DTIC

A Comparison of Interoperability Architectures within Object Oriented Multi-User Domains

Paul Prekop

DSTO-TR-0669

] APPROVED FOR PUBLIC RELEASE

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

A Comparison of Interoperability Architectures within Object Oriented Multi-User Domains

Paul Prekop

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0669

ABSTRACT

MUDs (Multi-User Domains/Dungeons) are multi-user, text-based, networked computer environments, in which users can interact with each other in a type of text based virtual reality. While traditionally used for text based adventure games, or social worlds, MUDs are increasing being used as the backbone for Computer Supported Collabrative Work (CSCW) or Anchor Desk Systems. Within the military domain Anchor Desks provide a single contact point for specialist areas (for example, logistics or intelligence, etc.) within a military organization. This paper explores the issues of interoperability between MOOs (Object Oriented MUD), and describes three different prototyped techniques for MOO Interoperability.

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

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

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR-010-539
May 1998

APPROVED FOR PUBLIC RELEASE

A Comparison of Interoperability Architectures within Object Oriented Multi-User Domains

Executive Summary

MUDs (Multi-User Domains/Dungeons) are multi-user, text-based, networked computer environments, in which users can interact with each other in a type of text based virtual reality.

Within the military domain MUDs have been used as a backbone for Anchor Desks. Anchor Desks provide a single contact point for specialist areas (for example, logistics or intelligence, etc.) within a military organisation. Anchor Desks are also responsible for the collection, storage and dissemination of information related to that specialised area.

When used as a backbone for Anchor Desks, MUDs have been used to provide the basic user/user communication and user/object interaction facilities, as well as providing support for basic data storage.

When using a MUD to support an Anchor Desk, a key design decision is; how should the organisation be divided into the MUD? Should all the Anchor Desks within a Command Centre used the one MUD. Should all Anchor Desks of the same type, for example logistics, inhabit the one MUD, regardless of their location, or should only one MUD be used within the whole military organisation?

Regardless of how an organisation is divided into a MUD, allowing users to interact with users and objects out side their local MUD is a vital function if MUDs are to be successfully used to support Anchor Desks.

This paper address the issue of integrating, two (or more) MUDs together, so that the services, or data provided by one MUD is available to the users of a remote MUD. Three different solutions to the problem of interoperability are developed, and the advantages and disadvantages of each is explored.

Authors

Paul Prekop
Information Technology Division

Before joining DSTO in January 1997, Paul was CML/CAL Project Officer at Deakin University's School of Engineering and Technology. Prior to that, he worked as a Senior Administrative Programmer at Monash University's Gippsland Campus.

Contents

1. Introduction

MUDs (Multi-User Domains/Dungeons) are multi-user, text-based, networked computer environments, in which players (users) can interact with each other, or other objects, in a shared space.

The shared space provided by the MUD often follows normal architectural conventions of rooms, exits, doors, etc. Users are free to navigate the space in an intuitive fashion. For example to exit a room you must use a door. MUDs also include objects (which can range in complexity and function), which users are free to manipulate, or use, in any way consistent with the MUD and the rules of the object. For example, you can't eat an object which can't be eaten.

For the user, the MUD provides a type of text based virtual reality. The space you as a user inhabit, is also inhabited by other users and other objects. The view of the virtual reality is shared by all users within it, through the common descriptions of the objects, users and interactions provided to all users within that space.

While recreational MUDs have been extremely popular for many years, there has been a growth in the use of MUDs for non-recreational use. This trend was highlighted by the inclusion of a workshop - "Design and Use of MUDs for Serious Purposes" at the 1996 Computer Supported Cooperative Work (CSCW) conference [6].

Perhaps the most popular, serious use for a MUD is as backbone for CSCW systems. When used as a backbone for CSCW systems, the MUD provides the basic user/user communication and user/object interaction facilities. The CSCW system then sits 'on top of' the MUD kernel and provides the user with specific CSCW tools and features.

Within the military domain MUDs have been used to support Anchor Desks. Anchor Desks provide a single contact point for specialist areas (for example, logistics or intelligence, etc.) within a military organisation. Anchor Desks are also responsible for the collection, storage and dissemination of information related to that specialised area. As with CSCW systems, MUDs have been used to provide the basic user/user communication and user/object interaction facilities, as well as providing support for basic data storage.

When using a MUD to provide basic user/user communication, user/object interaction and basic data storage, a key design decision is; how should the organisation be divided into the MUD?

