Distributed component software security issues on deploying a secure electronic marketplace

Stefanos Gritzalis
Department of Informatics, Technological Educational Institute (TEI) of Athens, Athens, Greece
John Iliadis
Department of Information & Communication Systems, University of the Aegean Research Unit, Athens, Greece
Spyros Oikonomopoulos
Department of Informatics, Athens University of Economics and Business (AUEB), Athens, Greece

Keywords
Distributed data processing, Electronic funds transfer, Computer software, Security, Computer architectures

Abstract
A secure electronic marketplace involves a significant number of real-time transactions between remote systems, either for commercial or for authentication purposes. The underlying infrastructure of choice to support these transactions seems to be a distributed component architecture. Distributed component software (DCS) is the natural convergence of client/server network computing and object oriented technology in a mix providing reusability, scaleability and maintainability for software constructs. In DCS a client acquires references to objects provided by components located to remote machines and invokes methods of them as if they were located in its native environment. One implementation also provides the ability to pass objects by value, an approach recently examined also by others. The three major models in the distributed component software industry are OMG's CORBA, Sun's Enterprise Java Beans, and Microsoft's DCOM. Besides these, we will discuss the progress for interoperable DCS systems performed in TINA, an open architecture for telecommunications services based on CORBA distributed components. In this paper the security models of each architecture are described and their efficiency and flexibility are evaluated in a comparative manner. Finally, upcoming extensions are discussed.

Information Management & Computer Security
8/1 [2000] 5–13

[ISSN 0968-5227]

1. Introduction

Distributed component software (DCS) is the extension of client/server architectures with object oriented technology in mind. In the classical client/server model a client machine uses the computing power of a server by calling procedures located in the latter (Remote Procedure Call). The server, who typically has access to a database system, returns data to the client who, rarely after performing some extra operations on them, displays them to the end user (Staamann, 1997). Several problems arise though, when the load on the server becomes heavier. Increased local (around the server) network traffic and slow processing are some of the obvious consequences in such a case.

To address these problems the idea of distributed object calls surfaced. In this model the client calls functions of objects, located on remote machines, as if they were in his own process space. The underlying infrastructure takes care of network-location transparency and load balancing (forwarding object reference requests to machines with less load). The general model found in all three architectures presented in this paper is schematically described in Figure 1.

As seen in Figure 1, the client application calls a method of an object residing in a server component. The call is made as if the object were in the caller's process space. What actually happens, though, is that the call is forwarded to a surrogate "stub" of the called object, whose responsibility is to forward requests on the server side, through the underlying object broker infrastructure. The stub "marshals" the method parameters and passes them to the object broker, which passes the request to a skeleton object on the server. The server "unmarshals" the parameters, performs all necessary up-calls to the actual object implementation, and passes back the return parameters to the client in an identical manner.

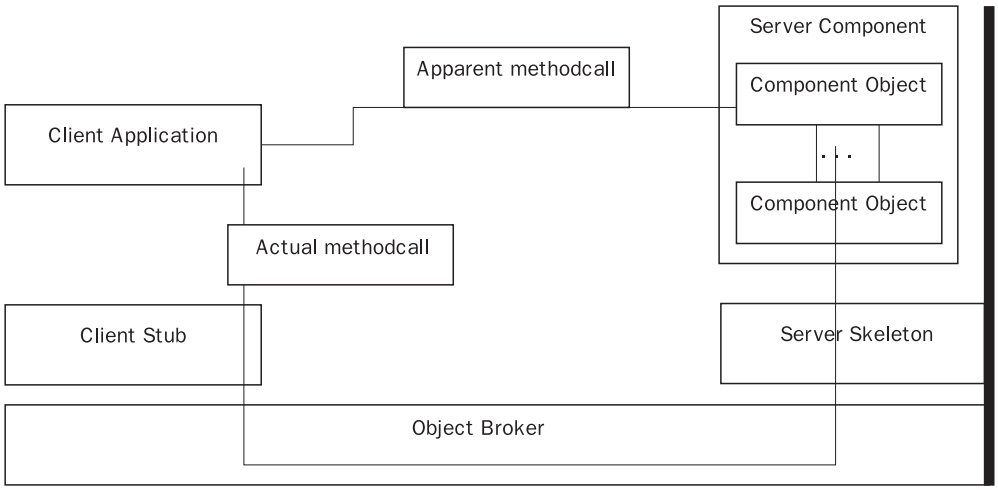
Electronic commerce transactions, on the other hand, involve the establishment of a number of client-server-like constructs in order to be completed. With the expansion of e-cheque and e-cash systems, and the participation of Registration and Certification Authorities (Gollmann, 1999) for the implementation of digital signatures, the required number of these constructs in order to complete a transaction will significantly grow.

