AR-009-480

DSTO-TN-0025

# Development Of An Interoperable, Survivable Pilot DCE Application

B. McClure and D. O'Dea



DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

# Development Of An Interoperable, Survivable Pilot DCE Application

*B. McClure and D. O'Dea*

**Information Technology Division**
**Electronics and Surveillance Research Laboratory**

DSTO-TN-0025

**ABSTRACT**

This report discusses some problems encountered in developing and porting a pilot Distributed Computing Environment (DCE) program to several different platforms. The pilot program demonstrates the interoperability, portability and survivability features of OSF's DCE.

**APPROVED FOR PUBLIC RELEASE**

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

*Published by*

*DSTO Electronics and Surveillance Research Laboratory*
*PO Box 1500*
Salisbury. South Australia, Australia 5108

*Telephone: (08) 259 7053*
*Fax: (08) 259 5619*


*AR-009-480*
*December 1995*

**APPROVED FOR PUBLIC RELEASE**

# Development Of An Interoperable, Survivable Pilot DCE Application

## EXECUTIVE SUMMARY

Under the charter of the Distributed Systems Technology Task (ADF 94/151) and the "Defence Demonstrator" project of the Co-operative Research Centre for Distributed Systems Technology (DSTC), development is planned of several application demonstrators based on **OSF's Distributed Computing Environment (DCE).** These programs will be expected to demonstrate the interoperability, portability, survivability and security features of DCE. In support of this activity a pilot DCE application was developed and ported to multiple platforms.

The features to be demonstrated by the pilot are *survivability* against server failure, *portability* and *interoperability*. For the purposes of this report, **survivability** requires that a client program that utilises the services of a server program can continue processing in the event of a server or communications failure. **Portability** requires that a program developed on one type of computer can be recompiled on another type of computer with a minimum of changes to the source code associated with that program. Ideally, no changes should be required in the source code. **Interoperability** requires that a program running on one type of computer can communicate with a program running on another type of computer, without any consideration or knowledge of what type of computer it is communicating with.

The pilot program allows the user to run the client program on HP, DEC Alpha, SUN or OS/2 machines, and obtain services from a server running on any of those platforms. Further, if a failure were to occur such that the client/server communications were interrupted, then the client program would automatically establish communications with another server (if available) and continue processing. Thus survivability against server failure is achieved using replicated interchangeable servers. This capability is simplified by the use of stateless server functions; hence the problems of maintaining consistency between replicated servers are not addressed.

This report discusses some problems encountered in developing and porting the pilot program. Some of the potential pitfalls of cross platform DCE development are addressed and solutions to some of the problems encountered are recommended.

The experience gained throughout this exercise will be a useful precursor to the further DCE development to be conducted under the Distributed Systems Task (ADF 94/151).

# DEPARTMENT OF DEFENCE

DM93/6081

Canberra ACT 2600

SERIAL NO: 2649

Defense Technical Information Center
8725 John J. Kingman Road, Suite 0944
FT Belvoir VA 22060 - 6218

Attention: DTIC-BCR

For Information: CONDS (W)

***AUSTRALIAN DEPARTMENT OF DEFENCE PUBLIC RELEASE DOCUMENTS***

1. Please find enclosed 2 copies of each of the Australian Department of Defence reports listed below. They are forwarded for either your retention or for distribution.

| AR NUMBER | | |
|---|---|---|
| 009-232 | 2 copies | Unlimited/Unclassified |
| 009-327 | 2 copies | Unlimited/Unclassified |
| 009-347 | 2 copies | Unlimited/Unclassified |
| 009-358 | 2 copies | Unlimited/Unclassified |
| 009-359 | 2 copies | Unlimited/Unclassified |
| 009-374 | 2 copies | Unlimited/Unclassified |
| 009-421 | 2 copies | Unlimited/Unclassified |
| 009-423 | 2 copies | Unlimited/Unclassified |
| 009-424 | 2 copies | Unlimited/Unclassified |
| 009-433 | 2 copies | Unlimited/Unclassified |
| 009-436 | 2 copies | Unlimited/Unclassified |
| 009-442 | 2 copies | Unlimited/Unclassified |
| 009-444 | 2 copies | Unlimited/Unclassified |
| 009-445 | 2 copies | Unlimited/Unclassified |
| 009-454 | 2 copies | Unlimited/Unclassified |
| 009-455 | 2 copies | Unlimited/Unclassified |
| 009-456 | 2 copies | Unlimited/Unclassified |
| 009-460 | 2 copies | Unlimited/Unclassified |
| 009-465 | 2 copies | Unlimited/Unclassified |
| 009-471 | 2 copies | Unlimited/Unclassified |
| 009-478 | 2 copies | Unlimited/Unclassified |
| 009-480 | 2 copies | Unlimited/Unclassified |

Judith M Kennett

Manager
Document Exchange Centre
Defence Information Services Network Ofi

23 Apr 1996

# GLOSSARY

| | |
|---|---|
| ADF | Australian Defence Force |
| API | Application Programming Interface |
| CDS | Cell Directory Service |
| CRC | Cooperative Research Centre |
| DCE | Distributed Computing Environment |
| DDMS | Distributed Database Management System |
| DES | Data Encryption Standard |
| DOS | Disk Operating System |
| DSTC | Distributed Systems Technology Cooperative Research Centre |
| DTS | Distributed Time Service |
| GUI | Graphical User Interface |
| HPFS | High Performance File System |
| IP | Internet Protocol |
| LAN | Local Area Network |
| OSF | Open Software Foundation |
| POSIX | Portable Operating System Interface |
| RPC | Remote Procedure Call |
| rpcd | Remote Procedure Call Daemon (server) |
| TCP | Transport Control Protocol |
| UDP | User Datagram Protocol |
| UUID | Universal Unique Identifier |
| WAN | Wide Area Network |

# CONTENTS

# FIGURES

# 1. INTRODUCTION

Under the charter of the Distributed Systems Technology Task (ADF 94/151) and the "Defence Demonstrator" project of the Co-operative Research Centre for Distributed Systems Technology (DSTC), development is planned of several application demonstrators based on **OSF's Distributed Computing Environment (DCE).** These programs will be expected to demonstrate the interoperability, portability, survivability and security features of DCE. In support of this activity a pilot DCE application was developed and ported to multiple platforms. The purpose of this report is to highlight the problems encountered during the development and porting of the pilot program.

The features to be demonstrated by the pilot are survivability against server failure, portability and interoperability. For the purposes of this report, **survivability** requires that a client program that utilises the services of a server program can continue processing in the event of a server failure or a communications failure. **Portability** requires that a program developed on one type of computer can be recompiled on another type of computer with a minimum of changes to the source code associated with that program. Ideally, no changes should be required in the source code. **Interoperability** requires that a program running on one type of computer can communicate with a program running on another type of computer, without any consideration or knowledge of what type of computer it is communicating with. The pilot program developed allows the user to run the client on HP, DEC Alpha, SUN or OS/2 machines, and obtain services from a server running on any of those platforms. Further, if a failure were to occur such that the client/server communications broke down, then the client program would automatically establish communications with another server (if available) and continue processing.

