Non-repudiation Evidence Generation for CORBA using XML

Michael Wichert
GMD - German National Research Center for Information Technology
SIT - Institute for Secure Telecooperation
64295 Darmstadt, Germany
wichert@gmd.de

David Ingham and Steve Caughey
Department of Computing Science
Newcastle University
Newcastle upon Tyne
NE1 7RU, UK
[dave.ingham, s.j.caughey]@ncl.ac.uk

Abstract

This paper focuses on the provision of a non-repudiation service for CORBA. The current OMG specification of a CORBA non-repudiation service forces the programmer to augment the application with calls to functions for generating or validating evidence. Furthermore, the application itself has to manage the exchange of this evidence between parties and its storage. The paper describes our design for a generic CORBA non-repudiation service implementation. Our approach provides a separation between the application business logic and the generation of evidence allowing non-repudiation support to be incorporated into applications with the minimum of programmer effort. Our design is described in this paper using the example of ordering goods over the Internet. The non-repudiation service provides the parties with evidence proving that the transaction has taken place. This proof is a XML document based on the proposed IETF Internet standard Digital Signatures for XML.

1. Introduction

It is predicted that U.S. business trade on the Internet will explode from $43 billion in 1998 to $1.3 trillion in 2003, meaning that more than 9% of total U.S. business sales will be done on the Internet in four years time [3]. The first e-commerce systems were based on point to point communication, where the buyer communicated directly to the seller and submitted orders using e-mail or Web forms. The complexity is now increasing with tighter integration with back-office systems to allow order data to be electronically transferred between businesses. The order phase is only one aspect of a typical transaction, for example, an interaction could include product selection from a categorised catalogue, electronic payment, and in the case of soft goods, electronic delivery.

One of the popular e-commerce models is the electronic marketplace which enables purchasers to select products from multiple vendors. This idea is being extended in the EU-funded MultiPLECX project (Multi-party Processes for Large-scale Electronic Commerce Transactions) which is looking at the issues involved in linking together federated marketplaces [8]. Figure 1 shows an example in which a single categorised product catalogue is available at all linked marketplaces. Each marketplace has a set of 'local' buyers and sellers but the marketplaces are inter-connected to allow buyers to see products from sellers hosted at remote marketplaces. For the buyer, this has the advantage that a broad selection of products are available through his home marketplace which provides a well-understood policy concerning payment, and terms and conditions of delivery.

These more complex models place additional requirements on the underlying technology; simple e-mail and Web communication is not sufficient. Distributed object technology is a good candidate for building such systems as it provides a strong separation between a service interface and its implementation thereby facilitating the construction of inter-organisation applications. The currently available technologies include Microsoft's DCOM (Distributed Component Object Model) [1] and the Object Management Group's CORBA (Common Object Request Broker Architecture) [10]. Trading systems with interfaces specified in CORBA can easily be integrated in existing enterprise resource planning systems (ERP) to avoid multiple manual data input. This enables a continuous data flow from the vendor's back end system to the client's system possibly via several marketplaces using a Web browser with its familiar user interface.

In an Internet based trading system, communication between the parties is notoriously insecure; transferred data could be eavesdropped or worse still, tampered with by an attacker. Because business data is often sensitive, it should be protected by security mechanisms like