For business to customer communication the most prevalent scheme seems to be through the customer's Web browser. On the server side though, authenticating customers, performing financial transactions, initiating order completion and carrying out other e-commerce related tasks, requires the use of object services on several remote machines. For business to business transactions, applications will heavily use remote object services directly, be it within the client organisation itself or on the provider organisation machines. Note that "client" and "provider" are roles that both organisations interchange during an electronic transaction.

Vendors like Sun, Microsoft, IBM and independent organisations present extensive examples for this type of constructs and how their own distributed component architectures provide the infrastructure that is necessary to implement such applications. Each of these infrastructures implements a different security scheme.

This paper addresses the security issues involved in distributed component architectures. In section 2 we provide a brief overview of the threats that a DCS system faces. In section 3 we comment on the CORBA specification and in particular the security mechanisms it specifies, in order to confront the threats presented in section 2. In section 4 we describe the DCOM model for

Figure 1
Distributed object method calls



implementing DCS and we discuss the security scheme this model incorporates. In section 5 we present the Enterprise Java Beans distributed component architecture and we discuss the security infrastructure it relies on, in order to confront with the threats that DCS faces. In section 6 we provide a comparative evaluation of the aforementioned schemes for implementing DCS. Finally, in section 7 we present our conclusions.

2. Threats in distributed component software

The threats that distributed component software systems have to deal with, include the following (Spinellis *et al.*, 1999):

- unauthorised disclosure of information;
- violation of data or code integrity;
- denial of service;
- repudiation of user's actions;
- malicious code, achieving user's annoyance;
- traffic analysis.

Distributed systems have characteristics that render more difficult the successful addressing of the aforementioned threats. Specifically, DCS systems are dynamic; their internal structure as well as the software components they comprise of, may change without altering the system itself or the capabilities and interfaces it provides. This renders the task of providing a uniform security infrastructure more difficult. In addition, DCS systems can be quite heterogeneous from a security point of view. They comprise of multiple, usually different security policy domains, the interoperability of which is hard to achieve. Moreover, the security mechanisms applied in DCS systems are layered. Special care has to be taken when designing layered security mechanisms as the potential for violating them by taking advantage of loose points in their boundaries is high (Gollmann, 1999). Finally, the administration of DCS security mechanisms is rather a difficult task because a uniform administration interface is not always possible to design and implement.

3. CORBA

CORBA (OMG, 1999), (CORBA, 1998a) is the specification of a framework that allows the interaction between users and objects in a transparent way, independent of the underlying operating systems and hardware. The Object Request Broker (ORB) introduced by CORBA is a "software bus" (Gollmann, 1999) that provides the ability for users and objects to interact. Communication, naming transaction, querying and security services are among the ones that are provided transparently by CORBA to DCS. CORBA itself does not provide functionality; it provides interfaces to be used by a variety of programming languages and a variety of environments. CORBA provides an abstraction layer to the applications as far as the network communication is concerned. The communication protocols supported by CORBA are the following:

- General Inter-ORB Protocol (GIOP);
- Internet Inter-ORB Protocol (IIOP) (CORBA, 1998b); and
- Secure IOP (SECIOP) which is used in combination with the previous two. It is part of the CORBASEC specification (CORBA, 1998c) and it is meant to provide

Information Management &
Computer Security
8/1 [2000] 5–13

an integrated security specification and well-defined interfaces for the CORBA security services.

