

Progress Report:

Automated Source Migration From OODCE to CORBA

Student: Daniel Jensen
Advisor: Dr. Rod Fatoohi

San Jose State University &
NASA Ames Research Center

June 1, 2000



TABLE OF CONTENTS

1	INTRODUCTION	4
1.1	OBJECTIVE	4
1.2	CORBA AND THE ALTERNATIVES	5
1.3	STRATEGIES FOR MIGRATING TO CORBA	6
1.3.1	<i>Where to Draw the Line?</i>	6
1.3.2	<i>Our Selected Strategy</i>	6
1.3.3	<i>Other Strategies for Migrating to CORBA</i>	7
1.3.4	<i>Combining Two Strategies</i>	8
2	TECHNICAL OBJECTIVES	9
2.1	SIMPLE AND THREADED MATH EXAMPLES	9
2.2	SIMPLE IMPLEMENTATION OF THE ECS SUBSCRIPTION SERVER INTERFACE.....	9
3	PROCESS	10
3.1	SIMPLE AND THREADED MATH EXAMPLES	10
3.2	THE ECS SUBSCRIPTION SERVER INTERFACE.....	11
4	DESIGN	12
4.1	INVOCATION FLOW DIAGRAM.....	13
4.2	MIGRATION PROCESS DIAGRAM	15
5	MIGRATION PROCESS - STEPS	16
6	PREVIOUS RESEARCH	17
7	GENERATING A CORBA INTERFACE DEFINITION (IDL) FILE	20
7.1	FEASIBILITY.....	20
7.2	THE TOOL: COTS OR CUSTOM?	21
7.3	THE ELEMENTS OF IDL CONVERSION	22
7.4	OTHER ISSUES	23
8	THE CLIENT	24
8.1	MISCELLANEOUS DETAILS	24
8.2	BASIC FORM OF A INTER-MIDDLEWARE INVOCATION	25
8.3	ADVANCED TYPE MAPPING	25
8.4	CODE EXAMPLES	26
8.4.1	<i>Header Files</i>	26
8.4.2	<i>Source Files</i>	26
9	THE SERVER	27
9.1	CODE EXAMPLES	28
9.1.1	<i>Header Files</i>	28
9.1.2	<i>Source Files</i>	28

10	DCESERVER	29
11	OBSERVATIONS	30
11.1	CHANGES IN VISIBROKER.....	30
11.2	PROBLEMS WITH SKELETONS AND STUBS.....	31
11.2.1	<i>Returning references (pointers, handles, etc.)</i>	31
11.2.2	<i>Single-Element Sequences</i>	31
12	CONCLUSIONS	32
12.1	MINOR PROBLEMS	33
12.2	UPCOMING WORK	33
13	LISTINGS OF AUTOMATICALLY GENERATED CODE	34
13.1	CLIENT GLUE HEADER (MATH INTERFACE)	35
13.2	SERVER GLUE HEADER (MATH INTERFACE)	36
13.3	CLIENT GLUE IMPLEMENTATION (MATH INTERFACE)	38
13.4	CLIENT GLUE HEADER (SUBSCRIPTION INTERFACE)	42
13.5	SERVER GLUE HEADER (SUBSCRIPTION INTERFACE)	44
13.6	CLIENT GLUE IMPLEMENTATION (SUBSCRIPTION INTERFACE)	49
13.7	SERVER GLUE IMPLEMENTATION (SUBSCRIPTION INTERFACE)	59
14	LISTINGS OF FRAMEWORK CODE	65
14.1	SERVER.H.....	65
14.2	DCESERVER.H	66
14.3	DCESERVER.CPP.....	68
14.4	TYPEMAPPERS.CPP	72
15	OUTPUT	78
15.1	OUTPUT FROM THE SIMPLE MATH EXAMPLE	78
15.2	OUTPUT FROM THE THREADED MATH EXAMPLE	79
16	ORBS WITH C++ BINDINGS	80
17	REFERENCES	81

1 INTRODUCTION

1.1 Objective

The objective of this exercise is to examine the portability of OODCE-based source code to CORBA, primarily by attempting to port several example interfaces and their implementations, but also by considering general issues that we're exposed to in the process.

1.2 CORBA and the Alternatives

The purpose of this study is not to resolve which strategy (CORBA, DCOM, RPC, etc.) is best, but rather to measure the effectiveness of a specific CORBA sub-strategy. Still, let us quickly review our reasoning for looking more deeply into CORBA.

EOSDIS has a number of significant issues to resolve, not the least of which is its complexity above the middleware layer. Middleware will not save EOSDIS so long as it is burdened by a lack of a consistent Application Programming Interface (API). Though the EOSDIS vision was to have a consistent API, expediency weakened this vision as schedules for the components of this API slipped. In stating this, we only reflect an apparent consensus on the history of EOSDIS, and are not attempting to analyze or second-guess the engineering management of EOSDIS in the past.

The two dominant middleware standards, at least as far as C++ applications is concerned, are CORBA and COM/DCOM (henceforward referred to as COM)¹. CORBA is recognized as the dominant *distributed* computing architecture, though COM deserves consideration due to its dominance as a component architecture:

Microsoft's COM is the dominant component architecture, and the OMG's CORBA is the dominant remoting architecture. Although several other products, such as Java RMI, have established a share of the market, they show no signs of supplanting the positions held by COM and CORBA.²

COM is a noteworthy alternative for those considering migration from DCE, since it inherits its RPC mechanism (DCOM) from DCE, however:

1. COM hasn't been ported to UNIX³. A complete port is a problematic goal since COM is tightly coupled with a number of Microsoft Windows technologies.⁴ Extracting COM from Windows is, in part, like extracting *Windows* from Windows.
2. COM's object interface is COM, which inherits nothing from DCE. Rather, OODCE and CORBA appear to have more in common in this regard than OODCE and COM.

¹ With all due recognition to DCE.

² COM and CORBA, Side by Side – Architectures, Strategies, and Implementations; by Jason Pritchard, Ph.D.; page 21.

³ Efforts are being made by [Microsoft](#) and [Software AG](#) to port DCOM to a variety of UNIX platforms. ECS is based on an SGI/IRIX implementation of HP's OODCE. Microsoft and Software AG have no plans for supporting DCOM on SGI/IRIX. In contrast, three CORBA ORBs with C++ bindings currently support SGI/IRIX.

⁴ See COM and CORBA, Side by Side, page 23.

1.3 Strategies for Migrating to CORBA

1.3.1 Where to Draw the Line?

The closest thing ECS provides to a universal API or abstraction layer is OODCE. This layer has two important strengths:

- OODCE provides a simple interface, relative to DCE.
- It is reported that very little EOSDIS application code undermines the OODCE interface by making direct calls to DCE.⁵

1.3.2 Our Selected Strategy

The most natural approach to porting OODCE applications to CORBA is to encapsulate CORBA client stubs and server skeletons within OODCE-style classes. Such encapsulation would be performed by an *idl++* replacement compiler. This strategy is relatively simple because it calls for relatively few changes: it doesn't require modification of compiler code or application code, and it leverages existing CORBA IDL compilers. It does, however, call for the development of an IDL converter and a simple IDL interface compiler, not to mention a support framework of definitions and classes.

⁵ We are concerned, however, that OODCE may have commonly been “extended” by EOSDIS code. Such extensions of OODCE may undermine the simplicity of the interface. For example: the DCEFilePassword class.

1.3.3 Other Strategies for Migrating to CORBA

There are several alternatives to the strategy taken in the this report:

1. Follow the same basic strategy, but from a DCE-to-CORBA perspective. See the section entitled “Previous Research” for details.
2. Modify the OODCE *idl++* compiler so that it generates CORBA stubs and skeletons instead of DCE stubs, while not altering the interface it currently generates. This is an elegant solution, but it requires extensive modification of compiler code; effectively, a hybrid of the *idl++* and *idl2cpp* compilers. It is also an inflexible option, as it doesn’t leverage diverse CORBA implementations, but rather it supports only its own.
3. Develop a utility that generates CORBA stubs and skeletons that can be placed under the interfaces generated by *idl++*. This involves modifying the CORBA *idl2cpp* compiler rather than replacing the OODCE *idl++* compiler, and would probably be more difficult to accomplish. This option is inflexible for the same reason that the previous option is inflexible: it enforces a single CORBA implementation.
4. Modify *idl++*⁶ to generate DCE stubs that use IIOP rather than RPC. This is not actually a strategy for replacing DCE or OODCE or migrating to CORBA, but rather a strategy for replacing the RPC protocol. Implementing CORBA support at the IIOP level would probably mean developing a hybrid between *idl++* and *idl2cpp* from the ground up, not to mention developing a suite of support headers and libraries.
5. Develop a utility that converts OODCE code (in the client and server *application* code; not the auto-generated code) to CORBA code. This strategy is problematic because it calls for modification of application code, and probably extensive modification at that.
6. Use a DCE-CORBA bridge. This would enable CORBA clients to interact with DCE servers, but would inhibit system growth or migration because it would not facilitate interaction between legacy DCE clients and new CORBA servers. See the next section, “Combining Strategies”, for further discussion.

⁶ The source code for *idl++* is the property of NASA, but is not immediately accessible to us at this time.

1.3.4 Combining Two Strategies

Our source-level migration strategy may be coupled with a DCE-CORBA bridge to capitalize on the complementary strengths of each approach.

- **Bridging:**

- Access OODCE servers with CORBA clients.
- Dynamic: doesn't require recompilation.
- Facilitates creation of new CORBA clients and gradual conversion of legacy clients to CORBA.

- **Source-Level Migration:**

- Run OODCE client and server code over CORBA.
- Static: requires minimal runtime overhead.
- Facilitates a more rapid migration of OODCE clients and servers to CORBA.

The two approaches can be combined by way of a three-stage transition plan:

- 1. Start by using source-level strategy to gradually move clients to CORBA.**
 - Continue accessing DCE servers via bridge.
- 2. Gradually move outer-tier servers (called only by non-servers) to CORBA.**
 - Access these servers directly (via CORBA).
- 3. Finally, move all remaining servers to CORBA simultaneously.**

Because this issue is outside the scope of this report, we will not address it any further.

2 Technical Objectives

2.1 Simple and Threaded Math Examples

Our initial technical objective is to execute the *simple and threaded math examples* without changing any application code, which consists of the following source files:

- `client.C`
- `server.C`
- `<interface>_<ver>_Mgr.C`
 - `math_1_0_Mgr.C`
 - `matha_1_0_Mgr.C`

2.2 Simple Implementation of the ECS Subscription Server Interface

Our second technical objective is to port a trivial implementation of the ECS subscription server interface, without changing any implementation source code.

There are two good reasons for working with a trivial implementation:

1. We don't have access to the actual implementation code.
2. Using the actual implementation code would introduce introduce issues beyond the scope of this exercise.

3 Process

3.1 Simple and Threaded Math Examples

The interfaces, implementation, client, and server code used in this example were provided for us; hence we did not determine the configuration described herein.

The examples were executed in the VisiBroker v.3.3 environment.

The steps of test execution are as follows:

1. Process each interface (IDL) file with our "Converter" utility, which performs two tasks:
 - a. Replaces the *idl++* utility with respect to high-level code generation.
 - b. Converts DCE IDL to OMG IDL.
2. Process the resulting OMG IDL with the *idl2cpp* utility provided with VisiBroker. This step replaces the use of *idl++* with respect to low-level code generation.
3. Create client and server projects consisting of legacy code and the out put of steps 1a and 2.
4. Build the client and server executables.
5. Launch the VisiBroker *osagent* and naming service.
6. Launch the server application(s).
7. Launch the client application(s).
8. Record server and client output.

The examples provided for this exercise use two similar interfaces: *math* and *matha*. For the purposes of CORBA, these interfaces are identical. The example is constructed such that a server program may register one of two pairs of objects with the ORB, depending on how that server program is launched. Since the client expects to bind to both object pairs, we launch the server program twice (each launch in a separate process).

Once the server processes are launched, we run the client program and watch its output. Our objective is to see that the client binds successfully to all server objects, and succeeds in receiving the proper results from each server object's member functions.

3.2 The ECS Subscription Server Interface

The implementation, client, and server code used in this example were developed, tested over OODCE and SunOS, and provided by Sandeep Gawai, a graduate student at San Jose State University.

This example was executed in both the VisiBroker v.3.3 and v.4.0 environments.

The basic execution steps for this example are identical with those of the previous (math) example.

