Proceedings of International Conference on Distributed
Systems, Software Engineering and Database Systems

Supporting an object-based API for DCE

Chu-Sing Yang, Shu-Chin Su Chen, C.C. Chen, and David Lin
Computer and Information Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, ROC
sschen@dcs.ccl.itri.org.tw

Abstract

In this paper, we investigate approaches to supporting friendly and efficient application programming interface for the Distributed Computing Environment(DCE) which is from Open Software Foundation. Application programming interface(API) provided by DCE is designed in C language, and the IDL file must implement in C language. Therefore ,if we want to develop object-oriented application by using DCE/API, we cannot utilize the orient-object facilities in our distributed application by using C style.

We take the advantages of object-oriented concept to reduce the difficult of application design. We encapsulate the function of API provided by DCE and provide an object-oriented interface for programmers to design distributed object-oriented programming. In addition, we also provide Stub Generator to analyze C++ API, and generate stub in C++ style. So the programmer can develop distributed application more conveniently in DCE by using C++ API and Stub Generator.

Keyword: Distributed Computing Environment, Interface definition Language, Application programming interface, and objected-oriented

1. Introduction

Information has been considered as an important asset by many of today's enterprises. Being able to affect enterprise-wide information and efficiently process the data is the key to enhance the competitive edge of the enterprise. As the personal computers, workstations and fast networks emerged, there is an increasing need to access data stored at different systems. Besides, as down sizing and right sizing are gaining more attention recently, next generation applications must require to retrieve data stored and processed by dozens or even hundreds of platforms connected by high performance computer networks.

In the Network, we need to have a powerful middleware to handle all the communication protocols, resources, data, and other among the computers. Open Software Foundation(OSF) had developed a middleware system called Distributed Computing Environment(DCE). A distributed computing environment enables users to take advantage of heterogeneous system resources without having to understand the specifies of how various systems and peripherals interoperate in networked systems.

Open Software Foundation(OSF) was founded in early 1980 which is a non-profit organization. The major mission was established and promoted the open system of client/server architecture. Due to the market need and technology trend, the OSF and X/Open had merged to form a new organization called Open Group in Feburary of 1996. The OSF/DCE had been developed in early 1986. At that time, the most of programming languages are procedure languages, such as Pascal, C, Fortran, etc. The C language is the most popular one. That's why C language was chosen for developing the DCE. DCE was composed by remote procedure call, security service, distributed time service, cell directory service, and distributed file system, and every service has its own application interface. Therefore, before users develop a program, they have to study all the application interface libraries first. Unfortunately, all the services of the DCE have huge and complicate libraries, and application developers need to use these library functions to develop their programs. So they have to spend lots of time to study the library functions before they start to do the developments. The result is the developing time will be very long.

Our approach is to provide a friendly C++ application interface on top of the DCE which provide an easy to use, debug, and maintain features, furthermore to reduce the developing time, as shown in Figure 1-1.

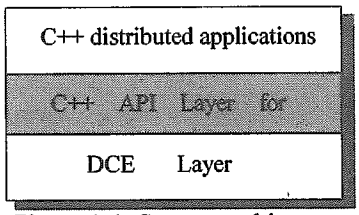

Figure 1-1: System architecture

The C++ application interface has two layers, one is the high level layer, the other one is low level layer. The low level layer calls the DCE library interface directly and the high level layer calls through the low level layer class only, as shown in figure 1-2. Developers can select all the functions by their demands.

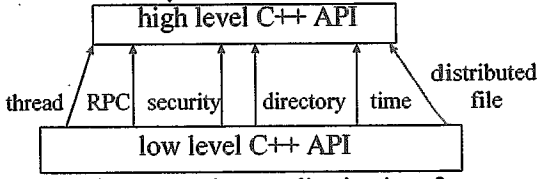

Figure 1-2: C++ application interface

2. Distributed Computing Environment

We selected Open Group OSF/DCE as implementation environment and built a C++ application interfaces on the top of DCE, which included stub generator, C++ application interfaces of RPC and Security Service.

Stub generator uses Lex and Yacc[5] to analyze C++ header file. C++ application interface of RPC and security service are provided by their encapsulated libraries [1,2,6,7,8,9,10,11].