Should a MUD used to support a CSCW system, for example, be divided on a work group bases, so each nominal work group within the organisation has its own MUD? Or should the division be based on the location of the users, or should the whole organisation use the one MUD?

When using a MUD to support an Anchor Desk, the problem is similar. Should one specialised anchor desk inhabit one MUD? Should all the Anchor Desks within a Command Centre use the one MUD? Should all Anchor Desks of the same type, for example logistics, inhabit the one MUD, regardless of their location, or should only one MUD be used within the whole military organisation?

Regardless of how an organisation is divided into a MUD, allowing users to interact with users and objects outside their local MUD is a vital function if MUDs are to be successfully used as the kernel for either a CSCW or Anchor Desk system. This paper addresses the issue of integrating, two (or more) MUDs together, so that the services, or data provided by one MUD is available to the users of a remote MUD.

This paper first examines the use of MUDs as CSCW backbones. Several of the advantages and disadvantages of using MUDs as CSCW backbones are explored. Next the design issues raised by MUD interoperability are explored, and an ideal architecture is described. (This architecture is ideal for the needs of this specific project, not necessarily for all MUD interoperability projects). Finally this paper describes three different interoperability architectures developed, the performance of each, and how well each meets the goals of the ideal architecture.

1.1 Why MUDs?

While the primary uses of MUDs are still text based adventure games, or as text based social worlds, MUDs have increasingly been used successfully as the backbone for interactive CSCW systems.

Using a MUD as the backbone for a CSCW system affords the CSCW developer several advantages. The major advantage is in being able to use the existing MUD's features to handle the base communication, user control, object

movement and user/user and user/object interactions. These functions are vital parts of any CSCW system, and by using a MUD, developers can reduce development time and effort.

As well as providing the object/user and user/user communication framework, several MUDs are extendible; the landscape or environment of the MUD can be expanded by its users or the CSCW developers. This powerful feature can be used to provide a high level of customisation and expendability to a CSCW system.

MUDs are also available for nominal or no costs. For example, Xerox's LambdaMOO, is public domain, available for several UNIX platforms, and is shipped with full source code.

However, use of an existing MUD as a backbone for CSCW systems does have several drawbacks.

All current MUDs are based around a central server. While this architecture is simple to implement, it does create a bottleneck when dealing with many users, because all user/user and object/user interactions must pass through the central server, regardless of the relative locations of the users. The problem can be exacerbated by the nature of graphical CSCW systems, which can generate a lot of complex change requests and action notifications. In a central server architecture these change requests and action notifications must flow from the generating user to the server, then from the server to any interested/relevant users.

The text only interface is another drawback of using MUDs as a CSCW backbone. While several projects have developed WWW or other GUI interfaces for MUDs, they are often tied to a specific MUD, and as yet there is no standard. Also the development of proprietary WWW or GUI interfaces adds an additional layer of complexity to a CSCW development.

2. Other MUDs Projects

Currently, there are several projects using a MUD as the backbone for CSCW development, these include;

- NRAD's C2MUVE system [1]
- The CVW system at MITRE [2]

- Project Jupiter at Xerox's Palo Alto Research Centre [3]
- CAPER/CAETI Architectures Project [4]

Of these projects only the CAPER/CAETI Architectures project and NRAD's C2MUVE project have explored MUD interoperability.

The CAPER/CAETI project has so far focused on the user and usage issues of MUD interoperability. While they have defined the higher levels of MUD interoperability (how it should look to a user, rules controlling the movement of objects between MUDs etc.) they have not defined any of the lower level functionality needed by MUD interoperability.

3. MOO Interoperability

The MUD selected for this project was Xerox's LambdaMOO. MOO (MUD, Object Oriented) was originally developed by Stephen White at the University of Waterloo, and enhanced by Pavel Curtis at Xerox PARC. Through out the rest of this paper the term MOO is used to describe the specific MUD selected. While most of the material covered by this paper is equally applicable to other specific MUDs, some issues, like object inheritance, or the concept of library functions, may only be applicable to the specific MOO selected.

MOO interoperability can be seen as the functions provided by a MOO which allow users to interact with objects parented by a remote MOO. Objects can include any MOO object; users, user developed objects, rooms, etc. Interaction can range from viewing an object's property to executing an object's verb.

An interoperability architecture, then, describes the different architectural components (objects, servers, etc.), the services they provide, and the interaction between the components which support MOO interoperability.