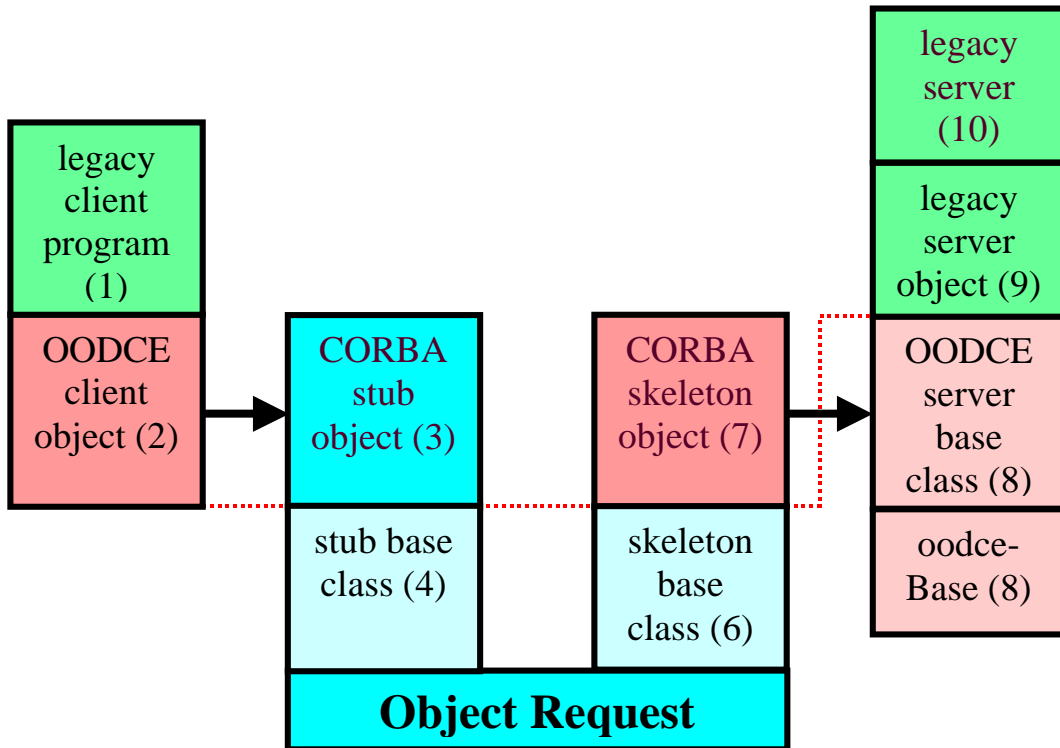
4 Design

Because our objective is to port OODCE applications without modifying source code, the top-level code remains unchanged, and must interact with a CORBA ORB. Most of the hard work is performed by CORBA skeletons and stubs generated by a CORBA utility such as *idl2cpp*. Before any skeletons or stubs can be generated, OODCE IDL files must be converted to OMG (CORBA) IDL. To enable the OODCE legacy code to communicate with the CORBA skeletons and stubs, an intermediate object layer must be generated from the OODCE IDL files.

In addition:

1. OODCE data types must be mapped to CORBA data types (see *Definition Conversions* in the diagram).
2. Services must be supplied where CORBA doesn't provide an equivalent to an OODCE class (e.g., DCEServer and DCEPthread).

4.1 Invocation Flow Diagram



Description of the Invocation Flow Diagram

Key:

- Green – legacy code.
- Red – conversion objects generated by “Converter” utility.
- Pink – abstract base classes generated by “Converter” utility.
- Blue – objects and services provided by the CORBA implementation.
- Light Blue - abstract base classes provided by the CORBA implementation.

The flow of a method invocation moves from left to right in the diagram. Invocation begins in the legacy client program (1), where a method of an OODCE client object (2) is invoked.

The OODCE client object preforms the necessary conversions of non-primitive types, represented by the arrow pointing to the CORBA stub object (3).

By way of the CORBA stub object’s base class (4), the invocation is passed across the ORB (5) to the corresponding skeletal base class (6).

The skeletal object (7) built upon the base class then preforms the necessary conversions of non-primitive types, represented by the arrow pointing to the OODCE server base class (8) and a special purpose class developed for this project, named *oodceBase* (8).

Finally, a legacy server implementation object (9) is invoked within the legacy server process (10).

Return data and “out” parameters are returned along the same path, in the opposite order.

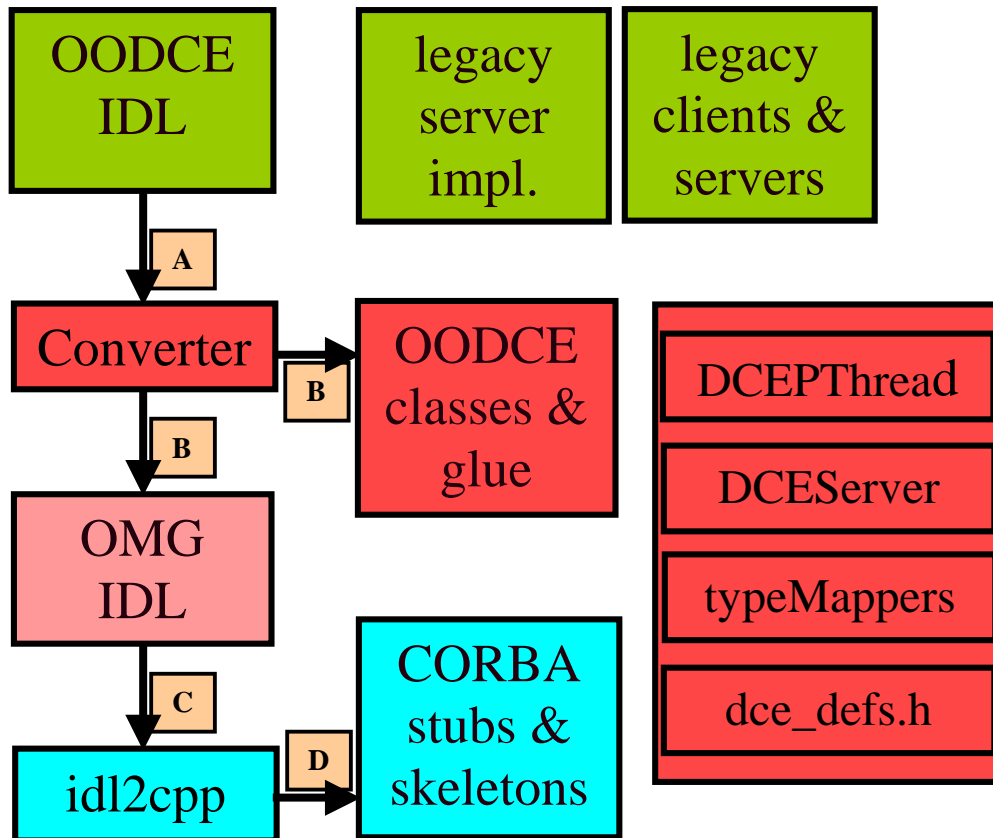
The four columns represent middleware zones (left to right):

- I. OODCE client zone.
- II. CORBA client zone.
- III. CORBA server zone.
- IV. OODCE server zone.

The arrows between OODCE and CORBA zones represent the critical type mapping functionality.

The red dotted line represents the boundary between concrete objects and their base classes.

4.2 Migration Process Diagram



The migration process diagram displays legacy code (in green) at the top, with VisiBroker utility and output (in blue) at the bottom, and our contributions in red (code) and pink (data) between.

The four basic steps illustrated in the diagram are outlined below:

- A. The migration process is initiated feeding OODCE IDL into the “Converter” utility.
- B. The “Converter” utility generates two types of data:
 - i. OMG IDL.
 - ii. OODCE classes & glue.
- C. The newly generated OMG IDL is fed into the VisiBroker *idl2cpp* utility.
- D. *Idl2cpp* generates the CORBA stubs and skeletons to be used under the auto-generated OODCE classes and glue.

5 Migration Process - Steps

- 1) Presume there is a pre-existing OODCE application, including an IDL file, headers, stubs, compiler-generated sources, and application sources. Make a copy of the entire file set for our use.
- 2) Run the OODCE IDL file through the *Converter* utility that generates the following files:
 - <interface>.idl is an OMG (CORBA) IDL file.
 - <interface>S.H is the header file included by legacy server code.
 - <interface>S.cpp is the source file which provides type mapping and other facilities on the server side.
 - <interface>C.H is the header file included by legacy client code.
 - <interface>C.cpp is the source file which provides type mapping and other facilities on the client side.
- 3) Run the new <interface>.idl file through a CORBA IDL compiler (such as *VisiBroker's idl2cpp*). This generates a number of CORBA-style stub and skeleton files which we do not modify.
- 4) Create a project, using a number of pre-constructed files that provide definitions and functionality:
 - Dce_defs.h CORBA-friendly definitions for DCE types.
 - unistd.h an empty placeholder file.
 - RPCErr.H an empty placeholder file.
 - Pthread.H OODCE-compatible threading definitions.
 - DCEPthread.lib OODCE-compatible threading library.
 - Server.H DCEServer replacement definitions for main program file.
 - DCEServer.h DCEServer replacement definitions for general use.
 - DCEServer.lib DCEServer replacement library.
 - TypeMappers.h Type mapping facility definitions & declarations.
 - TypeMappers.h Type mapping facility implementation code.
 - <interface>.idl an empty placeholder file.

6 Previous Research

The author of this paper recently discovered (in March 2000) that very similar work had already been performed several years ago. The paper [Software Tools for Automating the Migration From DCE to CORBA](#),⁷ authored by Aniruddha Gokhale, Douglas C. Schmidt, and Stanley Moyer, describes their research, development, and conclusions.

Much of our development work mirrors that of the above-cited collaboration between Washington University and Bellcore (now Telcordia Technologies, Inc.). The primary difference is that the objective of the authors of the paper was to port from DCE to CORBA, where our objective is to port from OODCE.

We have attempted to contact each author, and have succeeded with Dr. Schmidt, but have not acquired any additional information on their project.

Their paper points out a number of porting issues that concern us:

1. CORBA doesn't support idempotent functions. This may be significant, as many EOSDIS functions are idempotent. For instance, 6 of the 18 methods in *DsSdDCESubscrIDL.idl* are idempotent. Fortunately, the idempotent feature is intended primarily for unreliable protocols such as UDP. When using TCP, a client that has received a reply from a server method can be sure it has executed that method exactly once. In this case, method idempotence is much less important.⁸

⁷ Dated August 3, 1998.

⁸ See [Power Programming with RPC](#); by John Bloomer; pub. O'Reilly, 1992; pages 109-10, 198.

2. CORBA doesn't support DCE context handles transparently.⁹ 14 of the 43 EOSDIS interfaces, including *DsSdDCESubscrIDL.idl*, contain context handles. Each of these interfaces uses a single context handle to define a pointer type that refers to itself. In the case of the subscription server interface, this context handle is either passed to or returned by each of its 18 methods. The purpose of these context handles are to maintain session information between a client and a server. The client holds a context handle to identify itself and its current session with the server, but cannot make any other use of the handle. The server generates the handle, and uses it to store session data. The client and server depend on DCE stubs to manage the context handle, specifically to call a server-implemented routine in the case of a connection failure.¹⁰ CORBA does not provide this service, so we must somehow provide it above the skeletons and stubs, yet below the legacy application code. To emulate DCE functionality, we need to find a way to detect a connection failure. Perhaps a simple timeout mechanism would be adequate for demonstration purposes. A method could be defined in a base class that could be inherited by the `<interface>_<ver>_Mgr` class in the `<interface>S.H` file, and overridden by manager code. How, then, would that method be called? Every time a server method is called, a threaded timer would be restarted. If that timer reaches a predetermined value, the aforementioned context rundown procedure would be executed.
 - a. This timeout management code be declared in the remote methods of the `<interface>_<ver>_Mgr` class in the `<interface>S.H` file, and implemented (defined) in a dedicated source file. The server application code would no longer implement the remote interfaces, but a slight variation of them. This could be facilitated by modifying the CORBA skeletons.
 - b. Instead of having the `<interface>_<ver>_Mgr` class inherit from an automatically generated CORBA skeleton, we could have the `<interface>_<ver>_Mgr` class contain an instance of a subclass of that skeleton. The contained subclass would, upon being called, call an `<interface>_<ver>_Mgr` method, but it would also contain context management code.

⁹ Gokhale, Schmidt, & Moyer, page 6, col. 1.

¹⁰ See Guide to Writing DCE Applications, 2nd Ed.; by John Shirley, Wei Hu, and David Magid; pub. O'Reilly and Associates; pages 240-3.

3. The authors advise to “avoid the use of pointers in DCE interfaces since CORBA doesn’t provide a mechanism for specifying this in a portable and transparent manner.”¹¹ Pointers and handles are used commonly in the EOSDIS interfaces (in at least 33 idl files and 2 acf files). In the case of the subscription server interface, the pointers (or handles) used are the `DsSdDCESubscrHandle` context handle (18 instances) and the `DsTShStringIDL` string pointer (4 instances).
4. The authors advise to “avoid using DCE ACF attributes (e.g., `comm_status` and `fault_status`)”.¹² 9 of 11 EOSDIS ACF files use one or more of these attributes.
5. *Be careful of using DCE “varying arrays”, which only pass a part of the array from client to server.*¹³ One such array appears in a single EOSDIS IDL file.
6. The authors also found:

“Another drawback in moving from DCE to CORBA was that we were not able to duplicate the security of our DCE-based application in CORBA. This is not a fault of the tool, per se, but rather a shortcoming of current CORBA products.”¹⁴

This is still true for most ORBs. The only major ORB product that supports the CORBA security model is Iona’s *Orbix*.

¹¹ Gokhale, Schmidt, & Moyer, page 6, col. 1.

¹² Gokhale, Schmidt, & Moyer, page 5, col. 2.