2.1 DCE Architecture

OSF's Distributed Computing Environment provides a wide range of computer services to applications regardless of the location of the user, the application, or the required resources, as shown in Figure 2-1.

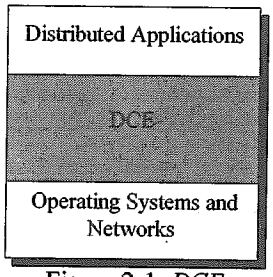

Figure 2-1: *DCE*

DCE consists of remote procedure call, security service, distributed service, cell directory service and distributed file system. DCE architecture is shown as Figure 2-2, and each component is described as follows[1]:

◆ RPC extends familiar local procedure calls, masking much of the underlying network complexity. It works consistently across DCE implementations on heterogeneous systems, and can support different network technologies.

◆ Security Service provides security-sensitive distributed computing.

-- Authentication of identities allows users to identify one another so that they may trust each other.

-- Authorization checking is carried out using access control lists(ACLs) to control access to distributed services and resources.

◆ Thread Service provides support for creating and managing multiple sequential flows of execution within a single process in a computer.

◆ DTS provides precise, fault-tolerant clock synchronization for computers connected in Local Area Networks (LANs) and Wide Area Networks (WANs) in a distributed environment.

◆ CDS provides directory (or naming) support which allows DCE services and applications to easily locate and look up information about objects in a distributed environment, such as files, services, etc.

◆ DFS is the key DCE information-sharing component, and is built upon the fundamental services. It masks the distribution of file systems, making it easy to work with remote files for users and application developers. Users are able to log to any computer in the global DCE environment to access DFS files the same way.

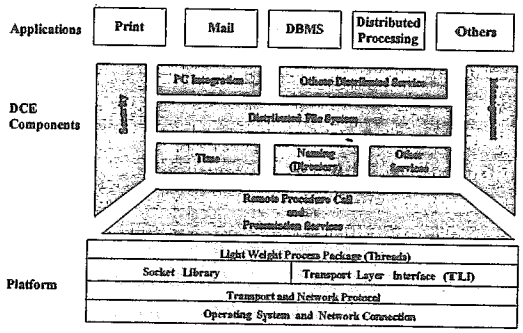

Figure 2-2: DCE Architecture

2.2 *DCE/RPC*

Remote Procedure Call (RPC) is the core of DCE. RPC is based on an openly specified architecture that provides a common base for the independent development of distributed computing products, applications, and servers. RPC works consistently across implementations on heterogeneous systems, providing interoperability - a key benefit of open systems. It is a typical client/server model. An RPC interface is a contract for a set of remote procedures to be offered by servers and to be used by clients, as shown in Figure 2-3.

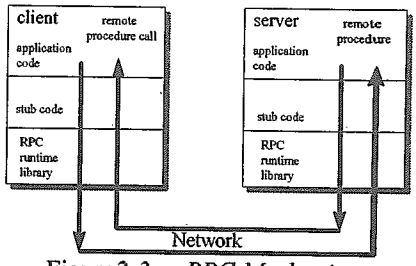

Figure*2-3: RPC Mechanism*

The following steps describe the control flow of RPC:

1. The client application code makes a RPC, passing the input arguments to its stub.
2. The client RPC runtime transmits the input arguments over the network to the server RPC runtime, which dispatches the call to the appropriate server stub.
3. The remote procedure executes and returns any results to the server stub.
4. The server RPC runtime transmits the results over the network to the client RPC runtime, which dispatches them to the client stub.
5. The client stub uses its copy of the RPC interface to unmarshall the result and pass them to the calling code.

2.3 *DCE/Security*

A distributed computing environment encourages the free flow of information from where it is stored to where it is needed, and the sharing of the network services and resources. But the computer networks are unfortunately susceptible to relatively easy security breaches by running network monitoring software on the workstation or by using dedicated network monitoring devices. So application developers should be able to build in the desired security into new distributed applications in an easy way. Security for application of distributed computing consists of identities, authorization of authenticated principals for using services and resources, and guarantees of integrity and privacy of messages sent over the network.