An attempt has been made to include useful information and pointers to reference material, in addition to the direct findings of the authors, to make this report more useful to application developers and system managers.

# 2. BACKGROUND

This section is provided as a brief primer for readers unfamiliar with the DCE technology and terminology.

## 2.1 The history of DCE

DCE is a development environment for open distributed processing. It was developed by the Open Software Foundation (OSF) which is a consortium of over 360 members including commercial, government, and university groups. Distributed computing is not new. Several thousands of sites around the world claim to be running distributed computing services. However, these implementations have few relevant standards, are based on proprietary solutions, and only offer partial interoperability solutions, relying on the expertise of system developers to patch systems together.

The main challenges of any distributed system are listed below:

- Interoperability.
- Consistent support for heterogenous architectures from multiple vendors.
- Identification and management of distributed services and resources.
- Provision of security.
- Support for end users, application developers and system administrators.
- High availability and good performance.

OSF issued its request for DCE technology in 1989. A year later, after several submissions and reviews, an OSF DCE technology was selected. It then took another two years for OSF to deliver version 1.0 of its DCE product. The first OSF DCE vendor products began to emerge in late 1992 to mid 1993, but these versions suffered teething problems and reliable OSF DCE implementations were not widely available until late 1993 to early 1994.

## 2.2 Basic services provided by DCE

The DCE services provide a platform independent communications mechanism called Remote Procedure Call (RPC). Several vendors offered an RPC mechanism before DCE was available, including SUNs TI-RPC[1]. In the current DCE release from OSF, RPC services are implemented using sockets. Only the Transmission Control Protocol (TCP) and the User Data Protocol (UDP) are supported in the general release.

Multi-processing is important for any communications system's performance and is provided in DCE by the use of POSIX threads. A common Application Programming Interface (API) is provided for thread related operations. The actual implementation of the threads varies from platform to platform, as some operating systems already have kernel level threads.

The DCE also provides a Cell Directory Service (CDS). The main role of the CDS is to provide a location independent naming service. This allows clients to find servers in the network, without having IP addresses hard coded. Some other attributes are provided to enable sophisticated look up schemes to be used to locate services.

The DCE provides a security service. The Authentication service is based upon Kerberos version 5, which uses the DES encryption standard. This security service enables authentication and authorisation to be performed. The US domestic version of DCE allows all application information in RPCs to be encrypted.

The DCE provides a Distributed Time Service (DTS) which provides a globally (cell-wide) common, monatonicaly increasing time.

The DCE services are integrated. For example, the security service makes use of CDS and DTS. Figure 1. depicts the relationship between the DCE services.

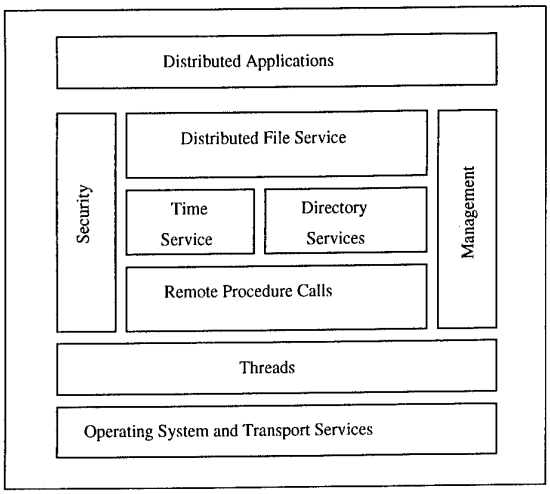



*Figure 1. The services provided by OSF's DCE.*

DCE services are based upon the concept of a cell. A group of computers can be configured into a single DCE cell. They then share a common security registry that maintains security information for all users, computers, and other principles such as application servers. Communication with computers outside the cell is possible, but allows restricted authentication and authorisation services. Figure 2. depicts a typical DCE cell setup.