This very broad definition of MOO interoperability, and interoperability architecture, raises three key issues; the level of interoperability support (what MOO functions should an interoperability architecture provide, how do they integrate into the MOO, etc.); the type of object distribution (where should objects execute); and architectural services (what additional services should the architecture provide, and where should they be located). Each issue is explored in more detail in the following sections.

3.1 Level of Interoperability Support

The level of interoperability support describes what type of interoperability functionality should be provided by the architecture and how the functionality fits within the MOO.

Should, for example, the interoperability architecture provide a set of additional room verbs which allow users to interact across MOO boundaries, for example a Remote Who (RWHO), or a Remote Page (RPAGE) command. Or should the interoperability architecture provide a set of generic 'net aware' objects that encapsulate interoperability functions. Or should it simply provide some low level library functions which facilitate interoperability.

The primary approach taken by this project was to develop low level, general purpose library functions. This approach provided a great deal of flexibility, because the low level library functions provide the connectivity, any style of object can be shared across MOO boundaries. Also, by using low level library functions, interoperability is not tied to specific objects, so any existing objects can be converted.

However, the low level library functions approach has meant that all higher level functionality still must be implemented. This may mean developing new object to provide user level interoperability, or modifying existing objects (like rooms, or players) to provide them with across MOO functionality.

Also because the library functions are called directly as verbs, any modification to the library function's parameters, would mean modifying all the places in which it is used.

3.2 Type of Object Distribution

Another key interoperability issue is the type of object distribution; where and how should the object execute. Should MOO interoperability implement a form of Remote Verb Invocation (RVI); conceptually similar to the RPC/RMI model where an object's verb is executed on the remote host under the control of the local host and local user? Or should it implement some form of Remote Code, Local Execution (RCLE) model; conceptually similar to the Applet Model, where the object's code or binary is transferred from the remote host to the local host, and executed as a local object?

The RVI model of interoperability provides a very powerful architecture for developing distributed objects. Conceptually it works like RPC/RMI architectures, where an object on one MOO can make a remote verb call to an object on a remote MOO. However one major problem with RVI style of connectivity is providing a user context in which the remote object executes.

All objects within a MOO run within a user context. That context is encapsulated within the object, and controls what functions an object can or cannot perform. For example a 'wizard' object, even when used by a non-wizard, is capable of more functions, than a non-wizard object. User contexts also control which users can perform what functions. For example a 'programmer' user is capable of executing commands a non-programmer cannot.

The issue for RVI, is what user context to use, and how to enforce it? Should a RVI call to a remote verb execute with a user context equal to the requesting user's, or should it run at a higher or lower level of security? Also should users on the remote host be aware that a RVI call is in progress, and should they be able to effect it?

While RCLE style of distribution is intuitively ideal for many interoperability functions, for example moving a player, or an object between MOOs, it does include a high level of complexity. Most of this complexity is caused by object inheritance and object identification.

The MOO supports run time binding of child objects with their parents. So when a verb of a child object is executed, any parent verbs which provide some of the child verb's functionality are also called. The issue for RCLE style of object distribution, is how to deal with this style of object inheritance.

The second problem RCLE causes is that of object identification. Each object is uniquely identified on its local MOO by an object number. Object numbers are only guaranteed to be unique within a MOO. When objects are moved between MOOs some method of dealing with possible object number clashes must also be used.

3.3 Architectural Services

Architectural services describe what additional services to support or enhance interoperability the architecture should provide, and where (at what level or within which component) the service should be placed. Architectural services

add value to the architecture by providing services above those provided by the object distribution method.

Basic architectural services should include; MOO and object location/naming services, and object trader services.

There are generally two types of MOO and object naming approaches; federated or domain. Within federated naming, all object names are unique within the federation of interconnected MOOs. For example, an object called ObjOne on BigMoo may be registered as Cookie via the naming service. The object name Cookie would be unique within the federation and any references to Cookie would be resolved back to the MOO name and the MOO unique object name.

With federated naming it may also be possible to register multiple physical objects under the one logical name and implement some type of dynamic resolution. For example ObjOne on BigMoo, WizOObject on BigMoo, and WizOObject on LittleMoo may all be registered as Cookie. The name resolution service could (based on some policy) decide which physical object to resolve to. This decision may, for example, be based on simple rules, or it may be based on the load of the respective MOOs, or it may be based on the how current the object is.