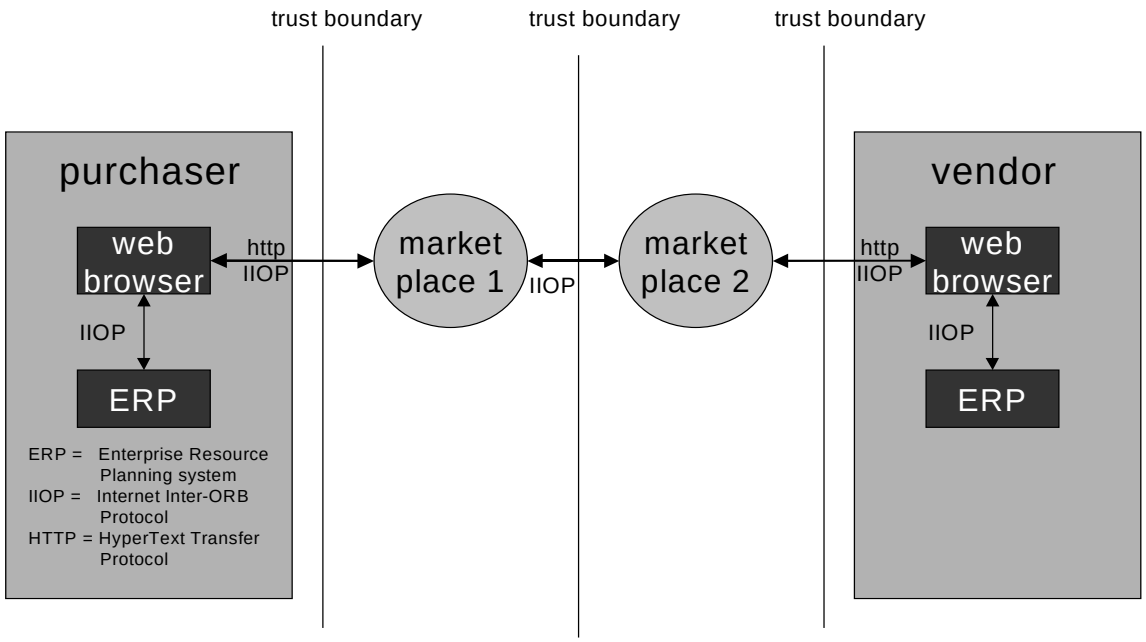

Figure 1. Data Flow in a Trading System

encryption and digital signatures to provide privacy, integrity and authenticity of the data. In trading, several partners are involved with different interests and there is often only limited trust between them. This leads to a need for a system that provides non-repudiation to compensate for the lack of trust between different companies across trust boundaries. Non-repudiation is necessary for commercial transactions like placing an order by a client or updating an internet catalogue that is located at a market place by a vendor.

Non-repudiation is provided by generating an unforgeable evidence that is held by each party that can be used after the fact to resolve disputes. If the non-repudiation system includes both proof of origin and proof of delivery then both business partners have equal rights and none of them is preferred in contrast to traditional e-commerce systems where vendors of goods are in a preferred position.

To provide security for CORBA applications the Object Management Group (OMG) has specified the CORBA security services [9]. This specification describes interfaces to services that provide access control, integrity, confidentiality, non-repudiation and others. These services are designed to be as transparent as possible so as to minimise the impact on the business logic of the application. However, in the case of the non-repudiation service, the programmer has to augment the application with calls to functions for generating or validating evidence. Furthermore, the application itself has to manage the exchange of this evidence between parties and its storage. This paper describes the design of a non-repudiation service that relieves this programmer burden by separating the evidence generation and management from the business logic of the application. The application programmer has to identify for which method invocations non-repudiation is required and the non-repudiation service automatically generates the appropriate evidence, transparently transfers it between parties, verifies its authenticity and stores it. Orthogonal to the management of the evidence is the issue as to what form the evidence should take.

The remainder of the paper is structured as follows: Section 2 gives an overview of the background technologies of non-repudiation and CORBA. Section 3 describes our design in two parts, first the mechanics of evidence management and secondly the form of the evidence itself. Finally, we draw conclusions.

2. Background

This section gives a short introduction into non-repudiation and CORBA. It also explains the specification of the CORBA non-repudiation security service.

2.1 Non-repudiation

Non-repudiation is one of five security services defined by the International Organization for Standardization (ISO), the others being authentication,

access control, data confidentiality, and data integrity [4]. According to ISO, non-repudiation is defined as follows:

> "The Non-repudiation service involves the generation, verification and recording of evidence, and the subsequent retrieval and re-verification of this evidence in order to resolve disputes. Disputes cannot be resolved unless the evidence has been previously recorded." [5]

Non-repudiation itself can be split in different types [6], the main types being:

- non-repudiation with proof of origin;
- and non-repudiation with proof of delivery.

The *non-repudiation with proof of origin* service provides the recipient of the data with evidence proving that the sender has sent the referenced data at a certain time. The n*on-repudiation with proof of delivery* service is also often called non-repudiation of receipt. It provides the sender of the data with evidence that proves that the recipient has received the referenced data at a certain time but it does not prove that the recipient has also processed the data.

Usually the evidence is generated using asymmetric cryptography where the data is digitally signed with the private key. The proof is sent to the communication partner where the digital signature is verified with a public key digital certificate issued by a trusted third party. It is important for the recipient to store the evidence to resolve later disputes. It is also possible to use symmetric cryptographic algorithms for creating a proof. But then an online trusted third party is needed to digitally sign the data with its secret key (notary service).