¹³ Gokhale, Schmidt, & Moyer, page 6, col. 1.

¹⁴ Gokhale, Schmidt, & Moyer, page 11, col. 1.

7 Generating a CORBA Interface Definition (IDL) File

A modified copy of the <interface>.idl file must be generated for CORBA compatibility. This CORBA-compliant *.idl* file will be processed by the *idl2cpp* utility.

7.1 Feasibility

DCE-to-CORBA idl converters exist, however, not every DCE concept maps to CORBA:

In both IDLs there are concepts which are specific to the particular middleware environment. For example, CORBA's context expressions and DCE's pipes. The DCE operation attributes like `idempotent`, `broadcast` are not applicable in CORBA. DCE clients can maintain state of the server by using context handles which have no equivalence in CORBA. Most of DCE's interface implementation attributes cannot be represented in CORBA. Some of these features can be bridged if required, via extra-IDL concepts and programs providing specific services for it.¹⁵

The question arises, *of what significance are these obstacles to translation?*

Our answer is:

- A. `idempotent` methods are not of significant use in a TCP/IP environment.
- B. The `broadcast` attribute does not appear to be used by ECS.
- C. The ECS IDL files do not appear to use DCE pipes.
- D. Context handles can be facilitated with fairly straightforward management code.
- E. We can emulate `size`, `string`, and `pointer` attributes easily.
- F. We have not attempted to handle attributes not used by the subscription service IDL, such as `comm_status` and `fault_status`.
- G. We have not attempted to handle data types not used by the subscription service IDL, such as *varying arrays*.

¹⁵ Whitepaper on DCE and CORBA Interoperability:
<http://www.ismnet.com/resource/dce-corba.html>

7.2 The Tool: COTS or Custom?

To achieve our objective, we need an IDL conversion utility. Since we don't have access to such a utility at this time,¹⁶ we must develop one for this project.

Two advantages to creating our own conversion tool are:

1. We can define type mapping precisely, rather than depending on someone else's mapping.
2. A custom-made tool can be integrated with our other utilities.

¹⁶ A partial solution has been developed by Sun Microsystems for OMG. Since it still requires a code generation back end be developed for it to be fully functional, and since it makes significant use of UNIX system calls (our implementation happens to be on Windows), we are at this time pursuing other options. We may make use of it at a later date.

7.3 The Elements of IDL Conversion

Some of the more commonly required changes (for interfaces such as the *math* examples and the EOSDIS subscription server) include:

1. Remove the global `uuid` attribute and create an equivalent constant in the interface it identifies.
2. Remove the global `version` attribute and create an equivalent constant in the interface it describes.
3. Remove all square brackets.
4. Replace the combined `[in, out]` method attributes with `inout`.
7. Change pointers to single-element CORBA sequences.
8. Change arrays to unbounded sequences.
9. Change handles to sequences of sequences.
10. Insert a typedef statement for each sequence type introduced.
11. Change `import " ;` statements to `#include <>` statements, and move them out of the interface into global space.
12. Change occurrences of the DCE `byte` type to the CORBA `octet` type.
13. Change indeterminate arrays with the `size_is` attribute (called *conformant arrays* in DCE) to CORBA unbounded sequences.
14. Define `context_handle` as `sequence <octet>`.
15. Remove all occurrences of `[idempotent]`. CORBA does not support it. Though this feature may impact the reliability of remote procedure calls, it does not appear to affect the interface, or even the set of possible outcomes.
16. Simple structs which are introduced only to emulate DCE *conformant arrays* are replaced by unbounded octet sequences (the CORBA equivalent of *conformant arrays*). The original struct is expected to have been designed with specific member types and names.
17. Each reference to a custom type defined in an imported interface is prefixed with “<interface>:”.
18. Each interface definition block is closed with a semicolon.

These are the changes introduced by our “Converter” utility. We may add other changes in future versions of the utility:

19. Convert *small* and *unsigned small* to *octet*.
20. Convert *hyper* and *unsigned hyper* to *octet[4]*.

7.4 Other Issues

Visibroker's idl2cpp.exe generates a pair of header files for each interface, even if multiple interfaces share a single module¹⁷. This means that the module is defined in each header file, leading to a multiple definition problem if more than one of these headers is included in a single source file. Because the OODCE examples being tested do just that, and also because the examples don't involve modules anyway, we won't use modules in our implementation, though we will address the use of modules.

¹⁷ Observed in v.3.3; not tested for v.4.0.

8 The Client

CORBA applications typically instantiate a client object by calling a binding function generated by an IDL compiler. OODCE client applications are more elegant, inasmuch as they instantiate a client object by doing just that: *instantiating* a C++ object from a class generated by idl++. This is a convenient distinction when porting from OODCE to CORBA, because it means that a wrapper class can be created over stub code (generated by an IDL compiler) that makes the stub code look just like an OODCE class.

8.1 Miscellaneous Details

In the file <interface>C.H, the #includes must be replaced with the following:

```
#include <dce_defs.h>

#include "<interface>_c.hh" // Interfaces generated by idl2cpp
```

The *dce_defs.h* file contains CORBA-compatible definitions for DCE data types. It is similar to the *idlbase.h* file, but it #includes *corba.h* and replaces NDR types with CORBA types. For instance:

```
// typedef ndr_long_float      idl_long_float ; (REPLACED)

typedef CORBA::Double         idl_long_float ;
```

In <interface>C.H, add the following data member to the class <interface>_var.

```
<module>::<interface>_var theObject;
```

The method declarations themselves remain unmodified.

We create a corresponding source file, which we arbitrarily name <interface>C.C. This file contains constructor implementations which call `CORBA::ORB_init()` and `<module>::<interface>::_bind()`. According to the CORBA specification (section 4.6), there is no problem with calling `CORBA::ORB_init()` more than once in the same process.

8.2 Basic Form of a Inter-Middleware Invocation

In cases where DCE types map directly to CORBA types (e.g., the math example), we insert a pair of statements into each interface method that connects the OODCE-alike class to the CORBA stubs underneath:

```
<return type> ret = theObject-><method name>(
    <parameter 1>,
    <parameter 2>,
    ...
    <parameter n> );

return ret;
```

Member functions that use the `void` return type would of course omit any reference to returned data.

8.3 Advanced Type Mapping

In cases where parameter or return types do not map directly (e.g. the subscription interface), the solution gets a bit more complex:

```
<return type> ret = theObject-><method name>(
    <parameter 1 type mapper>(<parameter 1>),
    <parameter 2 type mapper>(<parameter 2>),
    ...
    <parameter n type mapper>(<parameter n>) );

return <return type mapper>(ret);
```

This is as far as we have implemented a solution. The problem can become yet more complex where parameters and/or return types are structs or pointers thereto. The likely solution is to recursively call type mappers as needed for each struct member. We have used this recursive logic, for instance in parsing structured typedefs, but we have not yet extended that logic to type mapping.

8.4 Code Examples

8.4.1 Header Files

We have included the following code listings of client header files generated by *Converter* at the end of this report:

- A. [mathC.cpp](#) (*math* interface)
- B. [DsSdDCESubscrIDLC.H](#) (subscription interface)

8.4.2 Source Files

We have included the following code listings of client source files generated by *Converter* at the end of this report:

- C. [mathC.cpp](#) (*math* interface)
- D. [DsSdDCESubscrIDLC.cpp](#) (subscription interface)
- E. [typeMappers.cpp](#)

9 The Server

In the server case, we perform a similar modification of the OODCE class which server objects are based upon. We perform similar `#include` replacements in the `<interface>S.H` file:

```
#include <dce_defs.h>

#include "<interface>_s.hh" // Interfaces generated by idl2cpp.exe
```

Instead of the base class `<interface>_<version>_ABS` inheriting from `DCEObj` and `DCEInterfaceMgr`, we have it inherit from our own `oodceBase` class and link it to a subclass of the CORBA skeleton class `_sk_<module>::_sk_<interface>`. For example, the CORBA skeleton subclass holds a reference to the OODCE abstract class (a subclass of `oodceBase`):

```
class CORBA_DsSdDCESubscrIDL : public _sk_DsSdDCESubscrIDL {
private:
    DsSdDCESubscrIDL_1_0_ABS* oodceLayer;
...
};
```

Likewise, the `oodceBase` class holds a reference to the base class of the skeleton class (`servantBaseType`):

```
#ifdef USING_POA
    typedef PortableServer_ServantBase* servantBaseType;
#else
    typedef CORBA::Object* servantBaseType;
#endif USING_POA

class oodceBase
{
protected:
    servantBaseType servant;
...
};
```

As in the client case, we may alter the constructor declarations, but we make sure the interface is not broken for OODCE applications.

The interfaces and implementations of the server methods (member functions) need not be modified.

9.1 Code Examples

9.1.1 Header Files

We have included the following code listings for server header files generated by *Converter* at the end of this report:

- A. [mathS.H](#) (*math* interface)
- B. [DsSdDCESubscrIDLS.H](#) (subscription interface)

9.1.2 Source Files

We have included the following code listing of A server source file generated by *Converter* at the end of this report:

- F. [DsSdDCESubscrIDLS.cpp](#) (subscription interface)
- G. [typeMappers.cpp](#)

10 DCEServer

A special class needs to be developed to emulate the *DCEServer* class and *theServer* object used by OODCE server programs. *theServer* is a Global Server Object (GSO) provided in the OODCE library. To emulate it, an OODCE emulation library must be provided (that contains GSOs such as *theServer*).

This class would probably be implemented in the same way regardless of the level at which an OODCE-to-CORBA abstraction layer is placed, since its definition does not vary from application to application.

Note: the *orb* and *boa* (or *poa*) member variables may be made accessible globally to applications, but this will work only for applications that use *theServer*.

Server applications that use *theServer* access its prototype via the [Server.H](#) replacement header file, which may be included only once in a given server program. The header which contains the actual class details is [DCEServer.h](#), and the implementation file is [DCEServer.cpp](#).

11 Observations

11.1 Changes in VisiBroker

With CORBA, the term ‘standard’ might lead to unrealistic expectations, because the CORBA standard has been a moving target. The VisiBroker product illustrates this issue very well.

When we ran our first round of tests (of the math examples) in September and October 1999, we were using VisiBroker v3.3. When we resumed testing in April 2000, VisiBroker v.4.0 had become available, so we ran our tests using both versions.

The most difficult obstacle in upgrading our solution to support v.4.0 was the task of adopting the POA interface, which should lead to more server portability, but for us the timing of this upgrade was unfortunate.

A significant, perhaps temporary change, is that the code generated by *idl2cpp* appears to lead to scope ambiguity (the compiler is somehow lead to allow references to data with ambiguous scope) and ambiguity of indirection (the compiler is somehow lead to allow non-pointers to use the ‘new’ operator as though they are pointers).

Significant revisions have also been introduced to the COS naming service.

11.2 Problems with Skeletons and Stubs

11.2.1 Returning references (pointers, handles, etc.)

Skeletons and stubs generated by *idl2cpp* do not always contain copies of parameters, but contain only references to parameter data which is passed in as a reference. This is not a problem in and of itself, but it requires discipline in their use.

In our design, when legacy code returns a data reference, that reference is passed across to skeletal subclass code which passes that reference to a type mapping function. The type mapping function returns its results, often a reference, down to the ORB via its skeleton. Whenever a reference is returned, there is a risk that the data to which it refers may fall out of scope, rendering the reference invalid. Care must be taken to eliminate this risk, perhaps through a careful use of static or member data.

11.2.2 Single-Element Sequences

Skeletons and stubs generated by *idl2cpp* contain an apparent bug in their implementation of single-element CORBA sequences: if an attempt is made to use an index other than one on such code, the program will crash with any issuance of an exception, though it appears that an exception is intended by the generated code. This came to our attention as we used single-element sequences to replace OODCE pointers and handles. Since offsets can be applied to pointers in C++, we attempted to use an index on the skeletons and stubs implementing the sequences that represented such pointers. This was a mistake on our part, but one that was made more difficult to recognize by an apparent bug in *idl2cpp*.

12 Conclusions

The principal of automated source-level migration from OODCE to CORBA is sound, but designing and implementing a complete solution is a challenge. We have successfully ported the ECS Subscription interface to CORBA with a fair amount of difficulty, but porting the entire set of ECS interfaces available to us would present us with issues which we chose to bypass thusfar, and almost certainly some of which we are completely unaware.

We have developed an automated solution, so that the same work need not be repeated for every interface of every application. The two automation tools we have developed for this investigation are:

We have implemented this solution on several platforms:

2. VisiBroker v.3.3 on Windows 98
3. VisiBroker v.3.3 on Windows 2000
4. VisiBroker v.4.0 on Windows 98
5. VisiBroker v.4.0 on Windows 2000

With regard to migrating more realistic examples of legacy code, we read from Gokhale, Schmidt, and Moyer:

It is hard to port from DCE to CORBA since many features do not map directly. Therefore, to achieve some degree of portability to transition from DCE to CORBA it is necessary to avoid certain DCE features. In this context, portability focuses on writing DCE applications that may some day need to port to CORBA.¹⁸

This is, of course, an unlikely scenario. Most real legacy DCE and OODCE code could not have been designed with CORBA-friendly constraints. We know that the EOSDIS project is no exception. The EOSDIS IDL and ACF files are packed with non-portable and semi-portable elements.