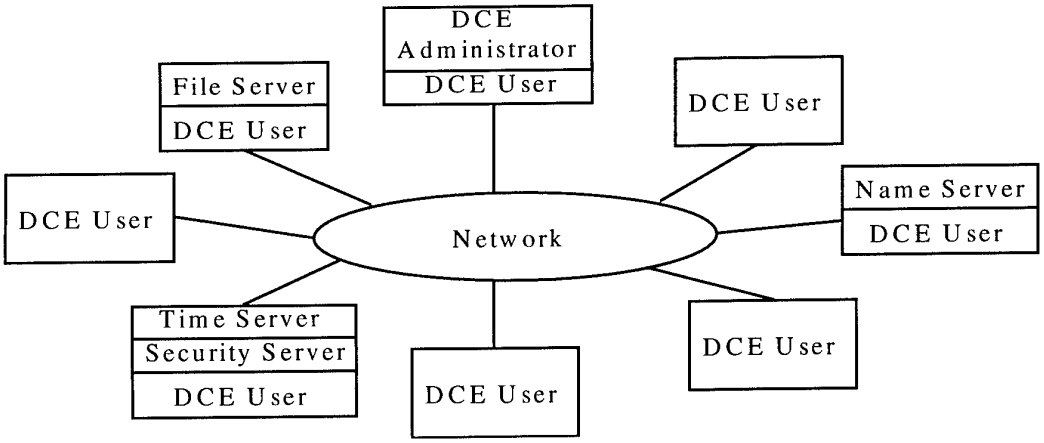


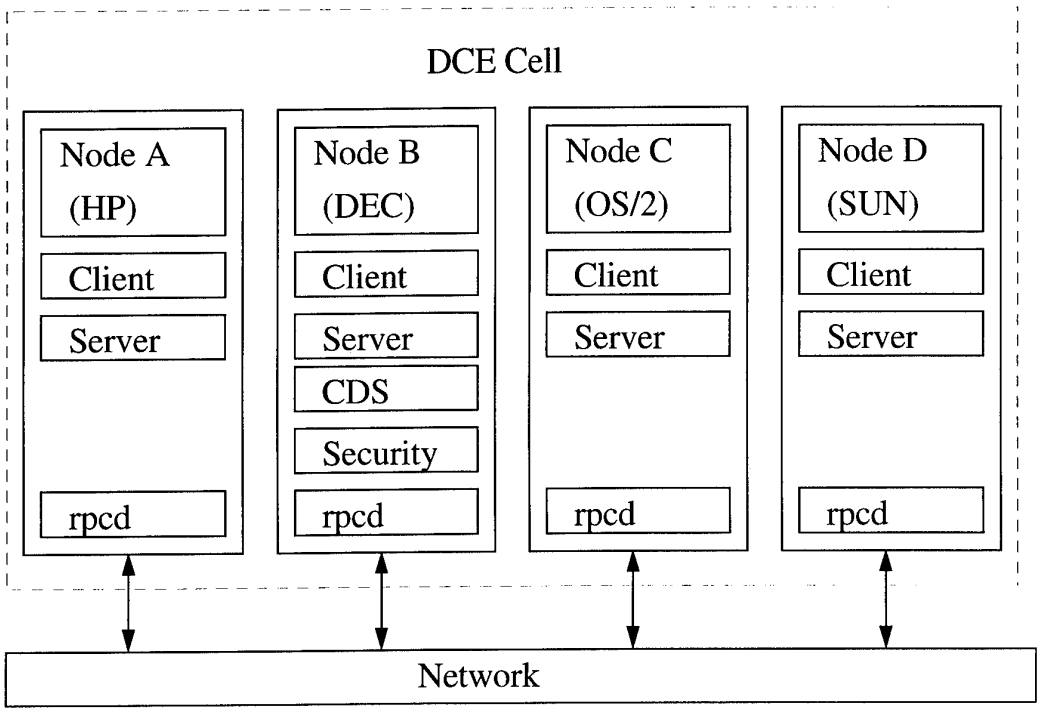

*Figure 2. A distributed computing cell.*

See [2] for further DCE background information.

# 3. APPROACH

## 3.1 Client/Server Architecture

The initial pilot program was developed under the OS/2 operating system, on IBM PS/2 hardware. It was based on an example program [3] . In this example program, the client program is invoked with two or more parameters which are sent to the server via DCE RPC, added by the server and passed back to the client where the result is displayed on standard output.




NOTE: The Remote Procedure Call Daemon (rpcd) is present on every computer participating in a DCE cell. The **rpcd** keeps a record of servers running on a single computer.

*Figure 3. The client / server architecture utilised for the pilot system.*

For the purpose of demonstrating the survivability feature the client was modified to continuously send two random numbers to the server to be summed and the sum to be returned. This behaviour simulates stateless but continuous client - server interaction. If the server maintains state, that is, retains information between procedure calls, maintaining data consistency between replicated servers becomes complex. Refer to [4,5] for example for some solutions to this problem in the context of Distributed

Database Management Systems (DDMS). The program was modified to make the client able to recover from server errors (for example: time-outs, server stopped, or communications failure), and locate and use another server if one exists within the DCE cell. If no other server is reachable by the client, it will endlessly attempt to find a server until one is found or the client is terminated by the user.

A very simple failure detection mechanism is used in the pilot program: a timeout. Getting failure detection to work reliably in a real, non-trivial system is a difficult problem and many researchers have suggested solutions [6,7]. Using a timeout still requires a choice of timeout value. If the timeout is too short, many false alarms will occur. If the timeout is too long the client program will be unnecessarily delayed, waiting to determine that the server is in fact unavailable. The choice of the correct value will be influenced by system requirements and system architecture. The approach taken to this choice for the pilot is outlined in section 4.2.

When the OS/2 version was satisfactory, it was ported to HP-800 series (HP-UX 9.0), then DEC Alpha AXP (OSF/1 v2.0) and finally SUN (Solaris 2.4) platforms. The three UNIX versions were kept source code compatible; that is, only the makefiles differed from platform to platform. However, the UNIX version diverged from the OS/2 version particularly in the area of signals - see below.

## 3.2 Simplifications

A number of complications in the porting process were avoided by the following simplifications of the pilot application:

(i) The client did not interact with a Graphical User Interface (GUI),

(ii) All program code was written in the 'C' language,

(iii) The server was stateless,

(iv) Authentication was not implemented.

Items i and ii above are likely to be of concern to developers of real applications. There is a discussion of the GUI issue under the next section. Development of DCE programs using languages other than 'C' are expected to be addressed by subsequent reports developed under this task. Item iii is discussed in section 3.1. The last simplification is not likely to affect the ability of real programs to be ported to the

various platforms. This is because the DCE security API is not closely related to the hardware or operating system implementation.

# 4. FAILURE RECOVERY

In the process of setting up the communications between a client and server, the client must make several RPCs to the DCE servers, such as the security server, CDS and rpcd. Any one of these RPCs could potentially fail. Good client software should detect any such failures and take appropriate action (such as alerting the system administrator). The pilot program was designed only to guard against application server failure and thus the prime concern was to detect failures with RPC calls to the application server and respond by finding another server.

## 4.1 Detecting Server Failure

It was established in section 3.1 that reliably detecting server failure in the presence of unpredictable network delays, is very difficult. For the pilot, a simple scheme was used. Failure detection in the pilot is based upon a timeout. The value of the timeout was reduced from the default value to make the client program respond quickly to server failure. This was done for the purpose of demonstrating the recovery feature. The timeout was changed using the rpc_mgmt_set_com_timeout() library call to a lower value (1 on a scale of 0 to 10). This value was determined by experiment with both a LAN and WAN connection. This value does not represent an optimal value, but simply one that works. Clearly, any distributed system that requires high availability in the presence of unpredictable network delays needs to be designed with a well thought out failure detection scheme. Such a scheme will depend upon the system requirements, system architecture and the communications environment in which the system must operate. Providing a reliable, generally applicable failure detection scheme is beyond the scope of this report.

## 4.2 Locating a Server on the same Computer after Failure

DCE Servers are found by clients using a process called **binding** (see section 4.3). In the first instance this was done on an OS/2 machine which was configured as a stand alone cell. This meant that all the client needed to do after an RPC time-out was re-bind to a different server running on the same computer. DCE allows for this by providing the rpc_ep_resolve_binding call, which randomly selects a binding from the end-point map [8] (the end-point map is a list of all servers running on a computer, maintained by rpcd). To facilitate the recovery, the client went through the entire

binding process after failure. Locating a server on the same machine is only of use if more than one server is likely to be running on a given computer.

## 4.3 The Binding Process

The binding process consists of the following steps:

(i) Begin the search for the server by passing an entry name and the interface UUID to the DCE runtime, which contacts CDS and attempts to find servers that match the criterion,

(ii) Selectively import a binding (ie: the IP address of the server) if the search was successful,

(iii) Resolve the binding to obtain a specific endpoint (ie: the specific IP port that the server is listening on - there may be several identical servers on the same node, with the same IP address, but listening on different ports).

(iv) Import the binding handle.

Figure 4. depicts a situation where two servers reside on a different computer (or node) than the client.

