An Architecture for Multilevel Secure Interoperability

Myong H. Kang, Judith N. Froscher, and Ira S. Moskowitz[1]

Center for High Assurance Computer Systems

Information Technology Division, Mail Code 5540

Naval Research Laboratory

Washington, DC 20375 USA

Abstract

As computer systems become distributed and heterogeneous, there is strong movement in the commercial sector to ease the problems of interoperability and security. Many standards have been proposed for these problems. However, the commercial sector has not shown strong interest in providing cost-effective high-assurance multilevel security (MLS) solutions to the relatively small communities (e.g., intelligence, military) that require them.

In this paper, we introduce a practical, cost-effective, and high-assurance secure solution for multilevel distributed and heterogeneous environments using COTS components. The solution is based on an MLS architecture that consists of commercial single-level hardware and software, and a few specialized security devices. We show how an MLS CORBA can be constructed from single-level CORBAs and two security devices; the NRL Pump and the Starlight Interactive Link. We also introduce the concept of MLS cooperative computing which is a way to semi-automate distributed computing among organizations at different security levels.

1. Introduction

Today's distributed computer systems are connected to large computer networks such as the Internet or corporate Intranets. They are heterogeneous in terms of hardware and software, applications, and programming languages. They address the growing needs of sharing information and resources within and across diverse computing enterprises.

To address the interoperability problem, the Common Object Request Broker Architecture (CORBA) was proposed. CORBA defines how objects are distributed across a distributed environment and how they interact. This computing paradigm promises component-based development, location and programming language independence, scalability and fault-tolerance, and software reuse. Note that the purpose of CORBA is to develop, adopt, and promote *standards* for applications in a distributed heterogeneous environment, not to promote a specific technology.

CORBA clients send requests to servers and servers return results. Clients do not need to know where a particular server is located nor which server is serving the request. An object request broker (ORB) mediates all accesses. The ORB receives requests from a client, finds the object implementation for the request, transmits the request to a server, receives the output from the server, and returns it to the client. In the CORBA framework, each object can be both client and server (figure 1). In figure 1, client A sends request *r* to server B, and client B (since B is both a client and a server) in turns send a request to server C to answer *r*.

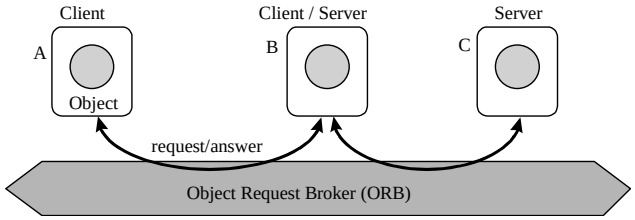

Figure 1: Clients and servers in CORBA

[1] Research supported by ONR.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1997		**00-00-1997 to 00-00-1997**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
An Architecture for Multilevel Secure Interoperability	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Center for High Assurance Computer Systems,4555 Overlook Avenue, SW,Washington,DC,20375**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**11**	

Security is an important issue in distributed computing because extensive communication between a client and a server, which may be located across enterprises, is necessary. There are many kinds of security concerns such as the integrity and confidentiality of information, the accountability of users, and the availability of information and resources. To address these concerns, CORBASecurity [11] specifies security functionality such as identification and authentication (I&A), access control, auditing, the security of communication between objects, non-repudiation, and the administration of security information.

One important problem that CORBA does not solve is the portability of applications in a distributed computing environment. Recently, the Java programming language on Java virtual machines was introduced. The Java idea has gained popularity among software developers and Internet content providers because both the software development and maintenance costs, and the cost for the clients to access services, can be reduced substantially. For example, if a service provider provides a Java applet that can communicate to server applications, end users who have Java-enabled Web browsers can download applets and related libraries dynamically and utilize the service. In this way, the service providers have a much lower burden for software maintenance, update, and distribution. In addition, the end users (i.e., clients) do not need any special software or hardware except a Java-enabled Web browser.

Since the applets, which can be downloaded from remote sites, may have potentially insidious behavior, significant efforts were made to provide reasonable security. For example, digital signature for applets and "sand box" security restrict the ability of untrusted applets that are running on Web browser.

Both CORBA and Java are attractive ideas for distributed computing for both the commercial and government sectors. Currently, there are substantial efforts to combine these two approaches. Since both approaches are so popular, CORBASecurity and Java security features are very important aspects for end users and developers. However, a security aspect that neither CORBASecurity nor Java security address is high-assurance multilevel security (MLS). High-assurance MLS has never been a primary security concern for the commercial sector, and we believe this lack of interest will continue for some time. Therefore, cost-effective, high-assurance MLS systems will not be available from the commercial sector in the near future. However, there are communities (e.g., intelligence and defense) that need high-assurance distributed MLS solutions today. To address high-assurance MLS in a distributed computing environment, NRL has started the Distributed Object Security Architecture (DOSA) project. The purposes of this project are

1. to introduce a high-assurance MLS distributed architecture for multilevel secure interoperability,
2. to ensure that MLS users will be able to use the most advanced commercially available information technology as well as legacy resources securely and affordably,
3. to examine existing security technologies and to recommend how they can be used in the MLS distributed computing framework, and
4. to identify additional security mechanisms or computing paradigms that need to be developed for distributed high-assurance MLS computing.