The common secure interoperability (CSI) specification defines the standards for secure interoperability when using the General Inter-ORB Protocol or the Internet Inter-ORB Protocol. It contains standard security mechanisms, associated cryptographic algorithms and details of the SECIOP protocol messages when using these mechanisms and algorithms. Finally it defines the security functionality supported when ORB interoperation takes place, using these security mechanisms.

CORBA security

The ORB provides protection against unauthorised access, masquerading, bypassing of security controls, disclosure of information, violation of information integrity and facilities to allow for user accountability. CORBA applications may be security-aware or security-unaware. In the first case, applications can co-operate with the ORB in order to deploy an integrated layer of security services; in this case, applications can even control the way these security services operate, and adjust the parameters of their operation. In the second case, security has to be put in place and security parameters have to be adjusted without the co-operation of the application itself.

CORBA security depends on a distributed trusted computing base (TCB) in order to ensure that the security mechanisms will not be tampered with and the security policy will be enforced. However, one should take under consideration that this distributed TCB includes also the underlying hardware and operating systems, involved in a DCS system. This constitutes a fundamental problem, since this TCB may not actually provide the desired level of security.

The key security features included in the CORBA specification include the following:

1 Identification and authentication of principals.
2 Authorisation and access control. Privilege attributes are given to principals after the authentication concludes and they include the identity of the principal, roles assigned to the principal, groups the principal belongs to, a security clearance, objects and operations that can be performed on these objects (capabilities) as well as other privileges defined by the applications.
3 CORBA supports privilege delegation in order to allow an object in a chain of objects act on behalf of the principal that initiated the chain. It should be mentioned that only security-aware CORBA applications can specify delegation options. Delegation (CORBA, 1998c) can be simple, composite, combined and traced. These different types of delegation offer a fine control on the way the privileges are being delegated and the level of auditing that can be performed concerning this delegation by the original holder of privileges.
4 Security auditing, which is based on system audit policies (logging security events pertinent to system activity) and application audit policies (events defined by the security applications).
5 Integrity and confidentiality of communication between objects. The CORBA specification provides interfaces that can handle a number of different cryptographic components, used to implement data integrity and confidentiality mechanisms. These cryptographic components are transparent to the applications themselves.
6 Non-repudiation. CORBA provides interfaces for the implementation of non-repudiation services. Non-repudiation of origin and non-repudiation of receipt are supported, however the successful completion of a request to an object could be repudiated because the generation of evidence of the aforementioned completion is not supported.
7 Security policy. Security policies are enforced either by the ORB on object invocation, or by the applications themselves.
8 Security mechanisms independence. It is possible to replace some security mechanisms independently. Some of these mechanisms are: authentication and security association; access decision policies; audit and non-repudiation.

Telecommunications information network architecture (TINA)

Distributed component architectures are expected to be used in a vast variety of applications. A major effort to combine their ease of deployment and flexibility with the capacity of high speed networks is the Telecommunications information network architecture (TINA) (Staamann, 1997). TINA is an effort to establish an open architecture for telecommunication services based on distributed components. TINA is expected to play a significant role in e-commerce transactions involving electronic merchandise. A broad range of applications

Information Management &
Computer Security
8/1 [2000] 5–13

in this category will heavily depend on digital multimedia information. Video-image servers and digital libraries are some of the most representative examples.

The underlying infrastructure will be based on CORBA compliant products. Of particular interest is TINA's security architecture, CrySTINA (Staamann *et al.*, 1998). It is aligned with the CORBA security specification, but it still considers enhancements necessary when it comes to converge security policies and capabilities of different systems, in order to ensure the preservation of security permissions through different security contexts. The central idea is that of a security association setup service, responsible for establishing a security association between client and server object. The security association setup service provides authentication, security parameters (e.g. cryptographic keys) exchange, and security policy negotiation. The exact way this negotiation is to take place has not yet been defined.

