Distributed Component Architectures Security Issues

Giorgos Gousios Efthimia Aivaloglou
Stefanos Gritzalis

Department of Information and
Communication Systems Engineering
University of the Aegean, Samos, Greece
cs980[0,1]1@icsd.aegean.gr, sgritz@aegean.gr

23rd November 2004

Abstract

Enterprise information systems and e-commerce applications are tightly integrated in today 's modern enterprises. Component architectures are the base for building such multi-tier, distributed applications. This paper examines the security threats those systems must confront and the solutions proposed by the major existing component architectures. A comparative evaluation of both security features and implementation issues is carried out to determine each architecture's strong points and drawbacks.

Keywords: Components, Component Architectures Security, CORBA, J2EE, .NET

1 Introduction

The intrusion of the Internet and e-commerce in enterprise information systems has stirred those systems away from monolithic design and deployment. Modern enterprise applications are based on multi-tier architectures that distribute the business logic, and therefore the load, in multiple sites. Such a design allows resource reuse, good scalability by adding new machines that operate with existing software and security as a consequence of multiple gates and gatekeepers being linearly placed in front of data. Component architectures are mainly targeted in supporting this kind of distributed applications.

A component is a small binary object or program that performs a specific function efficiently and is designed in such a way to easily operate with other components and applications. From a programmatic point of view, a component is a 'physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces' [19]. Components differ from libraries or objects in that they are loosely connected to an application (no compile time linking) and in that they are often distributed across a domain. Existing component architectures serve as a medium for executing calls to a remote component and to provide the runtime environment that makes remote procedure calls efficient and secure.

The general notion of a component architecture is described in Figure 1. The application only needs to know about the component's interface. This is usually knowledge assumed at the application development phase, although it can be obtained dynamically. Interfaces are either part of the programming language (e.g. Java) or are described using a formal declaration language (e.g. the Interface Definition Language (IDL)). After acquiring an interface, the application executes a call using the methods described in the interface. The call is transparently forwarded to the component server by the *request broker*, a software medium between the caller and the called component that acts

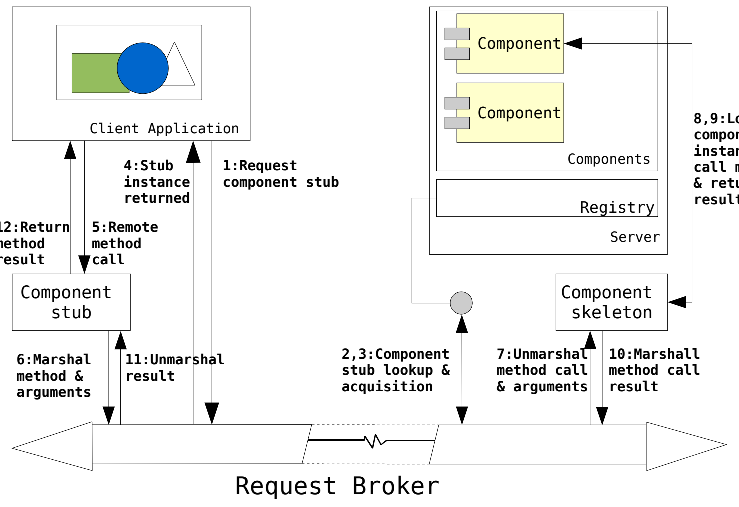


Figure 1: The sequence of events in a generic component architecture

as a software bus and is mainly responsible for transferring method arguments and returned results. Method arguments and results are *marshaled* (or else *serialized*) in a form suitable for the request broker used. The component server usually maintains a *registry* of the components it offers, which in turn stores information about the component and the component *stub*. A stub is the intermediate between the caller and the transport mechanism, while the *skeleton* serves the same purpose at the server side.

Security for a component architecture is a necessary prerequisite. Since most component invocations result in sending data over a (possibly untrusted) network, using commonplace networking technology, security mechanisms should be included in order to ensure message confidentiality and integrity. Also, these architectures are commonly used to implement business logic using data stored in databases, so access control mechanisms, based on user identity or system credentials, need to be in place to protect data from misappropriation. Furthermore, system provided services such as auditing are required to be implemented inside a component architecture to compensate the need of accountability and provide fine-grained logging and possibly non-repudiation services.

This paper discuses security issues rising in distributed component architectures. The architectures analyzed are OMG CORBA, the J2EE platform and Microsoft's .NET. Section 2 is about the general security threats in a distributed component environment. Sections 3, 4 and 5 provide an overview of each component architecture and an introduction to its security scheme. Section 6 describes how the main security requirements are covered by each of the architectures. Finally, in Section 7, a comparative evaluation of the security features offered by the three technologies is conducted along with a presentation of implementation specific issues, with references to the security threats presented in Section 2.

Prior studies

Component architectures are an actively studied subject in both academic and business environments. The CORBA security service [14] [6] and its applications [20] in various domains has attracted most of the security researcher's attention. Little work has been done for the other two architectures, mostly from industry consultants [17]. This paper tries to compare and contrast these technologies both in terms of security features offered in response to defined security threats and in terms of implementation issues that are of great concern if a system is to be deployed.

2 Security Threats

By carefully examining the generic component architecture in Figure 1, one can draw useful conclusions about the security issues threatening such a system. All component calls are directed through the middleware which most times operates using common networking infrastructure such as TCP/IP. A malicious user could eavesdrop the network connection thus tracking down all interchanged messages. Since most request brokering protocols are open standards, the malicious user could decode the messages and steal identity information about the calling user. Such information could be used in order to invoke components masquerading as an other user. Security could be compromised even more if the malicious user manages to bypass security checks, due to bad authorization setup, and tamper with system and application logs. A nearly complete list of security threats in component architectures, including the threats applying to web services [15], can be summarized into the following:

1. Disclosure of confidential information. Eavesdropping on an insecure communication line, so gaining access to confidential data. Authorized or unauthorized users get access to important information.