The rest of the paper is organized as follows. We introduce a generic multilevel security model for enterprises in section 2. We then introduce our approach to solve MLS information management requirements in section 3, especially the architectural side of our approach. Section 4 discusses how MLS cooperative applications can be organized in MLS distributed environments. We present the status of our prototype effort and a specific example of our approach in section 5. Section 6 contrasts our approach to a few proposed approaches to meet MLS distributed requirements. We summarize this paper in section 7.

2. Multilevel Secure Enterprise Model

Enterprises need to protect their computing assets (i.e., information and resources) from perceived threats. This can be achieved by establishing protection domains [10] that contain groups of related components. Protection domain boundaries can be enforced by both physical and logical separation. Protection domains can be organized as hierarchies (levels) of protection domains. For example, a protection domain, P1, may contain two protection subdomains, P11 and P12 so forth. Both P11 and P12 are governed by the protection policy enforced for all of P1. Components within the smallest protection domain are assumed to trust each other and have no need for security services. Using this protection domain concept, we can define several security models.

The multilevel secure enterprise model is based on the principle of multilevel security, which can be stated as follows:

(1) the enterprises determine the levels of trustworthiness (clearances) for their users,

(2) information is marked with a sensitivity label, and

(3) access control decisions are made based upon the level of user's trustworthiness and the sensitivity of the information.

Traditionally, an MLS computer system has a specific access policy that is based on a widely accepted information security policy. This policy can be summarized as "no high level information is allowed to pass to lower level users/processes and lower level information should be available to higher level users/processes" (e.g., Bell-LaPadula policy (BLP)). This is graphically illustrated by the diode symbol. Figure 2 illustrates secure enterprise model with two security levels.

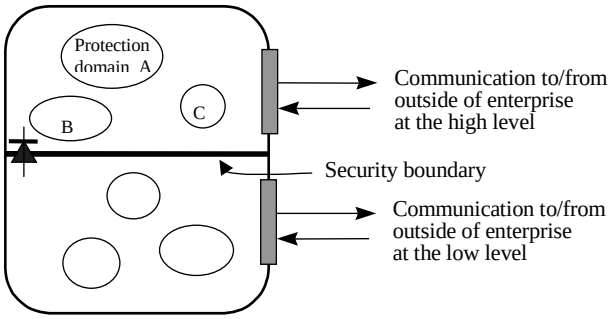

Figure 2: Two-level secure enterprise model

There are two important points to observe in the above example:

- There are various threats at each protection boundary. Different protection mechanisms such as discretionary access control (DAC), domain and type enforcement (DTE) [2], firewall or ORB gateway, and cryptographic techniques for identification and authentication (I&A), non-repudiation, and secrecy of communication across protection boundary are needed for effective security.
- There are various strengths of protection mechanisms. A smaller protection boundary in figure 2 does not necessarily imply a weaker protection boundary. For example, some department protection boundaries (which are smaller boundaries than the enterprise protection boundary) may deploy stronger protection mechanisms than that for the enterprise protection boundary.

Hence, it is important for the security designer to understand the various threats against each protection domain and choose the correct security mechanisms with the right strength for the respective protection boundary.

An enterprise may consist of geographically distributed protection domains so that a single Intranet no longer suffices. In this case, some distributed protection domains may form a virtual protection domain through protected communication links (either by physical protection or by cryptographic techniques)[2]. Hence, we can easily extend figure 2 to a multilevel secure virtual enterprise computing model as in figure 3.

Again, distributed portions of a virtual domain need a secure communication protocol among them. In the following sections, we concentrate on how this multilevel virtual protection model can be realized.

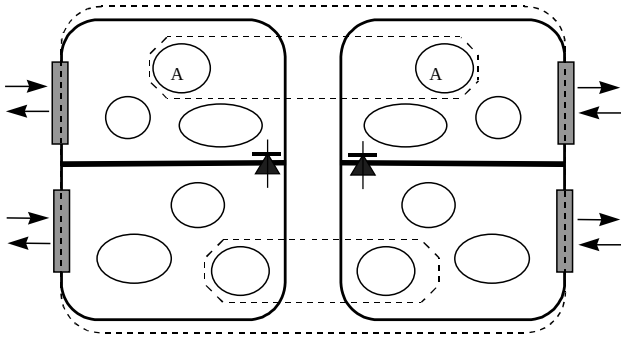

Figure 3: Two-level virtual enterprise model

3. DOSA (Distributed Object Security Architecture)

Information systems exist to enable users to access and manage information. Hence an MLS information architecture should answer the following two questions.

- How can a user access globally distributed data based on their clearance and need to know without leaking information?
- How can a high-level user integrate information from various locations at different security levels using (multilevel or single-level) applications without introducing security vulnerabilities?

Hence, throughout this paper, we try to answer the above two questions in DOSA and for the traditional MLS approach.

A practical way to achieve multilevel secure interoperability is to compose single-level distributed systems that enforce domain security policies to provide multilevel functionality in a secure way (i.e., the

[2] Interoperability models of CORBASecurity 1.1 (chapter 8) address the possibility of this configuration.

multiple single level (MSL) approach). Our solution is based on the following observations.