4. DCOM security model

DCOM (Horstmann *et al.,* 199; Microsoft, 1996) is Microsoft's extension to the component object model (COM), its own standard for building components (Rogerson, 1997; Dulepet, 1998). DCOM provides the necessary infrastructure for COM objects to communicate over a network; it is the distributed equivalent of interprocess communication, which occurs between processes on a single machine. As in the other approaches, communication takes place in a transparent way both for the client and the server component.

DCOM was first implemented in Microsoft's Windows NT Operating System. Its security model relies heavily on the security capabilities supplied by Windows NT (Concalves, 1998). Implementations on other platforms will use the security providers of these platforms in a similar or identical way. Microsoft claims that most UNIX implementations of DCOM will be security-compliant with the NT implementation.

At the core of the DCOM security scheme there is the notion of an Access Control List. DCOM implements an extended ACL, which includes components and associated users. This offers to application developers location, authentication and authorisation transparency. The user's authentication credentials are retrieved by the underlying operating system and are checked against the ACL. If the user does not have the necessary credentials to access a certain object or invoke a method then his/her request is rejected, before invoking the method or accessing the object in any way.

The latter applies to applications and components, which are not security-aware and have to rely solely on the DCOM authentication and authorisation model. However, DCOM provides support, through the operating system's libraries, for security-aware components to develop their own security mechanisms for authentication, authorisation, policies and auditing. These application-level mechanisms operate on top of the ones operated by the underlying operating system.

Using the user credentials stored in Windows NT Servers in order to authenticate and control the access of clients requesting the invocation of DCOM objects in a Windows NT domain does work in Intranets. Using the mechanisms provided by Windows NT for ensuring data protection can also work in Intranets. However, this might not be the case with wider networks such as the Internet because the number of potential users that have to be stored in each and every NT Server that hosts components and lies within a group of co-operating Servers in a distributed environment is much higher and probably the users are not known *a priori*. Therefore the cost of administration and the total cost of ownership of such a system is very high. To cope with this, Microsoft has expanded the security mechanisms offered by NT in order to incorporate widely deployed methods for authentication, authorisation and data protection, such as SSL (Freier *et al.*, 1996).

Components of DCOM security infrastructure

DCOM distinguishes between four fundamental aspects of security:

1. *Access security*. DCOM provides process-level access control, through the mechanisms provided by the underlying operating system. Security-aware applications may extend this access control functionality to include their own controls on top of it, possibly operating on a much higher level.
2. *Launch security*. Previous to accessing objects, object creation should be secured, primarily to protect the server from denial of service attacks. As previously described, DCOM checks for security credentials before the object is involved, i.e. before their creation process is launched.
3. *Identity*. DCOM provides for impersonation, where the object is

Information Management &
Computer Security
8/1 [2000] 5–13

identified with the caller. Any subsequent calls are limited by the caller's privileges. This approach, while useful, cannot be used in environments with a large number of distributed users, like the Internet, where the information concerning the identity of callers is not always located locally (in Microsoft Windows NT user databases), or even when the callers are not known *a priori*. Another implication of impersonation is the possibility of a malicious component to use the security privileges of the caller to perform actions otherwise not allowed. To prevent this, DCOM provides the caller with the capability to indicate what the called object is allowed to do with the security token it obtains. These options are:

- *Anonymous*. The object is not allowed to obtain the identity of the caller. The safest approach, but also, the most restrictive for the server component.
- *Identify*. The identity of the caller (i.e. user name) can be detected by the component but no impersonation is allowed.
- *Impersonate*. Only one object at a time can be identified with the caller and perform actions locally. This is an unsecured approach, still a powerful one.
- *Delegate*. The object can delegate the caller's identity to other objects in order for them to operate, using the initial caller's rights. Even remote operations can be performed. This option will probably be supported in the version of Microsoft Windows NT which will succeed version 4.