Authentication of identities allows principals to identify one another so that they may trust each other. Once principals are authenticated, authorization checking is carried out. DCE uses access control lists(ACLs) to control access to distributed services and resources. ACLs are associated with each service or resource. An ACL contains a list of names of principals and the types of operations they are permitted to perform.

3. Stub generator

DCE RPC interfaces are generated from a formal interface definition written using the DCE Interface Definition Language(IDL). RPC interfaces are implemented by 'stub routine', which are generated automatically by an IDL compiler(see Figure 3-1)[11].

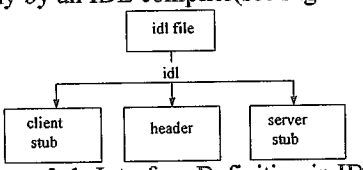

Figure 3-1: Interface Definition in IDL

Stubs handle the interfaces of remote procedures with the main body of the application program. The stub routines implement the interface required for RPCs. Client and server stubs are linked with the client and server code respectively. The client stub packages ('marshalls') the arguments to the call, transmits the data to the server, and waits for the server's reply. The server stub unpacks ('unmarshalls') the arguments, calls the required procedure, packages the results, and sends the reply to the client. Results are returned to the calling program, as shown in Figure 3-2.

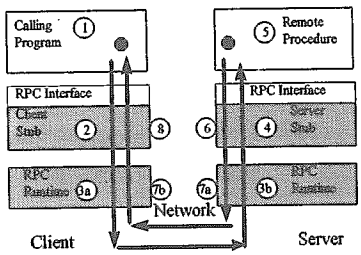

Figure 3-2: RPC and Stub operation

3.1 *Design of stubgen*

Due to DCE RPC's stub can only handle C functions, and IDL compiler can only interpret C API. So we built a stub code which can analyze and execute C++ header file. The stub code was created by the stubgen, which was built upon the DCE RPC stub. The client C++ application code 'marshalls' the input arguments - prepares arguments for transmission and dispatches the call to the server . The server stub uses its stub code interface to 'unmarshall' the input arguments - disassemble incoming network data and convert it into application data in the format that the local system understands and pass them up to the C++ application code. All the operations go through C++ stub code object-oriented model, which the users do not need to know the lower level long and complicate library functions. Figure 3-3 illustrates the operation.

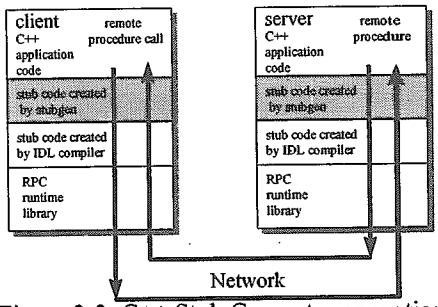

Figure 3-3: C++ Stub Generator operation

stubgen generated two header files, one IDL file, and two Stub routines. The details are described as follows:

1. *_f.h: copies a C++ header file which does not include "typedef", "define" and other declarations.
2. *_s.h: packs class parameter and private data into structure format and records them. This header file records the "typedef", "define", and other declarations.
3. *_i.idl: generates stub file to communicate with RPC runtime.
4. *_cs.cc: client site stub program.
3. *_ss.cc: server site stub program.

3.2 *Constraint of Stub generator*

In this section, we characterize the constraints of DCE and stubgen. The memory address can be different for the shared data at each node in a heterogeneous environment. Therefore, it is difficult to maintain data consistency. To summarize, we use the following two constraints for DCE.

1. No public data within the class:

Cannot have public data in the class. If public data exist in the class, its memory address may be different for each node in the network. This will cause the data to be inconsistent.

2. No "inline" type format for object:
All the "inline" function in the class, the C++ compiler will convert to macro format. If the object is in remote site, which has "inline" function in the class, then the compiler would not find its code in the remote site.
In our prototype system, stub generator can not handle the following.

1. No complicated data structure:
Due to the stub generator can not marshall and unmarshall the data structure of linked list format. Currently, the system can not handle the linked list data structure.

2. C++ Application interfaces are only available for Remote Procedure Call and Security service.

3.3 How to use the stub generator

The way to use the stub generator can be described from the input and output.