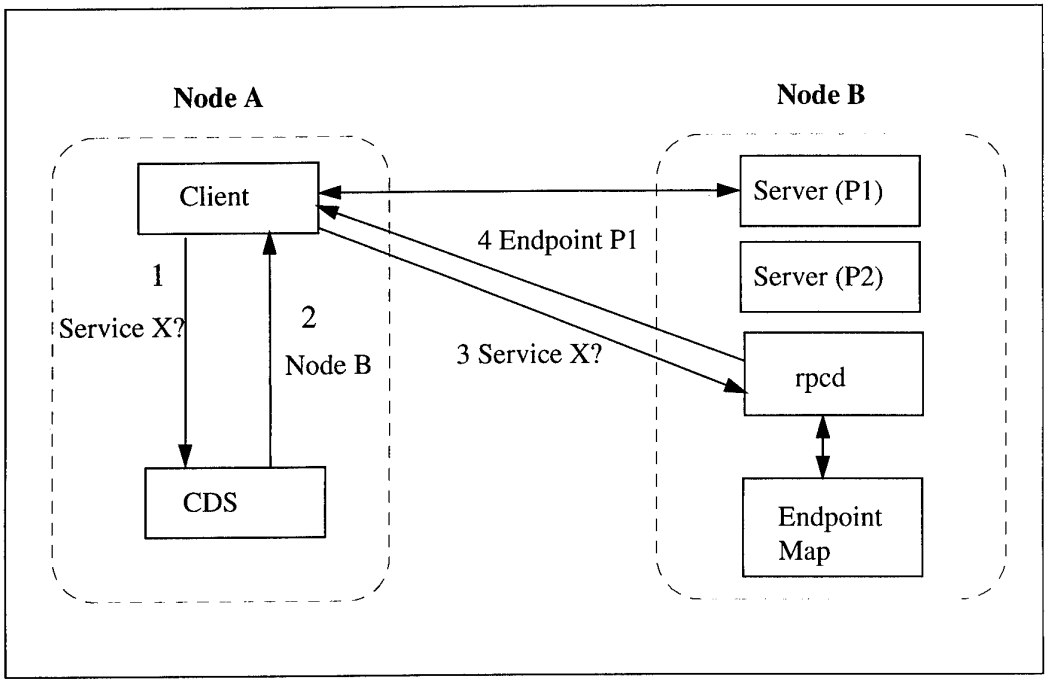



*Figure 4. The binding process.*

As the diagram shows, the CDS informs the client program that a server resides at Node B. From this, the client may select to *resolve* the binding to a particular endpoint (or port address) and then import the binding handle. The call to resolve the binding handle uses a pseudo random algorithm for selecting the endpoint. If the binding handle is not resolved, then the first endpoint available will always be used. The binding handle then enables the client to specify the server in subsequent RPC calls. The binding handle is retained until either it becomes invalid (server failure) or the client program is finished communicating with that server.

## 4.4 Locating A Server On Another Computer

The client can find another server on a different host via the CDS. To facilitate this, servers must first export the service (or *interface*) to the CDS in a way that will allow clients on other hosts to find it. In this case, the server was exporting the 'Implicit'

interface to a directory 'Servers', with the entry name of MathI. This system did work, but even with a low time-out value, it could take significant amounts of time for the client to rebind successfully to a new server. One reason for this is that previously failed servers leave stale entries in the CDS and endpoint map. Stale entries point to servers which are no longer running at the same location. This problem is discussed under Section 5.2 Clean Server Termination.

# 5. PORTING ISSUES

In porting the pilot program from the initial OS/2 implementation to the various UNIX platforms, several issues and difficulties arose. These were dealt with by a variety of methods. The problems encountered and the solutions utilised/developed are each described in turn below.

## 5.1 Porting From OS/2 To UNIX

When the pilot program was ported from OS/2 to UNIX there were several changes that had to be made to the source code before it would compile. These are common porting issues not directly related to DCE.

(i) Carriage Returns had to be stripped from the end of each line of source code.

(ii) Some of the #include statements in the code referred to DOS style file names. These had to be modified to the full name of the header files. This problem can be avoided by using the HPFS version of the DCE header files.

(iii) Some references to conio.h had to be removed. For example, the kbhit() function is not available under UNIX. In the original program, the client will terminate if the user presses a key. In the UNIX version, the user must press 'Control C' to stop the client program.

## 5.2 Clean Server Termination

In a development environment, clients and servers are often started and stopped. Each time a server is stopped it will leave a stale entry in the CDS and the end-point map of the host on which it is running. The server can be written in such a way as to remove the endpoint upon failure or termination but it is generally not recommended that the CDS entry be removed because of the overhead that this imposes on the CDS and because of the slow propagation rate of CDS changes. The CDS database is designed on the premise that there will be few servers exporting and removing entries from the CDS but many clients performing searches. In a large DCE cell, the CDS can

become strained and exhibit poor performance if servers are continually exporting and removing entries each time they are started and stopped.

In the pilot program, several signals were caught to enable the server to detect if the user had pressed 'Control C' and this allowed the server to remove the endpoint before terminating. Of course this can only be done if the server terminates properly, not if it crashes. In the initial version (OS/2) the signals were caught with a signal handler and the cleanup code was implemented in the signal handler. See the OS/2 server listing for details. This approach of directly catching signals is only valid for OS/2 and caused some problems when porting to the UNIX platforms. See Section 5.3 for details.

## 5.3 Signal Interface Compatibility

The requirement to catch signals caused portability problems, as there is no generally portable way to deal with signals in DCE. The standard UNIX way to deal with signals is to install a signal handler regardless of whether the signal is synchronous or not. This technique is not applicable to DCE-based applications. Synchronous signals are caused by an error in the program such as a floating point overflow or an illegal instruction. Asynchronous signals are caused by external events such as a packet arriving at an Ethernet port. No synchronous signals were handled by the Demonstrator. Installing signal handlers is not recommended in DCE applications since DCE uses POSIX threads (pthreads). POSIX threads are incompatible with signal handlers for asynchronous signals, because if two threads try to install signal handlers for the same (asynchronous) signal, then the last one to install the handler will catch the signal. Furthermore, DCE RPC calls will behave unpredictably in the interrupt state of a processor (ie. in a signal handler). For more information on handling signals in DCE refer to chapter 6 (Programming with Threads) in [9].

The recommended way to deal with asynchronous signals for DCE programs is to create a separate thread that blocks the signals of interest using sigprocmask() and then calls sigwait(). Sigwait() returns when one of the blocked signals occurs. The sigwait() routine was in fact used for the HP-UX, OSF/1 and SUN implementations of the pilot program. However, as stated in the previous section this is not how signals were handled in the OS/2 pilot program.