4 *Connection policy*. It concerns the protection of transmitted data. An entity may choose to protect only the integrity of data or their confidentiality as well. This choice can be altered dynamically. Connection policy also concerns authentication. Before laying the ground for protected communication, the identity of the caller should be authenticated. After that an impersonation level of the ones previously described can be selected. Clients are being authenticated at a first stage by the security providers of the underlying operating system. Authentication credentials are cached in order to avoid having the authentication process repeated. This, however, raises security issues regarding the secure storage of the cached authentication credentials.

5. Enterprise Java Beans (EJB)

EJB is Sun's implementation of distributed component architecture. What seems most promising with EJB is Sun's decision to avoid defining an underlying infrastructure. A set of standard application program interfaces (API) provides access to existing infrastructure services. Thus developers can use their preferred infrastructure environment, be it COM, CORBA, or any other, to communicate with Enterprise Java Beans Components. Security issues emerging from the use of these infrastructures are explicitly dealt with in other parts of this paper. At the core of EJB is a container, where components reside in the EJB server. The container is responsible for all object manipulation, from registering and creating object instances to co-ordinating transactions.

Apart from supporting a large set of existing middleware infrastructures, Sun provides Java applications with a way to directly invoke methods on EJB server components residing in a remote Java virtual machine (JVM). These EJB components implement one or more remote interfaces and are called remote objects. The action of invoking a remote method on such an object is called remote method invocation (RMI). RMI is another extension to the classic remote procedure call communication method, using stub and skeleton objects as described in the introduction.

The most unique feature of RMI is the ability to pass objects as method parameters by value. Objects can be either remote (as described above) or local (that do not implement any remote interfaces). In the case of remote objects, what is actually downloaded is the bytecode of the stub object of Figure 1. Thus the client can dynamically extend its set of types to include the new object. The stub object of course does nothing more than forwarding client requests to the skeleton object residing on the other side. In the case of local objects the actual bytecode is downloaded and executed on the remote VM, providing certain services locally. Note that this can be bi-directional (e.g. to allow the server to perform certain callbacks on the client).

Enterprise Java Beans security

Security in EJB is handled by the container. No explicit security coding is needed for server components, since every component method is associated with a list of users legitimate to execute it. This association is defined in a set of AccessControlEntry objects, used by the EJB container, which is

Information Management &
Computer Security
8/1 [2000] 5–13

used in order to check the user security privileges regarding the server component. The security model thus resembles the one of DCOM, since a form of an access control list is also used.

The major security concern has to do with RMI itself. The possibility to download executable content to the client machine inherently includes the possibility to (Java, 1998):

- attack the *integrity* of the system;
- violate the user's *privacy*;
- limit system resources *availability*;
- achieve user's *annoyance*.

By implication, one has to carefully consider what system resources and to what extent may be made available to a downloaded server object, without endangering the system's security and at the same time guaranteeing the usefulness of the executable content. One of the primary system goals for supporting distributed object in Java is to "...preserve the safety provided by the Java runtime environment" (RMI, 1997a). Towards this end, Sun has extended the RMI specification to fit well in the Java language Security Model. A brief description of the original Java Security Model vis-à-vis mobile executable content (applets) follows.

JAVA's security model provides a customisable sandbox in which JAVA programs run. This model restricts applet's actions in a dedicated area of the Web browser. Within its sandbox, the applet may do anything it wants but cannot gain access to the user's file systems, network connections or other resources.

Applets are loaded from the network by the applet ClassLoader that receives the bytecode instruction stream and converts it into internal data structures that represent the applet's classes. The ClassLoader invokes the Bytecode Verifier (Yellin, 1995) before running a newly imported applet. The Verifier subjects each applet class to a number of safety tests (Sun, 1997): checks the bytecode to ensure that it does not forge pointers, (Sun, 1997) or access objects using incorrect type information as well as any other actions that could lead to partial corruption of the security mechanism. The security manager is an abstract class that enforces the boundaries around the sandbox. Whenever an applet tries to perform an action that could corrupt the local system, the JVM first asks the security manager if this action can be performed safely. If the security manager does not approve, a security exception is raised and an error message is written to the JAVA console.