1. When information at lower security levels is located on another system in a distributed computing environment, a read-down becomes a write-down because a request must be sent to the ORB, which must send a message to the lower level system; this violates the security policy. Automatic ways to generate requests and send them to other systems at lower security level are vulnerable to Trojan Horse attacks.
2. Information in distributed (non-MLS) systems is often replicated to other systems to enhance performance and increase availability.
3. A copy of the same application or information can reside in systems at many different security levels. But, in general, the users at different levels of trustworthiness (clearances) do not sit side by side in the workplace. The levels of trustworthiness usually determine the security level of the enclave in which users reside and access information.

For example, DoD's current operational infrastructure consists of several physically separate networks, operating at different system-high levels. Users working at different security levels need access to information and services at lower security levels (i.e., in a separate network). To achieve secure interoperability within this infrastructure requires strong control of information flow across physical security boundaries. Our approach aims to use this infrastructure as the foundation for an MLS architecture that consists of single-level CORBAs (i.e., at least one ORB per security level). From a system of systems perspective, this approach produces an MLS system consisting of multiple single level (MSL) systems. MLS client-side and server-side solutions, that make low information and services available securely to high level users and applications, control the flow of information across security boundaries.

Architectural Overview

The MLS server-side solutions are SINTRA (Secure INformation Through Replicated Architecture) [5, 7] and the NRL Pump [6, 8, 9] both from the Naval Research Laboratory (NRL). NRL has demonstrated the effectiveness of physical separation to provide security, and replication for information sharing in several SINTRA relational database prototypes. Also NRL's efforts produced a one-way store-and-forward device, the NRL Pump, to enable secure and reliable communication from a low security level system to a higher level system.

The advent of asynchronous messaging and asynchronous replication servers for databases have made the SINTRA approach, in conjunction with the NRL Pump, quite tractable. Communication from a low security level to a higher level does not introduce security vulnerabilities; however, information flow from high to low is prohibited. In other words, asynchronous communication from a low security level to a higher level can be allowed in an MLS environment if all responses from high to low are blocked. Commercial availability of asynchronous communication services in general allows this approach to be used for making low information available to high systems and applications. In other words, the NRL effort has produced an MLS server that uses COTS products with a small security device, the NRL Pump, that enables the untrusted *applications* at high-level systems to access lower-level information securely as shown in figure 4.

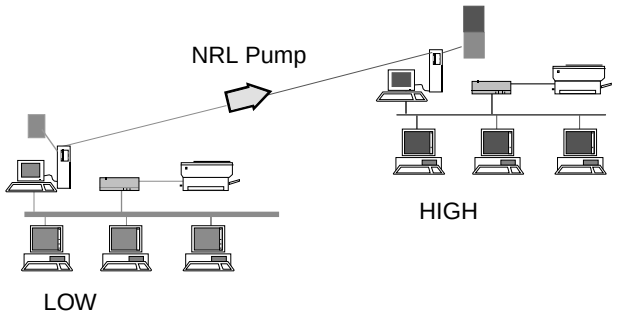

Figure 4: An MLS server that consists of COTS products and the NRL Pump

Some lower level information that is regularly used by higher level applications can be replicated from the lower level to the higher level systems. However, there is some unpredictable information that is needed by higher level users from time to time. Also sometimes, the volume of information prohibits replicating all lower level information to high level systems. One way to accommodate these needs is to equip the high-level users with high-assurance MLS workstations for accessing lower-level resource.

The Australian Defense Science and Technology Organization (DSTO) has developed a security device, (Starlight) Interactive Link [1]. This device allows a user of a COTS X Windows workstation to redirect the output of his keyboard to login to lower level servers to browse, send messages or have data sent to the higher enclave for future analysis. Throughout this paper, a client workstation that is enabled with this Interactive Link is called a Starlight-client. For example, a user in a Secret enclave could establish a connection to unclassified and send email or browse the Internet (see

figure 6). The Australian effort has produced an MLS client that uses COTS products with a small security device that enables MLS connectivity for high-level *users*. In other words, a high level user can establish simultaneous connections to systems at many different security levels through Starlight interactive link.

Our overall approach is to develop an MSL CORBA with conventional (system-high single-level secure) ORBs running at each security level to provide overall MLS services. The NRL Pump can be configured as a CORBA object and multilevel clients (e.g., Starlight-client) can be configured as CORBA clients. Of course if the Starlight-client is located at $level_j$, then the highest security level of the multilevel workstation is $level_j$. CORBA asynchronous communication services (e.g., OrbixTalk) in conjunction with the NRL Pump can send information from low level systems to high level systems. In such cases, the NRL Pump must have interfaces for CORBA's asynchronous communication services (see figure 5).

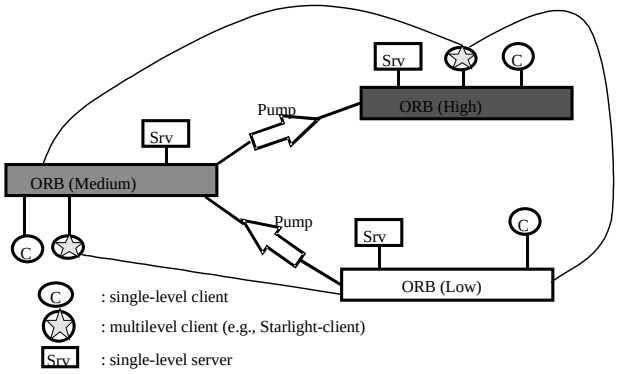

Figure 5: Overall approaches of CORBA-compliant MSL

The architecture-based security shown in figure 5 provides strong separation that can protect against catastrophic security problems. Other security mechanisms, such as firewalls and CORBA gateways can be easily integrated into the above architecture (between enclaves or between ORBs) to ensure that only authorized users have access to resources within an enclave. Encryption, intrusion detection, role-based access control, type enforcement mechanisms, and other need-to-know discretionary access control mechanisms and Java security also play important roles in protecting information both within and between physically separate enclaves. The ability to take advantage of single-level distributed computing functionality and to develop mission critical applications independent of security solutions makes this approach extremely attractive for near-term operational use.