2. Violation of data or code integrity. Tampering with the communication line by injecting or removing data. Replacing downloaded code with malicious code. Provide malicious code as application code.

3. Misappropriation of protected resources. Security breach that allows unauthorized user to use protected services.

4. Compromise of accountability. Destroy auditing information or non-repudiation proof. User masquerading someone else so as to attribute actions to wrong person.

5. Misappropriation the compromises availability. Denial of service attacks. Physical communication line attacks.

In order to respond to these security challenges, (except from attacks against system availability, which are assumed to be prevented by the underlying network or operating systems services) all component architectures offer a vast set of security related features. Although the implementation details in these features may defer, the general principles are more or less the same. Identification of principals is established using authentication mechanisms, while access controls can prevent unauthorized access to objects. Information confidentiality and integrity can be enforced using communication security mechanisms. Misuse of the system can be detected by keeping audit logs which, in conjunction with non-repudiation services can respond to the need for accountability. Finally, security policy administration methods are of great importance when the complex security policy expression rules needed for middleware systems are put into practice. All the above considered, the security features each platform must provide are presented categorized as follows:

1. Identification and authentication of principals (human users or objects) to the system or mutual authentication between objects.

2. Authorization - object access control.

3. Delegation of privileges during a chain of calls.

4. Security of communication between objects, message protection.

5. Security auditing.

6. Non-repudiation of data origin and receipt.

7. Administration of security policy.

It should be made clear that both CORBA and J2EE are platforms defined by specifications. The difference is that while the J2EE specification defines the minimal set of requirements for the complying systems, the CORBA specification defines the maximum set. All J2EE products are required to include all the functionality specified, while, to the best of our knowledge, there is no CORBA security service product implementing the full functionality of the security specification. The .NET platform, while built upon open standards, is only fully implemented by Microsoft until now, so we examine Microsoft's implementation details.

3 Common Object Request Broker Architecture (CORBA)

3.1 Introduction

Common Object Request Broker Architecture (CORBA) is a component framework that allows the design and implementation of distributed object-oriented applications in a standardized manner that guarantees portability and interoperability. Several implementations of CORBA exist, in a variety of programming languages and environments, which all implement a common set of interfaces, described in the framework specification. CORBA components may be implemented in a variety of languages; interoperability is preserved as long as their interfaces are formally described using the Interface Definition Language (IDL).

The key component in the CORBA architecture is the *Object Request Broker (*ORB*)* [1], which provides the mechanisms for applications and objects to interact, even if they reside in heterogeneous environments. The communication protocols used are the General Inter-ORB Protocol (GIOP) and its Internet Inter-ORB Protocol (IIOP) specialization for TCP/IP based networks. Apart from ORB, the specification defines a collection of object services, which reside outside the ORB core and provide basic capabilities for using and implementing objects. The CORBA security service is part of this set of components.

3.2 Security Overview

To respond to the need of securing CORBA systems while maintaining their portability and interoperability, OMG has developed the CORBA Security Service specification [2]. The specification defines two levels of security functionality, so that any product that implements the specification needs to be compliant with a level. Level 1 provides a first level of security for applications that are unaware of security, while Level 2 provides more security facilities and deals with applications which are in position to interact with the security service in order to specify the exact security features used. The CORBA Security Service does not enforce the use of any specific security mechanisms. Moreover, the service interfaces are defined in order to allow different security service implementations to be substituted and incorporated on a single ORB.

CORBA Security Service specification includes the *Common Secure Interoperability* (CSI) protocols to support the establishment of secure communication between objects, and therefore enable interoperability even between different CORBA security service implementations. Three functionality levels of secure interoperability are defined, the highest of which (CSI level 2) supports identity and privilege based policies with controlled (from the initiating principal) delegation and provides the full functionality of CORBA Security Service. The main communication protocol defined in CSI is Secure Common Inter-ORB Protocol (SECIOP), which is used in combination with the GIOP/IIOP protocol, enabling secure transition of GIOP messages. CSI is independent of the security technology used to provide message protection. The specification provides interfaces for the use of existing security protocols (SSL[10], Kerberos[13], SPKM[4] and SESAME[7]) as a medium for secure interoperability.

A higher level protocol defined by the CSI specification is the Security Attribute Service (SAS), which is designed to be used in environments where transport layer security such as that available

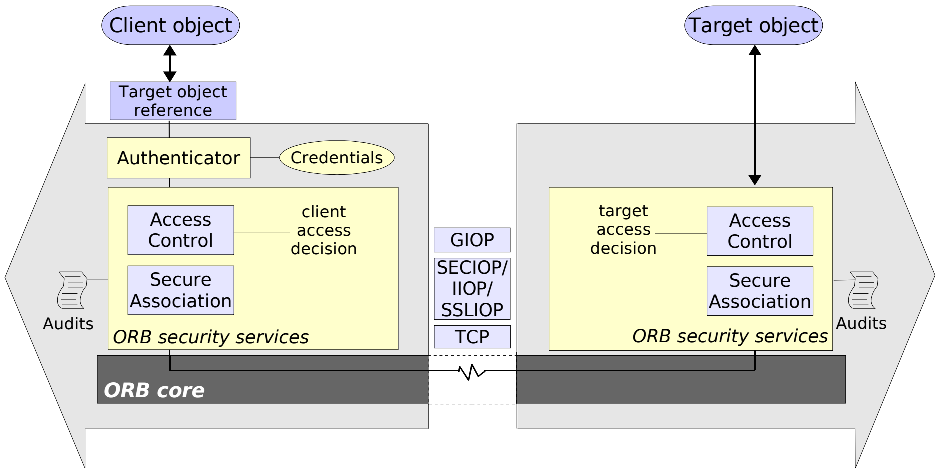


Figure 2: The main components of CORBA security.