Even were we to be able to migrate applications that use the OODCE interface layer, more uncertainties remain because the EOSDIS project extends the OODCE interface layer with DCE-based implementations. This means that simplifying assumptions we have made in the design of our migration strategy are not entirely valid. Rather, the DCE-to-CORBA work of Washington University and Bellcore may prove more applicable, though we know their strategy makes assumptions that do not apply to the EOSDIS environment.

Though problems exist with the source-level migration strategy employed by both projects, one may be well-suited as a complement to a bridging strategy. Source-level

¹⁸ Gokhale, Schmidt, & Moyer, page 5, col. 2.

migration could be applied where the legacy code passes a portability test, and then bridging might be used where the source code itself cannot be ported in this manner.

12.1 Minor Problems

It appears that CORBA *modules* cannot be implemented at this time because Visibroker's `idl2cpp` appears to create multiple definition conflicts when a module contains multiple interfaces (see section on `idl`). This does not hinder our ability achieve the goal of source-level migration in the ECS case.

12.2 Upcoming Work

Our next steps are expected to be:

1. Facilitate mapping of structured types using a recursive algorithm wherein “type mapper” functions are invoked for each struct member as necessary.
2. Port a more realistic implementation of the subscription interface. This may be the actual ECS implementation, if available, or else a simple client/server implementation that mirrors the basic functionality of the ECS subscription system.

13 Listings of Automatically Generated Code

Code is formatted and comments trimmed in some places to enhance readability.

13.1 Client Glue Header (math interface)

```
/*-----  
  
    mathC.H  
  
-----*/  
  
#ifndef __math_1_0_Class_Included__  
#define __math_1_0_Class_Included__  
#include <dce_defs.h>      // (added for CORBA port)  
#include "math_c.hh"      // Interfaces generated by idl2cpp.exe  
                          // (added for CORBA port)  
  
#include <math.h>  
  
class math_1_0 {  
  
protected:  
    math_var theObject;  
    bool initialized;  
    DCEUuid uuid;  
    void init(DCEUuid& to = NullUuid);  
  
public:  
    // Define Class Constructors  
    math_1_0(DCEUuid& to = NullUuid);  
    math_1_0(rpc_binding_handle_t bh, DCEUuid& to = NullUuid);  
    math_1_0(rpc_binding_vector_t* bvec, DCEUuid& to = NullUuid);  
    math_1_0(DCENSiObject* nsi_obj, DCEUuid& to = NullUuid);  
    math_1_0(unsigned char* name,  
              unsigned32 syntax = rpc_c_ns_syntax_default,  
              DCEUuid& to = NullUuid);  
    math_1_0(unsigned char* netaddr,  
              unsigned char* protseq, DCEUuid& to = NullUuid);  
    math_1_0(DCEObjRefT* ref) {}  
  
    // Member functions for client  
  
    double  
    add(  
        idl_long_float a,  
        idl_long_float b );  
  
    double  
    subtract(  
        idl_long_float a,  
        idl_long_float b );  
  
    double  
    multiply(  
        idl_long_float a,  
        idl_long_float b );  
  
    double  
    divide(  
        idl_long_float a,  
        idl_long_float b );  
};  
#endif
```

13.2 Server Glue Header (math interface)

```
/*-----  
  
    mathS.H  
  
-----*/  
  
#ifndef __math_1_0_Mgr_Class_Included__  
#define __math_1_0_Mgr_Class_Included__  
#include <dce_defs.h>  
#include "math_s.hh"  
#include <math.h>  
  
class math_1_0_ABS : public _sk_math {  
private:  
    math_1_0_ABS(DCEObj& obj, DCEUuid& type):  
        _sk_math(type.getUuid()) {}  
public:  
    // Declare Class Constructors  
    math_1_0_ABS(DCEUuid& obj, DCEUuid& type):  
        _sk_math(type.getUuid()) {}  
    math_1_0_ABS(DCEUuid& type):  
        _sk_math(type.getUuid()) {}  
  
    // Declare Class pure virtual member functions  
    // These correspond to the remote procedures  
    // declared in math.idl  
    // These need to be implemented by the developer  
  
    virtual double add(  
        /* [in] */ idl_long_float a,  
        /* [in] */ idl_long_float b  
    )= 0;  
    virtual double subtract(  
        /* [in] */ idl_long_float a,  
        /* [in] */ idl_long_float b  
    )= 0;  
    virtual double multiply(  
        /* [in] */ idl_long_float a,  
        /* [in] */ idl_long_float b  
    )= 0;  
    virtual double divide(  
        /* [in] */ idl_long_float a,  
        /* [in] */ idl_long_float b  
    )= 0;  
};
```

```

class math_1_0_Mgr : public math_1_0_ABS {
public:
    // Declare Class Constructors
    math_1_0_Mgr(DCEUuid& obj):
        math_1_0_ABS(obj) {}
    math_1_0_Mgr():
        math_1_0_ABS(NullUuid) {}

    // Declare Class member functions
    // These correspond to the remote procedures
    // declared in math.idl
    // These need to be implemented by the developer

    virtual double add(
        /* [in] */ idl_long_float a,
        /* [in] */ idl_long_float b
    );
    virtual double subtract(
        /* [in] */ idl_long_float a,
        /* [in] */ idl_long_float b
    );
    virtual double multiply(
        /* [in] */ idl_long_float a,
        /* [in] */ idl_long_float b
    );
    virtual double divide(
        /* [in] */ idl_long_float a,
        /* [in] */ idl_long_float b
    );
};
#endif

```

13.3 Client Glue Implementation (math interface)

```
/*-----  
  
    mathC.cpp  
  
-----*/  
  
#include "stdafx.h"  
#include "mathC.H"  
#include <CosNaming_c.hh>  
  
//-----  
  
extern CosNaming::NamingContext_var context;  
char* nameService = "SJSU";  
  
extern void printNamingHints(const char* reason);  
extern int initServices(    CosNaming::NamingContext_var& context,  
                          char* nameRoot);  
  
//-----  
  
math_1_0::math_1_0(DCEUuid& to)  
{  
    try {  
        uuid = to;  
        init(uuid);  
    } catch(const CORBA::Exception& e) {  
        cout << "CORBA Exception during orb init: " << e << endl;  
    }  
}  
  
//-----  
  
math_1_0::math_1_0(rpc_binding_handle_t hh, DCEUuid& to)  
{  
    try {  
        uuid = to;  
        init(uuid);  
    } catch(const CORBA::Exception& e) {  
        cout << "CORBA Exception during orb init: " << e << endl;  
    }  
}  
  
//-----  
  
math_1_0::math_1_0(rpc_binding_vector_t* bvec, DCEUuid& to)  
{  
    try {  
        uuid = to;  
        init(uuid);  
    } catch(const CORBA::Exception& e) {  
        cout << "CORBA Exception during orb init: " << e << endl;  
    }  
}  
  
//-----
```

```

math_1_0::math_1_0(DCENsiObject* nsi_obj, DCEUuid& to)
{
    try {
        uuid = to;
        init(uuid);
    } catch(const CORBA::Exception& e) {
        cout << "CORBA Exception during orb init: " << e << endl;
    }
}

//-----

math_1_0::math_1_0(unsigned char* name, unsigned32 syntax, DCEUuid& to)
{
    try {
        uuid = to;
        init(uuid);
    } catch(const CORBA::Exception& e) {
        cout << "CORBA Exception during orb init: " << e << endl;
    }
}

//-----

math_1_0::math_1_0(unsigned char* netaddr, unsigned char* protseq, DCEUuid& to)
{
    try {
        uuid = to;
        init(uuid);
    } catch(const CORBA::Exception& e) {
        cout << "CORBA Exception during orb init: " << e << endl;
    }
}

//-----

void
math_1_0::init(DCEUuid& to)
{
    initialized = false;

    if (initServices(context, nameService) != 0) return;

    CosNaming::Name_var name = new CosNaming::Name();
    name->length(1);
    name[(CORBA::ULong) 0].id = (const char*) to.getUuid();

    // Resolve the Name within the server
    CORBA::Object_ptr object = context->resolve(name);
    cout << "Resolved: " << object << endl;

    while (!initialized)
    {
        try
        {
            // Bind to a math server object.

            theObject = math::_narrow(object);

```

```

        initialized = true;
    }
    catch(const CORBA::Exception& e)
    {
        cout << "CORBA Exception during bind: " << e << endl;
    }
    if (!initialized)
        Sleep(0);    // relinquish remainder of time slice.
}
}

/*-----
Member function (aka 'method') stubs.
-----*/

double math_1_0::add(
    idl_long_float a,
    idl_long_float b
)
{
    // the following code replaces the code generated by IDL++.
    if (initialized)
        return theObject->add( a, b );
    else
    {
        cout << "cannot perform operation without binding.\n";
        return -1.0;
    }
}

double
math_1_0::subtract(
    idl_long_float a,
    idl_long_float b
)
{
    // the following code replaces the code generated by IDL++.
    if (initialized)
        return theObject->subtract( a, b );
    else
    {
        cout << "cannot perform operation without binding.\n";
        return -1.0;
    }
}

double math_1_0::multiply(
    idl_long_float a,
    idl_long_float b
)
{
    // the following code replaces the code generated by IDL++.
    if (initialized)
        return theObject->multiply( a, b );
}

```

```
        else
        {
            cout << "cannot perform operation without binding.\n";
            return -1.0;
        }
    }

double math_1_0::divide(
    idl_long_float a,
    idl_long_float b
)
{
    // the following code replaces the code generated by IDL++.

    if (initialized)
        return theObject->divide( a, b );
    else
    {
        cout << "cannot perform operation without binding.\n";
        return -1.0;
    }
}
```

13.4 Client Glue Header (subscription interface)

```
/*-----  
  
    DsSdDCESubscrIDL.C.H  
  
-----*/  
  
#ifndef __DsSdDCESubscrIDL_1_0_Class_Included__  
#define __DsSdDCESubscrIDL_1_0_Class_Included__  
#include <dce_defs.h>           // (added for CORBA port)  
#include "typeMappers.h"  
#include "DsSdDCESubscrIDL_c.hh" // Interfaces generated by idl2cpp.exe  
                                // (added for CORBA port)  
  
#include "shared.h"  
#include <math.h>  
  
#ifdef USING_POA  
# include <stubs\CosNaming_c.hh>  
#else  
# include <CosNaming_c.hh>  
#endif  
  
/*    define OODCE-friendly types for use by OODCE client code. */  
  
typedef void* DsSdDCESubscrHandle;  
typedef char* DsTShStringIDL;  
typedef byteArray DsTShByteBuffer;  
  
class DsSdDCESubscrIDL_1_0 {  
  
protected:  
    DsSdDCESubscrIDL_var theObject;  
    CosNaming::NamingContext_var context;  
    bool initialized;  
    DCEUuid uuid;  
    void init(DCEUuid& to = NullUuid);  
  
public:  
    // Define Class Constructors  
    DsSdDCESubscrIDL_1_0(DCEUuid& to = NullUuid);  
    DsSdDCESubscrIDL_1_0(rpc_binding_handle_t bh, DCEUuid& to = NullUuid);  
    DsSdDCESubscrIDL_1_0(rpc_binding_vector_t* bvec, DCEUuid& to = NullUuid);  
    DsSdDCESubscrIDL_1_0(DCENsiObject* nsi_obj, DCEUuid& to = NullUuid);  
    DsSdDCESubscrIDL_1_0(unsigned char* name,  
        unsigned32 syntax = rpc_c_ns_syntax_default,  
        DCEUuid& to = NullUuid);  
    DsSdDCESubscrIDL_1_0(unsigned char* netaddr,  
        unsigned char* protseq, DCEUuid& to = DCEUuid("test servant") );  
    DsSdDCESubscrIDL_1_0(DCEObjRefT* ref)  {}  
  
    // Member functions for client  
  
    DsSdDCESubscrHandle CreateNewSubscription(  
        idl_byte data[],  
        idl_long_int len );  
  
    DsSdDCESubscrHandle CreateKnownSubscription(  
        idl_long_int eventID,  
        idl_byte client[],  
        idl_long_int len );  
  
};
```

```

DsSdDCESubscrHandle CreateSubscriptionCollector(
    idl_byte client[],
    idl_long_int len );

void DestroySubscription(
    DsSdDCESubscrHandle* c );

DsTShStringIDL DistSubmit(
    DsSdDCESubscrHandle c );

DsTShStringIDL DistCancel(
    DsSdDCESubscrHandle c );

DsTShStringIDL DistUpdate(
    DsSdDCESubscrHandle c );

DsTShByteBuffer* DistGetData(
    DsSdDCESubscrHandle c );

DsTShStringIDL DistSetData(
    DsSdDCESubscrHandle c,
    idl_byte data[],
    idl_long_int len );

DsTShByteBuffer* DistGetAllSubscriptions(
    DsSdDCESubscrHandle c );

DsTShByteBuffer* DistGetClientSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte client[],
    idl_long_int len );

DsTShByteBuffer* DistGetEventSubscriptions(
    DsSdDCESubscrHandle c,
    idl_long_int eventID );

DsTShByteBuffer* DistGetDateSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte date[],
    idl_long_int len );

DsTShStringIDL DistCancelAllSubscriptions(
    DsSdDCESubscrHandle c );

DsTShStringIDL DistCancelClientSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte client[],
    idl_long_int len );

DsTShStringIDL DistCancelEventSubscriptions(
    DsSdDCESubscrHandle c,
    idl_long_int eventID );

DsTShStringIDL DistCancelDateSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte date[],
    idl_long_int len );

void Test(
    DsSdDCESubscrHandle c );
};
#endif