Advantages of the MSL approach are as follows.

- It is a sound framework for balanced assurance that can adopt strong and weak security mechanisms at the appropriate protection boundaries. Hence, it is easier to build high-assurance systems.
- It isolates security enforcement into simple mechanisms. Therefore, it enables the use of existing security mechanisms and the development of operational solutions independent of security solutions.
- It enables the secure use of unmodified COTS client and server products and the use of new protocols. This facilitates migration to new standards, exploitation of technology advances, and reduces training, maintenance and system cost.

The Starlight-client in CORBA Environments

Adapting the Starlight-client to a CORBA environment poses no problem. To understand why CORBA does not affect the Starlight-client, we need to understand how the Starlight-client works. The Starlight-client, which uses the X window system, works as follows:

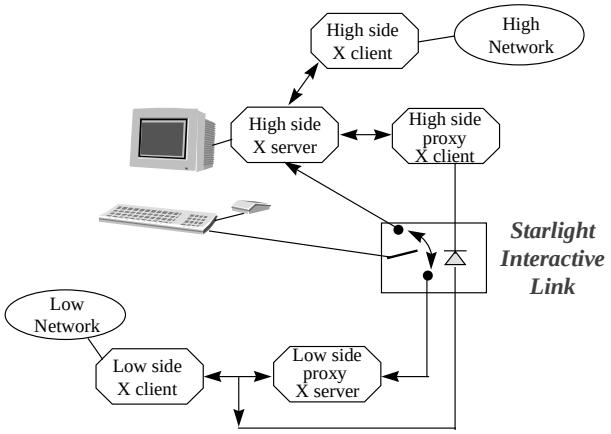

Figure 6: The Starlight-client

In figure 6, the X client is a user program or application (e.g., Netscape) that knows how to talk to the X server. The X server is a local program that controls display resources. In other words, the X server also acts as a mediator between X clients and the resources (e.g., windows, cursors, fonts) of the local system.

The goal of the Starlight-client is to run any X client (i.e., application) on the low network but display low output on the high network. To achieve this goal, in general, low X client, high X server, and bi-directional

communications between them (e.g., display geometry, color support) are necessary. However, the data diode in the Starlight interactive link does not allow any communication from the high side to the low side. Therefore, the Starlight-client uses a high-side proxy X client and low-side proxy X server to mimic the real low-side X client and high-side X server.

In a CORBA environment, High and Low networks in figure 6 can be replaced with High and Low ORBs. In that case, Low and High X-clients should be CORBA-enabled X-clients. It is not too difficult to construct ORB-enabled X-clients. One such example is ORB-enabled applets running on X-window based Web-browsers.

The NRL Pump in the CORBA environment

The NRL Pump is an one-way device that delivers messages from Low to High. It is designed to be:

- an application independent device,
- used in conjunction with unmodified COTS applications that support asynchronous communications.

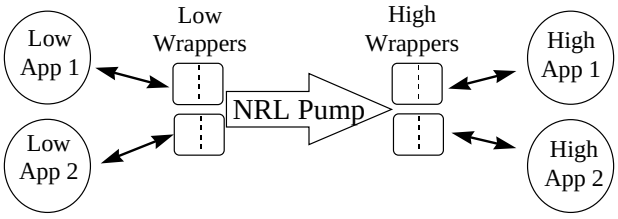

Figure 7: A typical configuration of the NRL Pump

In a typical NRL Pump configuration (figure 7), the low level to high level communication is actually a communication between application programs running on a lower level computer (sender) and application programs running on a high level computer (receivers). Note that application programs that expect some data values coming back as a result of computation cannot be used with the NRL Pump because the NRL Pump does not allow data values to pass from High to Low. Therefore, the application programs that can utilize the NRL Pump are those which can function with ACKs (or no ACKs) from the receiver program. In our SINTRA prototype, we were able to use the NRL Pump with the Sybase replication server that requires a little more than just ACKs (e.g., last commit ID) by generating the required information without actually accessing the Sybase SQL server that is located on the high side [3].

In general, an application program (i.e., sender) that functions with ACKs is able to ensure reliability at the application level by receiving ACKs from the receiver in the specific format determined by the application protocol. If the sender program directly sends a message to the NRL Pump, the NRL Pump, which is an application independent device, cannot return an ACK in the format specified by the application protocol. Therefore, wrappers, which are application-specific, are needed to ensure the correct formatting of application ACKs. In other words, wrappers satisfy both the application-specific protocol and the NRL Pump protocol.

