Supporting Secure and Transparent Delegation in the CORBA Proxy Platform π^2

Zoltán Nochta, Rainer Ruggaber and Taufiq Rochaeli
Institute of Telematics, University of Karlsruhe, Germany
[nochta\ruggaber\rochaeli]@telematik.informatik.uni-karlsruhe.de

Key words: transparent delegation, security, CORBA, mobility

Abstract: π^2 is a generic CORBA proxy platform that is used to support applications in mobile and wireless environments. π^2 consists of two proxies which are transparently integrated into the application. Due to the broadcast characteristics of wireless communication, mobile users have very high security requirements. Since the CORBA security service as the standard approach for securing CORBA applications does not support the transparent integration of proxies between client and server, we introduce proprietary solutions with end-to-end authentication based on the services of a public-key infrastructure.

1. INTRODUCTION

Today's innovations in the field of processors, displays and battery techniques promote the use of portable, mobile computers. Hence, the way to nomadic computing is paved. An infrastructure for nomadic users allows connectivity anywhere upholding the environment the nomadic user prefers. Therefore, such an infrastructure must provide dedicated access nodes (AN) to utilize different network technologies as depicted in Figure 1. Nomadic users can transparently migrate between these different network technologies without obviously noticing any change in the communication behavior of their computers (mobile node, MN) apart from different delays. However, progress in developing wireless communication architectures cannot compete with the introduction of high performance wired networks. This challenge has to be faced in the middleware providing for adaptation in order to match the characteristics of the wireless link.

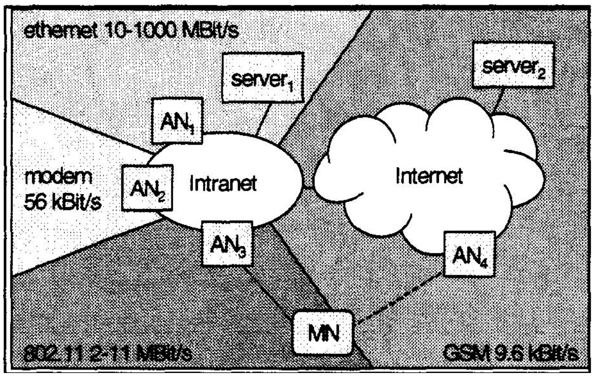

Figure 1. Nomadic computing scenario

Our solution for this problem area, the proxy platform π^2, is based on a special equipment on the borders between wireless and wired domains, where proxies act on behalf of the nomadic users. Proxies can help to reduce communication requirements for the wireless link and, therefore, integrate nomadic users into distributed applications. They have to bridge the protocols used in the wired domain and in the wireless domain, hence dealing with address and format translation, Furthermore, these proxies may be enhanced by components allowing value-added services to support context- and location-awareness or caching.

Proxy-architectures and π^2 as well suffer from security problems. Proxies that are transparently integrated into the data stream prohibit the use of end-to-end security mechanisms like authentication or encryption. The use of such mechanisms leaves the proxy behind useless, because the proxy is not able work on encrypted data. The decryption of the transmitted data is not possible due to the transparent integration of the proxy. In this paper we present special tailored solutions to the security requirements of mobile users.

2. THE CORBA PROXY-PLATFORM π^2

The proxy platform π^2 acts as a mediator between a CORBA client and a CORBA server [1]. π^2 works similarly to a request level bridge, but allows a more common solution for problems often occurring in mobile and heterogeneous environments and the integration of further functionality to provide value-added services.

With this architecture we are heading for a *client-service* paradigm instead of the client-server paradigm. In the client-service paradigm the client is not using or addressing a special server but a service. This service can be located anywhere and can be offered by different servers thus providing more flexibility with service selection.

2.1 CORBA

In CORBA, the communication infrastructure is an object bus, the Object Request Broker (ORB). Whilst the communication architecture within one ORB is not specified, there is a standard for the data exchange between ORBs, the General Inter-ORB Protocol (GIOP) and its adaptation onto the Internet protocol suite, the Internet Inter-ORB Protocol (IIOP). Since GIOP assumes a connection-oriented, reliable transport connection, IIOP is based on TCP/IP. A feature of GIOP frequently used in π^2 is the so called `service_context`. The `service_context` is realized as an array, that is transparently transmitted from one ORB to the other. This enables the exchange of data between client and server ORB, transparently to the application.