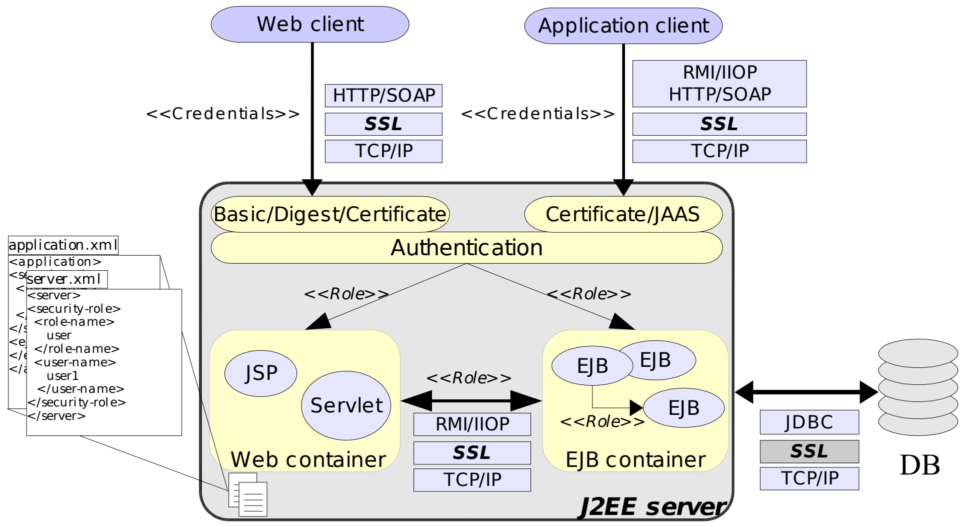
via SSL/TLS or SECIOP, is used to provide message protection. SAS protocol enables interoperable authentication, delegation and privilege functionality that may be applied to overcome corresponding deficiencies in an underlying transport.

4 Java 2 Enterprise Edition (J2EE)

4.1 Introduction

J2EE is a multi-tier component based architecture which extends the standard Java platform to support distributed, large-scale applications. The back-end tier consists of a database, accessed using the JDBC specification. The core of the architecture, the middle tier(-s), is described by the *Enterprise Java Beans (*EJB) specification. The front-end tier is often implemented as a web interface or a standalone application. The web interface makes use of the *Servlet* and JSP specifications to provide web services. The application interface is a normal user application written in any language that supports the *Internet InterORB protocol (*IIOP*)*. Each tier is associated with a *container*, a program that loads, executes and possibly serves the components it holds, while offering some value added features, for example naming or auditing services.

The EJB specification defines EJBs to be *'components of distributed transaction-oriented enterprise applications'*. EJBs can either be session EJBs, which implement transactions and are executed on the behalf of a single client in the context of a session, or entity EJBs, which map to database records and allow an object view of the underlying data. A recent addition to the specification also defines the existence of message driven EJB's which are mainly responsible for database updates upon client message receipt. An important feature of the EJB specification is the inherent support for distributed transactions. The EJB container is responsible to maintain instances of EJB's, to provide standard services described by the J2EE specification, e.g. naming and messaging services, and to handle the communication details with clients or other servers. EJBs are invoked using the RAMI-IIOP protocol for remote invocations or their home interface for local invocations. Using the industry-standard IIOP protocol allows interoperability with existing CORBA-based client and server applications.



```
<server>
<security-role>
 <role-name>
   user
 </role-name>
 <user-name>
   user1
  </user-name>
</security-role>
</server>
```

Figure 3: The main components of J2EE security.

4.2 Security Overview

The J2EE platform is paying important attention to security by design. The programming model followed attempts to leverage existing security services, rather than defining new ones, and provide a 'secure-by-default' approach to application components. The specification defines three important terms for security:

- Principal: A principal is an entity that can be authenticated, using an authentication technology, by its principal name and authentication data.
- Credential: A credential stores or references information for *security attributes* (e.g. access permissions, auditing information) and are provided by the external mapping of principal identity to an enterprise-wide security system (e.g. Kerberos, X509 certificates). Credentials can be used to authenticate a principal to a J2EE service.
- Security Domains: A security domain, or else a realm, is a logical part of the system where a specific security policy is enforced.

The J2EE platform urges for, albeit does not mandate, a *declarative* security model. All security related resources are declared in a specification-defined XML variant by the application deployer, instead of being hardcoded into the components. Security services are almost exclusively provided by the container. This allows flexibility in the deployment process and abstracts the business logic from the security policy that should be enforced to components that materialize it.

