DEVELOPING CORBA-BASED DISTRIBUTED SCIENTIFIC APPLICATIONS FROM LEGACY FORTRAN PROGRAMS

Janche Sang
Dept. of Computer and Info. Science
Cleveland State University
Cleveland, OH 44115
Phone: (216) 687-4780
Email: sang@cis.csuohio.edu

Chan Kim
Isaac Lopez
NASA Glenn Research Center
Cleveland, OH 44135
Phone: (216) 433-8715
Email: {Chan.M.Kim, Isaac.Lopez}@grc.nasa.gov

1 INTRODUCTION

Recent progress in distributed object technology has enabled software applications to be developed and deployed easily such that objects or components can work together across boundaries of the network, different operating systems, and different languages. A distributed object is not necessarily a complete application but rather a reusable, self-contained piece of software that cooperates with other objects in a plug-and-play fashion via a well-defined interface. The Common Object Request Broker Architecture (CORBA), a middleware standard defined by the Object Management Group (OMG)[9], uses the Interface Definition Language (IDL) to specify such an interface for a transparent communication between distributed objects. Since IDL can be mapped to any programming language, such as C++, Java, Smalltalk, etc., existing applications can be integrated into a new application and hence the task of code re-writing and software maintenance can be reduced.

In OMG's object model, an object is an encapsulated entity with a distinct immutable identity whose services can be accessed only through interfaces defined in IDL[16]. Clients issue the requests to objects to perform services on their behalf. The implementation and location of each object are hidden from the requesting client. Communication between clients and objects is provided by the Object Request Broker (ORB), a key component of CORBA architecture. Upon compiling IDL file, ORB generates the stub and the skeleton through which a client can invoke a method on a server object, which can be on the same machine or across a network. The ORB is responsible for finding an object that can implement the request, passing it the parameters, invoking its method, and returning the results to the client. In this process, the client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an objects interface.

Since its inception, CORBA has been widely accepted as the middleware standard for distributed object computing. It relieves distributed application developers of cumbersome task of dealing with issues due to the heterogeneous computing environments. It provides a standard-based interface to facilitate transparent exchange of management information for computer and communication network[6, 12]. TeleMed[5] is an effort to demonstrate the use of a multimedia electronic medical record over a wide area network. It is designed as a distributed object system in which the various healthcare components are dealt with as objects and distributed via the CORBA standard. CORBA-based distributed object technology is considered the key to integrating legacy applications in highly dynamic business environments [15]. Industry-specific stories on successful applications of CORBA are reported in OMG web site. OMG's Domain Technical Committee (DTC) aims at developing domain-specific CORBA services and technologies in various industries.

Many scientific applications in aerodynamics and solid mechanics are written in Fortran. Refitting these legacy Fortran codes with CORBA objects can increase the code reusability. For example, scientists could link their specific applications to an objectified vintage Fortran programs such as Partial Differential Equation (PDE) solvers in a plug-and-play fashion. Many standalone Fortran applications developed to analyze the performance of an individual component of the engineering system can be converted to CORBA objects and then be combined with other objects to design the entire system. The work in [8] attempts to provide a collaborative design and simulation environment based on this concept. A CORBA- based software environment is developed in [13] to couple two independently developed codes written in Fortran and C++ in modeling a thermomechanical problem. A computationally intensive Fortran application can also be decomposed into several pieces, made into CORBA objects, and distributed over several machines to speed up the computation. Unfortunately, CORBA IDL to Fortran mapping has not been proposed and there seems to be no direct method of generating CORBA objects from Fortran without having to resort to manually writing C/C++ wrappers.

In this paper, we present an efficient methodology to integrate applications written in Fortran into a distributed object framework. Issues and strategies regarding the conversion and decomposition of Fortran codes into CORBA objects are discussed. Our goal is to keep the Fortran codes unmodified as much as possible. To reduce the programming effort in code wrapping, we design and implement a conversion tool which takes the Fortran application program as input and generates C/C++ header file and IDL file. We also evaluate the performance of the client-server computing and identify possible overheads.

2 METHODOLOGY

One method of wrapping a legacy application is to encapsulate the entire legacy codes into a single object. Programmers only need to provide a server which invokes the wrapped legacy-code object when receiving a request from a client. This straightforward method is suitable for small-scale applications which only have one module or entity.