For a trading system both non-repudiation with proof of origin and proof of receipt are necessary. The non-repudiation service is required to provide the following functions:

- evidence generation;
- evidence delivery;
- evidence verification;
- evidence storage;
- evidence retrieval;
- and evidence re-verification.

2.2 CORBA

CORBA [10] was introduced by the Object Management Group (OMG) in 1991. The OMG was founded in 1989 as a non-profit organisation which now includes more than 800 members. Its main task is to standardise CORBA in a specification that allows interoperability between products of different software vendors. CORBA provides a mechanism for objects with interfaces defined in an Interface Definition Language (IDL) to communicate with each other no matter in what language they are implemented nor on what platform they are located. The objects invoke methods on a server object through the Object Request Broker (ORB) using the Inter-ORB Protocol (IOP). The IOP was specialised for objects to communicate over the Internet in the specification of the Internet Inter ORB Protocol (IIOP). Using CORBA, method invocations can be performed in a distributed environment where the client does not have to know where the server object is located whether locally or across a network. The ORB finds the object if only a reference to it is known, transfers a method call with its parameters to the server, invokes the method and returns the result to the client.

2.3 CORBA non-repudiation service

The CORBA security service specification describes the following services [9]:

- identification and authentication;
- authorisation and access control;
- security auditing;
- security of communication;
- and non-repudiation.

Implementations of the first four services are available from a number of vendors but, at the time of writing, no non-repudiation service is known to exist. There are a number of reasons for this. Firstly, the importance of non-repudiation is not currently widely acknowledged by software manufactures and Internet traders alike. This is reinforced by the fact that support for non-repudiation is an optional part of the specification and therefore vendors are able to sell compliant products without it. The specification is also somewhat incomplete in that mechanisms for evidence delivery, evidence storage and interoperability of non-repudiation evidence is missing.

The specification of a non-repudiation service by OMG is based on an ISO standard [6] that specifies the non-repudiation service functionality in terms of:

- evidence generation and verification;
- evidence storage and retrieval;
- and delivery authority.

Evidence generation and verification are specified in detail and several functions are defined to generate or verify non-repudiation evidence (e.g., generate_token, verify_token). The evidence is generated in the form of a non-repudiation token that cannot be repudiated later. Any holder of such a token can use the non-repudiation service to verify the evidence and may store it in case of

