. In this paper, considering the first case and optimized. Others are difficult to predict the size of marshaling buffer and the check of marshaling buffer size is need once.

In this paper, we add the routine that analyzes the overall storage requirements of every message at compile time and maintain the enough free space before marshaling the data into message buffer. Thus, we have with profit reduced the unnecessary check of marshaling buffer size or continuously expand the message buffer, then the performance of compiler is increased.

Ⅳ. Conclusion

The development of distributed application in the standardized CORBA(Common Object Requ-est Broker Architecture) environments reduces the developing time and maintaining cost of the systems. Because of these advantages, the deve-lopment of application is being progressed in the several fields using the CORBA environments.

The programmers in the CORBA environments usually develop the application programs using the CORBA

IDL(Interface Definition Language). The IDL files are compiled by IDL compiler and translated into the stubs and skeleton codes which are mapped onto particular target langua-
ge. The stubs produced by IDL compilers processes the marshaling a data into message buffer. Before a stub can marshal a data into its message buffer, the stub must ensure that the buffer has at least enough free space to contain the encoded representation of the data. But, the stubs produced by typical IDL compilers check the amount of free buffer space before every atomic data is marshaled, and if necessary, expand the message buffer. These repeated tests are wasteful and incidence of overheads, especi-
ally if the marshal buffer space must be contin-
ually expanded. Thus, the performance of the application program may be poor.

In this paper, we remove the routine that checks the required marshal buffer space for every atomic data with the traditional IDL compiler, add the routine that analyzes the overall storage requirements of every message at compile time, and then maintain the enough free space before marshaling the data into message buffer. Thus, stub code is optimized and the performance of application program is increased.

The study of future is to increase the perform-
ance of fixed-size and variable argument marsh-
alling.

[References]

[1]OBJECT MANAGEMENT GROUP. The CommonObject Request Broker: Architectureand Specification, 2.0 ed., July 1995.

[2]SRINIVASAN, R. RPC: Remote procedure call protocol specification version 2. Tech. Rep. RFC 1831, Sun Microsystems, Inc., Aug. 1995.

[3]Lee Jinho, Lee Gunyoung, Jeong Taemyung, "Design the IDL-to-Java Compiler for Java ORB System" KOREA INFORMATION SCI-
ENCE SOCIETY Journal Vol.5, No.8 1998.8

[4]SUN MICROSYSTEMS, INC. ONC+ Developer's Guide, Nov. 1995. SUNSOFT, INC. SunSoft Inter-ORB Engine, Release 1.1, June 1995. ftp://ftp.omg.org/pub/interop/iiop.tar.Z.

[5]SRINIVASAN, R. XDR: External data representation standard. Tech. Rep. RFC 1832, Sun Microsystems, Inc., Aug. 1995.

[6]NETBULA, LLC. PowerRPC, Version1.0 ,1996.

http://www.netbula.com/products/powerrpc/

[7]SUN MICROSYSTEMS, INC. 1992. ftp://ftp.omg.org/pub/OMG_IDL_CFE_1.3.

[8]O'MALLEY, S., PROEBSTING, T. A., AND MONTZ, A. B. USC: A universal stub compiler. In Proceedings of the Conference on Communications Archite ctures, Protocols and Applications (SIGCOMM) (London, UK, Aug. 1994), pp. 295--306.

[9]JANSSEN, B., AND SPREITZER, M. ILU 2.0 alpha Reference Manual. Xerox Corporation, May,1996. ftp://ftp.parc.xerox.com/pub/ilu/ilu.html

[10]A. Leinwand and K. F. Conroy, Network Man-agement., Addison-Wesley Publihing Company, Inc., pp. 17-36 ,1996

[11]I.Jacobson , Object-Oriented Software Engi-
neering, Addison-Wesley Publishing Com pa-
ny, Inc., 1992