2.2 Architecture of π^2

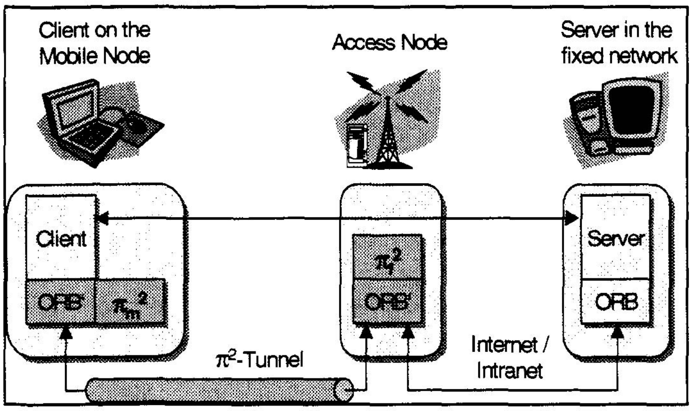

Figure 2. π^2 Architecture

π^2 consists of two different types of proxies. π_m^2 is located on the mobile node and is integrated into the ORB of the client application. π_f^2 is placed on the access node. The two proxies encapsulate the wireless link and can thus hide the characteristics of the wireless link. π_f^2 is realized as a combination of a CORBA client and a CORBA server. In order to enable transparent integration into applications both π_f^2 and π_m^2 require a modified ORB (ORB').

Method invocations of the client application are transparently redirected to π_m^2 in the client ORB. After being processed in π_m^2 the requests are tunneled to π_f^2. π_f^2 restores the request and passes it to the server. The response of the server is inversely directed through π^2. The end-to-end semantics of CORBA method invocations are not modified by π^2, because the same invocation type is used on all sub-connections. When using π^2 the client as well as the server applications remain unchanged. This allows a transparent integration of π^2 into existing CORBA-applications. For the support of different or unknown applications a reconfiguration of π^2 is not necessary since π_f^2 uses the dynamic CORBA interfaces (DII and DSI). π^2 provides generic disconnection handling and network handover mechanisms, that are especially useful in a wireless environment where sudden disconnections can occur [2].

2.3 Invocation handling

π^2 supports several invocation handling modes. Four of them are considered here for integration of security features.

In the *simple forwarding mode* π_f^2 passes the request to the server that was specified by the client in the initial request. In the more advanced modes the proxy π_f^2 can redirect client requests to a different server than the specified one to support the client-service paradigm. Another field of application is fault-tolerance or the transparent change of server interfaces. In the *static redirection mode* a given server object reference is always mapped to the same server implementation. In the *dynamic redirection mode* object references can be mapped to different server implementations in run-time. In the *caching mode* π_f^2 can respond to client requests without contacting the server. This is useful in scenarios where operation results can be cached or computed by π_f^2.

These advanced invocation handling modes prohibit the use of end-to-end encryption as the proxy is not able to decrypt the request and carry out the redirection. A solution providing security in π^2 has to take these invocation handling modes into consideration to retain the functioning of π^2.

3. PROVIDING SECURITY IN π^2

In this chapter we introduce two approaches to provide secure and transparent delegation in π^2. These approaches are based on the services of a managed X.509 [3] public-key infrastructure.

In order to retain the transparency of π^2 as far as possible, different security mechanisms and PKI services have to be integrated into the ORB.

3.1 Using the services of a PKI

In order to prevent malicious attacks against signature schemes and to provide non-repudiation functionality of digital signatures, keys must be managed by a trustworthy PKI. The most important management services of such a PKI are: user registration, key-generation, user certification, certificate distribution, cross-certification, certificate revocation, key recovery, key-update and key history [4]. In addition, the PKI should provide application developers with proper interfaces, in order to use and integrate the PKI services transparently into the applications. Respectively to the requirements of both of our concepts we use a PKI implementation which manages two different key-pairs and certificates for each user. We assume that the private keys are stored in a secure personal environment, e.g. in a smart-card.