Previous work [3] demonstrates that the NRL Pump works well with COTS applications such as database replication servers and FTP. However, in a CORBA environment, we want to demonstrate how the NRL Pump works with applications that use CORBA style communication. The CORBA specification mainly focuses on a synchronous programming model[3]. One CORBA service that provides asynchronous communication is the event service. OMG plans to provide more asynchronous programming models in the near future[4]. Since the event service is intended to deliver events not messages, some implementations may not be able to handle long messages (i.e., the time and capacity limits of storage may not support large messages).

We chose OrbixTalk[5] as our first CORBA style[6] application that will be incorporated with the NRL Pump because

- it provides asynchronous messaging functionality, and
- it is a basis for a range of application areas, such as multicasting and event-driven programming,
- it is the basis for the Orbix event service that is an implementation of CORBA's event service. Hence, the *suppliers*, *consumers*, and *event channel* of CORBA's event service directly correspond to the *talkers*, *listeners*, and *network* of OrbixTalk (see figure 8).

OrbixTalk in conjunction with the NRL Pump will provide a capability for multicasting and event services across security boundaries. It is important to support

---

[3] A client sends a request to a server and waits for the reply from the server.

[4] Recently, the Object Management Group (OMG) issued a RFP for an Asynchronous Messaging Service.

[5] OrbixTalk is trademark from IONA Technologies Ltd.

[6] Only contract between client and server applications is interface definitions in CORBA IDL.

event-driven programming and multicasting across security boundaries because they are popular paradigms in distributed computing environments. For example, situation changes at lower levels that can trigger/alert higher level activities can be programmed in an event-driven model across security boundaries.

OrbixTalk de-couples talkers (i.e., message generators) and listeners (i.e., message receivers), and forms an uni-directional (asynchronous) message stream. In general, *m* talkers can talk to *n* listeners through a single message stream. Also the talker does not have explicit knowledge of listeners and the listener does not need to know about the talker. OrbixTalk allows shared information to be organized according to topics. The talker simply specifies the topic it wants to talk about and sends out messages, and a listener specifies the topic it wants to listen to and receives messages.

Of course, an OrbixTalk implementation introduces some bi-directional communication. For example, OrbixTalk maintains a service that translates topic name to IP addresses through the "directory enquiries server". This server communicates to both talkers and listeners. However, OrbixTalk in conjunction with the NRL Pump will produce a secure reliable one-way channel. One method to implement OrbixTalk across security boundaries is shown in figure 8.

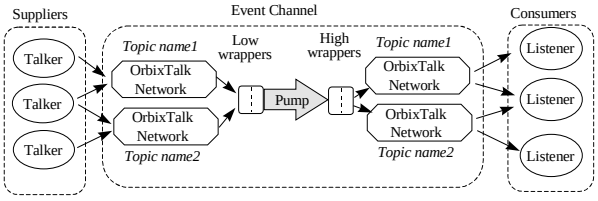

Figure 8: A simple configuration of the NRL Pump and OrbixTalk where the dotted boxes represent CORBA's event service.

In the above scheme, a low wrapper is a proxy for listener(s) that represents potentially many high listeners. There may be one low wrapper per topic or one low wrapper for many topics. High wrapper is a proxy for talker(s) that potentially represents many low talkers. In the above scheme, low talkers talk without knowing that there are high listeners. The high administrator can set up low wrappers (i.e., register for topics that are of interest to the high listeners) through a Starlight-client.

Combining the NRL Pump and the Starlight-client in the CORBA environment

As we saw in the above, the NRL Pump, Starlight-client, and CORBA are similar in that one side has a client (sender) with a server (receiver) proxy, and the other side has a client (sender) proxy with a server (receiver). All three components can be configured as follows: the NRL Pump resides between server-side applications, and the Starlight Interactive Link resides between X clients. In other words, the two security devices connect ORBs at different security levels to form an MLS CORBA.

Note that our approach is different from other distributed MLS approaches (see section 6) in that our approach realizes a distributed MLS system by composing multiple single-level system-high systems with a few security devices (e.g., NRL Pump, Starlight-client). Our approach successfully provides

- a way for users to access globally distributed data based on their clearance and need to know through MLS workstations (e.g., Starlight-client) and
- a mechanism for a high-level user to integrate information from different locations at security levels through replication of lower level information to higher level systems via one-way devices (e.g., NRL Pump).

4. MLS Collaborative Computing

We have introduced a distributed MLS *architecture* that is cost-effective, practical, and secure. However, a distributed architecture alone cannot meet the needs of users in distributed environments. What we need is a new MLS distributed computing *paradigm* that can help users at different locations and at different security levels cooperate. Users may independently work on activities that together provide a solution for some enterprise level problem without knowing details about the problem. These activities combine to support the enterprise business processes. Ad-hoc collaboration (by arbitrarily exchanging files or e-mail) does not satisfy the information management needs of enterprises. Global information management strategies based on a sound distributed architecture are the foundation for effective distribution of complex applications that are needed to support ever changing operational conditions across security boundaries. In other words,

- distributed information management activities should be defined,
- these activities should be distributed to the appropriate organizations at the right security levels, and
- administrators should be able to control and monitor those activities.

For some time, enterprises have described their activities as collections of business processes and have tried to streamline those processes within a single security level. Modeling business dynamics as a set of processes has allowed enterprises to use information technology more effectively. A workflow management system (WFMS) assists users in managing dependencies among activities and monitoring business processes.