The second style of MOO and object naming is domain naming. Within domain naming the object is registered as being present on a specific MOO. For example, the object ObjOne on BigMoo, may be named as Cookie on Collins (Collins may be the registered name for BigMoo). The object's name is only unique within a MOO (domain), and any calls to the object name must also include its registered MOO (domain) name. For example to access ObjOne on BigMoo (registered as Cookie on Collins), the fully qualified name would be used.

The MOO interoperability architecture should also provide an object trading service. Object trading services are conceptually similar to trader services provided via OMG's CORBA architecture [5]. Within CORBA, objects are registered as providing a specific services, clients (in CORBA a client is a program which users the services provided by a server) query the trader by passing it a description of the service required. The trader returns a reference to an object which can provide the needed service.

Within the MOO interoperability architecture, traders perform a very similar function. Objects are registered as providing a specific service or specific data, a

user can query the trader (via some intermediate object). The trader returns the name of an object capable of providing the required service, or the required data.

4. Interoperability Architectures

This section describes the work done in developing a MOO interoperability architecture. The first section describes the minimal architecture requirements of any MOO interoperability architecture, and the second section describes the three different architectures developed, and compares each with the minimal architecture requirements.

4.1 Minimal Architectural Requirements

The architectural requirements describe the basic functionality any MOO interoperability architecture developed for this project would need to provide to fulfil the project's goals.

The key requirements include;

- Provide RVI style of connectivity.
- Provide support via low level, library functions.
- Be low maintenance.
- Be scalable - no hard limits on the number of MOOs or objects.
- It should provide a method of Object and/or MOO Registration.

The low maintenance goal describes how much additional development time the architecture adds to the development of interoperable MOO objects. Does a large amount of additional MOO (or other) code need to be developed to make the MOO object accessible? Is the object or MOO registration process complex and time consuming? Do considerable changes to the architecture need to be made each time a new MOO or object needs to be registered?

4.2 Architectural Prototypes

Three different architectural prototypes were developed; Peer To Peer, Message Switching Server, and Client/Server via CORBA.

As well as phototyping different interoperability architectures, each prototype also made use of different sets of underlying technologies. The different

functionality afforded by these different technologies is only identified when they have a direct effect on the functionality of the architecture.

The three different architectures are described in detail in the following sections.

4.2.1 Peer to Peer

The first architecture prototyped was a simple Peer to Peer architecture. Each MOO was connected to every other MOO in the federation, and several simple, interactive, across MOO functions were provided. These functions included; RWHO (remote who), RPAGE (remote page) and RCOPY (a copy command which copied an object from one MOO to the other). This architecture is illustrated graphically by figure 1.

This architecture allowed users to interact with each other (in a very limited way) across MOO boundaries. It also provided a very crude form of RCLE style distribution, via the RCOPY command. Users could copy an object from a remote MOO to their local MOO, and execute the object locally.

The object inheritance problem associated with RCLE style distribution was solved by restricting the objects that could be moved to children of objects available on all MOOs within the federation. The problem of object identification also associated with RCLE style distribution was solved by re-registering the remote object as a local object as part of the remote copy process.

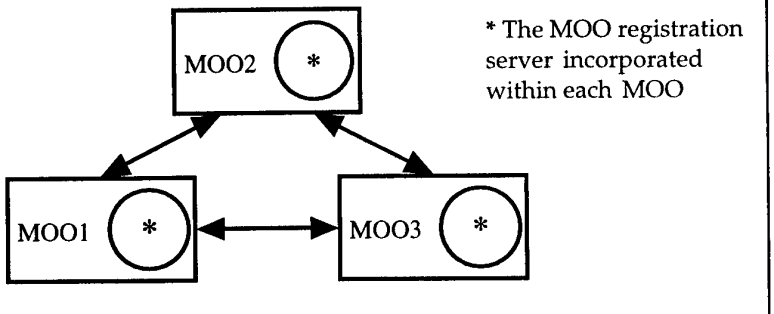

Figure 1. Peer to Peer architecture

Since this architecture didn't provide direct (interactive) access to remote objects, no object registration/naming, or object trader services were needed. However MOO registration services were still needed, because users needed to identify a remote MOO, in order to direct their across MOO functions.

MOO registration was achieved by incorporating a MOO naming service within each MOO, as shown in figure 1. The MOO naming service mapped the well known MOO name to the unique, implementation dependent MOO information, for example the host machine of the MOO, the port it listened on. No formal method to ensure the uniqueness of a well known MOO name within the network was developed.

4.2.1.1 Architectural Performance

While very simple to develop and deploy, this architecture provided only limited MOO interoperability. Also building the architecture from MOO object and library functions, did limit the performance of this architecture.