- Input of stubgen:

C++ language is class and its a user-defined class. Classes provide data hiding, guaranteed initialization of data, implicit type conversion for user-defined types, dynamic typing, user-controlled memory management, and mechanisms for overloading operators. It has already had enough information for stub. So the input of stubgen is a C++ header file only.Stubgen based on this header file to analyze and generate demand file.

- Output of stubgen:

Stubgen analyzes the definition of C++ classes and generates *_f.h, *_s.h, *_i.idl, *_cs.cc(client stub), and *ss.cc(server stub). The *i_idl uses DCE/IDL compiler to compile and generate RPC runtime stub. The *_cs.cc and *_ss.cc use the C++ compiler to generate the object files, as shown in Figure 3-4.

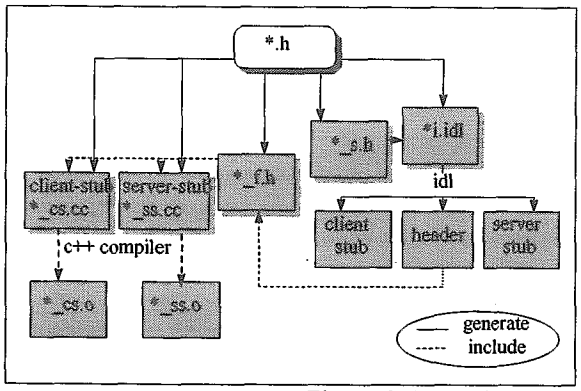

Figure 3-4: stub generator flow diagram

4. RPC/C++ Implementation

DCE/RPC uses some data structures to record and control its status. These data structures are described as follows[1,2]:

- binding handle: point to the binding information. This binding information records binding related information, such as server address, network protocol, object UUID, etc.
- uuid: The universal unique identifier(UUID) is the mechanism to assure the uniqueness of the interface. It is generated by an interactive utility, uuidgen. uuidgen is used to identify network object, such as user, group, node, cell, etc.
- protocol sequence: User uses it to select its own network protocol. DCE provides three kinds of network protocols: (1) ncacn_ip_tcp using the TCP protocol. (2) ncadg_ip_udp using the UDP protocol, and (3) ip using UDPprotocol.
- profile entry: It is used to set the service priority. Each entry contains server's binding informations.
- endpoint: It is an address of the server's instance, like a connection port.

The above informations are encapsulated in the RPC classes and each data structure is independent. In our system, C++ application interface classes do not have clear class hierarchy relationship, but they all depend on each other, as shown in Figure 4-1.

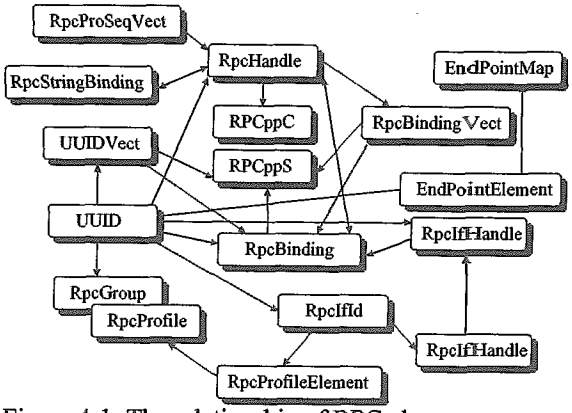

Figure 4-1: The relationship of RPC classes

The above classes belong to low level or thin C++ API classes. We need to have high level C++ interface upon these low level C++ API, which provides users a friendly interface to access the system. So the users do not need know the complicated library functions in the low level. The high level C++ interface has RPCppC and RPCppS. One is for client site, and the other one is for the server site, as shown in Figure 4-2.