3.2 Securing the sub-connections

The most important security requirement in all invocation handling modes of π^2 are the strong authentication and message protection of the client-to-proxy and proxy-to-server communication. The implementation of two separate SSL [5] sockets securing the complete CORBA IIOP communication and the integration of a comprehensive PKI that manages and distributes X.509 public-key certificates that are being used by SSL in its authentication process, can meet this basic requirements. [6] discusses the most important technical problems and describes an implemented solution of the PKI integration in SSL/IIOP.

3.3 A protocol to build end-to-end security associations

One drawback of the separation the sub-connections described above is the fact that the end-to-end authentication respectively the delegation of identities is not supported. Clients do not know which server will respond to their requests, because the proxy redirects method invocations unless operating in simple forward mode. On the other hand servers targeted by a proxy do not know who originally generated the request they have to respond to. In many cases, a server has to know who was the initiator of an invocation and vice versa, clients should know who is their server counterpart. In addition, if an intruder or a malicious system administrator of the proxy node knows both session keys used by the SSL sockets, he can read and alter any messages without being detected.

In order to protect end-to-end message integrity and authenticity effectively, clients and servers have to share a secret (key) unknown to the proxy. This key can of course only protect the integrity and authenticity of message fragments not changed by the redirection.

Since the CORBA security service as the standard approach to securing CORBA applications does not support the transparent integration of proxies between client and server [7], we introduce proprietary solutions.

In the following we describe a novel high-level protocol which works on the ORB layer and provides mutual end-to-end authentication, message integrity and non-repudiation functionality between ORBs. Technically, this protocol is placed on top of the SSL layer and is transparent to the access-node (s. fig. 3). This means that every message protected by this protocol will be sent through the SSL socket which protects the whole message against wiretapping and more.

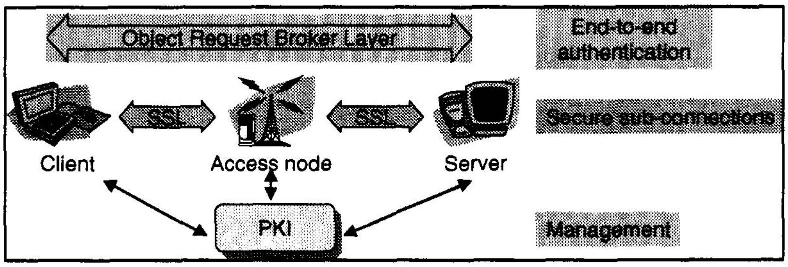

Figure 3. Security in the proxy platform

The protocol starts with mutual authentication. In this phase a secret will be exchanged. The authentication process is a challenge-response which is based on digital signatures and on the services of a well managed PKI. The shared secret is used to generate one-time-keys for computing message authentication codes (MACs). Optionally, in order to prevent the pick-up of sensible message fragments by the proxy such fragments can be encrypted with this key.

The information used to build a security association between client and server is sent in a security context (SC) which is part of the `service_context`. Figure 4 shows the main protocol steps.

Fig. 4: Protocol steps

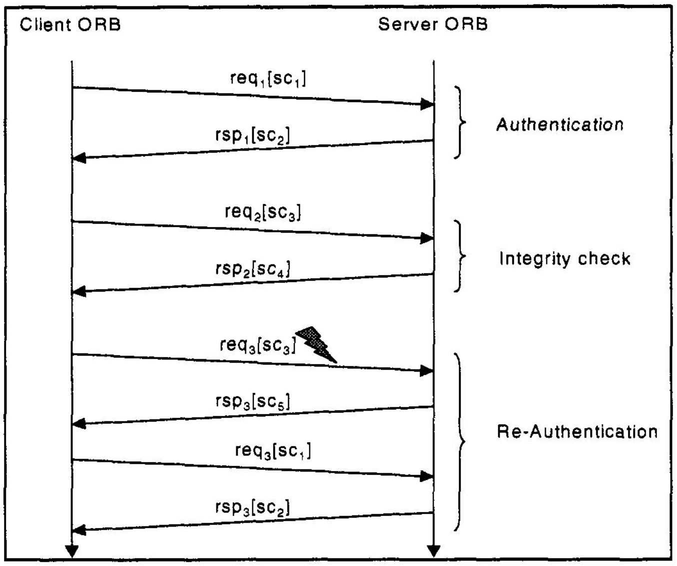

Figure 4. Protocol steps

At the first protocol step a client ORB sends his first method invocation which is automatically converted to an IIOP request (req_1) to the server. Req_1 is extended with the security context SC_1. SC_1 consists of the following mandatory fields:

Table 1. Security context SC1

Field	**Description**
Message Type (MT)	Contet ID (in this case 1)
R_n	16 byte random number generated by the client
CA certificate (CC)	Certificate of the CA the client trusts
Verification certificate (VC)	Certificate containing the client's public key used for signature verification
Encryption certificate (EC)	Certificate containing the client's public key used for encryption
Signature	Digital signature of the sequence {MT, R_n, CC, VC, EC, method name, method parameters}

After getting this request, the server first validates the signature and checks the validity of the encryption certificate (EC) the client has sent. If both the signature and the EC are valid the server generates a random number R_s, encrypts it with the public encryption-key of the client, then creates SC_2 (s. Table 2) and sends the response (rsp_1) back to the client. The most important field of SC_2 is the signature field. The sequence signed by the server includes not only the plain-text fields of SC_2 and the name and the results of the last method invocation, but also R_n in order to prevent replay-attacks.

Table 2. Security context SC2

Field	Description
Message Type (MT)	Context ID (in this case 2)
$E_{Cpub}(R_s)$	Random number (R_s) encrypted with the clients encryption key
CA certificate (CC)	Certificate of the CA the server trusts
Verification certificate (VC)	Certificate containing ther server's public key used for signature verification
Signature	Digital signature of the sequence {MT, R_n, $E_{Cpub}(R_s)$, CC, VC, method name, invocation results}

After responding the client, the server creates a so called Client Information Object (CIO). This object stores security information about a particular client, for example, the fingerprints of the certificates the client used to authenticate himself. (s. the right side on fig. 5). This information will be used in later protocol steps. Additionally, each CIO contains two different pseudo-random-number generators (PRNG). PRNGs are used to produce and validate one-time-keys which protect messages exchanged by client and server. $PRNG_S$ is used to generate one-time-keys protecting messages *sent* by the server. Unlike $PRNG_S$ $PRNG_r$ generates one-time-keys for the verification of *received* messages sent by the particular client. The random number R_S generated by the server and sent to the client is used to initialize both $PRNG_S$ and $PRNG_r$.

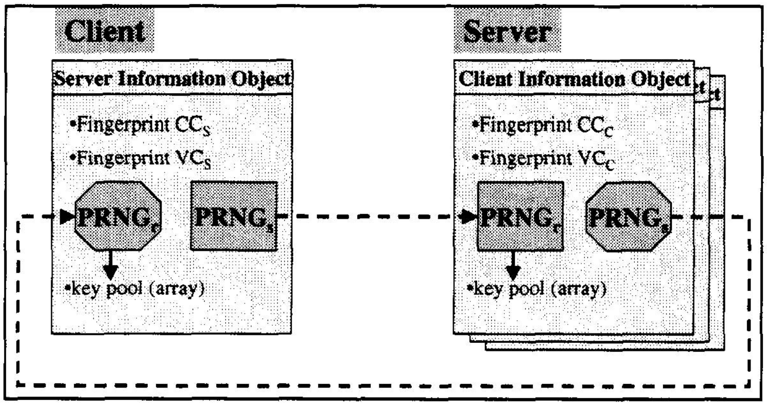

Figure 5. Client and Server Information Objects

In order to speed-up the protocol and to support the out-of-order delivery of requests, $PRNG_r$ generates 16 one-time-keys. These keys are stored in the so called key-pool which is an array. After a received key has been found in the key-pool it will be always deleted, a new key will be generated by $PRNG_r$ and added to the key-pool. The key-pool is only used to check one-time-keys a particular client used to compute a MAC of the message (s. later protocol steps). The CIO does not store any keys that will be sent by the server in future protocol steps.