The Java Development Kit 1.2 (JDK1.2) includes a set of new features (Gong *et al.*, 1997; Gong, 1997) which introduce another modus operandi for the Java security system. JDK 1.2 will consist of the following new protection mechanisms: security policy, access permissions, protection domains, access control checking, privileged operation and Java class loading and resolution.

The security policy introduced by JDK1.2 is instantiated at JVM startup and may be altered *a posteriori* via secure methods. Permissions, in JDK 1.2, are not granted to classes but to protection domains. The latter consist of all the objects that correspond to a principal who has been authorised by the system. One of the future adjustments pertinent to the protection domains is the inclusion and usage of user authentication and delegation information, in order to be able for a piece of code to have different permissions when executed by different principals. User authentication information inclusion and usage has already been tested and performed to a certain degree (Balfanz and Gong, 1997), (Gritzalis and Katsikas, 1996).

Java itself does not address efficiently the problems that might arise from denial of service attacks. However, research is being performed in that field in order to provide Java with the capability to limit the percentage of CPU or memory available to a specific applet (Gorrieri and Marchetti, 1998).

The RMI security model relies on the Java security infrastructure to build the security features it needs. The very first requirement a client must satisfy to load classes from remote machines is the presence of a security manager, either defined by the application or the RMISecurityManager provided by RMI itself. If that requirement has not been satisfied, stub loading is not allowed (RMI, 1997b).

There is a RMI-specific class loader used to load stubs, skeletons, and other classes needed by them, the RMIClassloader. Local classes are loaded first, followed by stubs loaded from the same or a remote machine. Any subsequent calls to corresponding objects force loading from the original source with the same security privileges. Classes used directly by these objects are also loaded by the RMIClassLoader and are subject to the same restrictions. What should be noted is that after access to a resource has been granted to an object, there currently is no way to restrain the object from abusing it.

Information Management &
Computer Security
8/1 [2000] 5–13

6. Comparative evaluation

In this section we compare various characteristics of the three aforementioned architectures. The criteria we will use, arise from the previous descriptions of the three architectures and hopefully highlight some of the subtle security aspects and issues concerning the three architectures.

Security policy – flexibility

All three systems provide a clear set of security services. CORBA does it through its specification, Java Beans through its API, and DCOM through its four aspects of security. A flexible security policy can be implemented, using the aforementioned services and featuring a number of security levels. CORBA provides for a wide range of detailed security services and security levels to which these services belong. CORBA provides a solid framework for a scalable DCS security scheme. The extensibility of the CORBASEC as well as its ability to replace certain functional parts of the security scheme (e.g. security mechanisms) is remarkable. Java Beans seems to be quite flexible also, since user permissions are assigned to object methods rather that the objects alone. On the other hand, DCOM assigns the responsibility to extend the security mechanisms to the developer (e.g. in order to make certain object methods more secure than others).

Infrastructure dependency

By this we mean the ability of an architecture to maintain its security features when ported to arbitrary platforms. EJB has been designed with portability in mind. The security services API (provided by the AccessControlEntry objects), as any other API, is expected to be provided by all vendors implementing an EJB container. Moreover, EJB supports portability at the cost of an underlying "virtual machine" that is not an integral part of the operating system. Increased portability is an advantage but usually it results in decreased efficiency in terms of application speed, system resources available to the applications themselves and most important it raises the potential of attacks when a security aspect is well established on one platform but more vulnerable on others. On the other hand, CORBA is platform neutral by definition since it only provides a specification of the architecture. Commercial and non-commercial CORBA platforms provide their own implementations of the CORBA security services. These platforms are usually platform dependent, since they are designed for specific operating systems. However, the CORBA specification ensures interoperability on the security services level. Finally, DCOM is the most platform-dependent architecture when it comes to security services. We are still waiting to see an implementation on Macintosh and UNIX that will provide adequate security support.