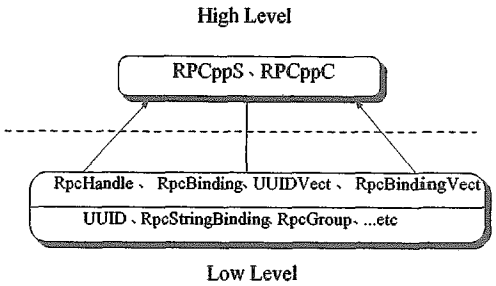

Figure 4-2: The relationship of the Low Level and High Level Classes

Although DCE application programs can be

executed in the heterogeneous environment, the calling code and the called remote procedure are not linked together, they communicate indirectly, through an RPC runtime, over the network. An RPC runtime manages the network and provides library function to access runtime operation[7]. The runtime operation is described as follows:

1. communication operation :
   build binding, which client and server's program uses it to request and communicate data.
2. name service interface operation (NSI) :
   provide RPC server's information to the programmer through name service. RPC server uses NSI operation to broadcast server interface, object, and address information which are recorded in name service database.
3. endpoint operation :
   allow server to add or delete endpoint.
4. authentication operation :
   provide authentication, protection level, and authorization for distributed applications.
5. UUID operation :
   provide some operation to build and maintain UUID.

Now, according to the above information, RPC C++ API can be grouped as follows[12]:

◆ RpcHandle 、RpcStringBinding 、RpcBinding :
manage and maintain binding handle, string binding, and other related informations.

◆ RpcProfile 、RpcGroup 、RpcProfileElement 、RpcIfId :
allow user to add or delete entry from name service database.

◆ EndPointElement 、EndPointMap :
manage and maintain endpoint element. Informations of the endpoint element have binding handle, interface ID, object UUID, and annotation.

◆ UUID :
manage and maintain UUID.

◆ RpcBindingVect 、RpcIfIdVect 、RpcProtVect 、RpcStatusVect 、UUIDVect :

Figure 4-3 shows the relationship of C API, RPC runtime operation and RPC C++ classes.

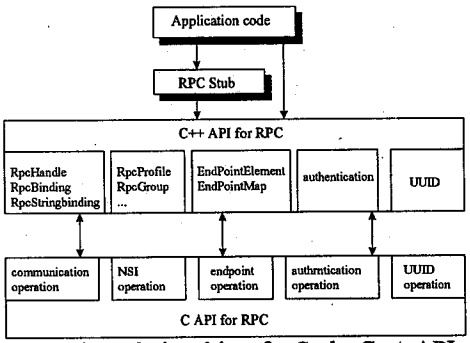

Figure 4-3: The relationship of Stub, C++ API, and C API

5. Security/C++ Implementation

DCE provides comprehensive security support for security-sensitive distributed computing. Application interface of Security Service provides authentication, authorization, data integrity, and data privacy. The characteristics of the components are described as follows[2,7]:

- authentication: allows users to identify one another so that they may trust each other.
- authorization: using access control lists to control access to distributed services and resources.
- data integrity: using cryptographic data checksums to determine whether data was modified or corrupted while passing through the network.
- data privacy: ensure the encrypting /decrypting data as it is transferred across a network.
- access control list (ACL) : contains a list of names of principals and the types of operations they are permitted to perform.
- principal key : comes from the password of principal. It is used to verify the access right of the principal.
- principal, group, organization (PGO) :

The registry database of Security Service is composed of the following three container objects:

- principal--includes principal name.
- group--records group name and members of group's principal.
- organization--records organization name and members of organization's principal.
- login context: contains principal name, UUID, group of principal, and access right of the account.
- account: contains person, group, and organization name 。

Security classes encapsulate the above data structure to manage and maintain resources.

The interface of security service can be grouped into five categories[7]. The characteristics of the security API are described as follows:

1. Registry API: provides a binding mechanism to link with registry database.
2. Login API: builds login context.
3. ACL API: manages and maintains access control lists 。
4. Key Management API: verifies the access right.
5. ID Map API: analyze global name.

Based on the above API categories, the implementation's classes are described as follows:

◆ SecRgyBinding, SecRgyAcct, SecRgyPGOItem, and SecRgyPGO : manages and maintains registry database.

◆ SecLoginContext: builds login context.

◆ SecAclLocalIf, SecRdaclNetworkIf, and SecAclMgmtIf: provides ACL network interfaces and managemet programs.

- SecKeyMgmt: manages the account key.
- SecIDMap:analyzes global name and maps with id.

The relationship of the security classes is shown in Figure 5-1.