When CS_2 (in rsp_1) was received, the client first validates the server's signature and then decrypts R_s. At this point the mutual authentication is completed, client and server share the secret R_s. Since public-key encryption is used, R_s is unknown to the proxy and to the rest of the world.

Before sending the next request, the client creates a Server Information Object (SIO) which is very similar to the CIO object (s. the left side of fig. 5). As mentioned above the shared secret R_s is used as initial seed for both pseudo-random-number generators ($PRNG_r$ and $PRNG_s$). As shown in fig. 5 the SIO also contains a key-pool which will be filled up by the $PRNG_r$. This PRNGr must be exactly the same pseudo-random-number generator implementation as $PRNG_s$ on the server side which is used by the server to generate the one-time-keys for sent messages. Fig. 5 depicts the relationship between $PRNG_s$.

The communication continues with client requests (req_2) and server responses (rsp_2). The client ORB adds the security context SC_3 to each request which protects integrity and authenticity of the request message (s. Table 3).

Table 3. Security Contents SC3

Field	Description
Message Type (MT)	Context ID (in this case 3)
Client certificate fingerprit (FP)	Fingerprint of the client's verification certificate
MAC	Message Authentication Code of the sequence MT, i, FP, method name, method parameters} computed with the one-time-key K_i

In order to compute the MAC protecting the message the client generates a K_i key. These keys are being generated by PRNGs of the particular SIO object. K_0 is generated with the aid of the seed value R_s. Every other K_i key used in further messages is generated by the PRNGs automatically. The MAC of the message sequence { MT, i, FP, method name, method parameters} are computed using the key K_i. This sequence contains i which is the position-index of the particular K_i in the server's key-pool generated by the server's $PRNG_r$.

After receiving a protected message the server first searches for a CIO object storing the fingerprint (FP) of the client. If such a CIO exists the server checks the MAC. The MAC is valid only, if K_i matches with the ith element of the key-pool stored in the CIO object. In this case, the server deletes K_i from his pool and responds to the client. Server responses are protected by SC_4 which is very similar to SC_3 as shown in Table 4.

Table 4. Security Contents SC4

Field	**Description**
Message Type (MT)	Context ID (in this case 4)
Server certificate fingerprint (FP)	Fingerprint of the server's verification certificate
MAC	Message Authentication Code of the sequence {MT, i, FP, method name, results} computed with the one-time-key K_i

After receiving a protected server response, the client first checks the MAC. The MAC is valid only, if K_i used by the server matches with the ith element of the key-pool stored in the particular SIO object. In this case, the client generates K_{i+1} and sends a new protected request (i is then replaced with i+1) to the particular server.

If any error occurs during the communication (s. req_3 on Fig. 4), for example, the client's fingerprint is unknown to the server or there was an error (attack) by computing the MAC, the server initializes a new authentication process by sending a message with the security context SC_5 (s. in Table 5).

Table 5. Security Contents SC5

Field	**Description**
Message Type (MT)	Context ID (in this case 5)
CA certificate (CC)	Certificate of the CA the server trusts
Vertification certificate (VC)	Certificate including the server's public key used for signature verification
Signature	Digital signature of the sequence {MT, CC, VC}

Receiving a message with SC_5, the client must re-authenticate himself. In order to do that, the client must re-start the authentication process by sending the failed req_3 request embedding SC_1 again.

3.4 Providing delegation with X.509 certificates

An alternative approach to solve the delegation problem is the modified use of restricted proxies as proposed by B. C. Neuman [8]. In order to provide this delegation mechanism in π^2, the certificate verification procedure of existing SSL implementations has to be modified. At this point, we only describe the main steps that are necessary to propagate the client's identity to the server.

After establishing an SSL connection to the proxy, the client ORB can authorize π_f^2 to act on his behalf The client ORB signs a sequence S including the fingerprint of the signature verification certificate of π_f^2, the fingerprint of the client's signature verification certificate and the expiration time of S. Then the client ORB sends S to the proxy by embedding it in the first CORBA method invocation using the `service_context`. After that the proxy finds a suitable implementation of the target and establishes an SSL connection to that server. Normally, in the mutual SSL authentication process one X.509 certificate of each participant is needed. In this case, the proxy does not only send its own certificate to the server, but also the signed S. Then the targeted server verifies the proxy′s certificate which is signed by the trusted certification authority of the PKI. After that he verifies the client's signature on S and compares the fingerprint of certificate presented by the proxy with the fingerprint of the proxy's certificate sent in S. Finally, the server checks the expiration time of S. This way, the server knows who initiated the given revocation. The server's access control decisions can be made by using this securely delegated identity.