User vulnerability

This criterion is closely related to the previous one. We make the distinction though, to explicitly address the issue of security-unaware users who connect to objects of the three architectures in order to use their services and the risk this imposes. We have analysed how objects passed through RMI conform to the Java Sandbox security architecture. The security-unaware user is provided with an adequate level of security and with the safety of the code execution, which is provided by the Java virtual machine (JVM). In CORBA the security-unaware user is also provided with a basic level of services, while the security-aware user can further control and adjust the parameters of the respective security mechanisms. Furthermore, the security-aware user can use additional services, such as delegation. On the contrary, DCOM expects security awareness from the users, in order to operate the security mechanisms it offers and implement security services even at the lowest level. The difference between the basic level of protection offered by CORBA and EJB to the security-unaware users is that in EJB the users are provided from the JVM with mobile code safety as well.

The fact that there is no support for an advanced audit filtering system in the models we have examined probably renders the task of security management a more difficult one, when it comes both to security-aware and security-unaware users.

Intra-architecture security preservation

This is a question that naturally comes to mind since both Microsoft and Sun along with major CORBA implementers are about to provide the capability to communicate with objects embedded in each other's architecture schemes. What will probably happen is a convergence between EJB and CORBA with developers creating products allowing EJB servers to run over CORBA engines. Iona's OrbixHome v1.1 (Iona, 1998) already supports such an integration. This approach does not seem to impose any security problems as long as the container provides a robust implementation for the EJB security API. Regarding DCOM,

Information Management & Computer Security
8/1 [2000] 5–13

according to Sun, interaction between COM clients and EJB servers does not make a difference for the EJB server. We were not able to designate how a user communicating with COM is to be authenticated by the AccessControlEntry object though. COM communication with CORBA objects is still a development process so there is not yet an issue of security preservation between these two architectures.

Interoperability

CORBA has not yet been tested that much in the implementation field; at least as much as the other models and especially DCOM. One should take under consideration the fact that the CORBA specification is too large and detailed to be implemented even partially, without running the risk of unforeseen software bugs and vulnerabilities that may come up later on. Furthermore, at the present time a large number of commercial ORBs are being implemented and marketed. The promised interoperability level does not seem to reflect itself in practice, at least for the time being and to the degree described in the specification.

CrySTINA is moving towards improvements of its CORBA-like security scheme, when it comes to interoperability of security features of different systems. It is expected that CrySTINA will provide a much more interoperable security scheme, which is probably going to be a subset of CORBASEC, because secure interoperability in TINA networks is of paramount importance.

Trusted computing base

Remote method invocation significantly extends current distributed object architecture with its support for dynamically loaded stubs and skeletons and its capability to pass object code for execution to other virtual machines. The RMI security model conforms to the original Java security model, which has been and continues to be under continuous public scrutiny and improvement by Sun. However, one should not neglect the fact that RMI does make assumptions regarding the security functionality provided by the underlying operating system and hardware. One of the most fundamental security features of Java is its type safety. However, without the proper memory protection and segregation mechanisms from the operating system, it would be much more difficult to achieve.

These assumptions are also made by the other models we have examined. DCOM, in particular, relies heavily on the security provided by the underlying operating system, most often Microsoft Windows NT. While this is a serious advantage in such environments, clients residing on less secure operating systems seem to be unguarded from the server side.

Transparency

All the models we have examined hide the complexities of the underlying mechanisms, for networking and security, among others. The level of transparency varies according to the model but this is a feature that they could improve even more. Certainly, security-aware distributed components can make better use of the security interfaces and mechanisms provided by the models we have examined, at the cost of the complexity the application programmers have to face.

7. Conclusions

In this paper we have considered a number of distributed component architectures and their security features to support the underlying infrastructure of Secure Electronic Transactions. Authentication and Electronic Payment, along with a number of Corporate and Business to Business services are likely to rely heavily on this infrastructure, in order to establish a secure electronic marketplace. We have also briefly presented TINA, an architecture offering a range of solutions for the up-and-coming transactions involving digital merchandise.

We have seen that there are a lot of security issues still to be examined, in distributed component software systems. However, the advances that have been performed in this field recently do provide a solid framework for studying further their vulnerabilities and improving the interoperable and distributed security infrastructures that DCS systems need to operate securely. Thus, along with ease of deployment, flexibility and scalability, vendors and individual organisations in the electronic marketplace should also be expected to improve and implement robust security infrastructures, in order to provide real value for their customers.