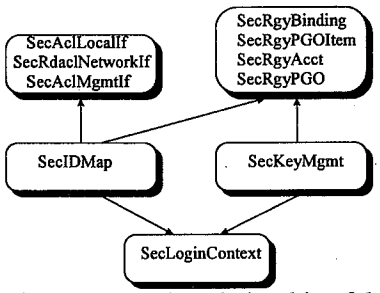

Figure 5-1: The relationship of the security classes

These classes are provided by Security Service/C++ API, which are built upon the Security Service / C API, as shown in Figure 5-2.

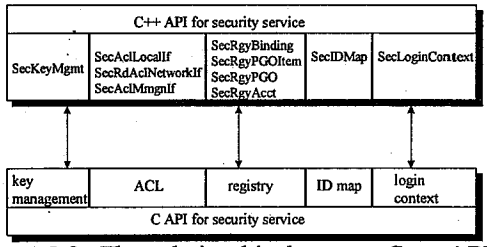

Figure 5-2: The relationship between C++ API and C API

6. Example application

In this chapter, we used addition and subtraction of array as our example. In this example, the programmers only have to write four files. Each file is described in the following:

client.cc declares the variable op to use arithmetic class, and uses this class to build the initial value of op object. This initial value is the status value of the client when it is set up.

server.cc declares the variable op to use arithmetic class, and uses this class to build the initial value of op object. This initial value is the status value of the server when it is set up.

arithmetic.cc has two remote files. One is for addition, and the other one is for subtraction.

```
#include <stdio.h>
#include <DCE/rpcpp.h>
#include <DCE/sec_context.h>
#include <DCE/sec_keymgmt.h>
const unsigned short ARRAY_SIZE = 10;
typedef long long_array[ARRAY_SIZE];

class arithmetic:public RPCppC,RPCppS{
    private:
            long_array c;
    public:
    arithmetic(){};
    arithmetic(unsigned_char_t *entry,rpc_if_handle_t if_h,
            unsigned_char_t *protseq,UUIDVect *uuid_vect=NULL)
        :RPCppS(entry,if_h,protseq,uuid_vect){};
    arithmetic(unsigned_char_t *entry,rpc_if_handle_t if_h,UUID *uuid=NULL)
        :RPCppC(entry,if_h,uuid){};
    void sum_arrays(handle_t handle,long_array a,long_array b);
    void diff_arrays(handle_t handle,long_array a,long_array b);
    long *get_result() { return c; };
    rpc_binding_handle_t Get() { return RPCppC::GetBindingHandle();};
};
```

Program 6-1 arithmetic.h

```
/* FILE NAME: client.cc */
/* This is the client module of the arithmetic example. */
#include <stdio.h>
#include "arith_f.h"      /* header file created by Stubgen compiler */
```

```
long_array a ={100,200,345,23,67,65,0,10,20,0};
long_array b ={4,0,2,3,1,7,5,9,6,8};

main ()
{
    int i;
    SecLoginContext context(2);
    unsigned_char_t         *princ_name="chencc";
    unsigned_char_t         *entry_name="/.:/arithmetic_serverhost";
    arithmetic     op(entry_name,arith_v0_0_c_ifspec);
    unsigned32     status;

    printf("entry name = %s\n",entry_name);
    rpc_binding_set_auth_info((handle_t)op.Get(),princ_name,
                        rpc_c_protect_level_pkt,rpc_c_authn_dce_secret,
                        (rpc_auth_identity_handle_t) context,
                        rpc_c_authz_dce,&status);
    op.sum_arrays(handle,a, b);                 /* A Remote Procedure Call */
    puts("sums:");
    for(i = 0; i < ARRAY_SIZE; i++)
        printf("%ld\n", op.get_result()[i]);
    op.diff_arrays(op.Get(),a, b);              /* A Remote Procedure Call */
    puts("diff:");
    for(i = 0; i < ARRAY_SIZE; i++)
        printf("%ld\n", op.get_result()[i]);
}
```