The lack of a central server to provide MOO registration (and possible object registration if MOO interoperability was expanded) would limit this architecture's expendability, and make it difficult to include higher level architectural services like object traders.

4.2.1.2 Closeness to Minimal Architectural Requirements

The prototype implementation of this architecture failed to provide many of the features needed by the ideal architecture, including the provision of RVI style of commands, a method of object registration.

4.2.2 Message Switching Server

The second prototype developed, conceptually modelled a packet switching network. All interaction between MOOs within the federation, were via well defined messages, with the flow of messages between the federated MOOs being controlled by the Message Switching Server (MSS). This architecture is illustrated graphically in figure 2.

Typically, a MOO would create a message (in order to fulfil a user's request) and pass the message to the MSS. The MSS would interrogate the message, and pass it to the MOO specified in the destination address, or perform the action specified in the message itself (two of the architecture's services were physically located within the MSS). If the message was passed on to a MOO, the MOO would perform the required action and package a return message, using the return address of the original message as the destination address, and it own address as the return address. This message would then be passed to the MSS, which would pass it to original MOO.

While at the lower level this architecture was built around well formatted messages, for the MOO object developer, it provided a powerful RVI interface. MOO objects could remotely execute the verbs of remote objects, pass parameters to remote verbs, and receive any returned values.

The RVI interface provides a low level MOO programming library, as well as a generic object. The generic object was used as the parent for any objects that need to support RVI calls. The generic object contained several MOO verbs and code stubs needed to support RVI calls.

The problem of providing a user context for RVI style of distribution was addressed through the use of a control connection. When the MSS connected to each MOO, it connected through a control connection. The control connection was simply a user account, with wizard access. Any RVI calls would run within the context of the control connection.

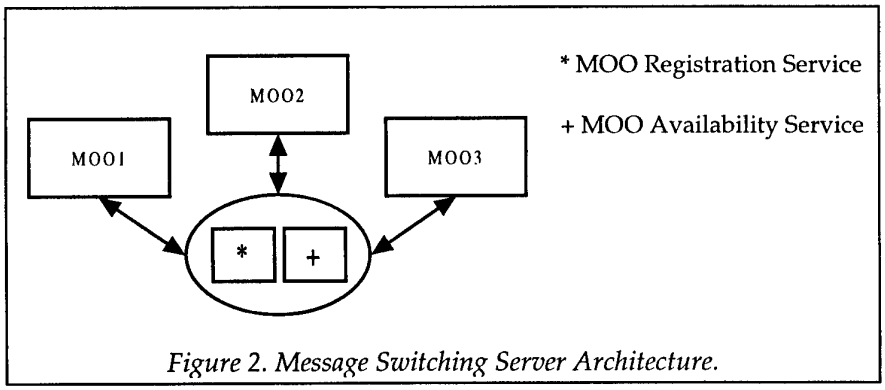

Figure 2. Message Switching Server Architecture.

As illustrated by figure 2, the MSS architecture also provided a MOO registration service, which was physically located within the MSS. All MOOs were registered with the MOO registration service, and identified by a unique well known name. This well known name was used as a part of RVI call.

As well as providing a MOO registration service, the MSS architecture also provided a MOO availability service. Located within the MSS, (see figure 2), the MOO availability service, could report on which MOOs were currently available (i.e. connected), and which MOOs were registered but not available (i.e. not currently connected).

Although not explicitly implemented in this prototype, the structure of this architecture did provide an incomplete form of domain naming. When RVI calls were made, the registered, well known MOO name, the object number and the

name of the verb to be executed, were used to form a fully qualified address. The Object number was unique within the MOO (domain), and the verb name was unique within the object.

Although not implemented, this architecture would also lend it self to federation style naming. With the central server architecture providing for a clean implementation of a naming service, with possible support for dynamic name resolution.

4.2.2.1 Architectural Performance

Among the most powerful aspects of this architecture was its ease of use (for the MOO object developer) and its low impact on MOO performance. From the MOO object developer's view, RVI calls were performed through a few simple library functions; and there was little difference in the amount of effort needed to make a local or remote verb call. Also because the complexities of message routing was handled by the MSS, the RVI calls had little performance impact on the MOOs.

However the MSS architecture does require that only objects which are children of the generic object are able to make and accept RVI calls. This requirement may prevent existing objects from being shared.