5 Microsoft .NET

5.1 Introduction

.NET is an initiative driven by Microsoft that aims to provide desktop users with easy access to web services. Contrary to common belief, the main parts of the .NET framework are open specifications and Microsoft's implementation of .NET is only extensions to the standard. As most enterprise platforms today, .NET is a multi-tier platform supported by a backend database and allowing for web or

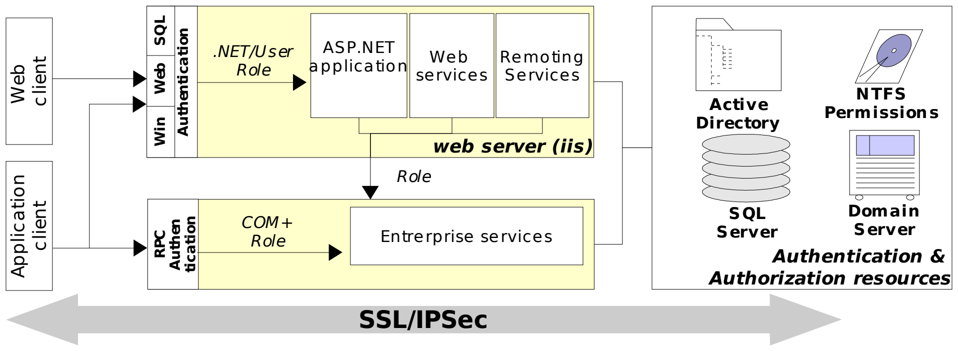


Figure 4: The main components of .NET security.

application interfaces to process data. .NET is specifically targeted to supporting web services; it includes integrated support for XML-RPC and SOAP while extending the already existing ASP dynamic web content technology [16] [12].

Components in .NET are classes that implement the `IComponent` interface. Typically, components have properties and can receive and send events. Components are hosted by *containers* and a container can host multiple *sites* of similar components. A component is only allowed to communicate through its site with external resources. Components targeted for use at the middle tier are referred to as *serviced* components and can use both .NET and COM+ [1] specific functionality (Enterprise Services). .NET is responsible for providing additional services such as object pooling and caching to serviced components. Serviced components are packaged and distributed in *assemblies*, which can also contain component resources and security policies..NET can encapsulate existing COM components into assemblies or expose .NET components to COM. Remote invocation of serviced components is performed using the *Remoting* mechanism or Web Services. Remoting is the process of sending objects by value or by reference to a remote component and thus implement a remote call and can be performed over HTTP (using SOAP) or TCP communication channels.

5.2 Security Overview

The security model used in .NET is heavily based on existing infrastructure. The basic security services (authentication, resource access control) are provided by the operating system, while extensions also require Microsoft programs to be installed (eg naming services). This is an approach that can harm interoperability but the number of services bundled with Microsoft server software seems to leverage the situation. .NET can also use Kerberos for basic security requirements.

Security in .NET is mainly based on *roles*, *policies* and Access Control Lists (ACLS). ACLS associate available resources with permissions a user or a group of users has on a particular resource. ACLS are implemented by the NTFS filesystem. Permissions can include filesystem operations or code execution permissions. A role is named set of principals that have the same privileges, while a principal is the identity of a single user along with the groups she belongs to. A policy is a configurable set of rules that .NET follows when determining the permissions to grant to principals.

It should be made clear though that the .NET framework is the authority that enforces a security policy rather than Windows security. Windows security is the second layer of security that is applied. For example the security policy may allow an assembly to access the file system but ACLS on the files don't allow access to the assembly calling principal.

[1]COM+ is a seamless merge of the traditional COM architecture, the DCOM architecture which allows distributing components to multiple sites and the Microsoft Transaction server which is used for distributed transactions with some value-added services, for example the Microsoft message queue.

6 Overview of security features

6.1 Authentication

CORBA The authentication methods available at each CORBA security service product depend on the implementation, since the specification does not dictate any particular mechanisms that should be used. During authentication, a principal normally supplies its security name, the authentication information needed by the particular authentication method and its privilege attributes. The most usual types of authentication information required from principals are passwords or cryptographic keys, especially for system entities [14]. The privilege attributes are determined by the access policies enforced by the system and include the principal's access identity, roles, groups, security clearance and capabilities on specific objects. The specification also includes identity attributes for other purposes, like enabling principal accountability, signing messages or component usage charging. The Credentials object (Figure 2) is created after the authentication to store these attributes, along with information for establishing secure associations.

J2EE The authentication methods supported in a J2EE environment depend on the end client. In the case of web clients there are three forms of authentication: HTTP basic authentication (clear text username and password), form based authentication (custom looking authentication form) and HTTPS authentication where authentication data are encrypted using the SSL protocol before being send to the client. Application clients use the methods provided by the application container, which are not strictly defined by the specification. Other concepts of authentication provided by the J2EE platform are the concept of lazy authentication, where a user can navigate through a web or application interface until a protected resource is requested, and the concept of guest user account where an EJB can only perform a constrained set of functions. A J2EE system can also be integrated with enterprise authentication mechanisms (e.g. Kerberos), although this is not strictly required. EJB containers can either rely on the front end tiers to acquire authentication data or implement authentication using the CSIv2 protocol.

.NET The .NET platform authentication depends on the client. Web clients can authenticate using the classic methods provided by HTTP or by using the centralized Passport service. Application clients can use NTLM (the authentication service provided by Windows), Web Services or Kerberos authentication. Both clients can use certificate authentication. Principal identities can be stored in the server's operating system user database or at an external resource, like a database table or the Active Directory service. Authentication for applications directly targeting Enterprise Services is performed by the serviced component container and is based on the RPC authentication method introduced by DCOM/COM+. An important thing about authentication in .NET is that not all authentication methods provide the same functionality after authentication. Some authentication methods can be used to transfer identities from the authenticating process to another (see delegation) while others cannot. Web services that use serviced components can also authenticate their users using custom XML tags in the SOAP message header, and the authentication is done by the component container. Web services security in .NET adheres to the WS-SEC specification [8].