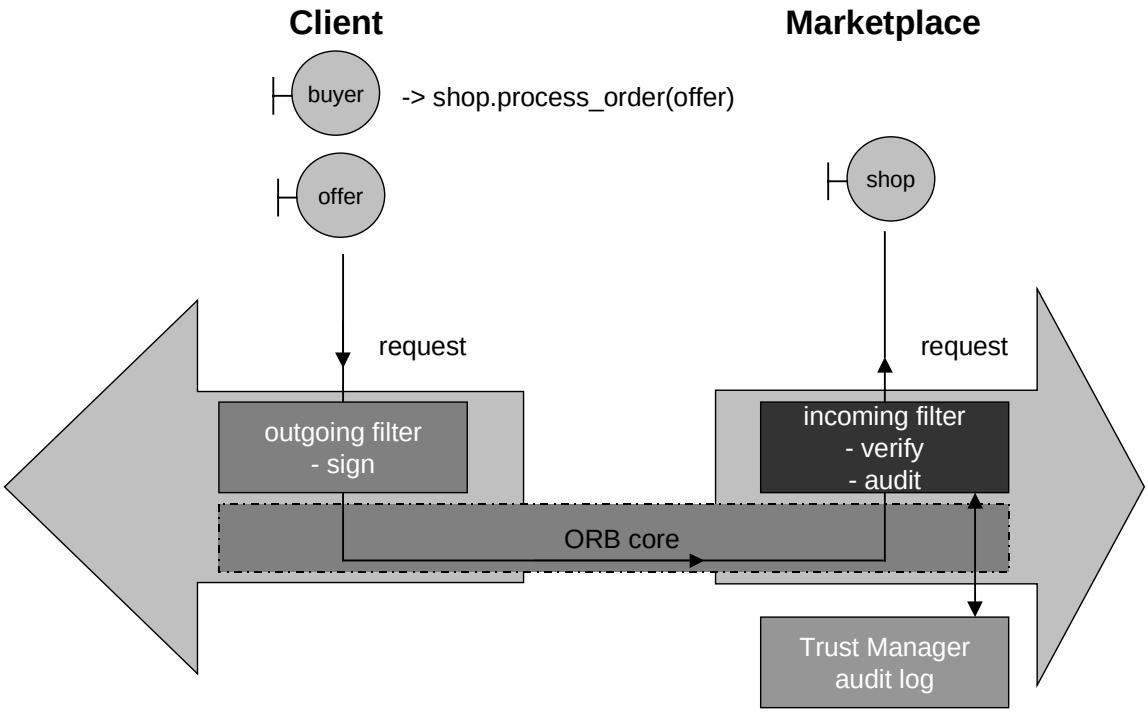

Figure 2. Non-repudiation of an order

later disputes. Because the specified CORBA non-repudiation services are under the control of the applications rather than used automatically on object method invocations the possessor of a token is normally the application that generated it. The specification lacks of a description of storage and retrieval of tokens that means that the application is responsible for administering the tokens. It is also not specified how the token is sent to the recipient but it is proposed to pass it on an invocation as a parameter to a request. According to the specification, a non-repudiation token should be composed of the following components:

- non-repudiation policy (or policies) applicable to the evidence;
- type of action or event;
- parameters related to the type of action or event;
- date and time of action or event;
- digital signature or secure envelope.

Though the token components are defined the format of it is not. This will lead to interoperability problems between products from different vendors. Depending on the chosen cryptographic technology non-repudiation tokens are generated as secure envelopes in the case of symmetric cryptography (requiring a trusted third party) or in the case of asymmetric cryptography the content is digitally signed with the initiator's private key and the corresponding public key is certified by a trusted certification authority. It is not specified what digital signature algorithms and formats to use, nor is it clear how a secure envelope should be composed. The conclusion is that the non-repudiation service for CORBA specified by OMG describes a general approach how a non-repudiation service could be implemented but it doesn't specify an interface to a non-repudiation service that satisfies interoperability criteria.

3. A Transparent non-repudiation service

One of the main design goals for our CORBA non-repudiation service was the minimisation of the burden of evidence management on the application programmer. To this end, we separate the format and specification of evidence from its creation, transport and storage. Application objects, such as an order, are responsible for defining the format of their non-repudiation evidence and application programmers tag object interfaces to indicate the requirement for evidence generation. The non-repudiation service then automatically requests an object to provide evidence where appropriate, transports it between the parties, verifies its authenticity and finally stores it securely. We shall first describe these evidence

management mechanisms before discussing the format of the evidence itself.

In the CORBA environment, object interfaces are defined using language-independent IDL, as previously mentioned. The IDL is pre-processed to create language-specific *stubs*. A client application invokes a method on a stub object and the underlying ORB-core transparently passes the request to the target object which may reside in a remote process. Our non-repudiation service uses the concept of *filters* (or *interceptors*) to transparently manage evidence generation, transport, verification and storage. Filters operate by trapping method invocations as they flow through the ORB. Using filters it is possible to manipulate method invocations at four points:

- as the request leaves the client (*pre-request*);
- when the request arrives at the server but before it is passed to the target object (*pre-dispatch*);
- after the target object has serviced the request but before the response leaves the server (*post-dispatch*);
- and before the response is returned to the client (*post-request*).

The filter interface provides access to the name of the invoked method, to the target object to which the method belongs and to the parameters of the invocation. The C++ signature of the preRequest filter is shown below[1]:

```
virtual bool preRequest (
              CORBA_Object_ptr p,
              CORBA_ULong reqId,
              const char* op,
              OBBuffer& buf);
```

Our service operates by using a pre-request filter (the *outgoing filter*) to create appropriate evidence, sign it and append it to the request. A pre-dispatch filter (the *incoming filter*) removes the evidence, verifies it and store it for later use. The invocation is then passed up to the application for process as normal.

Consider the example of a Marketplace with a categorised catalogue offering goods for sale on the Internet. A Client selects some goods he intends to buy. The Marketplace then sends an offer to the Client and the Client has to decide whether to accept or reject the offer. If it is accepted, e.g., with a button press in a Web browser user interface, then the offer is sent to the Marketplace. This is done with the invocation of the method `process_order` on the object `shop` with the object `offer` passed as parameter as shown in Figure 2.

[1] This C++ code is taken from the filter mechanism that we have added to the ORBacus 3.1 ORB from OOC. See <URL:http://www.ooc.com/>.