While the central message server does provide a very clean architectural model, the use of a central server does have inherent limitations. If more complete object registration features were included, or if the number of federated MOO, or the amount of message traffic was to increase, server performance may degrade.

4.2.2.2 Closeness to Minimal Architectural Requirements

The MSS architecture comes very close to being an ideal architecture. It provides most of the key features, including; provision of RVI style of distribution, provision of low level, library functions, its low maintenance, and its extensible.

However the current implementation of the prototype does not provide any object registration functions. It also restricts the objects which can be shared to children of the generic broker object, which may limit its usefulness.

4.2.3 Client/Server via CORBA

The final prototype was built with CORBA. While perhaps the most complex architecture, it is by far the most promising. CORBA products provide many of the architecture services that are needed, as well as providing seamless integration of MOO objects, and MOO users with objects outside the MOO.

The Client/Server via CORBA architecture was built in two parts, the server side and the client side. The server side of the architecture is responsible for exposing MOO objects, via CORBA, to the outside world. The client side of the architecture is responsible for providing MOO users with the ability to make calls to CORBA objects.

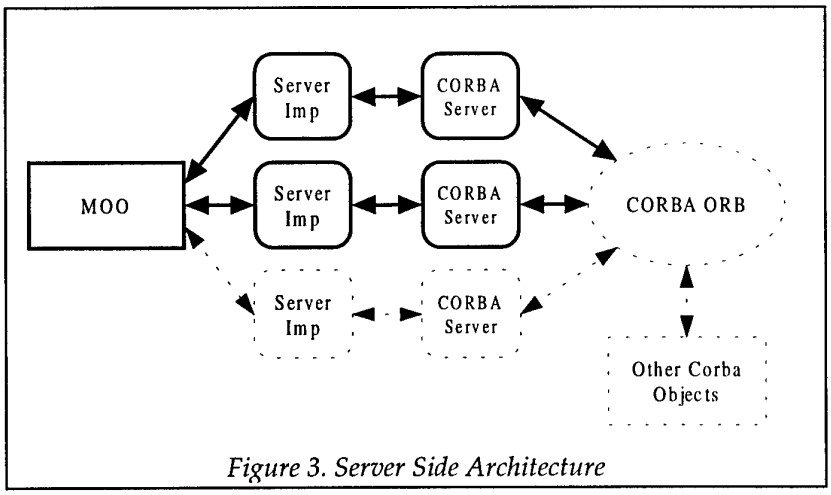

Figure 3. Server Side Architecture

Figure 3 illustrates the server side of the architecture. For each MOO object to be exposed via CORBA, a server implementation (Server Imp) and CORBA server components are created.

The server implementation component is responsible for maintaining a connection with the MOO, executing object verbs, and gathering values returned by the verbs. The CORBA server component is responsible for maintaining a connection with the CORBA ORB, translating requests from the CORBA ORB into actions for the server implementation component.

Once a MOO object has been exposed by a CORBA server, it is accessible to any CORBA compliant client, regardless of the client's location.

Figure 4 illustrates the client side of this architecture. Client connections from a MOO to the CORBA ORB are performed via connection persistent proxies. To

connect to a CORBA server, the MOO first requests a new proxy from the Client Proxy Server (CPS). The CPS creates a new client proxy for the exclusive use of the requesting MOO user. The proxy translates the MOO depending requests into CORBA requests, and CORBA return values, into MOO values.

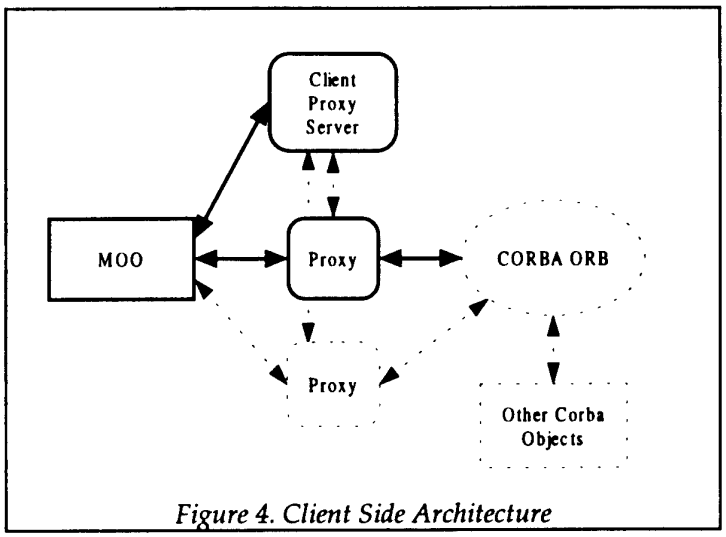

Figure 4. Client Side Architecture

As with the server side of this architecture, the proxy can be used to communicate with any CORBA compliant object, within, or external to a MOO.