6.2 Authorization

CORBA Access control security policies of CORBA are divided in two layers: object invocation access policies, which are enforced automatically on object invocation, and application access policies, which are enforced within the object implementation. CORBA security service specification includes only the object invocation access control service, whose basic components are the Access Decision Functions. Client side access decision functions define the conditions that allow the client to invoke the specified operation to the target object, while target side access decision functions define the conditions that allow the object to accept the invocation. The information used for access control is the client's privilege attributes and the target object's control attributes, like the object's classification or access control lists. The specification includes a wide range of access policies, including access control list schemes, label-based schemes and capability schemes.

J2EE The authorization mechanisms provided by the J2EE platform are used to control access to code based on identity properties and to identify the principal calling a component. Code access control is done by the container in order to not permit components to use methods that can compromise the container security. Principal authorization is done in order to protect security domains from unauthorized access. Each container in the J2EE environment serves as an authorization boundary so that all incoming requests must first pass authorization tests. The authorization process is abstracted from the code using *security roles*, that describe groups of principals that should have the same set of permissions. Each principal is associated with a security role, according to the credentials it obtained during authentication. For each component, one or more *method permissions* map security roles on methods provided by the component. The authorization mechanism also allows components to be called with different credentials than those obtained by the caller.

.NET Authorization in .NET can either be role-based or resource(ACL)-based. The latter case is mainly used to protect filesystem resources from being accessed from malicious code or to secure physical files or devices. In the former case, users are mapped to roles by the administrator. Roles can contain Windows groups and user accounts and some roles are already provided by the .NET platform. The administrator can define new roles and associate roles with access permissions to serviced components, methods or interfaces. The access control operation is done by the component container before any component activation or component method call, utilizing the role and the 'user-in-role' lists. Enterprise services authorization is based on predefined or definable COM+ roles.

6.3 Delegation

CORBA CORBA security service supports privilege delegation in order to allow to an object in a chain of objects act on behalf of the principal that initiated the chain. The initiating principal or the intermediate objects can specify restrictions on the privileges to be delegated or the target objects that can use those privileges. The interfaces that are specified for delegation control allow five types of delegation policies: No delegation, simple, composite, combined privileges and traced, each defining a different set of privilege attributes being transferred. Applications can be unaware, as the ORB can enforce many delegation controls automatically according to the delegation policy specified, or security-aware so as to specify themselves the delegation controls they need. Applications are not yet allowed to specify controls such as target restrictions.

J2EE Principal identities in the J2EE are propagated by default; a called component knows about the identity of the caller who initiated a chain of calls. The propagation can be done using different mechanisms for different occasions, for example propagated data can be an X.509 certificate when there is trust relationship between the caller and the component container or a CSIv2 token when using server only authentication. However, this process does not require delegation of credentials, since mapping of identities to credentials is not required to be done by the EJB container. As a consequence, a called component cannot identify the credentials used by the principal that initiated the request without using an external credentials mapping mechanism (eg Kerberos).

.NET Delegation in .NET depends on the security service used by the hosting platform. After authenticating, and if the server is configured to, the authenticated principal 's token is attached to the server thread that serves the client. In order to be able to be delegated, this token must have network credentials. Most authentication types in .NET allow delegation provided that certain circumstances are fulfilled, for example when the authenticated account is a domain account. The .NET documentation also suggests an one-hop credential transferring scheme, called impersonation, which relies on writing user credentials to a common store while the user is online. Serviced components (Enterprise Services) automatically delegate principal identities when calling other serviced components.

6.4 Communication security

CORBA For the establishment of secure interactions between objects, the CORBA Security Service Specification includes the notion of *secure associations*. The characteristics of a secure association depend upon the client and target object's individual security policies, which specify the strength of integrity and confidentiality protection needed, and the common security mechanisms available between them. The mechanisms used to establish secure associations depend on the implementation. All CSI conforming implementations support Kerberos and, optionally, SPKM and SESAME. Message protection might be provided through SECIOP or by using IIOP over an SSL protected channel. A problem that has been identified with CORBA security protocols and architecture and is that it does not easily integrate with firewalls [14]. Solutions, mainly through changes at the ORB level, have been specified [3].

J2EE The component invocation security in J2EE is built on top of IIOP over SSL. Both the client and the container must support the SSL or the equivalent TLS protocol and be configured to use a trusted common certificate authority. The client can be pre-configured for secure invocation or learn about it using the CSIv2 field of an EJB's IOR registration information. Using SSL, confidentiality and integrity of the transferred data are also pertained. The specification encourages the exchange of information between the application designer and the application deployer so as to avoid using secure communication mechanisms for invocations that do not really require it.

.NET Remoting security depends on the communication channel; should it be an HTTP communication the security mechanisms are provided by the IIS server, else wire-level security can be used. In the former case, SSL is used to secure communications while on the later IPsec or SSL are recommended.

6.5 Auditing

CORBA CORBA security specification identifies two categories of audit policies: system audit policies, which control the auditing of ORB and CORBA Security Service events, and application audit policies, which control which events are audited by applications. System audit policies are enforced automatically over all applications. The specification defines a set of event families and event types within each family. It includes how audit records are generated and then written to audit channels, but not how they are filtered, analyzed or kept secure afterwards.

J2EE The J2EE specification does not define a standard auditing mechanism. It is left to the container provider to support auditing using a custom solution.

.NET .NET implements auditing by using the Windows logging service. An application can implement custom logging for tracking user level actions using the provided logging classes. Also, framework level logging is supported to report failed resource access requests or intentional attempt to misuse resources, with an adjustable logging detail.