The OS/2 implementation of DCE does not support the sigprocmask() or sigwait() system calls described above but rather allows asynchronous signals to be caught using handlers and synchronous signals to be caught using the TRY / CATCH / FINALLY mechanism. Because of these differences, the OS/2 pilot program caught the Control C signal by installing a signal handler. If source code compatibility is

desired between OS/2 and other DCE platforms, then some of the signal handling code should be switched in or out (for example by the use of #define statements) depending upon the target platform.

Details of the OS/2 signal handling interface are described in [10].

## 5.4 IDL Types

When porting from HP-UX to DEC OSF/1, a problem was encountered with IDL types. In this instance, the type "long" in the IDL file was translated to idl_long_int in the mathi.h file by the IDL compiler. The DEC 'C' compiler then declared an error when the type of the arguments to the RPC function implementation in the server did not match that of the arguments declared in mathi.h. The HP compiler did not declare this to be an error. On reading chapter 17 (Interface Definition Language) of the OSF DCE Application Development Guide, it became apparent that all types are translated into IDL types. To ensure portability and to reduce platform dependant bugs the following rules should be observed:

(i) Use standard types in the IDL file (int, long, char, etc.)

(ii) Use equivalent types in the client and server code (idl_long_int, idl_char, etc.)

(iii) Cast all constants in RPC calls to correct IDL type (idl_char, etc.)

## 5.5 The Xlib Problem

The following is from [11].

"Code that can work properly in a multi-threaded environment, the kind of environment you get when you start a server with more than one active thread at a time, is called "thread-safe" code. All the code in a multi-threaded library must be thread-safe. The standard UNIX libraries that come with DCE are mostly thread-safe, but most other libraries are not. For example, the X and Motif libraries are not thread-safe, so all calls to these libraries must be isolated, either by using locks or by ensuring that all calls to those libraries are made inside a single thread."

In general, client code will be multi-threaded, but only one thread will be controlled by the programmers code. The other threads will operate within the stub (the stub is code generated automatically from the interface definition). Provided that the client code does not explicitly create a thread and the stub code does not use the Xlib library (unlikely) then the above conditions should be satisfied. However, there are

situations where it is necessary for a server to display graphical information or for a client to also act as a server and thus have multiple threads. These kind of problems were avoided during development of the pilot program. One solution for this problem is as follows:

(i) Ensure that only one thread makes calls to the X library

(ii) Create common data structures that represent the 'state' of the graphical interface.

(iii) Keep these structures as simple and as few as possible

(iv) Protect these structures with mutexes as necessary

### 5.6 Useful Utilities

During the development of the pilot program, it was often required to know the state of the end-point map or the CDS entries that related to the server. The following two commands display this information and were found to be useful:

(i) rpccp show entry /.:/Servers/MathI

This displays the current bindings in the CDS. When there are bindings in CDS for which there are no endpoints - the lookup will fail. (Servers/MathI is the name exported by the server to the CDS for the pilot program).

(ii) rpccp show mapping | grep Implicit

This displays the current endpoints on the local host, which relate to the pilot program. (Implicit is the name of the interface). If there are stale endpoints in this list, which no longer point to valid servers then the lookup may time-out on the first attempt. Since the rpc runtime implements a pseudo-random algorithm for selecting bindings from the endpoint map, repeated binding attempts will eventually succeed in the presence of stale bindings.

## 6. RECOMMENDATIONS

The following recommendations are made based on the experience gained in developing and porting the pilot program.

(i) Standard IDL types should be used within the client and server code for all references to RPC arguments.

(ii) The use of signals and exceptions should be planned early in a development (detailed design phase) to avoid portability problems, particularly if an OS/2 implementation is envisaged. These problems may be avoided by the use of third-party DCE development tools such as Entera or Objdce.

(iii) Plan all uses of user interface related library calls. This can be a major portability issue, since a given user interface library may not be available on a given platform. In addition, most GUI libraries are not thread-safe, hence, multi-threaded calls to a GUI library should be avoided using the steps outlined in Section 5.5 or some alternative method.

# 7. CONCLUSION

This report has demonstrated the survivability, portability and interoperability features of the Open Software Foundation's Distributed Computing Environment. Survivability against server failure was achieved using replicated interchangeable servers. This capability was simplified by the use of stateless server functions; hence the problems of maintaining consistency between replicated servers has not been addressed. Some of the potential pitfalls of cross platform DCE development have been addressed and solutions to some of the problems encountered have been recommended.

The experience gained throughout this exercise will be a useful precursor to the further DCE development to be conducted under the Distributed Systems Task (ADF 94/151).

# 8. FUTURE WORK

Two Defence DCE Demonstrators are currently being developed by DSTO. Both of these systems will require integration of DCE and Xlib services. This will provide experience with the pitfalls of using Xlib and other libraries with DCE. It will also enable DSTO to further verify some of the claims made about DCE.

Porting the pilot application used for this report to an IBM mainframe environment may prove to be a useful addition to this report.

# 9. ACKNOWLEDGMENT

The work described in this report was conducted in collaboration with the Cooperative Research Centre for Distributed Systems Technology (DSTC). The authors wish to acknowledge the contribution of the DSTC in this task. The CRC for Distributed Systems Technology is funded in part by the Cooperative Research Centres Program through the Department of the Prime Minister and the Cabinet of the Commonwealth Government of Australia.

# REFERENCES


1. 'Solaris ONC: Design and Implementation of Transport-Independent RPC', Sunsoft Inc Technical Report, September 1991.

2. W. Rosenberry, D. Kenney and G. Fisher, 'Understanding DCE', O'Reilly and Associates, Inc, 1992.

3. Sample Code Developing DCE Applications for AIX, OS/2 and Windows, IBM International Technical Support Organisation (Austin), February 1994.

4. R. Elmasri and S. Navathe, 'Fundamentals of Database Systems', The Benjamin/Cummings Publishing Company, Inc., 1989.

5. M. Ozsu and P. Valduriez. 'Principles of Distributed Database Systems', Prentice-Hall International Inc., 1991.

6. Tushar Chandra and Sam Toueg 'Unreliable Failure Detectors for Reliable Distributed Systems', Proceedings of the tenth ACM Symposium on principles of Distributed Computing, pp 325-340. ACM press, August 1991.

7. L. Sabel and K. Marzullo 'Simulating Fail-Stop in Asyncronous Distributed Systems', Cornell University, 22 June 1994.

8. Open Software Foundation 'OSF DCE Application Development Reference', Prentice Hall PTR, 1993. p 2-98.

9. Open Software Foundation 'OSF DCE Application Development Guide', Prentice Hall PTR, 1993.

10. README file distributed with IBM Distributed Computing Environment (DCE) for OS/2, Version 1.0.

11. 'Objtran Programmer's Guide', Citibank Distributed Processing Technology, April 1994.