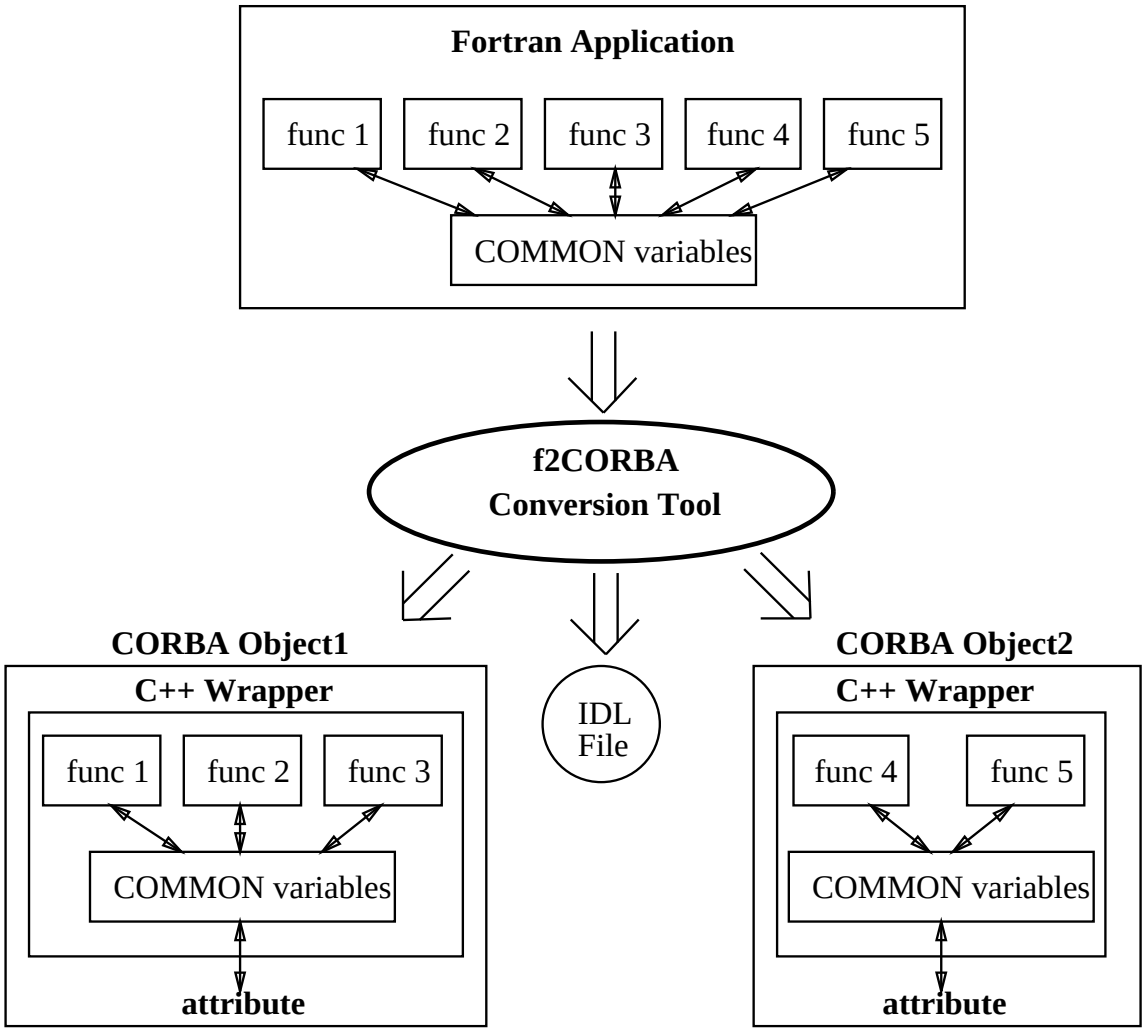

Figure 1: Functional Diagram of f2CORBA conversion tool

For a complicated applications, an alternative method which we are interested in is to decompose the codes into different function modules and wrap each module into a distributed object. This method can increase the code usability because each distributed object can be invoked by different applications as a plug-and-play software component. Figure 1 shows the conversion and decomposition mechanism we proposed. Our objective is to keep modification of the Fortran source codes to the minimum. The conversion tool takes the Fortran application program as input and helps programmers generate C/C++ header file and IDL file for wrapping the Fortran code.

In our current environment, programmers still need to determine by themselves how to decompose the legacy application into several reusable components based on the cohesion and coupling factors among the functions and subroutines. In the future, we plan to add an analyzer tool to help programmers extract objects from legacy codes. Earlier studies in object extraction can be found in [2] and in [10]. This topic is beyond the scope of the paper.