3.5 Implementation de tails

In order to support different operating systems the complete π^2 architecture is based on pure Java technology. In our prototype we used the ORB implementation of the Object Oriented Concepts (OOC). The Java version of ORBacus v. 3.3 is available with source code and it is free of charge for non-commercial use. This is important since we had to modify the ORB in order to provide proxy functionality (s. ORB' in fig. 2). OOC offers for his different ORBs a Java based SSL implementation, called FreeSSL. This product as well its source code are free available by OOC.

The public-key infrastructure utilized for our key-management purposes was the developer edition of the Entrust/PKI v. 4.0 which is free downloadable by Entrust Technologies. In order to integrate the PKI services into FreeSSL we took the Entrust's Java toolkit (Entrust/Toolkit for Java v. 5.1) which is also free available by Entrust.

The communication protocol we introduced applies the Java based crypto-library of IAIK (Institute for Applied Information Processing and Communications) including the PRNG implementations. The most important cryptographic algorithms applied in our prototype are RSA for asymmetric encryption and digital signatures, and H-MAC in order to compute MACs.

4. CONCLUSIONS

In this paper two solutions for the problem of secure and transparent delegation in the CORBA proxy platform π^2 are presented. π^2 supports applications in mobile and wireless environments by transparently integrating two proxies which encapsulate the wireless link, and thus are able to hide its characteristics.

Standard mechanisms providing end-to-end authentication and encryption like SSL or the CORBA security service do not work in a proxy architecture, as these leave the proxy behind useless. Therefore special tailored solutions are needed. Both solutions presented in this paper are using SSL and the services of a PKI in order to secure the sub-connections. SSL provides authentication and encryption on the sub-connections client-to-proxy and proxy-to-server. The protocol we first introduced provides end-to-end authentication which is based on a PKI and the use of asymmetric encryption to exchange a shared secret only known to client and server. This shared secret is used to initialize pseudo-random-number generators. The generated random numbers are used as one-time-keys in the computation of message authentication codes ensuring message integrity and authenticity. In the case of the redirection of request to a different server in an advanced invocation handling mode, this new server can force a new mutual authentication. The second approach which is a slight modification of the key-exchange procedure in SSL, makes it possible to delegate the client's identity to the proxy, securely.

The solutions fulfill the security requirements of the mobile user on the one hand and retain the functioning of π^2 by supporting the different invocation handling modes on the other hand.

REFERENCES

[1] R. Ruggaber, J. Seitz, M. Knapp: π^2 – a Generic Proxy Platform for Wireless Access and Mobility in CORBA, In Proceedings of the 19th Symposium on Principles of Distributed Computing (PODC'2000). Portland, Oregon, USA, July 2000

[2] Rainer Ruggaber, Jochen Seitz:
A Transparent Network Handover for Nomadic CORBA Users
In Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, Arizona, USA, April 2001

[3] ITU-T Recommendation X.509
Information Technology – Open Systems Interconnection – The directory: Public-Key and Attribute Certificate Frameworks, September 2000

[4] Carlisle Adams, Steve Lloyd:
Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations
Macmillan Technical Publishing, 1999

[5] Transport Layer Security Working Group
A. O. Freier, P. Karlton, P. C. Kocher: The SSL Protocol Version 3.0
Internet-draft November 18, 1996

[6] Z. Nochta, S. Abeck, G. Augustin, M. Becker, M. Friedmann:
Integration of Public-Key Infrastructures in CORBA-Systems (in German)
In Proceedings of the Conference “Communication Security” (KSI’2001), Germany, March 2001

[7] Object Management Group: Security Service Specification vl.7,2000

[8] B. Clifford Neuman:
Proxy-Based Authorization and Accounting for Distributed Systems
In Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS), Pittsburgh, May 1993