# Appendix A: Make files

## A.1 HP-700; HP-UX:

```
#
# Makefile for the HP-UX Version of the Implicit Binding Example.
#
#  Modified by B.McClure for HP_UX using info from the HP manual
#  and the objtran HP makefile template.
#

# interface name
INTFC     = mathi

# Set defaults for the DCE stuff. If this isn't where DCE is on your
# system, change these lines. These should be correct for most HP-UX
systems.
DCE_BIN = /opt/dcelocal/bin
DCE_INCLUDE = /usr/include/reentrant
DCE_LIB = /opt/dcelocal/lib
DCE_DIR = /opt/dcelocal

# All libraries for DCE
DCE_L = -ldce
# Other libraries
SYS_L = -lndbm -lM -lc_r
DCE_LIBS = $(DCE_L) $(SYS_L)

# command names
CC   = /bin/c89
IDL = $(DCE_DIR)/hptools/bin/idl
DEL = rm -f

# Debugging flags to use for C code
CDEBUG = -g
# Optimizer flags for all code
OPT =

# command flags
CFLAGS = -Aa +DA1.0 +DS1.0 $(OPT) -Dsigned= \
  -D_HPUX_SOURCE -D_REENTRANT \
  -I. -I$(DCE_INCLUDE)

# Flags to use when linking C code that contains no C++
C_LDFLAGS =  -W l,-z -W l,-Bimmediate -L$(DCE_LIB)

# Flags for IDL
IDLFLAGS = -keep all -I. -I$(DCE_INCLUDE)


# object files
SOBJS     = $(INTFC)_s.o $(INTFC)_m.o $(INTFC)_sstub.o
COBJS     = $(INTFC)_c.o $(INTFC)_cstub.o
```

```
# targets
all: $(INTFC)_s $(INTFC)_c

$(INTFC)_s: $(SOBJS)
  $(CC) $(SOBJS) -o $(INTFC)_s $(DCE_LIBS)
$(INTFC)_c: $(COBJS)
  $(CC) $(COBJS) -o $(INTFC)_c $(DCE_LIBS)
$(INTFC).h $(INTFC)_sstub.o $(INTFC)_cstub.o: $(INTFC).idl
$(INTFC).acf
  $(IDL) $(IDLFLAGS) $(INTFC).idl
clean:
  -$(DEL) core *.o $(INTFC)_?stub.c $(INTFC).h

# dependencies
$(INTFC)_m.o: $(INTFC)_m.c $(INTFC).h
$(INTFC)_s.o: $(INTFC)_s.c $(INTFC).h
$(INTFC)_c.o: $(INTFC)_c.c $(INTFC).h
```

## A.2 Alpha AXP; OSF/1 v2.0:

```
#
# Makefile for the Alpha Version of the Implicit Binding Example.
#
#  Modified by B.McClure for HP_UX using info from HP manual
#  and the objtran HP makefile template.
#  Modified by D O'Dea for OSF/1.
#

# interface name
INTFC    = mathi

# Set defaults for the DCE stuff. If this isn't where DCE is on your
# system, change these lines. These should be correct for most HP-UX
systems.
DCE_BIN = /opt/dcelocal/bin
DCE_INCLUDE = /usr/include/reentrant
DCE_LIB = /opt/dcelocal/lib
DCE_DIR = /opt/dcelocal

# All libraries for DCE
DCE_L = -ldce
# Other libraries
SYS_L = -lpthreads -lc_r -lm
DCE_LIBS = $(DCE_L) $(SYS_L)

# command names
CC  = /bin/c89
IDL = $(DCE_DIR)/bin/idl
DEL = rm -f

# Debugging flags to use for C code
CDEBUG = -g
# Optimizer flags for all code
```

```
OPT =

# command flags
CFLAGS = -Dalpha -D_REENTRANT -I$(DCE_INCLUDE)

# Flags to use when linking C code that contains no C++
C_LDFLAGS =  -W l,-z -W l,-Bimmediate -L$(DCE_LIB)

# Flags for IDL
IDLFLAGS = -keep all -I. -I$(DCE_INCLUDE)


# object files
SOBJS    = $(INTFC)_s.o $(INTFC)_m.o $(INTFC)_sstub.o
COBJS    = $(INTFC)_c.o $(INTFC)_cstub.o

# targets
all: $(INTFC)_s $(INTFC)_c

$(INTFC)_s: $(SOBJS)
  $(CC) $(SOBJS) -o $(INTFC)_s $(DCE_LIBS)

$(INTFC)_c: $(COBJS)
  $(CC) $(COBJS) -o $(INTFC)_c $(DCE_LIBS)

$(INTFC).h $(INTFC)_sstub.o $(INTFC)_cstub.o: $(INTFC).idl
$(INTFC).acf
  $(IDL) $(IDLFLAGS) $(INTFC).idl

clean:
  -$(DEL) core *.o $(INTFC)_?stub.c $(INTFC).h

# dependencies
$(INTFC)_m.o: $(INTFC)_m.c $(INTFC).h

$(INTFC)_s.o: $(INTFC)_s.c $(INTFC).h

$(INTFC)_c.o: $(INTFC)_c.c $(INTFC).h
```

## A.3 Sparc; Solaris 2.4:

```
#
# Makefile for the Sparc Version of the Implicit Binding Example.
#
#  Modified by B.McClure for HP_UX using info from HP manual
#  and the objtran HP makefile template.
#  Modified by D O'Dea for Solaris 2.4.
#

# interface name
INTFC    = mathi
```

```
# Set defaults for the DCE stuff. If this isn't where DCE is on your
# system, change these lines. These should be correct for most HP-UX
systems.
DCE_BIN = /opt/dcelocal/bin
DCE_INCLUDE = /usr/include/reentrant
DCE_LIB = /opt/dcelocal/lib
DCE_DIR = /opt/dcelocal

# All libraries for DCE
DCE_L = -ldce
# Other libraries
SYS_L = -lthread -lm -lnsl
DCE_LIBS = $(DCE_L) $(SYS_L)

# command names
CC  = /usr/local/opt/SUNWspro/bin/cc
IDL = $(DCE_DIR)/bin/idl
DEL = rm -f

# Debugging flags to use for C code
CDEBUG = -g
# Optimizer flags for all code
OPT =

# command flags
CFLAGS = -Aa +DA1.0 +DS1.0 $(OPT) -Dsigned= \
  -D_HPUX_SOURCE -D_REENTRANT \
  -I. -I$(DCE_INCLUDE)

# Flags to use when linking C code that contains no C++
C_LDFLAGS =  -W l,-z -W l,-Bimmediate -L$(DCE_LIB)

# Flags for IDL
IDLFLAGS = -keep all -I. -I$(DCE_INCLUDE)


# object files
SOBJS    = $(INTFC)_s.o $(INTFC)_m.o $(INTFC)_sstub.o
COBJS    = $(INTFC)_c.o $(INTFC)_cstub.o

# targets
all: $(INTFC)_s $(INTFC)_c

$(INTFC)_s: $(SOBJS)
  $(CC) $(SOBJS) -o $(INTFC)_s $(DCE_LIBS)

$(INTFC)_c: $(COBJS)
  $(CC) $(COBJS) -o $(INTFC)_c $(DCE_LIBS)

$(INTFC).h $(INTFC)_sstub.o $(INTFC)_cstub.o: $(INTFC).idl
$(INTFC).acf
  $(IDL) $(IDLFLAGS) $(INTFC).idl

clean:
  -$(DEL) core *.o $(INTFC)_?stub.c $(INTFC).h
```

```
# dependencies
$(INTFC)_m.o: $(INTFC)_m.c $(INTFC).h

$(INTFC)_s.o: $(INTFC)_s.c $(INTFC).h

$(INTFC)_c.o: $(INTFC)_c.c $(INTFC).h
```