6.6 Non-repudiation

CORBA CORBA security specification includes a stand-alone Non-Repudiation service that is designed for security-aware applications and used separately from the standard security service on the ORB level. The NR service, according to the specified non-repudiation policy, enables the creation of evidence related to the creation and the receipt of a message, as well as evidence relevant only in the context of the application itself, but does not provide evidence that a request on an object was successfully carried out, since this information is available only on the ORB level. NR service is based on the ISO Non-Repudiation Framework, only providing for evidence generation and verification, but not evidence storage, retrieval and delivery services, which are assumed to be provided by external components and authorities. The NR service is an optional feature for products compliant with the CORBA security service.

J2EE The J2EE specification does not define a standard non-repudiation mechanism, as a consequence of not providing an auditing mechanism.

.NET There is no standard non-repudiation facility provided by .NET.

6.7 Security administration

CORBA For the administration of principals the specification defines different kinds of privilege attributes like groups, roles and clearances. However, it does not provide interfaces for managing privilege attributes, which depend on the implementation. Concerning object security, the specification identifies three types of security-related domains: security policy domains, security technology domains and security environment domains, but includes management interfaces only for security policy domains. These interfaces allow administering of domain security policies for security association, delegation, and access control. Some administrative problems stemming from the lack of transparency of underlying security mechanisms used in CORBA security systems have been identified [14][6].

J2EE The J2EE specification mandates the existence of user interfaces for easily configuring security roles, role references and security domains. However, each J2EE container provider can implement a custom user interface and extend it to support vendor-specific enhancements to the security services.

.NET Security in .NET is expressed with a combination of programmatic and declarative statements. The programmer can specify some security attributes (special tags that are not part of the language syntax) in the code that guide .NET to activate some security mechanisms. The assembly constructor can also place security policies in the assembly that can be activated when the assembly component is deployed. The application server administrator can use existing tools to enforce other security policies than that provided in the assembly or the component code. Finally, the COM+ and .NET environments can place themselves predefined or user-defined security policies on the code.

7 Comparative Evaluation

7.1 Introduction

Comparisons among technologies on the same application field offer a means to contrast the functionality and features offered and thus understand each technology's weaknesses and strong points.

The comparison framework in this paper tries to provide a clear overview of the security features analyzed in detail in the previous sections and discusses issues that concern the practical implementation of security for all technologies.

7.2 Feature Comparison

Table 1 summarizes the security features offered by each technology. The table shows that all technologies can cover most of the requested security features. Specifically, authentication options for all types of clients are provided. It should be noted that the CORBA authentication scheme appears stronger by requiring authentication of all principals, either human users or components, while J2EE and .NET allow for trust relationships between components in the same site [11]. However, the J2EE implementation might be stricter and enforce authentication for all invocations. The authorization and delegation schemes of all technologies cover most requirements, with CORBA supporting a wider range of delegation policies, allowing for enhanced access control and accountability. All technologies use widely accepted and tested communication security mechanisms. Moreover, the use of CSI protocols, enables secure interoperability between CORBA and J2EE systems and between CORBA implementations that offer different sets of security mechanisms.

Both CORBA and .NET include auditing mechanisms. Although J2EE lacks a standard auditing service, all J2EE server vendors offer auditing facilities in their products. Non-repudiation seems to be the least supported feature, since even for CORBA it is optional for the compliant security service products. However, this feature is only needed by large organizations which most times use external enterprise-wide non-repudiation services. The means for security administration highly depend on the implementation for CORBA and J2EE systems, while in .NET standard methods are provided by the host platform. Security administration might prove tedious if an implemented application uses more than one technologies that co-operate in a site. The lack of a standard security administration method can impose significant load on the administrating team, forcing them to express and apply a single security policy in more than one ways.

Strengths and Weaknesses CORBA's major advantage is the fact that it is platform and security technology neutral. This enables CORBA deployments to be used as a medium for secure interoperability among heterogeneous applications. On the other hand, the CORBA security service is a very complicated specification that has not been fully implemented yet. J2EE is a coherent application framework which provides a standard way of expressing security but also allowing extensions to it; vendor applications implementing the standard are expected to be fully compliant. Thus, should the need arise, user applications can be easily upsized, especially considering that the platform is hardware independent. The major drawback of J2EE seems to be that, trying to be as compatible as possible, has left out important security related functionality such as auditing and integration with enterprise wide security systems. A standards-only J2EE product allows security to be only expressed and enforced in one way harming flexibility. .NET seems quite a promising platform. It incorporates all needed security features and is fully compliant with Microsoft current and prior security implementations. It can serve very well as a platform for extending existing Microsoft oriented applications. However, it appears to be too tightly connected to Microsoft technology, especially when regarding security. One of the strong points of both .NET and J2EE is that are both supported by security specification and deployment applications, which can be helpful when expressing complex security rules.

Development status J2EE and .NET are in active development status. CORBA development, while officially not stalled, is not keeping up in pace with its two competitors, having almost two years to introduce any major advancement. Since the version 2 of the J2EE that introduced CORBA interoperability, many CORBA vendors are moving their business towards J2EE. Today, most CORBA products are compatible with J2EE or even written in Java. On the other hand, J2EE is continuously evolving

Security features	CORBA	J2EE	.NET
1. Authentication	Mechanisms depend on CORBA security service implementation. No particular mechanisms specified.	The end client type designates the authentication mechanism used. Web authentication or custom made authentication schemes can be used. .	Many authentication mechanisms, mostly Windows depended. Can use web authentication.
2. Authorization	A variety of privilege types (roles, security clearance) and access policies (ACLs, label schemes) supported. Both client and target object access control.	Role-based access control, component method level granularity.	Uses predefined roles, component method level granularity or filesystem based ACLs.
3. Delegation	Various types of credentials delegation policies. Application specific delegation control supported.	Principal identity only delegation. Can use external mechanisms to delegate credentials.	Delegation ability based on authentication scheme. Most can delegate, under specific circumstances.
4. Communication security	Secure interoperability via CSI protocols. Message protection through SECIOP or SSL / IIOP.	SSL or TLS used to protect IIOP messages. Uses CSIv2 to discover target security capabilities.	Remoting calls are protected using SSL or IPsec depending on the communication channel.
5. Auditing	Supports generation of audit logs for predefined system events.	No standard system auditing mechanism.	Uses Windows logging service. Audit trails generated by the framework.
6. Non-Repudiation	Standalone service for non-repudiation aware applications providing evidence generation and verification.	No standard non-repudiation mechanism	No standard non-repudiation mechanism
7. Security administration	User management depends on the implementation. Interfaces provided for administering domain security policies.	Implementation must provide its own tools for managing users and security domains.	User management provided by Windows. Security policies can be specified in various locations including the code, the component package or the application server.