To enable enterprises that need to perform activities across security boundaries, an MLS workflow based on the architecture that was described in section 3 is needed. MLS WFMSs can support strategic and tactical planning processes, weapons acquisition processes, travel arrangement processes, distributed simulation administrative processes, command and control, mission planning, and logistic processes. These activities may take place at different security levels, and the data that the activities require may be classified at different security levels. Therefore, when the activities that make up these processes are incorporated into an MLS workflow, not only their functional dependencies but also their security constraints must be considered.

To develop MLS WFMSs, several types of tools that support and enforce the MLS security policy will be required. Tools for analyzing the dependencies among activities at different security levels support workflow design. An activity can be performed in many systems by many people (some site or activities may be preferred to other sites or activities) in a WFMS. Hence, a WFMS may need some scheduling scheme that can avoid failed and overloaded systems, and consider the preferred route without violating MLS constraints. Activities may be triggered by prior activities at different security levels or dynamically changing situations (potentially from a different security level). Each activity may need to report its current condition to a workflow monitor; they may also need to pass information from one activity to others. To support MLS workflow on an open object bus (e.g., CORBA), asynchronous messaging schemes (e.g., asynchronous multicasting) are needed in order to start activities at different security levels. A pragmatic and secure initialization process must be developed.

MLS WFMSs require MLS distributed information management schemes. In particular, they need long-lived, flexible, heterogeneous MLS transaction models that manage diverse operations in heterogeneous computing environments across security levels. A WFMS requires recovery mechanisms in case of a system failure. They need a scheme that can recover across heterogeneous environments and that can recover from the fail point; this type of recovery is known as forward recovery.

As mentioned above, MLS workflow activities need to send messages to the workflow monitor or to other activities at the same or higher security levels. Activities need to verify the authenticity and integrity of messages that they received. In some cases, secrecy of messages must be preserved. Cryptographic techniques for authentication, integrity, and secrecy among activities at the same security level are well known. However, those techniques need to be extended to support authentication, integrity, and secrecy among activities across the different security levels.

The key to the success of our efforts is maximum use of unmodified COTS CORBA-compliant clients and servers. This will facilitate the migration to new Information Technology (IT) standards, the exploitation of technology advances, and the reduction of training, maintenance, and system cost. CORBA, Java, and DOSA will be the foundation for MLS collaborative computing. As a first step toward this new computing paradigm, we prototyped an MLS CORBA that is described in section 5.

5. MLS CORBA Prototype

CORBA promises language and platform/OS independence among applications in a distributed environment. To demonstrate that the CORBA promise can be extended to MLS environments, we have built a prototype that uses two hardware platforms (SUN and PC) and two operating systems (Solaris and WindowsNT), two programming languages (C++ and Java), and Orbix/OrbixWeb[7]. To support multi-level event-driven programming and multicasting across a security boundary, we also incorporate OrbixTalk in conjunction with the NRL Pump.

The hardware configuration of our prototype is shown in figure 9.

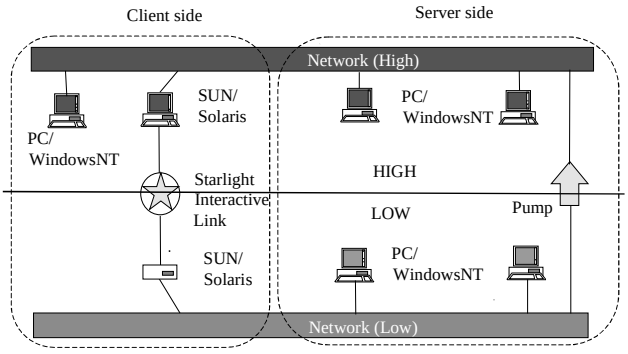

Figure 9: Hardware configuration of the prototype

[7] Orbix is a C++ implementation of the CORBA and OrbixWeb is a Java implementation of the CORBA.

In the client side, we have two different types of hardware (see figure 9) and two different sets of software. The client software on a PC/WindowsNT is CORBA-enabled Java client applications. Hence, OrbixWeb is required on that platform. The SUN/Solaris platform is equipped only with a Java enabled Web browser. Hence, CORBA-enabled Java applets (i.e., client applications) and related CORBA libraries should be dynamically downloaded from a server. Thus the two approaches are distinguished by one being statically CORBA-enabled and one being dynamically CORBA-enabled.

These approaches have their pros and cons. If each client is equipped with CORBA-enabled client applications (e.g., our PC/WindowsNT client), it reduces the download time. However, whenever the client application is updated, a new version has to be distributed and loaded to the system. Also each client may have to be equipped with the CORBA library. On the other hand, if the clients are equipped with a Java enabled Web browser and download CORBA-enabled client applets, then updating and distributing the client programs is much easier (especially for roaming clients).

Our servers are written in C++ and Java, running on a PC/WindowsNT platform. Lower level servers are also equipped with OrbixTalk for asynchronous message delivery from low level applications to the low Pump wrapper. Since low level servers replicate messages through OrbixTalk, the low level servers can be very generic (i.e., they do not have to know where the messages have to be delivered). All they have to do is to publish their results for any processes that are interested in them.

The process architecture of our prototype is shown in figure 10. The location of each process is straightforward except for the Pump wrappers. The wrappers can be located in any server platform. Note that we may have different wrappers for different applications.