```

13.5 Server Glue Header (subscription interface)

```
/*-----  
  
    DsSdDCESubscrIDL.S.H  
  
-----*/  
  
#ifndef __DsSdDCESubscrIDL_1_0_Mgr_Class_Included__  
#define __DsSdDCESubscrIDL_1_0_Mgr_Class_Included__  
#include <dce_defs.h>  
#include <DCEServer.h>  
#include "typeMappers.h"  
#include "DsSdDCESubscrIDL_s.hh"  
#include "shared.h"  
#include <math.h>  
  
/*    define OODCE-friendly types for use by OODCE server  
    manager code. */  
  
typedef void* DsSdDCESubscrHandle;  
typedef char* DsTShStringIDL;  
typedef byteArray DsTShByteBuffer;  
  
class DsSdDCESubscrIDL_1_0_ABS;  
  
class CORBA_DsSdDCESubscrIDL : public _sk_DsSdDCESubscrIDL {  
  
private:  
  
    DsSdDCESubscrIDL_1_0_ABS* oodceLayer;  
  
public:  
  
    // Declare Class Constructors  
  
    CORBA_DsSdDCESubscrIDL(  
        DCEUuid& obj, DCEUuid& type, DsSdDCESubscrIDL_1_0_ABS* ol);  
  
    CORBA_DsSdDCESubscrIDL(  
        DCEUuid& type, DsSdDCESubscrIDL_1_0_ABS* ol);  
  
    // Declare Class member functions  
    // These correspond to the remote procedures  
    // declared in DsSdDCESubscrIDL.idl  
    // These need to be implemented by the developer  
  
    CORBA::ULong  
    CreateNewSubscription(  
        const ConverterTypes::octetArray& data,  
        CORBA::Long len );  
  
    CORBA::ULong  
    CreateKnownSubscription(  
        CORBA::Long eventID,  
        const ConverterTypes::octetArray& client,  
        CORBA::Long len );  
  
    CORBA::ULong  
    CreateSubscriptionCollector(  
        const ConverterTypes::octetArray& client,
```

```

CORBA::Long len );

void DestroySubscription(
    DsSdDCESubscrIDL::DsSdDCESubscrHandlePtr& c );

DSTShBB_Interface::DsTShStringIDL* DistSubmit(
    CORBA::ULong c );

DSTShBB_Interface::DsTShStringIDL* DistCancel(
    CORBA::ULong c );

DSTShBB_Interface::DsTShStringIDL* DistUpdate(
    CORBA::ULong c );

DsSdDCESubscrIDL::DsTShByteBufferPtr* DistGetData(
    CORBA::ULong c );

DSTShBB_Interface::DsTShStringIDL* DistSetData(
    CORBA::ULong c,
    const ConverterTypes::octetArray& data,
    CORBA::Long len );

DsSdDCESubscrIDL::DsTShByteBufferPtr* DistGetAllSubscriptions(
    CORBA::ULong c );

DsSdDCESubscrIDL::DsTShByteBufferPtr* DistGetClientSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& client,
    CORBA::Long len );

DsSdDCESubscrIDL::DsTShByteBufferPtr* DistGetEventSubscriptions(
    CORBA::ULong c,
    CORBA::Long eventID );

DsSdDCESubscrIDL::DsTShByteBufferPtr* DistGetDateSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& date,
    CORBA::Long len );

DSTShBB_Interface::DsTShStringIDL* DistCancelAllSubscriptions(
    CORBA::ULong c );

DSTShBB_Interface::DsTShStringIDL* DistCancelClientSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& client,
    CORBA::Long len );

DSTShBB_Interface::DsTShStringIDL* DistCancelEventSubscriptions(
    CORBA::ULong c,
    CORBA::Long eventID );

DSTShBB_Interface::DsTShStringIDL* DistCancelDateSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& date,
    CORBA::Long len );

void Test(
    CORBA::ULong c );

};

```

```

class DsSdDCESubscrIDL_1_0_ABS : public oodceBase {
public:
    // Declare Class Constructors

    DsSdDCESubscrIDL_1_0_ABS(DCEUuid& obj, DCEUuid& type);

    DsSdDCESubscrIDL_1_0_ABS(DCEUuid& type);

    /* Declare Class pure virtual member functions.
     * These correspond to the remote procedures
     * declared in DsSdDCESubscrIDL.idl.
     * They must be implemented by the developer.
     */

    virtual DsSdDCESubscrHandle CreateNewSubscription(
        idl_byte data[],
        idl_long_int len )= 0;

    virtual DsSdDCESubscrHandle CreateKnownSubscription(
        idl_long_int eventID,
        idl_byte client[],
        idl_long_int len )= 0;

    virtual DsSdDCESubscrHandle CreateSubscriptionCollector(
        idl_byte client[],
        idl_long_int len )= 0;

    virtual void DestroySubscription(
        DsSdDCESubscrHandle* c )= 0;

    virtual DsTShStringIDL DistSubmit(
        DsSdDCESubscrHandle c )= 0;

    virtual DsTShStringIDL DistCancel(
        DsSdDCESubscrHandle c )= 0;

    virtual DsTShStringIDL DistUpdate(
        DsSdDCESubscrHandle c )= 0;

    virtual DsTShByteBuffer* DistGetData(
        DsSdDCESubscrHandle c )= 0;

    virtual DsTShStringIDL DistSetData(
        DsSdDCESubscrHandle c,
        idl_byte data[],
        idl_long_int len )= 0;

    virtual DsTShByteBuffer* DistGetAllSubscriptions(
        DsSdDCESubscrHandle c )= 0;

    virtual DsTShByteBuffer* DistGetClientSubscriptions(
        DsSdDCESubscrHandle c,
        idl_byte client[],
        idl_long_int len )= 0;

    virtual DsTShByteBuffer* DistGetEventSubscriptions(
        DsSdDCESubscrHandle c,
        idl_long_int eventID )= 0;

    virtual DsTShByteBuffer* DistGetDateSubscriptions(
        DsSdDCESubscrHandle c,

```

```

        idl_byte date[],
        idl_long_int len )= 0;

virtual DsTShStringIDL DistCancelAllSubscriptions(
    DsSdDCESubscrHandle c )= 0;

virtual DsTShStringIDL DistCancelClientSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte client[],
    idl_long_int len )= 0;

virtual DsTShStringIDL DistCancelEventSubscriptions(
    DsSdDCESubscrHandle c,
    idl_long_int eventID )= 0;

virtual DsTShStringIDL DistCancelDateSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte date[],
    idl_long_int len )= 0;

virtual void Test(
    DsSdDCESubscrHandle c )= 0;
};

class DsSdDCESubscrIDL_1_0_Mgr : public DsSdDCESubscrIDL_1_0_ABS {
public:
    // Declare Class Constructors
    DsSdDCESubscrIDL_1_0_Mgr(DCEUuid& obj):
        DsSdDCESubscrIDL_1_0_ABS(obj) {}
    DsSdDCESubscrIDL_1_0_Mgr():
        DsSdDCESubscrIDL_1_0_ABS(NullUuid) {}

    /* Declare Class virtual member functions.
     * These correspond to the remote procedures
     * declared in DsSdDCESubscrIDL.idl.
     * They must be implemented by the developer.
     */

    virtual DsSdDCESubscrHandle CreateNewSubscription(
        idl_byte data[],
        idl_long_int len );

    virtual DsSdDCESubscrHandle CreateKnownSubscription(
        idl_long_int eventID,
        idl_byte client[],
        idl_long_int len );

    virtual DsSdDCESubscrHandle CreateSubscriptionCollector(
        idl_byte client[],
        idl_long_int len );

    virtual void DestroySubscription(
        DsSdDCESubscrHandle* c );

    virtual DsTShStringIDL DistSubmit(
        DsSdDCESubscrHandle c );

    virtual DsTShStringIDL DistCancel(
        DsSdDCESubscrHandle c );

    virtual DsTShStringIDL DistUpdate(
        DsSdDCESubscrHandle c );
};

```

```

virtual DsTShByteBuffer* DistGetData(
    DsSdDCESubscrHandle c );

virtual DsTShStringIDL DistSetData(
    DsSdDCESubscrHandle c,
    idl_byte data[],
    idl_long_int len );

virtual DsTShByteBuffer* DistGetAllSubscriptions(
    DsSdDCESubscrHandle c );

virtual DsTShByteBuffer* DistGetClientSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte client[],
    idl_long_int len );

virtual DsTShByteBuffer* DistGetEventSubscriptions(
    DsSdDCESubscrHandle c,
    idl_long_int eventID );

virtual DsTShByteBuffer* DistGetDateSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte date[],
    idl_long_int len );

virtual DsTShStringIDL DistCancelAllSubscriptions(
    DsSdDCESubscrHandle c );

virtual DsTShStringIDL DistCancelClientSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte client[],
    idl_long_int len );

virtual DsTShStringIDL DistCancelEventSubscriptions(
    DsSdDCESubscrHandle c,
    idl_long_int eventID );

virtual DsTShStringIDL DistCancelDateSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte date[],
    idl_long_int len );

virtual void Test(
    DsSdDCESubscrHandle c );
};
#endif

```

13.6 Client Glue Implementation (subscription interface)

```
/*-----  
    DsSdDCESubscrIDL.cpp  
-----*/  
  
#include "stdafx.h"  
#include "DsSdDCESubscrIDL.H"  
  
//-----  
  
extern char* nameService;  
extern char* argv[];  
extern int argc;  
  
//-----  
  
DsSdDCESubscrIDL_1_0::DsSdDCESubscrIDL_1_0(DCEUuid& to)  
{  
    try {  
        uuid = to;  
        init(uuid);  
    } catch(const CORBA::Exception& e) {  
        cout << "CORBA Exception during orb init: " << e << endl;  
    }  
}  
  
//-----  
  
DsSdDCESubscrIDL_1_0::DsSdDCESubscrIDL_1_0(  
    rpc_binding_handle_t hh, DCEUuid& to)  
{  
    try {  
        uuid = to;  
        init(uuid);  
    } catch(const CORBA::Exception& e) {  
        cout << "CORBA Exception during orb init: " << e << endl;  
    }  
}  
  
//-----  
  
DsSdDCESubscrIDL_1_0::DsSdDCESubscrIDL_1_0(  
    rpc_binding_vector_t* bvec, DCEUuid& to)  
{  
    try {  
        uuid = to;  
        init(uuid);  
    } catch(const CORBA::Exception& e) {  
        cout << "CORBA Exception during orb init: " << e << endl;  
    }  
}  
  
//-----  
  
DsSdDCESubscrIDL_1_0::DsSdDCESubscrIDL_1_0(  

```

```

    DCENsiObject* nsi_obj, DCEUuid& to)
{
    try {
        uuid = to;
        init(uuid);
    } catch(const CORBA::Exception& e) {
        cout << "CORBA Exception during orb init: " << e << endl;
    }
}

//-----

DsSdDCESubscrIDL_1_0::DsSdDCESubscrIDL_1_0(
    unsigned char* name, unsigned32 syntax, DCEUuid& to)
{
    try {
        uuid = to;
        init(uuid);
    } catch(const CORBA::Exception& e) {
        cout << "CORBA Exception during orb init: " << e << endl;
    }
}

//-----

DsSdDCESubscrIDL_1_0::DsSdDCESubscrIDL_1_0(
    unsigned char* netaddr, unsigned char* protseq, DCEUuid& to)
{
    try {
        uuid = to;
        init(uuid);
    } catch(const CORBA::Exception& e) {
        cout << "CORBA Exception during orb init: " << e << endl;
    }
}

//-----

void
DsSdDCESubscrIDL_1_0::init(DCEUuid& to)
{
    initialized = false;

    try
    {
        debugPrint("Initializing ORB...");

        CORBA::ORB_ptr orb =
            CORBA::ORB_init(argc,argv);

        if (orb == NULL)
        {
            cerr << "ORB initialization failed." << endl;
            return;
        }

        debugPrint("ORB initialized.");
    }
}