Most Fortran applications use the COMMON block to facilitate the transfer of large amount of variables among several functions. The COMMON blocks play the similar role of global variables used in C. In the CORBA-compliant programming environment, global variables can not be used to pass values between objects. One approach to dealing with such problem is to put the COMMON variables into the parameter list. This requires a lot of modification of the Fortran source code which violates our design consideration. Our approach is to extract the COMMON blocks and convert them into a structure-typed attribute in C++. Through the attributes, each component can initialize the variables and return the computation result back to the client. With our conversion tool, programming effort can be greatly reduced because function headings, types, and even the COMMON blocks have been converted to C++ and IDL styles.

3 IMPLEMENTATION ISSUES

The conversion tool we proposed in the previous section consists of a parser and a code generator. The parser constructs the parsing trees from the input Fortran codes and the generator translates the trees into the C++/IDL codes. Instead of writing a language translator from the beginning, we implemented the conversion tool based on an existing Fortran-to-C converter called f2c[4]. We chose the f2c package because it is open source and has been widely used in the academic areas.

The f2c program translates the Fortran 77 codes to the C codes. Since our goal is to wrap the Fortran codes and provide the interface for both the client and the server, we only need to use f2c to extract and translate the codes of data types, variable declarations, and function headings. The function bodies and statements are no interests to us. However, the codes converted by f2c do not totally meet the IDL syntactic requirements. For example, IDL requires a tag in the structure type, while C does not. Unfortunately, f2c translates a Fortran COMMON block into a structure variable in C without a tag. Furthermore, the structure tag can be used as a type to define structure variables in IDL. In C, it has to put the keyword `struct` in front of the tag and this kind of declaration is not allowed in IDL.

Because of the difference between C and IDL, programmers need to manually edit the codes from f2c. This inconveniency for programmers motivated us to modify the f2c program. We added a few codes in f2c to generate the tag for a structure. Figure 2 shows an example of Fortran codes with the declarations of the COMMON blocks `cgcon` and `disp`. The corresponding codes in IDL which are translated by our modified f2c program are shown in Figure 3.

Like the example shown in Figure 2, most Fortran applications have several COMMON blocks. After decomposing the application into a few CORBA objects, the problem of passing the structure variables (i.e COMMON blocks) among servers needs to be solved. Our current approach is to merge all of the structures into another structure (eg. `lu_tag` in Figure 3) and use an `attribute` (eg. `lu_all` in Figure 3) to facilitate data transfer.

The above approach is based on the assumption that each server needs to access all of the COMMON blocks. However, a certain server may access parts of them. Currently, we are adding a

```
   integer nx, ny, nz
   integer nx0, ny0, nz0
   ...
   double precision  dxi, deta, dzeta
   double precision  tx1, tx2, tx3
   double precision  ty1, ty2, ty3
   ...
   common/cgcon/ dxi, deta, dzeta,
  >              tx1, tx2, tx3,
  >              ty1, ty2, ty3, ...
  >              nx, ny, nz,
  >              nx0, ny0, nz0,
   ...
   double precision dx1, dx2, dx3, dx4, dx5
   double precision dy1, dy2, dy3, dy4, dy5
   ...
   common/disp/ dx1,dx2,dx3,dx4,dx5,
  >             dy1,dy2,dy3,dy4,dy5,
   ...
```

Figure 2: Original Fortran Codes with Common Block Variables

```
struct cgcon_tag {
    double dxi, deta, dzeta, tx1, tx2, tx3, ty1, ty2, ty3, ... ;
    integer nx, ny, nz, nx0, ny0, nz0, ... ;
} ;
struct disp_tag {
    double dx1, dx2, dx3, dx4, dx5, dy1, dy2, dy3, dy4, dy5, ... ;
} ;
...
struct lu_tag {
    cgcon_tag cgcon_;
    disp_tag disp_;
    ...
} ;
interface Lu1 {
   attribute lu_tag lu_all;
   void lu1_comp();
} ;
interface Lu2 {
   attribute lu_tag lu_all;
   void lu2_comp();
} ;
```