While this architecture made use of several different low level communication methods, it provides the MOO object developer with a very simple, and very clean RVI interface. The RVI interface, physically implemented as a programming library, provides the MOO object developer with RVI, and Remote Property Calls functionality.

By default this architecture implemented a form of federated naming; all exposed MOO objects are identified by a unique, well known name. These names are registered with the CORBA Implementation Repository (IR). The IR can be queried by any CORBA compliant client.

Once registered, all calls to the MOO objects were via its registered name. This style of object naming is intrinsic to the CORBA implementation, where the focus is on the object not the location of the object.

Perhaps the most interesting functionality provided by the underlying CORBA technology is its ability to integrate with any CORBA compliant server or client. From the MOO object developer's view, there is no difference between a MOO

object exposed via CORBA, or a non-MOO object exposed via CORBA. The RVI, and Remote Property Calls library still provides the same functionality.

And from the view of a CORBA client (a CORBA client is an application, or part of an application which makes calls to a CORBA server), there is no real difference between a MOO object exposed via CORBA and a non-MOO object exposed via CORBA, the functionality afforded by both is the same.

4.2.3.1 Architectural Performance

The use of CORBA as a technology to support MOO interoperability provided a very flexible way to integrate, not only multiple MOOs, but also MOOs with external objects and external objects with MOOs.

The CORBA standard also describes a collection of standard network services; including, Object Trading and Object Location services. The functions these services provide would be very applicable to MOOs interoperability.

However, the Client/Server via CORBA architecture is complex, with a large number of components and high bandwidth and machine usage requirements. Also the current CORBA/JAVA standard has not been formalised (voting on the current revision will not close until September 1997).

4.2.3.2 Closeness to Minimal Architectural Requirements

The client/server via CORBA architecture is very close to being an ideal architecture. It supports most of the features identified as important, including; provision of RVI interaction, provision of low level, library functions, and object registration.

One key issue of this architecture is the amount of additional code needed to be developed to expose an object. While the code is relatively simple, and doesn't change much between different objects, there is no real way to generate a generic server which can expose all of the objects. So the amount of additional developer effort to expose a MOO object via CORBA is still quite substantial.

5. Conclusions

This paper examined the key issues of MOO interoperability, namely; what MOO functions should an interoperability architecture provide, the type of

object distribution and what additional services should the architecture provide.

Based on these issues, an ideal interoperability architecture was then described. This architecture was ideal for the specific goals of the project, and may or may not be ideal for other MOO interoperability projects.

Three different architectural prototypes were then explored, and their fitness to the ideal architecture was examined. As pointed out in the discussion, none of the prototypes developed so far, completely meet the requirements of an ideal architecture.

Failure to meet this requirement of the ideal architecture is due mostly to the object and MOO naming requirements of the ideal architecture. Any methods used to name objects and MOOs will depend, for the most part, on the environment into which the interoperability architecture is place, as well as exact user requirements, and the structure of the CSCW system which the MOO will support.

6. References

[1] Duffy , Lorraine and Shope, John. C2MUVE Homepage., SPAWAR Systems Center URL http://da5id.nosc.mil/c2muve.html

[2] Spellman, Peter J. Collaborative Virtual Workspace - a demonstration presented at CSCW'96., Mitre Corporation URL http:/www.mitre.org/resources/centers/advanced_info/g04e/cvw.html

[3] Curtis, Pavel., Dixon, Michael., Frederick, R., and Nichols, David A. The Jupiter Audio/Video Architecture: Secure Multimedia in Network Places. Xerox Palo Alto Research Center URL ftp://ftp.lambda.moo.mud.org/pub/MOO/papers/JupiterAV.ps

[4] Bobrow, D., Carlson, B., Rowley, M., Shirley M. CAPER Interoperability Standard. George Mason University URL http://www.nac.gmu.edu/caper/mudarch/mudio.html

(4) Schettler, Chuck. Open home page of CAPER. George Mason University URL http://www.nac.gmu.edu/caper/openpage.html

[5] Object Management Group, *A Discussion of the Object Management Architecture*, OMG, 1997.