```

```

#ifdef USING_POA

    //Get the manager Id
    PortableServer::ObjectId_var managerId =
        PortableServer::string_to_ObjectId("SubscriptionMgr");

    if (managerId == NULL)
    {
        cerr << "Failed to get an object ID." << endl;
        return;
    }

    debugPrint("Got the object ID.");

    // Locate a subscription server.
    // Give the full POA name and the servant ID.

    theObject =
        DsSdDCESubscrIDL::_bind("/subscription_poa",managerId);

    if (theObject == NULL)
    {
        cerr << "Failed to acquire an object reference." << endl;
        return;
    }

#else

#ifdef USING_NAMING_SVC

    debugPrint("Binding to name service...");

    context = CosNaming::NamingContext::_bind();

    debugPrint("Bind complete.");

    CosNaming::Name_var name = new CosNaming::Name();
    name->length(1);
    name[(CORBA::ULong) 0].id = (const char*) to.getUuid();

    debugPrint("Server object name:");
    debugPrint(name[(CORBA::ULong) 0].id);

    // Resolve the Name within the server

    CORBA::Object_var object = context->resolve(name);

    cout << "Resolved: [" << object << "]" << endl;

    while (!initialized)
    {
        try
        {
            // Bind to a DsSdDCESubscrIDL server object.

            cout << "Looking for server: [" << to.getUuid()
                << "]" << endl;

            // theObject = DsSdDCESubscrIDL::_bind(to.getUuid());
            theObject = DsSdDCESubscrIDL::_narrow(object);

            // theObject = DsSdDCESubscrIDL::_rebind(object);

```

```

        debugPrint ("Object narrowed successfully.");

        initialized = true;
    }
    catch(const CORBA::Exception& e)
    {
        cout << "CORBA Exception during bind: " << e << endl;
    }

    if (!initialized)
        Sleep(0);    // relinquish remainder of time slice.
}

#else !USING_NAMING_SVC

        theObject = DsSdDCESubscrIDL::_bind();

#endif !USING_NAMING_SVC
#endif !USING_POA

        debugPrint("Got the object.");
    }
    catch (const CORBA::Exception& e)
    {
        cerr << e << endl;
        return;
    }

    debugPrint("Object reference obtained?");

    initialized = true;
}

/*-----
Member function (aka 'method') stubs.
-----*/

::DsSdDCESubscrHandle
DsSdDCESubscrIDL_1_0::CreateNewSubscription(
    idl_byte data[],
    idl_long_int len )
{
    if (initialized)
    {
        debugPrint("Calling CreateNewSubscription()...");

        CORBA::ULong out =
            theObject->CreateNewSubscription(
                *to_octetArray(data, len),
                len );

        debugPrint("CreateNewSubscription() complete.");

        return (void*)out;
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
    }
}

```

```

        return (DsSdDCESubscrHandle) -1;
    }
}

::DsSdDCESubscrHandle
DsSdDCESubscrIDL_1_0::CreateKnownSubscription(
    idl_long_int eventID,
    idl_byte client[],
    idl_long_int len )
{
    if (initialized)
    {
        CORBA::ULong out =
            theObject->CreateKnownSubscription(
                eventID,
                *to_octetArray(client, len),
                len );

        return (void*)out;
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (DsSdDCESubscrHandle) -1;
    }
}

::DsSdDCESubscrHandle
DsSdDCESubscrIDL_1_0::CreateSubscriptionCollector(
    idl_byte client[],
    idl_long_int len )
{
    if (initialized)
    {
        CORBA::ULong out =
            theObject->CreateSubscriptionCollector(
                *to_octetArray(client, len),
                len );

        return (void*)out;
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (DsSdDCESubscrHandle) -1;
    }
}

void
DsSdDCESubscrIDL_1_0::DestroySubscription(
    ::DsSdDCESubscrHandle* c )
{
    if (initialized)
    {
        theObject->DestroySubscription(
            *to_uLongPtr(c) );

        return;
    }
    else

```

```

        {
            cout << "cannot perform operation without binding.\n";
            return;
        }
    }

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistSubmit(
    DsSdDCESubscrHandle c )
{
    if (initialized)
    {
        ConverterTypes::charPtr* out =
            theObject->DistSubmit(
                (CORBA::ULong)c );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistCancel(
    DsSdDCESubscrHandle c )
{
    if (initialized)
    {
        ConverterTypes::charPtr* out =
            theObject->DistCancel(
                (CORBA::ULong)c );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistUpdate(
    DsSdDCESubscrHandle c )
{
    if (initialized)
    {
        ConverterTypes::charPtr* out =
            theObject->DistUpdate(
                (CORBA::ULong)c );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

```

```

    }
}

::DsTShByteBuffer*
DsSdDCESubscrIDL_1_0::DistGetData(
    DsSdDCESubscrHandle c )
{
    if (initialized)
    {
        DsSdDCESubscrIDL::DsTShByteBufferPtr *out =
            theObject->DistGetData(
                (CORBA::ULong)c );

        return to_byteArray(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (DsTShByteBuffer*) -1;
    }
}

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistSetData(
    :DsSdDCESubscrHandle c,
    idl_byte data[],
    idl_long_int len )
{
    if (initialized)
    {
        ConverterTypes::charPtr* out =
            theObject->DistSetData(
                (CORBA::ULong)c,
                *to_octetArray(data, len),
                len );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

::DsTShByteBuffer*
DsSdDCESubscrIDL_1_0::DistGetAllSubscriptions(
    :DsSdDCESubscrHandle c )
{
    if (initialized)
    {
        DsSdDCESubscrIDL::DsTShByteBufferPtr *out =
            theObject->DistGetAllSubscriptions(
                (CORBA::ULong)c );

        return to_byteArray(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";

```

```

        return (DsTShByteBuffer*) -1;
    }
}

::DsTShByteBuffer*
DsSdDCESubscrIDL_1_0::DistGetClientSubscriptions(
    DsSdDCESubscrHandle c,
    idl_byte client[],
    idl_long_int len )
{
    if (initialized)
    {
        DsSdDCESubscrIDL::DsTShByteBufferPtr *out =
            theObject->DistGetClientSubscriptions(
                (CORBA::ULong)c,
                *to_octetArray(client, len),
                len );

        return to_byteArray(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (DsTShByteBuffer*) -1;
    }
}

::DsTShByteBuffer*
DsSdDCESubscrIDL_1_0::DistGetEventSubscriptions(
    :DsSdDCESubscrHandle c,
    idl_long_int eventID )
{
    if (initialized)
    {
        DsSdDCESubscrIDL::DsTShByteBufferPtr *out =
            theObject->DistGetEventSubscriptions(
                (CORBA::ULong)c,
                eventID );

        return to_byteArray(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (DsTShByteBuffer*) -1;
    }
}

::DsTShByteBuffer*
DsSdDCESubscrIDL_1_0::DistGetDateSubscriptions(
    :DsSdDCESubscrHandle c,
    idl_byte date[],
    idl_long_int len )
{
    if (initialized)
    {
        DsSdDCESubscrIDL::DsTShByteBufferPtr *out =
            theObject->DistGetDateSubscriptions(
                (CORBA::ULong)c,
                *to_octetArray(date, len),

```

```

        len );

        return to_byteArray(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (DsTShByteBuffer*) -1;
    }
}

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistCancelAllSubscriptions(
    :DsSdDCESubscrHandle c )
{
    if (initialized)
    {
        ConverterTypes::charPtr* out =
            theObject->DistCancelAllSubscriptions(
                (CORBA::ULong)c );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistCancelClientSubscriptions(
    :DsSdDCESubscrHandle c,
    idl_byte client[],
    idl_long_int len )
{
    if (initialized)
    {
        ConverterTypes::charPtr* out =
            theObject->DistCancelClientSubscriptions(
                (CORBA::ULong)c,
                *to_octetArray(client, len),
                len );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistCancelEventSubscriptions(
    :DsSdDCESubscrHandle c,
    idl_long_int eventID )
{
    if (initialized)
    {

```

```

        ConverterTypes::charPtr* out =
            theObject->DistCancelEventSubscriptions(
                (CORBA::ULong)c,
                eventID );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

::DsTShStringIDL
DsSdDCESubscrIDL_1_0::DistCancelDateSubscriptions(
    ::DsSdDCESubscrHandle c,
    idl_byte date[],
    idl_long_int len )
{
    if (initialized)
    {
        ConverterTypes::charPtr* out =
            theObject->DistCancelDateSubscriptions(
                (CORBA::ULong)c,
                *to_octetArray(date, len),
                len );

        return to_char(*out);
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return (::DsTShStringIDL) -1;
    }
}

void
DsSdDCESubscrIDL_1_0::Test(
    DsSdDCESubscrHandle c )
{
    // the following code replaces the code generated by IDL++.

    if (initialized)
    {
        theObject->Test(
            (CORBA::ULong)c );

        return;
    }
    else
    {
        cout << "cannot perform operation without binding.\n";
        return;
    }
}

```

13.7 Server Glue Implementation (subscription interface)

```
/*-----  
  
    DsSdDCESubscrIDLs.cpp  
  
-----*/  
  
#include "DsSdDCESubscrIDLs.H"  
#include "typeMappers.h"  
  
//-----  
  
CORBA::ULong  
CORBA_DsSdDCESubscrIDL::CreateNewSubscription(  
    const ConverterTypes::octetArray& data,  
    CORBA::Long len )  
{  
    debugPrint("In CreateNewSubscription!");  
  
    CORBA::ULong out = (CORBA::ULong)  
        oodceLayer->CreateNewSubscription(  
            to_idl_byte(data),  
            len );  
  
    return out;  
}  
  
CORBA::ULong  
CORBA_DsSdDCESubscrIDL::CreateKnownSubscription(  
    CORBA::Long eventID,  
    const ConverterTypes::octetArray& client,  
    CORBA::Long len )  
{  
    CORBA::ULong out = (CORBA::ULong)  
        oodceLayer->CreateKnownSubscription(  
            eventID,  
            to_idl_byte(client),  
            len );  
  
    return out;  
}  
  
CORBA::ULong  
CORBA_DsSdDCESubscrIDL::CreateSubscriptionCollector(  
    const ConverterTypes::octetArray& client,  
    CORBA::Long len )  
{  
    CORBA::ULong out = (CORBA::ULong)  
        oodceLayer->CreateSubscriptionCollector(  
            to_idl_byte(client),  
            len );  
  
    return out;  
}  
  
void  
CORBA_DsSdDCESubscrIDL::DestroySubscription(  
    DsSdDCESubscrIDL::DsSdDCESubscrHandlePtr& c )
```

```

{
    oodceLayer->DestroySubscription(
        to_ptr(c) );

    return;
}

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistSubmit(
    CORBA::ULong c )
{
    DsTShStringIDL out =
        oodceLayer->DistSubmit(
            (void*)c );

    debugPrint("Returned from DistSubmit:");
    debugPrint(out);

    return to_charPtr(out);
}

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistCancel(
    CORBA::ULong c )
{
    DsTShStringIDL out =
        oodceLayer->DistCancel(
            (void*)c );

    debugPrint("Returned from DistCancel:");
    debugPrint(out);

    return to_charPtr(out);
}

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistUpdate(
    CORBA::ULong c )
{
    DsTShStringIDL out =
        oodceLayer->DistUpdate(
            (void*)c );

    debugPrint("Returned from DistUpdate:");
    debugPrint(out);

    return to_charPtr(out);
}

DsSdDCESubscrIDL::DsTShByteBufferPtr*
CORBA_DsSdDCESubscrIDL::DistGetData(
    CORBA::ULong c )
{
    static ::DsTShByteBuffer* out = NULL;

    if (out != NULL) delete out;

    out = oodceLayer->DistGetData(
        (void*)c );
}

```

```

        return to_octetArrayPtr(*out);
    }

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistSetData(
    CORBA::ULong c,
    const ConverterTypes::octetArray& data,
    CORBA::Long len )
{
    DsTShStringIDL out =
        oodceLayer->DistSetData(
            (void*)c,
            to_idl_byte(data),
            len );

    debugPrint("Returned from DistSetData:");
    debugPrint(out);

    return to_charPtr(out);
}

DsSdDCESubscrIDL::DsTShByteBufferPtr*
CORBA_DsSdDCESubscrIDL::DistGetAllSubscriptions(
    CORBA::ULong c )
{
    static ::DsTShByteBuffer* out = NULL;

    if (out != NULL) delete out;

    out = oodceLayer->DistGetAllSubscriptions(
        (void*)c );

    return to_octetArrayPtr(*out);
}

DsSdDCESubscrIDL::DsTShByteBufferPtr*
CORBA_DsSdDCESubscrIDL::DistGetClientSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& client,
    CORBA::Long len )
{
    static ::DsTShByteBuffer* out = NULL;

    if (out != NULL) delete out;

    out = oodceLayer->DistGetClientSubscriptions(
        (void*)c,
        to_idl_byte(client),
        len );

    return to_octetArrayPtr(*out);
}

DsSdDCESubscrIDL::DsTShByteBufferPtr*
CORBA_DsSdDCESubscrIDL::DistGetEventSubscriptions(
    CORBA::ULong c,
    CORBA::Long eventID )
{
    static ::DsTShByteBuffer* out = NULL;

    if (out != NULL) delete out;