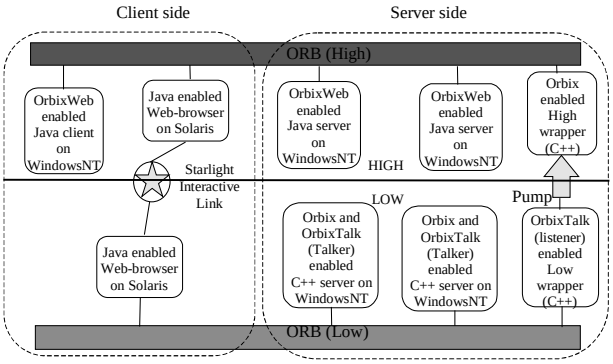

Figure 10: Process configuration of the prototype

At this time, our prototype uses regular ORBs. However, once CORBA security services are available, we plan to incorporate them into our prototype.

6. Traditional Approach to MLS Security

The approach we have taken in the DOSA project is quite different from the traditional TCSEC approach to MLS computing and may seem too good to be true. In this section, we contrast the DOSA approach to other traditional distributed MLS approaches.

When early security experts first introduced the concept of a trusted operating system and a trusted computing base (TCB), computers were a very costly resource and only very talented, highly trained technologists interacted with them. Building high assurance trusted operating systems became the primary goal for the MLS community. It was quite difficult and expensive to build these trusted operating systems and very hard to gain the necessary assurance that the trusted software enforced the security policy. At this time, the government's share of the information technology market was dominant.

With the introduction of the personal computer, computing power became more affordable for the common person. Today's computing resources are examples of a commodity whose cost has dropped while the capabilities of the products have increased over time. The government's share of the IT market is less than 10% today; hence, its ability to influence commercial IT products has become less. The computing model has also changed to a distributed object model.

MSL is more affordable, primarily because it promotes the use of commercially available products with a few simple specialized security components to enforce strong separation required by national MLS policy. User expectations to have the latest IT advances on their desktop can be realized, as well as access to legacy

resources. MSL facilitates the development of mission critical applications without having to address security restrictions imposed by special-purpose MLS operating systems. Traditional MLS approaches attempt to emulate evolving technology but do not keep up with rapidly evolving IT advances.

The MSL approach provides very strong separation and completely obviates a solution for the traditional assurance composition problem. Because users at different security levels access different copies of lower level information, traditional Trojan Horse attacks are not effective in a strict MSL approach (i.e., only upward flow is allowed). The need for downgrading introduces a vulnerability that malicious code can exploit. The protection afforded by the MSL approach is so effective because it minimizes resources shared across security levels.

The MLS community takes the Trojan Horses threat seriously. Trojan Horses can hide in untrusted software at different security levels and leak information from a higher to a lower level. Traditionally, the MLS community deals with this threat by

1. isolating single-level (untrusted) information and processes from those of different security levels
2. restricting write-down capability (i.e., only trusted processes can write-down when it is necessary).

An MLS operating system mediates access to a shared memory by users at different security levels. High level users or applications access lower level information via a read-down mechanism. A read-down mechanism is allowed because lower level applications/data cannot tell when the higher level applications/users read its information; hence, a lower level Trojan Horse cannot receive any information. Most MLS systems enforce BLP.

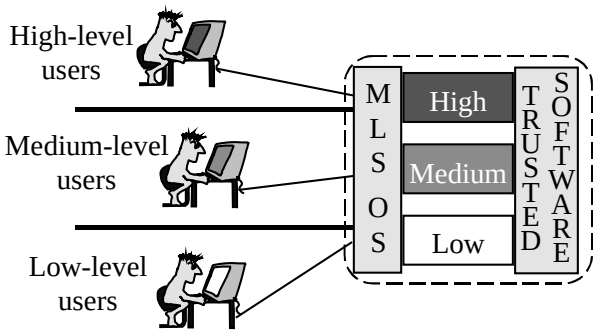

Figure 11: A traditional MLS system

In figure 11, high level users and applications can access and integrate lower level information through read-down mechanisms that are performed via an MLS operating system (OS). Trusted software can allow processes to read from and write to multiple security levels.

However, the read-down approach for accessing information from many security levels may leak information when the lower level information is located in different systems. If all computing systems in the distributed environment are MLS systems, they all have the same assurance level, and they all mediate access across the same security range (e.g., Top secret to Confidential), then it may be protected from Trojan Horse attacks. However, this is a very unlikely scenario in today's heterogeneous and distributed computing environments because

(1) it is inevitable that information from unclassified network will be accessed, and

(2) excellent COTS (untrusted) products, in terms of functionality, user interface, and cost, are available (i.e., it is too attractive not to use them).

To overcome the read-down problem in MLS distributed environments, other schemes have been proposed [4] (see figure 12).