[6] Design and Use of MUDs for Serious Purposes workshop: http://130.236.88.19/MUD/

A Comparison of Interoperability Architectures within Object Oriented Multi-User Domains

Paul Prekop

(DSTO-TR-0669)

DISTRIBUTION LIST

Number of Copies

AUSTRALIA

DEFENCE ORGANISATION

Task Sponsor

DGFD (Joint)	1

S&T Program

Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Director General Science Policy Development	1
Counsellor Defence Science, London	Doc Control Sheet
Counsellor Defence Science, Washington	Doc Control Sheet
Scientific Adviser to MRDC Thailand	Doc Control Sheet
Director General Scientific Advisers and Trials)	
Scientific Adviser Policy and Command)	1 shared copy
Navy Scientific Adviser	Doc Control Sheet and distribution list
Scientific Adviser - Army	Doc Control Sheet and distribution list
Air Force Scientific Adviser	1
Director Trials	1

Aeronautical and Maritime Research Laboratory

Director	1

Electronics and Surveillance Research Laboratory

Director	1
Chief of Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	Doc Control Sheet
Head, Information Architectures Group	1
Head, Information Warfare Studies Group	Doc Control Sheet

Head, Software Systems Engineering Group	Doc Control Sheet
Head, Year 2000 Project	Doc Control Sheet
Head, Trusted Computer Systems Group	Doc Control Sheet
Head, Advanced Computer Capabilities Group	Doc Control Sheet
Head, Computer Systems Architecture Group	Doc Control Sheet
Head, Systems Simulations and Assessment Group	Doc Control Sheet
Head, Intelligence Systems Group	Doc Control Sheet
Head, Command Support Systems Group	1
Head, C3I Operational Analysis Group	Doc Control Sheet
Head, Information Management and Fusion Group	1
Head, Human Systems Integration Group	Doc Control Sheet
Head, C2 Australian Theatre	1
Task Manager	1
Author	2
Publications and Publicity Officer, ITD	1

DSTO Library and Archives

Library Fishermens Bend	1
Library Maribyrnong	1
Library Salisbury	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Control sheet

Capability Development Division

Director General Maritime Development	Doc Control Sheet
Director General Land Development	Doc Control Sheet
Director General C3I Development	Doc Control Sheet

Navy

SO (Science), Director of Naval Warfare, Maritime Headquarters Annex, Garden Island, NSW 2000	Doc Control Sheet

Army

ABCA Office, G-1-34, Russell Offices, Canberra	4

Intelligence Program

DGSTA Defence Intelligence Organisation	1

Corporate Support Program (libraries)

OIC TRS, Defence Regional Library, Canberra	1
Officer in Charge, Document Exchange Centre (DEC),	1
US Defence Technical Information Center,	2
UK Defence Research Information Centre,	2
Canada Defence Scientific Information Service,	1
NZ Defence Information Centre,	1
National Library of Australia,	1

UNIVERSITIES AND COLLEGES

Australian Defence Force Academy	1
Library	1
Head of Aerospace and Mechanical Engineering	1
Deakin University, Serials Section (M list), Deakin University Library, Geelong, 3217	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1

OTHER ORGANISATIONS

NASA (Canberra)	1
AGPS	1
State Library of South Australia	1
Parliamentary Library, South Australia	1

OUTSIDE AUSTRALIA

ABSTRACTING AND INFORMATION ORGANISATIONS

INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts, US	1
Documents Librarian, The Center for Research Libraries, US	1

INFORMATION EXCHANGE AGREEMENT PARTNERS

Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards and Technology, US	1

SPARES	10
Total number of copies:	**65**

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

A Comparison of Interoperability Architectures within Object Oriented Multi-User Domains

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)

Paul Prekop

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-TR-0669	AR-010-539	Technical Report	May 1998

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
	94/150	ADF	28	6

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 Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Multi-user dungeons/domains (MUD), Computer supported cooperative work, Object-oriented system architecture, Interoperability, Virtual reality.

19. ABSTRACT

MUDs (Multi-User Domains/Dungeons) are multi-user, text-based, networked computer environments, in which users can interact with each other in a type of text based virtual reality. While traditionally used for text based adventure games, or social worlds, MUDs are increasing being used as the backbone for Computer Supported Collabrative Work (CSCW) or Anchor Desk Systems. Within the military domain Anchor Desks provide a single contact point for specialist areas (for example, logistics or intelligence, etc.) within a military organization. This paper explores the issues of interoperability between MOOs (Object Oriented MUD), and describes three different prototyped techniques for MOO Interoperability.