```

```

        out = oodceLayer->DistGetEventSubscriptions(
            (void*)c,
            eventID );

    return to_octetArrayPtr(*out);
}

DsSdDCESubscrIDL::DsTShByteBufferPtr*
CORBA_DsSdDCESubscrIDL::DistGetDateSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& date,
    CORBA::Long len )
{
    static ::DsTShByteBuffer* out = NULL;

    if (out != NULL) delete out;

    out = oodceLayer->DistGetDateSubscriptions(
        (void*)c,
        to_idl_byte(date),
        len );

    return to_octetArrayPtr(*out);
}

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistCancelAllSubscriptions(
    CORBA::ULong c )
{
    ::DsTShStringIDL out =
        oodceLayer->DistCancelAllSubscriptions(
            (void*)c );

    debugPrint("Returned from DistCancelAllSubscriptions:");
    debugPrint(out);

    return to_charPtr(out);
}

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistCancelClientSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& client,
    CORBA::Long len )
{
    ::DsTShStringIDL out =
        oodceLayer->DistCancelClientSubscriptions(
            (void*)c,
            to_idl_byte(client),
            len );

    debugPrint("Returned from DistCancelClientSubscriptions:");
    debugPrint(out);

    return to_charPtr(out);
}

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistCancelEventSubscriptions(
    CORBA::ULong c,
    CORBA::Long eventID )
{
    ::DsTShStringIDL out =

```

```

        oodceLayer->DistCancelEventSubscriptions(
            (void*)c,
            eventID
        );

        debugPrint("Returned from DistCancelEventSubscriptions:");
        debugPrint(out);

        return to_charPtr(out);
    }

ConverterTypes::charPtr*
CORBA_DsSdDCESubscrIDL::DistCancelDateSubscriptions(
    CORBA::ULong c,
    const ConverterTypes::octetArray& date,
    CORBA::Long len )
{
    ::DsTShStringIDL out =
        oodceLayer->DistCancelDateSubscriptions(
            (void*)c,
            to_idl_byte(date),
            len );

    debugPrint("Returned from DistCancelDateSubscriptions:");
    debugPrint(out);

    return to_charPtr(out);
}

void
CORBA_DsSdDCESubscrIDL::Test(
    CORBA::ULong c )
{
    oodceLayer->Test(
        (void*)c );

    return;
}

```

```

CORBA_DsSdDCESubscrIDL::CORBA_DsSdDCESubscrIDL(
    DCEUuid& obj,
    DCEUuid& type,
    DsSdDCESubscrIDL_1_0_ABS* ol )
{
    oodceLayer = ol;
    _object_name(obj.getUuid());
}

CORBA_DsSdDCESubscrIDL::CORBA_DsSdDCESubscrIDL(
    DCEUuid& type,
    DsSdDCESubscrIDL_1_0_ABS* ol )
{
    oodceLayer = ol;
    _object_name(type.getUuid());
}

//-----
DsSdDCESubscrIDL_1_0_ABS(DCEUuid& obj, DCEUuid& type)
{
    oodceBase(
        new CORBA_DsSdDCESubscrIDL(obj, type, this),
        obj.getUuid() );
}

DsSdDCESubscrIDL_1_0_ABS(DCEUuid& type)
{
    oodceBase(
        new CORBA_DsSdDCESubscrIDL(type, this),
        type.getUuid() );
}

```

14 Listings of Framework Code

14.1 Server.H

```
/*-----  
    Server.H  
-----*/  
#include <DCEServer.h>  
/*-----  
    theServer  
    This object emulates OODCE's "theServer" Global Server  
    Object (GSO).  
-----*/  
extern DCEServer* theServer;  
//-----
```

14.2DCEServer.h

```
/*-----  
  
    DCEServer.H  
  
    This class emulates the DCEServer class provided by OODCE.  
-----*/  
  
#include <corba.h>  
  
#ifdef USING_POA  
# include <stubs\CosNaming_c.hh>  
#else  
# include <CosNaming_c.hh>  
#endif  
  
#ifndef DCE_SERVER_H  
#define DCE_SERVER_H  
  
typedef void* pthread_addr_t;  
  
#ifdef USING_POA  
    typedef PortableServer_ServantBase* servantBaseType;  
#else  
    typedef CORBA::Object* servantBaseType;  
#endif USING_POA  
  
//-----  
  
class oodceBase  
{  
protected:  
  
    servantBaseType servant;  
    char name[64];  
  
public:  
  
    oodceBase(void):servant(0L) { *name = '\0'; }  
  
    oodceBase(    servantBaseType s,  
                char*          n ):servant(s)  
                { strcpy(name, n); }  
  
    servantBaseType getServant(void) {return servant;}  
    char* getName() {return name;}  
  
};
```

```

class DCEServer
{
public:
    static pthread_addr_t ServerCleanup (void* param);

    DCEServer(void) {serverInitialized = false;}

    void RegisterObject(oodceBase&);
    void Listen(void);
    void UseProtocol (unsigned char*) {};

private:

    int ServerInit(void);

    CORBA::ORB_ptr orb;
    bool serverInitialized;

#ifdef USING_POA
    PortableServer::POAManager_var rootManager;
    PortableServer::POA_var rootPOA, myPOA;
#else
    CORBA::BOA_var boa;
    CosNaming::NamingContext_var context;
    CORBA::Object_var nameServiceObj;
#endif USING_POA

};

#endif DCE_SERVER_H

```

14.3DCEServer.cpp

```
/*-----  
    DCEServer.cpp  
    This class emulates the DCEServer class provided by OODCE.  
-----*/  
  
#include <DCEServer.h>  
  
//-----  
  
extern char *argv[];  
extern int  argc;  
extern void debugPrint(char *message);  
  
//-----  
  
void printNamingHints(const char* reason)  
{  
    cout << reason << endl;  
    cout << "\t" << "-Make sure a Name Server process is running" << endl  
        << "\t" << "-Make sure you are setting the -SVCnameroot parameter"  
        << endl;  
}  
  
//-----  
  
int  
DCEServer::ServerInit(void)  
{  
    debugPrint("Initializing GSO...");  
  
    try  
    {  
        debugPrint("Initializing ORB...");  
  
        // Initialize the ORB and BOA  
        orb = CORBA::ORB_init(argc, argv);  
  
        if (orb == NULL) return -1;  
  
#ifdef USING_POA  
        debugPrint("Looking for the root POA ...");  
  
        /*  
        To implement objects using the POA, at least one POA  
        object must exist on the server. To ensure that a  
        POA exists, a rootPOA is provided during the ORB  
        initialization. This POA uses the default POA policies...  
        */  
  
        CORBA::Object_var obj =  
            orb->resolve_initial_references("RootPOA");  
  
        debugPrint("Resolved initial references.");  
  
        // get a reference to the root POA
```

```

rootPOA = PortableServer::POA::_narrow(obj);

debugPrint("Reference to root POA acquired.");

// Set policies for myPOA

CORBA::PolicyList policies;
policies.length(1);
policies [(CORBA::ULong)0 ] =
    rootPOA->create_lifespan_policy(
        PortableServer::PERSISTENT);

// Create myPOA with the right policies

rootManager = rootPOA->the_POAManager();
myPOA = rootPOA->create_POA(
    "subscription_poa",
    rootManager,
    policies );

debugPrint("POA created.");

#else

boa = orb->BOA_init(argc, argv);

try
{
    nameServiceObj =
        orb->resolve_initial_references("NameService");
}
catch(const CORBA::ORB::InvalidName)
{
    cerr << "Caught exception CORBA::ORB::InvalidName" << endl;
    printNamingHints("ORB::resolve_initial_references failed");
    return 1;
}

if (nameServiceObj == CORBA::Object::_nil())
{
    printNamingHints(
        "ORB::resolve_initial_references returned NULL");
    return 1;
}

debugPrint("Naming Service object acquired.");

// resolve_initial_references raises
// (InvalidName, CORBA::SystemException);

context = CosNaming::NamingContext::_narrow(nameServiceObj);

if (context == CosNaming::NamingContext::_nil())
{
    printNamingHints(
        "CosNamingContext::_narrow returned NULL");
    return 1;
}

debugPrint("Naming context acquired.");

#endif USING_POA

```

```

    }
    catch (const CORBA::Exception& e)
    {
        cerr << e << endl;
    }

    serverInitialized = true;

    return 0;
}

//-----
// this method is designed for use with MFC threads.

pthread_addr_t
DCEServer::ServerCleanup (void* param)
{
    return param;
}

//-----

void DCEServer::RegisterObject(oodceBase& oodceObject)
{
    servantBaseType servant = oodceObject.getServant();

    if (!serverInitialized)
    {
        debugPrint("Initializing Server...");
        ServerInit();
    }

    if (serverInitialized)
    {
        debugPrint("GSO initialized.");
    }

    try
    {
#ifdef USING_POA
        // Decide on the ID for the servant

        PortableServer::ObjectId_var managerId =
            PortableServer::string_to_ObjectId(
                oodceObject.getName());

        debugPrint("Object ID created.");

        // Activate the servant with the ID on myPOA
        myPOA->activate_object_with_id(managerId, servant);
        // myPOA->activate_object(servant);

        debugPrint("Servant activated?");

        // Activate the POA Manager.
        // hopefully there's no problem with repeating this step.

        rootManager->activate();

        debugPrint("Is servant ready?");

        cout << myPOA->servant_to_reference(servant)
            << " is ready." << endl;
#endif
    }
}

```

```

        debugPrint("POA manager activated.");
#else
        CosNaming::Name_var name = new CosNaming::Name();
        name->length(1);
        name[(CORBA::ULong) 0].id =
            (const char*) servant->_object_name();

        // Bind the Name to the server
        context->rebind(name, servant);

        // Export the newly created object.
        boa->obj_is_ready(servant);
#endif USING_POA
    }
    catch (const CORBA::Exception& e)
    {
        cerr << e << endl;
    }
}

//-----
void DCEServer::Listen(void)
{
    // Wait for incoming requests

    debugPrint("Listening ...");

    try
    {
        //Wait for incoming requests.

#ifdef USING_POA
        orb->run();
#else
        boa->impl_is_ready();
#endif USING_POA
    }
    catch (const CORBA::Exception& e)
    {
        cerr << e << endl;
    }
}

/*-----
    theServer
    This static object emulates OODCE's "theServer" Global
    Server Object (GSO).
-----*/

DCEServer* theServer = new DCEServer() ;

```

14.4 typeMappers.cpp

```
/*
 * typeMappers.cpp
 *
 */

#include "shared.h"
#include <dce_defs.h>
#include "typeMappers.h"

ConverterTypes::octetArrayPtr*
to_octetArrayPtr(idl_byte *in, idl_long_int length)
{
    debugPrint("Creating a new octet array ptr...");

    static ConverterTypes::octetArray *mid = NULL;

    if (mid != NULL) delete mid;

    mid = to_octetArray(in, length);

    static ConverterTypes::octetArrayPtr *out = NULL;

    if (out != NULL) delete out;

    out = new ConverterTypes::octetArrayPtr(1, mid);

    debugPrint("Octet array ptr created.");

    return out;
}

ConverterTypes::octetArray*
to_octetArray(idl_byte *in, idl_long_int length)
{
    debugPrint("Creating a new octet array...");

    ConverterTypes::octetArray *out =
        new ConverterTypes::octetArray(
            length,
            length,
            (CORBA::Octet*) in);

    debugPrint("Octet array created.");

    return out;
}

/*
 * This function is not used in the subscription server example,
 * because context handles are better represented by long
 * integers than by octet sequences.
 *
 */

ConverterTypes::octetArrayPtr*
to_octetArrayPtr(void **in)
{
    debugPrint("In to_octetArrayPtr() ...");
}
```

```

ConverterTypes::octetArray *mid =
    to_octetArray(*in);

ConverterTypes::octetArrayPtr *out =
    new ConverterTypes::octetArrayPtr(
        1,
        mid );

debugPrint("Exiting to_octetArrayPtr() ...");

return out;
}

ConverterTypes::uLongPtr*
to_uLongPtr(void **in)
{
    debugPrint("In to_uLongPtr() ...");

    ConverterTypes::uLongPtr *out =
        new ConverterTypes::uLongPtr(
            1,
            (CORBA::ULong*)in );

    debugPrint("Exiting to_uLongPtr() ...");

    return out;
}

/*
 * to_octetArray(void*,long)
 *
 */

ConverterTypes::octetArray*
to_octetArray(
    void *in,
    idl_long_int length )
{
    debugPrint("In to_octetArray ...");

    debugPrint((char*)in);

    static ConverterTypes::octetArray *out =
        new ConverterTypes::octetArray(
            length,
            length,
            (CORBA::Octet*)in);

    debugPrint("Exiting to_octetArray() ...");

    return out;
}

/*
 * to_octetArray(void*)
 *
 * Warning: this function presumes that void pointers
 * actually point to null-terminated (c-style) strings.
 * For other types of data, use the following function:
 *
 * to_octetArray(void*,long)

```

```

*
* This function is used commonly in the subscription server example,
* by both stubs and skeletons. This would give rise to a problem if
* clients expected to make use of the data referenced by the pointer,
* but DCE clients are not supposed to use the data referenced by
* context handles, so we're ok.
*
* Actually ... this function is not used in the subscription server
* example, because context handles are better represented by long
* integers than by octet sequences.
*/

```

```

ConverterTypes::octetArray*
to_octetArray(void *in)
{
    debugPrint("In to_octetArray ...");

    unsigned long length = strlen((char*)in);

    debugPrint((char*)in);

    static ConverterTypes::octetArray *out =
        new ConverterTypes::octetArray(
            length,
            length,
            (CORBA::Octet*)in);

    debugPrint("Exiting to_octetArray() ...");

    return out;
}