Figure 3: Converted Codes in IDL

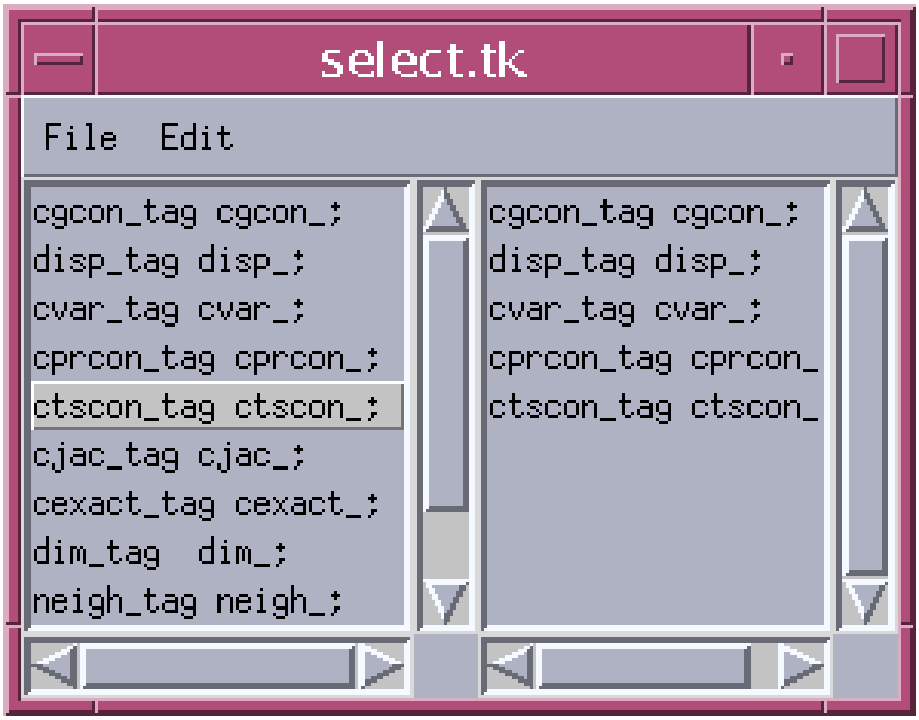

Figure 4: A Window Tool for Structure Variable Selection

GUI interface to ease the conversion task. Programmers can simply click on an item to select a structure variable from a list box to be a member in a structure-typed attribute (see Figure 4). To implement the GUI interface, we are using the tool Tcl/Tk [11] because of it's availability and portability. Tcl/Tk can be downloaded from the web and has been ported on most operating system platforms such as UNIX, NT, and Macintosh. Furthermore, most programmers can learn the fundamentals of Tcl/Tk and write Tcl/Tk script programs to do real work in a few days.

We have tested successfully the proposed conversion methodology on different CORBA packages: VisiBroker C++ [7] and MICO[1]. Because of availability and portability, we prefer using MICO rather than VisiBroker C++. For example, VisiBroker C++, a commerical software worth $2-3K, only works with Sparcworks C++ compiler on Sun Sparc/Ultra platforms. MICO, a public domain ORB with complete CORBA compliant implementation, relies on the GNU package and hence can be ported easily to any platforms such as Solaris, LINUX, Windows NT, etc.

4 PERFORMANCE MEASUREMENTS

In order to investigate the overhead produced in distributed object computing, we selected the LU and the BT benchmarks from the NAS Parallel Benchmarks (NPB) suite[3]. These benchmarks were devised by the Numerical Aerodynamics Simulation (NAS) program at NASA Ames Research Center. It has been widely used to study the performance of parallel computing. For

Benchmark LU	Traditional		1 Client - 2 Servers			
	Comp.	Total	Binding	Comp.	Comm.	Total
Sun Ultra	2.99	3.05	0.046	2.99	0.33	3.37
PC Linux	1.51	1.58	0.019	1.51	0.19	1.72

Table 1: Breakdown of Elapsed Time for running the Client-Server Benchmark LU

Benchmark BT	Traditional		1 Client - 2 Servers			
	Comp.	Total	Binding	Comp.	Comm.	Total
Sun Ultra	6.99	7.25	0.047	6.99	2.12	9.16
PC Linux	4.16	4.35	0.019	4.16	1.38	5.73

Table 2: Breakdown of Elapsed Time for running the Client-Server Benchmark BT

example, we used the benchmarks to evaluate the performance of a cluster of 32 Intel P6 workstations which are connected by a two-level tree-structure network[14]. The NPB 2.3 benchmarks are a set of eight problems which consists of five kernels which highlight specific areas of machine performance, and three pseudo-applications which simulate computational fluid dynamics(CFD). We briefly describe the LU and the BT benchmarks below.