Table 1: Summary of security features offered by the examined technologies

under the procedures of the Java Community Process. The next major additions to security in J2EE, as described in the J2EE specification will be an auditing system, support for more fine-grained data access control, implementation of API's that expose the underlying authentication and authorization mechanisms to the code, support for container based user registration to the J2EE services and the inclusion of a centralized deployment service. The .NET platform development is closed to public, so no future directions are known.

7.3 Implementation Issues

The following section tries to clarify the differences among technologies in the field of implementation. The criteria were selected so as to cover the implementation issues that concern security.

External Dependencies This criterion specifies how strongly each architecture depends on the underlying platform and existing security mechanisms in order to provide its security features. The CORBA security service specification is platform neutral and does not dictate the use of specific security technology. However, portability of existing security services depends on the implementation, although experience shows that most of them are designed for specific platforms. Both the J2EE specification and implementations are also platform independent, at least in theory. Libraries that implement security services are universally portable, except when using platform specific mechanisms. The .NET platform specification also cares about portability; however, the reigning current implementation, Microsoft's one, strongly depends on the security features provided by the Windows platform. Efforts have been carried out lately to provide a portable implementation for UNIX environments.

Transparency All technologies can hide details of the underlying security mechanisms from application developers. The applications can have no responsibility for security, as the security policy is automatically enforced at the ORB level for CORBA and by the container for J2EE and .NET. All architectures allow the implementation of security aware components, using specific methods that enable security checks to be carried out by the user code. The amount of control left out to the developer defers among architectures, with CORBA being the least restrictive.

Intra-Architecture Security Preservation Components of CORBA and J2EE can communicate using the IIOP protocol over SSL. In addition, both CORBA and J2EE can use the CSIv2 protocol stack, which enables interoperable authentication and integrity/ confidentiality mechanisms negotiation. Communication between .NET components and CORBA or J2EE components is not supported at the moment.

Security Scalability Security scalability is the ability to adapt to different levels of security requirements. All architectures provide some means for defining and applying security policies. The CORBA specification defines different levels of security functionality and secure interoperability, along with the mechanisms that can be used to achieve each level, while security functionality in J2EE can be extended by the use of supplementary services and mechanisms provided by implementations (J2EE specification provides only the minimal set of requirements). .NET is by default configured to be easy to use, although stricter security policies can be enforced by carefully removing permissions from selected users and assigning code execution restrictions to the appropriate components.

Development tools Good development tools can greatly assist developers in writing secure code, either by preventing them from using potentially dangerous constructs (e.g. `malloc`/ `realloc` in C) or by allowing them inspect running code properties (e.g. memory leak detectors). Since most times development tools depend on specific programming languages, CORBA has a definite advantage being language neutral. .NET may support many languages, but all these languages are expressive variations of the same class hierarchy. Microsoft has a good reputation in providing good development tools, such as the Visual Studio IDE and the .NET core supports common security oriented technologies for example memory management, that can both be used as a basis for secure programming. Finally, Java is known to be a secure programming environment. Both free and commercial tools exist that provide the Java developer with all necessary functionality to develop and test secure code.

Compatibility with existing applications Existing applications, often referred to as legacy applications, most times present an obstacle to the evolution of an information system; reimplementing them using modern technology can cost significantly both in terms of capital and productivity loss while continuously extending them leads to inconsistent systems whose security administration is painful. Is such environments CORBA's independence of programming language and platform pays off the price of high complexity. CORBA implementations exist in almost any possible platform and therefore can interconnect legacy systems to modern applications. On the other hand if the decision to re-implement the information system is made, all three technologies can be quite competent in the security field. It should be noted that all technologies are mostly compatible with their ancestors regarding security.

Cryptography - Public Key Infrastructure (PKI) support Cryptography in a distributed environment is a means for providing privacy to the end user and secure exchange of sensitive corporate data. A PKI architecture [5] provides the basic services for cryptographic applications such as generation, storage and distribution of public and private keys and certificate management and revocation. A distributed component architecture can benefit from a PKI infrastructure in many ways, for example to provide smart card services. Some basic cryptographic and PKI facilities that should be provided in such an architecture are some form of hashing functions (MDx, SHAx), shared key (DES, RC2) and public key (RSA, DSA) cryptography and certificate (X.509) management facilities. We are overviewing the main cryptography facilities offered by each technology:

CORBA: The CORBA security service specification [2] does not include any specific requirement on cryptography. Cryptographic services are expected to be provided in implementations for use with secure component invocation. However, a separate OMG specification enables the use of PKI services in CORBA components. This specification defines mechanisms to support a full range of certificate related services. Combining required cryptographic facilities with the PKI specification provides a good basis for implementing security services.