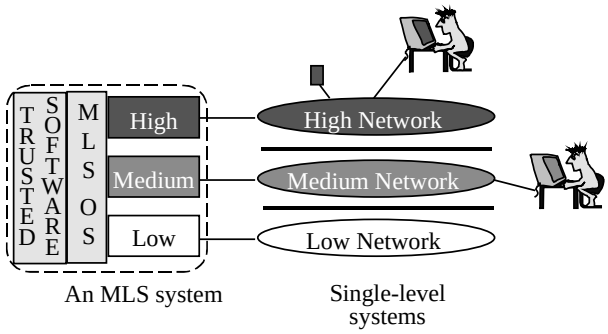

Figure 12: A distributed MLS system

In this approach, MLS systems are used as a mediator between users/processes at different security levels. When higher level users/applications need to access lower level information on other systems, the request for the information is sent down through the MLS OS/applications. Sending a high-level request to a lower level system is a write-down operation that violates BLP. Even though the MLS OS, MLS applications, and users are all trusted, it is not easy to tell what other untrusted processes are between the user and the trusted software. For example, when a user accesses an MLS database system, he/she usually does so through a graphical user interface (GUI) tool. It is very difficult to determine whether there is any other process that may reside between the GUI and the MLS database. In addition to building the trusted software, it is very difficult to establish a trusted path between the user and

the trusted process, and between trusted processes in today's multi-tier software architecture.

For these reasons, a high-assurance MLS ORB (if one is ever built) or MLS OSs cannot solve the secure interoperability problem. A high-assurance MLS ORB residing on an MLS operating system to enforce the non-by-passable property will inherit all the security vulnerabilities of the above approach.

7. Conclusions

In this paper, we have demonstrated a practical way to achieve multilevel secure interoperability through the MSL approach, which constructs MLS distributed systems from multiple single-level, distributed systems in conjunction with a few specialized security devices. A generic MLS distributed architecture is shown in figure 13.

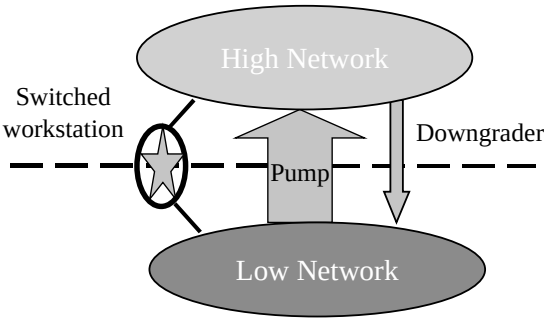

Figure 13: A generic MLS distributed architecture

In this architecture, switched workstations (e.g., Starlight) enable high-level users to access lower-level resources, one-way devices (e..g, NRL Pump) provide a secure way to replicate lower-level information for higher-level applications and users, and downgraders provide capability to violate the security policy for trusted software. We emphasize that downgrades should be performed locally by an authorized data owner. The philosophy of this architectural approach to MLS computing is to minimize the need for downgrading.

In this paper, we also described our prototype efforts to illustrate that cost-effective high-assurance MLS distributed solutions are possible. Our prototype, constructed from single-level ORBs, not only demonstrates a practical distributed MLS infrastructure but also shows how to manufacture MLS event-driven programming and multicasting from single-level counterparts.

Thus far, our effort in the DOSA project has concentrated upon setting up an MLS infrastructure for distributed applications. This is an important first step toward distributed MLS cooperative computing (e.g., MLS Workflow) running on the MLS infrastructure. MLS cooperative applications are a key to the success of MLS distributed computing.

References

1. Anderson, M., North, C., Griffin, J. Milner, R., Yesberg, J., and Yiu, K. "Starlight: Interactive link," 12th Annual Computer Security Applications Conference, San Diego, CA, 1996.
2. Boebert, W. E. and Kain, R. Y. "A practical alternative to hierarchical integrity policies," Proceedings of the 8th National Computer Security Conference, Gaithersburg, MD, 1985.
3. Froscher, J. N., Goldschlag, D. M., Kang, M. H., Landwehr, C. E., Moore, A. P., Moskowitz, I. S., and Payne, C. N. "Improving inter-enclave information flow for a secure strike planning application," Proceedings of 11th Computer Security Applications Conference, pp. 89 - 98, New Orleans, LA, 1995.
4. Jensen, C., *et. al*. "SDDM: A prototype of a distributed architecture for database security," Proceedings of Conference on Data Engineering, 1989.
5. Kang, M. H., Froscher, J. N., and Costich, O. "A practical transaction model and untrusted transaction manager for multilevel-secure database systems," Proceedings of 6th Annual IFIP WG11.3 Working Conference on Database Security, pp. 289 - 310, 1992.
6. Kang, M. H. and Moskowitz, I. S. "A Pump for rapid, reliable, secure communication," Proceedings of ACM Conference on Computer & Communication Security, pp. 119 - 129, Fairfax, VA, 1993.
7. Kang, M. H., Froscher, J. N., McDermott, J., Costich, O., and Peyton, R. "Achieving database security through data replication: The SINTRA prototype," Proceedings of 17th National Computer Security Conference, pp. 77 - 87, Baltimore, MD. 1994.
8. Kang, M. H. and Moskowitz, I. S. "A data Pump for communication," NRL Memo. Report 5540-95-7771, 1995.
9. Kang, M. H., Moskowitz, I. S. and Lee, D. C. "A network Pump," IEEE Transactions on Software Engineering, vol. 22, no. 5, pp. 329 - 338, 1996.
10. Moskowitz, I. S. and Kang, M. H. "An Insecurity Flow Model," Proceedings of New Security Paradigms Workshop, Cumbria, UK, 1997.
11. Object Management Group "CORBA Security," OMG document 97-02-20, 97-02-21, 1997.