- **Application LU** solves a finite difference discretization of the 3-D compressible Navier-Stokes equations by using a symmetric successive over-relaxation (SSOR) numerical scheme.
- **Application BT** is based on Beam-Warming approximate factorization which decouples the x , y, and z dimensions, resulting in three sets of narrow-banded, regularly structured systems of linear equations.

We used the sample-size serial version of the LU and the BT benchmarks (NPB2.3-serial) and decomposed each benchmark into two server objects. The client needs to contact these two servers one after the other to accomplish the task. The experiments were performed on a pair of Sun Ultra (170MHz, 128MB) running Solaris 2.6, and also on a pair of Intel P6 (400MHz, 512MB) running Linux 2.2.12. The client and server is connected through a 100BaseT LAN.

Table 1 and Table 2 show the breakdown of the elapsed time for running the benchmarks LU and BT, respectively. For the purpose of comparison, we also ran the original programs. The result shows that the time for service binding is small. However, the communication overheads, including the time for marshaling and unmarshaling data, between the client and the servers cannot be ignored.

5 CONCLUSION

Many scientific applications in aerodynamics and solid mechanics are written in Fortran. Refitting these legacy Fortran codes with CORBA objects can increase the code reusability. In this paper, we have presented a methodology to integrate Fortran legacy programs into a distributed object framework. Issues and strategies regarding the conversion and decomposition of Fortran codes into CORBA objects have been discussed. We also have implemented a conversion tool which takes the Fortran application program as input and generates C/C++ header file and IDL file. Tedious programming tasks for wrapping the codes can therefore be reduced. In the future, we plan to add more user-friendly GUI interfaces and to provide an analyzer tool to help programmers easily extract objects from legacy applications.

ACKNOWLEDGEMENTS

The authors would like to express their appreciation to management of the High Performance Computing and Communications Program and to the NASA R&T Base Program for supporting NPSS

REFERENCES

[1] A. Purder and K. Romer. MICO is CORBA. URL: http://www.mico.org.

[2] B. L. Achee and D. L. Carver. Creating Object-Oriented Designs From Legacy Fortran Code. *Journal of Systems and Software*, 39:179–194, 1997.

[3] D. Bailey, T. Harris, W. Saphir, and R. Wijngaart. The NAS Parallel Benchmarks 2.0. Technical report, NAS-95-020, NASA Ames Research Cneter, 1995.

[4] S. I. Feldman, D. M. Gay, M. W. Maimone, and N. L. Schryer. A Fortran-to-C Converter. Technical Report No. 149, Bell Laboratories, NJ, 1995.

[5] D. Forslund, J. George, E. Gavrilov, T. Staab, T. Weymouth, and S. Kotha. TeleMed: Development of a Java/CORBA-based Virtual Electronic Medical Record. In *Proceedings of the PacMedTek Symposium*, Aug. 1998.

[6] P. Haggerty and K. Seetharaman. The Benefits of CORBA-Based Network Management. *Communications of the ACM*, 41:73–79, Oct. 1998.

[7] Inprise, Corp. *VisiBroker for C++: Programmer's Guide, Version 3.3*, 1999.

[8] J. Lytle. The Numerical Propulsion System Simulation: A Multidisciplinary Design System for Aerospace Vehicles, NASA/TM-1999-209194. In *Proceedings of the 14th International Symposium on Air Breathing Engines sponsored by the International Society for Air Breathing Engines*, Sep. 1999.

[9] Object Management Group. *The Common Object Request Broker: Architecture and Specification, 2.3 ed.*, June 1999.

[10] C. Ong and T. Tsai. Class and Object Extraction from Imperative Code. *Journal of Object-Oriented Programming*, pages 58–68, Mar/April 1993.

[11] J. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Mass., 1994.

[12] J. Pavon, J. Tomas, Y. Bardout, and L. Hauw. CORBA for Network and Service Management in the TINA Framework. *IEEE Communications*, 36:72–79, March 1998.

[13] Sandia National Lab. Coupling Finite Element Codes Using CORBA-Based Environments. URL: http://www.cs.sandia.gov/HPCCIT/corba/impres.html.

[14] J. Sang, C. Kim, T. J. Kollar, and I. Lopez. High-Performance Cluster Computing over Gigabit/Fast Ethernet. *Informatica*, 23, 1999.

[15] Sun Microsystems, Inc. Distributed Object Technology In The Financial Services Industry, White Paper. URL: http://www.sun.com/swdevelopment/whitepapers/.

[16] S. Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications*, 35:46–55, Feb. 1997.