References

Balfanz, D. and Gong, L. (1997), "Secure multi-processing in Java", available at <http://www-Swiss.ai.mit.edu/~jbank/javapaper/javapaper.html>

Concalves, M. (1998), *Windows NT 4.0 Server Security Guide*, Prentice-Hall, Englewood Cliffs, NJ.

CORBA (1998a), *Common Object Services Specification*, available at <ftp://ftp.omg.org/pub/docs/formal/98-07-05.pd>

Information Management & Computer Security
8/1 [2000] 5–13

CORBA (1998b), Internet Inter-ORB Protocol v2.2, available at <ftp://ftp.omg.org/pub/docs/formal/98-07-01.pdf>

CORBA (1998c), Security Services Specification, available at <ftp://ftp.omg.org/pub/docs/ptc/98-01-02.pdf>

Dulepet, R. (1998), "COM security in practice" Microsoft Developer's Network Library, available at <http://premium.microsoft.com/msdn/library/techart/msdn_practicom.htm>

Freier, A., Karlton, P. and Kocher, P. (1996), SSL specification, available at <http://home.netscape.com/newsref/std/SSL.html>

Gollmann D. (1999), *Computer Security*, John Wiley & Sons, New York, NY.

Gong L. (1997), "New security architectural directions for Java", *Proceedings of IEEE COMPCON*.

Gong, L., Mueller, M., Prafullchandra, H. and Schemers, R. (1997), "Going beyond the sandbox: an overview of the new security architecture in the Java Development Kit 1.2", *Proceedings of the USENIX Symposium on Internet Technologies*.

Gorrieri, R. and Marchetti, R. (1998), "Applet watch-dog: a monitor controlling the execution of Java applets", *Proceedings of the IFIP SEC'98, Austrian Computer Society*, pp. 15-23.

Gritzalis, D. and Katsikas, S. (1996), "Towards a formal system-to-system authentication protocol", *Computer Communications*, Vol. 19, Elsevier, pp. 954-61.

Horstmann, M. and Kirtland, M. (1997), "DCOM Architecture", *Microsoft Developer's Network Library*, available at <http://www.microsoft.com/windows/downloads/bin/nts/dcom_architecture.exe>

Iona Information (1998), Orbix Home found at <http://www.iona.com/info/orbixhome.html>

Java Security (1998), available at <http://www-swiss.ai.mit.edu/~jbank/javapaper/javapaper.html>

Microsoft Corp. (1996), "DCOM Technical Overview", available at <http://www.microsoft.com/windows/downloads/bin/nts/dcomtec.exe>

Object Management Group (OMG) (1999), available at <http://www.omg.org>

RMI Specification (1997a), available at <http://www.javasoft.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html>

RMI Tutorial (1997b), available at <http://www.javasoft.com/docs/books/tutorial/rmi/index.html>

Rogerson, D. (1997), *Inside COM*, Microsoft Press, Redwood, WA.

Spinellis, D., Kokolakis, S. and Gritzalis S. (1999), "Security requirements, risks, and recommendations for small enterprise and home-office environments", *Information Management and Computer Security*, Vol. 7 No. 3, pp. 121-8.

Staamann S. (1997), "Overall integrity of service control in TINA networks", *Proceedings of the CMS'97 3rd IFIP TC6/TC11 International Joint Working Conference on Communications and Multimedia Security*, Chapman & Hall, London, pp. 3-15.

Staamann, S., Buttyan, L. and Wilhelm U. (1998), "Security in TINA", *Proceedings of the 14th IFIPSEC '98 International Information Security Conference*, published by Austrian Computer Society, pp. 111-22.

Sun Microsystems (1997), *Frequently Asked Questions – Applet Security*, available at <http://java.sun.com/sfaq/>

Yellin, F. (1995), *Low Level Security in Java*, available at <http://java.sun.com/sfaq/verifier.html>