The method invocation is intercepted by our *outgoing filter* which then generates a non-repudiation token that contains:

- date and time;
- type of non-repudiation (*non-repudiation with proof of origin*);
- method name (`process_order`);
- and a representation of the parameters (the `offer`).

Note: the representation of the parameters and the format of the evidence will be discussed further in later sections, we will first describe the basic operation of the mechanism.

To provide non-repudiable evidence, the token is digitally signed. The public key certificate and the digital signature are also appended to the token. For interactions which involve human participants, the information to be signed can be presented to the Client before the digital signature is applied. Typically, the Client will be requested to type in a PIN to unlock a smart card, where the private key is located. In other situations where the interaction does not involve human participants the signature can be applied automatically. Finally the filter appends the digitally signed token to the existing buffer of the request parameters.

The request is sent via IIOP to the *Marketplace* where the *incoming filter* verifies the signature on the token and sends the token to the *Trust Manager* which is responsible for storing and retrieving non-repudiation tokens. The *Trust Manager* thus has the task of evidence storage and retrieval defined by the CORBA non-repudiation service specification. The token is then removed from the parameter list before the message is passed up to the application level `shop` object. The token can be retrieved from the *Trust Manager* in case of disputes to prove that a specific action has taken place as claimed.

3.1 Representing user class objects

Passing instances of user classes as parameters in a non-repudiable method call requires special attention. Until very recently, CORBA has only supported the passing of user objects by reference whereas instances of basic types, such as integers are passed by value. In our example, this means that under normal circumstances it will be a reference to the `offer` rather than its state that is passed in the `process_order` invocation. Any invocations that the implementation of the `process_order` method makes on the `offer` object will result in remote communication back to the `offer` object in the client address space. Clearly, this is inappropriate

for the purposes of non-repudiation as including the object reference in the non-repudiation token will not provide evidence as to the state of the offer when the request to process it was made.

CORBA 2.3 maintains this default parameter passing model but also introduces the concept of *valuetypes*, which are user objects that are passed by value. Since, at the time of writing, CORBA 2.3 compliant ORBs are not widely available, we will first present a design assuming no support for valuetypes and then show how they can be used to simplify our design for CORBA 2.3 compliant ORBs.

Without ORB support for value types

Without ORB support for valuetypes, it is necessary to implement our own pass-by-value mechanism for the purposes of non-repudiation. The requirement is that the state of the parameters that are manipulated by the implementation of the method be the same as the states that are preserved in the non-repudiation token.

Our approach operates by tagging user classes that are required to be passed by value by inheriting them from a service-provided IDL interface, `ByValue`. The `ByValue` interface defines methods for packing the state of the object into a buffer and for unpacking the state from a buffer into the object. Illustrative IDL is shown below:

```
interface ByValue {
     void pack (in Buffer buffer);
     void unpack (in Buffer buffer);
};
```

User objects that inherit from `ByValue` implement their own class-specific pack and unpack routines. The pack and unpack operations must encode sufficient information so that packed objects can be recreated at the server side. The format of the packed data is configurable, further details are provided in the next section. The *outgoing filter* invokes the pack operation on each `ByValue`-derived parameter. The packed states are used in the non-repudiation token.

It is not sufficient to simply use the packed state for evidence but allow the implementation of the target method to use a reference to the original object within its processing. Instead a copy of the object is created in the server address space based on the packed state held in the token. This is done automatically by the *incoming filter*: for each user class parameter, a new object is created and the unpack operation is used to recreate its state from the packed data. Before passing the invocation up to the application, the filter modifies the object references in the parameter list to point to these newly created objects causing the implementation of the method to use the local copies rather than references to the originals. Unfortunately, this results in an altering of the semantics of the parameter passing model as there now exist two copies of the objects with no mechanism to ensure that their states are consistent. Application developers have to be aware of this feature when using the non-repudiation mechanism.

With ORB support for value types

The value type mechanism specified in CORBA 2.3 can be used instead of the aforementioned bespoke approach to provide non-repudiation evidence for user classes. Value types are defined at the IDL level and the ORB automatically serialises their state when they are passed as parameters to method invocations. The semantics of value types are very similar to our implementation in that a copy of the object is created in the server address space and there is no consistency mechanism between the copy and the original. The advantage of using the value type facility is that the semantics are well defined. The ORB provides default support for packing (marshaling) and unpacking (unmarshaling) value types. However, it is possible to provide custom marshaling code to control the representation of the packed objects.