Program 6-2 client.cc

```
/* FILE NAME: server.cc */
#include <stdio.h>
#include "arith_f.h"                    /* header file created by Stubgen compiler */

#define KEYTAB "/home/chencc/simple_server_tab"
main ()
{
    unsigned32              status;                         /* error status (nbase.h) */
    rpc_binding_vector_t *binding_vector; /*set of binding handles(rpcbase.h)*/
    unsigned_char_t         *entry_name; /*entry name for name service (lbase.h)*/
    unsigned_char_t         *princ_name="chencc";
    void *key;

    entry_name = (unsigned_char_t *)getenv("ARITHMETIC_SERVER_ENTRY");
    printf("entry name = %s\n",entry_name);
    arithmetic arith(entry_name,arith_v0_0_s_ifspec);
    rpc_server_register_auth_info(princ_name,rpc_c_authn_dce_secret,NULL,KEYTAB,&status);
    SecKeyMgmt keymgmt(rpc_c_authn_dce_secret,KEYTAB,princ_name,0);
    SecLoginContext context(princ_name,(sec_passwd_rec_t *)keymgmt.get_key(),sec_login_no_flags);
    arith.ServerListen();
}
```

Program 6-3 server.cc

```
/* FILE NAME: arithmetic.cc */
/* An implementation of the procedure defined in the arithmetic interface. */
#include <stdio.h>
#include "arith_f.h"                /* header file produced by Stubgen compiler */
```

```
void arithmetic::sum_arrays(handle_t handle,long_array a,long_array b)
    {
        int i;
        unsigned_char_t    *server_name;
        unsigned32   protect_level,authn_svc,authz_svc;
        error_status_t status;
        rpc_binding_inq_auth_client(binding,(rpc_authz_handle_t *) &pac,
                &server_name,&protect_level,&authn_svc,&authz_svc,&status);
        if (status==rpc_s_ok&&authz_svc==rpc_c_authz_dce)
        {
          for(i = 0; i < ARRAY_SIZE; i++)
              c[i] = a[i] + b[i];
        }
    }
void arithmetic::diff_arrays(handle_t handle,long_array a,long_array b)
    {
        int i;
        for(i = 0; i < ARRAY_SIZE; i++)
            c[i] = a[i] - b[i];
    }
```

Program 6-4 arithemtic.cc

7. Conclusion and Future work

In this work, we implemented a C++ API prototype system upon DCE, which allows programmers to implement an application system by object-oriented methodology. This API has inherit, overloading, object, and data privacy features. This C++ API can reduce programmers' learning and developing time, and provides users an easy way to manage and maintain the system.

Currently, the C++ API is only available for Remote Procedure Call and Security Service. Some techniques are still needed to handle more complicated data structures, such as linked lists. As a future step, we plan to (1) build the C++ Application interfaces for other DCE services, and (2) enhance the functions of stubgen to handle more complicated data structure.

References

I. Open Software Foundation, "Introduction to OSF DCE", Open Software Foundation,Cambridge,USA 1993

II. W. Lockhart, Jr., "OSF DCE Guide to Developing Distributed Applications", McGraw-Hill, Inc., 1994.

III. Bjarne Stroustrup, "The C++ Programming Language",2nd Edition, Addision Wesley, Inc.,1991.

IV. Willian Leddy and Arjun Khanna, "DCE++:A C++ API for DCE",Hal Cpmputer Systems,Inc.,1993

V. John R. Levine, Tony Mason and Doug Brown, "Lex & Yacc", O'Reilly & Associates Inc, October 1992.

VI. Open Software Foundation, "DCE Users Guide and Reference", Open Software Foundation, Cambridge, USA 1993.

VII. Open Software Foundation, "DCE Application Development Guide", Open Software Foundation, Cambridge, USA 1993.

VIII. Open Software Foundation "DCE Application Development Reference", Open Software Foundation, Cambridge, USA 1993.

IX. Open Software Foundation "OSF DCE Administration Guide", Open Software Foundation, Cambridge, USA 1992.

X. W. Rosenberry, D. Kenney and G. Fisher, "Understanding DCE," O'Reilly & Associates Inc, September 1992.

XI. Shirley, "Guide to Writing DCE Applications", O'Reilly & Associates Inc., 1992.

XII. Y.H. Du, "Object-based RPC Develop Environment", IE NSYSU,1995.