# Appendix B: Source Code

## B.1 Interface Definition File

```
[
uuid(0077F830-1100-1BFC-9250-10005A4F5444),
version(1.0)
]
interface mathi
{
  const short MAX_VALUES = 12;
  typedef long value_array_t[ MAX_VALUES ];
  void add (
    [ in ] value_array_t value_a,
    [ in ] long num_values,
    [ out ] long *sum
  );
}
```

## B.2 Client Source Code (OS/2)

```
/*****************************************************************************/
/* Module  : mathi_c.c                                                       */
/* Purpose : Client module for server mathi_s. Gets the values to be         */
/*         added from the command line and sends them to the remote add      */
/*         procedure together with the number of values passed and a         */
/*         pointer to the variable that will contain the result.             */
/*         Uses implicit binding.                                            */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <dce/rpc.h>
#ifdef _WINDOWS
#include <dce/dcewin.h>
#endif
#include "errchk.h"
#include "mathi.h"
#include <conio.h>
#include <dce/exchandl.h>
#include <dce/rpcexc.h>

#define ENTRY_NAME "/.:/Servers/MathI"   /* Server entry name.              */

void   Do_Add(int argc, char *argv[]);

int main (int argc, char *argv[] )
{
  while (!kbhit())
  {
```

```
        Do_Add(argc, argv);
    }
}

/* -------------------- Do_Add ---------------------- */
void
Do_Add(int argc, char *argv[])
{
 long int i, numv, sum;
  value_array_t va;
  rpc_binding_handle_t bh;
  unsigned32 status;
  rpc_ns_handle_t imp_ctxt;


  /* Set up the context to import the bindings.                    */
  rpc_ns_binding_import_begin ( rpc_c_ns_syntax_dce, ENTRY_NAME,
                        mathi_v1_0_c_ifspec, NULL, &imp_ctxt,
                        &status );
  ERRCHK ( status );

  /* Get the first binding handle.                                 */
  rpc_ns_binding_import_next ( imp_ctxt, &bh, &status );
  ERRCHK ( status );

  /* Resove the binding for multiple servers on this host  here*/
 rpc_ep_resolve_binding(bh,mathi_v1_0_c_ifspec,&status );
 ERRCHK ( status );

/* Set a low timeout value to detect server outages */
 rpc_mgmt_set_com_timeout(bh,1,&status);
 ERRCHK ( status );


  /* Store the binding handle bh in the global variable defined in .acf.    */
  global_handle = bh;

  /* Call add passing values, number of values and pointer to result.       */
  /* catch all exceptions and drop out to all catch keypressed */
  TRY
  {
        while (!kbhit())
        {
                va[ 0 ] = rand();
                        va[1] = rand();
                numv = 2;

                add ( va, ( long )numv, &sum );
                printf( "The sum of %d and %d is %d\n",
                                        va[0], va[1], sum );
        }
  }
  CATCH_ALL
  {
```

```
        printf("Lost server connection - attempting reconnection to any server\n");

    }
    ENDTRY
        /* Release the context.                    */
    rpc_ns_binding_import_done ( &imp_ctxt, &status );
    ERRCHK ( status );
}
```

## B.3 Server Start-up Source Code (OS/2)

```
/************************************************************************/
/* Module  : mathi_s.c                                                  */
/* Purpose : Perform the necessary setup for the server manager code    */
/*         module mathi_m.c. This module registers manager EPV,         */
/*         selects all protocol sequences available, advertises the server */
/*         and listens for RPC requests.                                */
/************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <dce/rpc.h>
#ifdef _WINDOWS
#include <dce/dcewin.h>
#endif
#include "errchk.h"
#include "mathi.h"
#include <dce/exchandl.h>
#include <signal.h>


#define MAX_CONC_CALLS_PROTSEQ  4      /* Max concurrent calls per protocol. */
#define MAX_CONC_CALLS_TOTAL    8      /* Max concurrent calls total.        */
#define ENTRY_NAME "/.:/Servers/MathI"  /* Server entry name.                */

/* ============ prototypes =============== */
void MyExit(int SIG_TYPE);


/* ============= GLOBALS ================ */
rpc_binding_vector_t *bv_p;     /* exit routine needs access */
```

```
/* ---------------------------- main ----------------- */
int main ( int argc, char *argv[] )
{
    unsigned32              status;
    int                     rc;

  /* Set up clean up routine */
  signal(SIGTERM,MyExit);
  signal(SIGABRT,MyExit);
  signal(SIGINT,MyExit);

  /* Register interface/epv associations with RPC runtime.              */
  printf("Registering server interface with RPC runtime...\n");
  rpc_server_register_if ( mathi_v1_0_s_ifspec, NULL, NULL, &status );
  ERRCHK ( status );

  /* Inform RPC runtime to use a supported protocol sequences.          */
  rpc_server_use_protseq ( "ncadg_ip_udp", MAX_CONC_CALLS_PROTSEQ, &status );
  ERRCHK ( status );

  /* Get the binding handle vector from RPC runtime.                    */
  rpc_server_inq_bindings ( &bv_p, &status );
  ERRCHK ( status );

  /* Register binding information with endpoint map.                    */
  printf("Registering server endpoints with endpoint mapper (RPCD)...\n");
  rpc_ep_register_no_replace ( mathi_v1_0_s_ifspec, bv_p, NULL,
              ( unsigned_char_t * )"Implicit math server, version 1.0, server 0",
              &status );
  ERRCHK ( status );

  /* Export binding information to the namespace.                       */
  printf("Exporting server bindings into CDS namespace...\n");
  rpc_ns_binding_export ( rpc_c_ns_syntax_dce, ENTRY_NAME,
                  mathi_v1_0_s_ifspec, bv_p, NULL, &status );
  ERRCHK ( status );

  /* Listen for service requests.                                       */
 TRY
  printf ( "Server %s listening...\n", ENTRY_NAME );
  rpc_server_listen ( MAX_CONC_CALLS_TOTAL, &status );
  ERRCHK ( status );
 CATCH_ALL
       printf("Error during server listen\n");
       printf("Server terminated\n");
       rpc_ep_unregister(mathi_v1_0_s_ifspec, bv_p,NULL ,&status);
 ENDTRY
}
```

```
/* ------------ MyExit -------------- */
/* Clean up after program termination (Does this catch cntrl brk ??)  */
void
MyExit(int SIG_TYPE)
{
   unsigned32              status;

  printf("Server terminated\n");
  rpc_ep_unregister(mathi_v1_0_s_ifspec, bv_p,NULL ,&status);
  if (status != 0)
       printf("Unable to remove endpoint map\n");
  signal(SIG_TYPE, SIG_DFL);
  exit(0);
}
```