J2EE: The J2EE specification does not include any specific requirements on cryptography. The basis of J2EE, the Java language, provides the required set of cryptographic related facilities to J2EE. All standard hash algorithms, shared and private key facilities are included. Also included are advanced certificate management (Certificate Revocation Lists, Certificate Chains and Certificate Store access), random number generators and, perhaps most important of all, the ability to change the default security infrastructure with a custom made one.

.NET: The .NET framework provides cryptographic facilities such as hash algorithms, shared key encryption, public key encryption and certificate handling. It can also use the PKI facilities included in recent versions of the Windows operating system though COM interoperability and thus support trusted third party access and certificate revocation lists.

User groups and professional support Support from technical groups plays an important role in the deployment of complex technologies like those presented. A well supported technology, within a security context, has some important advantages such as secure implementation paradigms and security vulnerabilities auditing. CORBA is well supported both from software vendors and from independent user groups. The problem seems to be that support is mainly focused on implementations and many of them, especially the free ones, do not get professional level support. J2EE is probably the most widespread of these technologies and there are a lot of profit and non-profit organizations that support it, implementation-specific user groups and the Java community in general. For .NET support is rather more focused, since it is a single framework and not a specification, and is mainly backed by the software producer. Professional support does not seem to be a problem in any of these technologies.

8 Conclusions

Component architectures are very important for today's enterprises because they are the basis for building integrated e-commerce and enterprise information systems. In this paper, the security features of common component architectures have been examined. For CORBA and J2EE the presentation and comparison were based on their specifications, while for .NET on Microsoft's implementation. The comparison pointed out that all platforms offer a satisfactory set of security features, though CORBA seems to be the most feature rich, covering most evaluation criteria and offering many alternatives for implementing security. However, it should be made clear that the security features of CORBA and J2EE depend on the implementation, being usually more extended than those described in the specification for J2EE and a subset of the specification for CORBA. The comparison results show that security in distributed component architectures ought to be a judging factor for choosing an architecture. While all three architectures examined offer a good amount of security-related functionality and are almost equivalent for basic needs, minor details differentiate them and make them more appropriate for certain application fields. Each architecture has its strong points and drawbacks and it is that point, the drawbacks, that should be more carefully examined.

Component architectures are an active research and development area, as proved by the continuously evolving systems and architectures that emerge. Existing architectures offer a wealth of security features but, do not put great effort on deployment. Further work in this area should be the development of cross-platform methods and mechanisms for implementing and deploying security policies in an abstract, yet flexible and detailed, way and the incorporation of non-repudiation services conforming to the ISO methodology.

Acknowledgments: The authors would like to thank Athanassios Vambakas (MSC Consultant) for his help and positive criticism on writing the .NET section.

References

[1] CORBA: Core Specification. Technical report, OMG, November 2002. ftp://ftp.omg.org/pub/docs/formal/02-11-01.pdf.

[2] CORBA Security Service Specification. Technical report, OMG, March 2002. ftp://ftp.omg.org/pub/docs/formal/02-03-11.pdf.

[3] CORBA Firewall Traversal Specification. Technical report, OMG, January 2003.

[4] C. Adams. The simple public-key GSS-API mechanism (SPKM). Technical report, IETF, October 1996. Online http://www.ietf.org/rfc/rfc2025.txt.

[5] Carlisle Adams and Steve Lloyd. *Understanding the Public-key Infrastructures: Concepts, Standards, Deployment Considerations*. New Riders, 1999.

[6] A. Alireza, U. Lang, M. Padelis, and M. Schumacher. The challenges of CORBA security. *Informatik aktuell*, 2000.

[7] E. Baize, S. Farrell, and T. Parker. The SESAME v5 GSS-API mechanism. Technical report, IETF, November 1996.

[8] Chris Caller. Web services security. Technical report, IBM Corp., Microsoft Corp., Verisign Inc., April 2002. http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.

[9] Linda G. DeMichiel. Enterprise JavaBeans specification. v2.1. Technical report, Sun Microsystems, 2002.

[10] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL protocol version 3.0. Technical report, IETF, November 1996. Online http://wp.netscape.com/eng/ssl3/.

[11] Bret Hartman. *Enterprise security with EJB and CORBA*. New York John Wiley Sons, 2001.

[12] Michael Dunner J.D. Meler, Alex Mackman and Srinath Vasireddy. *Building secure ASP.NET Applications*. Microsoft Corporation, November 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp.

[13] J. Kohl and C. Neuman. The Kerberos network authentication service (v5). Technical report, IETF, September 1993. Online ftp://ftp.isi.edu/in-notes/rfc1510.txt.

[14] Ulrich Lang and Rudolf Schreiner. *Developing Secure Distributed Systems with CORBA*. Artech house, 2002.

[15] Chen Li and Claus Pahl. Security in the web services framework. In *Proceedings of the 1st international symposium on Information and communication technologies*, pages 481–486. ACM, 2003.

[16] Microsoft Corporation. *The .NET framework SDK Documentation*, 2002. Included in the .NET SDK package.

[17] Denis Piliptchouk and Vince Dovydaitis. Securing .NET and enterprise Java: Side by side. *Computer Security Journal*, 18(3-4):71–83, Summer/fall 2002.

[18] Bill Shannon. Java 2 Platform, Enterprise Edition specification, v1.4. Technical report, Sun Microsystems, 2002.

[19] Diomidis Spinellis and Kostantinos Raptis. Component Mining: A Process and its Pattern Language . *Information and Software Technology*, 42(9):609–617, June 2000.

[20] C.M. Westphall and J. Da Silva Fraga. Authorization schemes for large-scale systems based on Java, CORBA and Web security models. In *Proceedings of ICON'99: IEEE International Conference on Networks*, pages 327–340. IEEE, 1999.