```

```

/*
* to_octetArray(void*,ConverterTypes::octetArray**)
*
* This function demonstrates an 'inout' strategy for
* avoiding the data loss problems associated with
* returning local data.
*
* Warning: this function presumes that void pointers
* actually point to null-terminated (c-style) strings.
* For other types of data, use the following function:
*
* to_octetArray(void*,long)
*/

```

```

ConverterTypes::octetArray*
to_octetArray(
    void *in,
    ConverterTypes::octetArray** out )
{
    debugPrint("In to_octetArray ...");

    unsigned long length = strlen((char*)in);

    debugPrint((char*)in);

    *out = new ConverterTypes::octetArray(
        length,
        length,
        (CORBA::Octet*)in);

    debugPrint("Exiting to_octetArray() ...");
}

```

```

        return *out;
    }

ConverterTypes::charPtr*
to_charPtr(char* in)
{
    debugPrint("Entering to_charPtr()...");

    ConverterTypes::charPtr* out =
        new ConverterTypes::charPtr(1, (CORBA::Char*)in);

    debugPrint("Returning from to_charPtr().");

    return out;
}

ConverterTypes::octetArrayPtr*
to_octetArrayPtr(const byteArray& in)
{
    static ConverterTypes::octetArray buffer;

    buffer = ConverterTypes::octetArray(in.length, in.length, (CORBA::Octet*)
in.bytes);

    static ConverterTypes::octetArrayPtr* out = NULL;

    if (out != NULL) delete out;

    out = new ConverterTypes::octetArrayPtr(1, &buffer);

    debugPrint("returning from to_DstShByteBufferPtr");

    return out;
}

/*
 * CORBA-to-DCE Mapping Functions
 */

/*
 * to_idl_byte
 *
 * In the subscription server example, this function is used to convert
 * data for all special parameters in the skeletons. This is because the
 * special data types used for parameters are all byte arrays.
 */

idl_byte*
to_idl_byte(const ConverterTypes::octetArray& in)
{
    static idl_byte *out = NULL;

    if (out != NULL) delete out;

    out = new idl_byte[in.length()];

    for (CORBA::ULong i = 0; i < in.length(); i++)
    {
        out[i] = in[i];
    }
}

```

```

        }

        return out;
    }

/*
 * to_void
 *
 * This function simply calls to_idl_byte and typecasts
 * the returned data.
 *
 * This function is not used in the subscription server example,
 * because context handles are better represented by long
 * integers than by octet sequences.
 */

void*
to_void(ConverterTypes::octetArray& in)
{
    return (void*) to_idl_byte(in);
}

void*
to_void(CORBA::ULong in)
{
    return (void*)in;
}

/*
 * to_ptr
 *
 * This function simply calls to_void and returns the
 * address of the returned data.
 *
 * This function is not used in the subscription server example,
 * because context handles are better represented by long
 * integers than by octet sequences.
 */

void**
to_ptr(ConverterTypes::octetArrayPtr& in)
{
    debugPrint("Converting ConverterTypes::octetArrayPtr to a void ptr ...");

    static void* out = NULL;

    if (out != NULL) delete out;

    out = to_void(in[0]);

    debugPrint("DsSdDCESubscrHandlePtr converted to void ptr:");
    debugPrint((char*)out);

    return &out;
}

void**
to_ptr(ConverterTypes::uLongPtr& in)
{
    debugPrint("Converting ConverterTypes::uLongPtr to a void ptr ...");

    static void* out = (void*)in[0];

```

```

        debugPrint("DsSdDCESubscrHandlePtr converted to void ptr:");
        debugPrint((char*)out);

        return &out;
    }

    /*
     *
     * In the subscription server example, this function is used to convert
     * return data from several stubs.
     *
     * The returned character string is null-terminated.
     */

    char*
    to_char(ConverterTypes::charPtr& in)
    {
        char* out = new char[in.length()];

        for (unsigned long i = 0; i < in.length(); i++)
        {
            out[i] = in[i];
        }

        out[in.length()] = '\0';

        return out;
    }

    /*
     * In the subscription server example, this function is used to convert
     * return data from several stubs.
     *
     */

    byteArray*
    to_byteArray(const ConverterTypes::octetArrayPtr& in)
    {
        return to_byteArray(in[0]);
    }

    /*
     * This function is not used directly by the subscription server
     * example, but it is called indirectly via the preceding function.
     *
     */

    byteArray*
    to_byteArray(const ConverterTypes::octetArray& in)
    {
        byteArray* out = new byteArray;

        out->length = in.length();
        out->bytes = new unsigned char[in.length()];

        for (CORBA::ULong i = 0; i < in.length(); i++)
            out->bytes[i] = in[i];

        return out;
    }

```

15 Output

15.1 Output From the Simple Math Example

```
C:\Inprise\math\application\Debug>start mathServer  
C:\Inprise\math\application\Debug>start mathServer *  
  
C:\Inprise\math\application\Debug>mathClient  
Looking for server: [0d51fb58-b1ec-11d0-8b04-9b9d3034aa77]  
result from math1 = 15.000000  
Looking for server: [530d1138-6838-11d2-b0f6-9b9d7912aa77]  
result from matha1 = 10.000000  
Looking for server: [cceba372-683b-11d2-a063-9b9d7912aa77]  
result from math2 = 50.000000  
Looking for server: [04295a00-683c-11d2-9041-9b9d7912aa77]  
result from matha2 = 4.000000  
  
C:\Inprise\math\application\Debug>
```

15.2 Output From the Threaded Math Example

```
C:\Inprise\math\application\Debug>start mathServer

C:\Inprise\math\application\Debug>start mathServer *

C:\Inprise\math\application\Debug>client
(1) Connecting to math1...
(2) Connecting to math1...
Resolved: Repository ID: IDL:math:1.0
Repository ID: IDL:math:1.0
  Object name: 0d51fb58-b1ec-11d0-8b04-9b9d3034aa77

  Object name: 0d51fb58-b1ec-11d0-8b04-9b9d3034aa77

(1) result from math1 = 15
(1) Connecting to math1...
Resolved: Repository ID: IDL:matha:1.0
  Object name: 530d1138-6838-11d2-b0f6-9b9d7912aa77

(2) result from math1 = 15
(2) Connecting to math1...
Resolved: Repository ID: IDL:matha:1.0
  Object name: 530d1138-6838-11d2-b0f6-9b9d7912aa77

(1) result from matha1 = 10
(1) Connecting to math2...
Resolved: Repository ID: IDL:math:1.0
  Object name: cceba372-683b-11d2-a063-9b9d7912aa77

(2) result from matha1 = 10
(2) Connecting to math2...
Resolved: Repository ID: IDL:math:1.0
  Object name: cceba372-683b-11d2-a063-9b9d7912aa77

(1) result from math2 = 50
(1) Connecting to matha2...
Resolved: Repository ID: IDL:matha:1.0
  Object name: 04295a00-683c-11d2-9041-9b9d7912aa77

(2) result from math2 = 50
(2) Connecting to matha2...
Resolved: Repository ID: IDL:matha:1.0
  Object name: 04295a00-683c-11d2-9041-9b9d7912aa77

(1) result from matha2 = 4
(2) result from matha2 = 4
GetExitCodeThread(): Invalid thread handle.
GetExitCodeThread(): Invalid thread handle.

C:\Inprise\math\application\Debug>
```

16 ORBS with C++ Bindings

Table created from direct investigation on October 28, 1999. Updated on April 11, 2000. ORBs supporting IRIX are featured in the first block; discontinued ORBs are in the third block.

Developer	Product	URL	Platforms
IRIX ORBs			
Borland	VisiBroker C++	http://www.borland.com/visibroker/	Solaris, HP-UX, AIX, IRIX, Digital UNIX, DG/UX, Linux, NT, Win-95/98, ...
BEA	WebLogic Enterprise	http://www.beasys.com/products/weblogic/enterprise/index.html	HP-UX, Solaris, NT, AIX, Sequent Dynix, Compaq Tru64 UNIX, IRIX
Non-IRIX ORBs			
Sun	NetDynamics	www.netdynamics.com	NT, Solaris, AIX
Iona	Orbix C++	http://www.ionaiportal.com/suite/orbixc.htm	Solaris, HP-UX, NT, Win-95/98
PeerLogic	LiveContent BROKER for C and C++	http://www.peerlogic.com/products/products.html	AIX, HP-UX, OpenVMS, Stratus VOS, Solaris, UnixWare, Win-95/NT
IBM	WebSphere Application Server, Enterprise Edition	http://www-4.ibm.com/software/webservers/appserv/enterprise.html	AIX, Solaris, NT
Decision Systems Group	Arachne ORB	http://www.arachne.org/	Sources for NT, UNIX, Mac
Secant	Extreme Enterprise Server for C++	http://www.secant.com/products/extreme_enterprise_server.html	NT, Solaris, ...?
Distributed Object Computing G., WUSTl	TAO	http://www.cs.wustl.edu/~schmidt/TAO.html	not found
Bionic Buffalo Corp.	Tatanka ORB	http://www.tatanka.com/prod/info/orb1.htm	Linux, Solaris, FreeBSD, or Windows
TIBCO	TIB/ObjectBus	http://www.tibco.com/products/messaging/objectbus/index.html	AIX, Solaris, NT, HP-UX, Digital UNIX?
AT&T	omniORB2	http://www.uk.research.att.com/omniORB/omniORB.html	Solaris, Win-95/NT, Linux
OOC	ORBacus	http://www.ooc.com/faq/orbacus.html	Sources for NT, UNIX
Expersoft	CORBAplus for C++	http://www.expersoft.com/Products/CORBAC/corbac.htm	NT, Win-95, UNIX
Discontinued ORBs			
TRW	UNAS for C++	http://www.trw.com/unas/	Digital UNIX, HP-UX, IRIX, Solaris, NT
Nortel	RCP-ORB	http://www.nortelnetworks.com/products/01/rcp-orb/	no access to information
Chorus	COOL-ORB	not found	not found
Sun	NEO	http://www.sun.com/software/neo/	product discontinued
HP	ORB Plus	Obsolete; not found	HP-UX, Solaris, NT
IBM	ComponentBroker/DSOM	Obsolete; see WebSphere	see WebSphere
Digital	ObjectBroker	sold to BEA Systems	see WebLogic Enterprise

17 References

DCE and OODCE

HP Object Oriented DCE C++ Class Library Programmer's Guide. November 1994. E1194; Second Edition.

Understanding DCE, by Rosenberry, Kenney, & Fisher. Pub. O'Reilly & Associates, Inc., May 1993.

Guide to Writing DCE Applications, by Shirley, Hu, & Magid. Pub. O'Reilly & Associates, Inc., Nov 1994.

CORBA

Instant CORBA, by Orfali, Harkey, & Edwards. Pub. John Wiley & Sons, Inc., 1997.

The *VisiBroker* tutorial at <http://www.borland.com/visibroker/tutorial/>.

ORB Core Feature Matrix: <http://www.vex.net/~ben/corba/orbmatrix.html>

CORBA Security Implementations

VisiBroker SSL Pack (not a CORBA Security Service implementation):

<http://www.borland.com/visibroker/ssl/>

Secant Extreme Enterprise Server:

http://www.secant.com/products/extreme_enterprise_server.html

Tatanka ORB:

<http://www.tatanka.com/prod/info/orb1.htm>

OrbixSecurity:

<http://www.iona.com/products/docs/orbixsecurity30whitepaper1.pdf>

ORBAssec SL2 (Java-based):

<http://www.adiron.com/ORBAssec2.1.html>

DCE-CORBA Bridging and Migration

Inprise DCE-CORBA Bridge:

<http://www.borland.com/entera/dcecorba/>

<http://www.borland.com/entera/papers/dce-corba/>

<http://www.borland.com/techpubs/entera/dce-corba10/index.html>

Whitepaper on DCE and CORBA Interoperability (focused on bridging):

<http://www.ismnet.com/resource/dce-corba.html>

Software Tools for Automating the Migration From DCE to CORBA, by Aniruddha Gokhale, Douglas C. Schmidt, and Stanley Moyer (Aug 3, 1998):

<http://makarov.east.hitc.com/middle/ISS-97.pdf>

Strategies for Migrating from DCE to CORBA, by Aniruddha Gokhale and Douglas C. Schmidt (Aug 3, 1998):

<http://makarov.east.hitc.com/middle/DCE2CORBA.pdf>

DCOM

Microsoft DCOM Page: <http://www.microsoft.com/Com/tech/DCOM.asp>

DCOM Technical Overview:

http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/backgrnd/html/msdn_dco_mtec.htm

Microsoft Windows Programming

Windows threading information is provided at

http://msdn.microsoft.com/isapi/msdnlib2.idc?theURL=/library/sdkdoc/winbase/prothred_86sz.htm.

MFC threading information is provided at

<http://msdn.microsoft.com/library/wcedoc/wcemfc/cwinthrd.htm>.

ECS (EOSDIS Core System)

Workshop Presentations:

http://edhs1.gsfc.nasa.gov/waisdata/toc/workshop_toc.html