## B.4 Server Manager Source Code (OS/2)

```
/*****************************************************************************/
/* Module  : mathi_m.c                                                       */
/* Purpose : Server manager code. Implements the add procedure that will be  */
/*         called from the client mathi_c.                                   */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "mathi.h"

/*****************************************************************************/
/* Procedure  : add                                                          */
/* Purpose    : Add numv values passed in array value_a placing the sum in   */
/*           total. Uses implicit binding.                                   */
/*****************************************************************************/

void add (
  value_array_t value_a,
  long numv,
  long *total )
{
  int i;

  /* Display the parameters that have been passed.                           */
  printf ( "Add has been called with numv = %d\n", numv );
  for ( i = 0; i < numv; i++ )
    printf ( "Value_a[ %d ] = %d\n", i, value_a[ i ] );

  /* Zero the accumulator.                                                   */
  *total = 0;

  /* Add the values in the array of size numv placing the result in total.  */
  for ( i = 0; i < MAX_VALUES && i < numv; i++ )
    *total += value_a[ i ];
}
```

**Development Of An Interoperable, Survivable Pilot DCE Application**

*B. McClure and D. O'Dea*

(DSTO-TN-0025)

DISTRIBUTION LIST

| | Number of Copies |
|---|---|
| *Defence Science and Technology Organisation* | |
| Chief Defence Scientist and members of the ) | 1 shared copy |
| DSTO Central Office Executive ) | for circulation |
| Counsellor, Defence Science, London | (Document Control sheet) |
| Counsellor, Defence Science, Washington | (Document Control sheet) |
| Senior Defence Scientific Adviser ) | 1 shared copy |
| Scientific Adviser - POLCOM ) | |
| Assistant Secretary Science Industry Interaction | 1 |
| Director, Aeronautical & Maritime Research Laboratory | 1 |
| Navy Scientific Adviser (NSA) | 1 |
| Scientific Adviser, Army (SA-A) | 1 |
| Air Force Scientific Adviser (AFSA) | 1 |
| *HQADF* | |
| Director, Operational Information Systems | 1 |
| DD-EW | 1 |
| *RAAF* | |
| CO, Electronic Warfare SQN | 1 |
| *Electronics and Surveillance Research Laboratory* | |
| Chief Information Technology Division | 1 |
| Research Leader Command & Control and Intelligence Systems | 1 |
| Research Leader Command, Control and Communications | 1 |
| Research Leader Military Computing Systems | 1 |
| Executive Officer, Information Technology Division | (Document Control sheet) |
| Head, Human Systems Integration Group | (Document Control sheet) |
| Head, Software Engineering Group | (Document Control sheet) |
| Head, Trusted Computer Systems Group | (Document Control sheet) |
| Head, Advanced Computer Capabilities Group | (Document Control sheet) |
| Head, Intelligence Systems Group | (Document Control sheet) |
| Head, Systems Simulation and Assessment Group | (Document Control sheet) |
| Head, Exercise Analysis Group | (Document Control sheet) |
| Head, C3I Systems Engineering Group | (Document Control sheet) |

| | |
|---|---|
| Head, Computer Systems Architecture Group | (Document Control sheet) |
| Head Command Support Systems Group | 1 |
| Mr B.McClure, Command Support Systems Group | 1 |
| Head Information Management and Fusion Group | 1 |
| Mr D.O'Dea, Information Management and FustionGroup | 1 |
| Publications and Publicity Officer, ITD | 1 |
| *Libraries and Information Services* | |
| Defence Central Library - Technical Reports Centre | 1 |
| Manager Document Exchange Centre (MDEC) (for retention) | 1 |
| Additional copies which are to be sent through MDEC | |
| DIS for distribution: | |
| National Technical Information Centre. United States | 2 |
| Defence Research Information Centre, United Kingdom | 2 |
| Director Scientific Information Services, Canada | 1 |
| Ministry of Defence, New Zealand | 1 |
| National Library of Australia | 1 |
| Defence Science and Technology Organisation Salisbury, Research Library | 2 |
| Library Defence Signals Directorate Canberra | 1 |
| AGPS | 1 |
| British Library Document Supply Centre | 1 |
| Parliamentary Library of South Australia | 1 |
| The State Library of South Australia | 1 |
| *Spares* | |
| Defence Science and Technology Organisation Salisbury, Research Library | 6 |

**DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION**
**DOCUMENT CONTROL DATA**

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
N/A

2. TITLE
Development Of An Interoperable, Survivable Pilot DCE Application

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)
Document (U)
Title (U)
Abstract (U)

4. AUTHOR(S)
B. McClure and D. O'Dea

5. CORPORATE AUTHOR
Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury SA 5108

| 6a. DSTO NUMBER | 6b. AR NUMBER | 6c. TYPE OF REPORT | 7. DOCUMENT DATE |
|---|---|---|---|
| DSTO-TN-0025 | AR-009-480 | Technical Note | December 1995 |

| 8. FILE NUMBER | 9. TASK NUMBER | 10. TASK SPONSOR | 11. NO. OF PAGES | 12. NO. OF REFERENCES |
|---|---|---|---|---|
| N9505/10/14 | ADF 94/151 | DGFD (Joint) | 38 | 11 |

13. DOWNGRADING/DELIMITING INSTRUCTIONS
N/A

14. RELEASE AUTHORITY
Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT
APPROVED FOR PUBLIC RELEASE

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT
No limitation

17. CASUAL ANNOUNCEMENT
No limitation

18. DEFTEST DESCRIPTORS
Distributed Processing
Computer Programming
Distributed Computer Systems
Computer Architecture

19. ABSTRACT
This report discusses some problems encountered in developing and porting a pilot Distributed Computing Environment (DCE) program to several different platforms. The pilot program demonstrates the interoperability, portability and survivability features of OSF's DCE.