3.2 Evidence format

Deciding on the format that the evidence should take is non-trivial; it is essential that there is no possibility of any misinterpretation of the token. The critical part of a token in this sense is the invocation parameter list. There are three requirements regarding the form of a serialised object state. Firstly, it must be complete, in that all of the salient information about the state of the object has to be included so that an identical clone of the object can be created at the server side. Secondly, the state has to be machine-understandable to allow the creation of the clone. Thirdly, the state of the object should be human-understandable otherwise validating and interpreting the evidence in case of dispute would require the availability of the software to ‘decode’ the binary information.

Consider an example where the parameter part of a non-repudiation token consists of three 16-bit integers like "0000 0000 0000 0001 0000 0000 0000 0010 0000 0000 0000 0011". It could be interpreted the following way: the client has ordered one item of an article with order number 2 for 3 dollars. Or it could be read as, the client has ordered three pieces of article with order number 1 for 2 pounds each, depending on the sequence. To overcome these possible misinterpretations, the implementations of the serialise and unserialise operations would have to be agreed upon (i.e., signed) by each party and stored safely. This would greatly complicate the evidence management system. Without care, software

versioning could also lead to possible misinterpretations. Conversely, an ASCII text representation of the object state could be interpreted correctly by humans but would be difficult for machine interpretation. Another possibility is to have two structures, one for the machine and one for the user. But it is impossible to be sure that both have the same meaning without interpreting both structures by the same entity. Our design goal is therefore to have a single representation that is both machine and human readable. To simplify inspection of stored evidence, tool support is necessary that can interpret the state in a way that it could not be misinterpreted. Additionally a conversion algorithm must exist that transforms the text structure into an Java or C++ object to allow processing of the non-repudiable data by the *incoming filter*. This is also important for transferring the data to an enterprise resource planning systems (ERP) for further processing.

Several languages comply with this requirement, for example XML (eXtensible Markup Language) and ASN.1. For our implementation, we have chosen XML due to the widespread availability of interpreters in different implementation languages. Furthermore, current trends suggest that XML is to be the de-facto standard structure description language of the future. XML [13] is specified by the World Wide Web Consortium (W3C) and is a subset of SGML (Standard Generalized Markup Language) [7]. It is a language that describes the structure of documents and data. The most important syntax element of XML is the markup which encodes a description of the document's layout and logical structure. Example markups are tags like <form> or </form> in HTML. XML could be used to describe the content of an object which is done by its pack method. The produced document can be displayed if required to the user after parsing it with an XML parser. It is also possible to reconstruct the original object when it is described in XML. Such a document is shown in the following example, which shows how the content of an offer object could be described.

```
<?xml version=´1.0´?>
<offer>
 <purchaser>
  <name>Tom Buyer</name>
  <address>
   15 Vendor St.,Shopcity
  </address>
 </purchaser>
 <article>
  <quantity>1</quantity>
  <order_number>1345</order_number>
  <name>KTM Sorento</name>
  <catogory>bicyle</category>
  <price>795</price>
  <currency>EUR</currency>
 </article>
</offer>
```

Not only can the object be described in XML but also all parameters of a method invocation or even the whole non-repudiation token. If this evidence token is a document described in XML then the applied digital signature could be integrated in XML as well. There are several initiatives specifying standards for digital signatures incorporated in documents. One important initiative is the proposed Internet standard *Digital Signatures for XML* [2]. It specifies the syntax of a digital signature within an XML document. A digitally signed non-repudiation token complying with this standard could look like:

```
<NR-token>
 <Token-data id="to-be-signed">
  <nr-type>origin</nr-type>
  <method>process_order</method>
 <parameters><offer>...</offer><parameters>
 </Token-data>
 <Signature>
  <Manifest>
   <Resource>
    <Locator href="#to-be-signed"/>
    <ContentType type="text/data"/>
    <Digest>
     <DigestAlgorithms>
      ...
     </DigestAlgorithms>
     <Value encoding="base64">
      pkKE6o2pK7EldfdiIK8Sfb5FjT3V=
     </Value>
    </Digest>
   </Resource>
   <OriginatorInfo>
    (identification information block)
    (keying material information block)
   </OriginatorInfo>
   <RecipientInfo>
    (identification information block)
    (keying material information block)
   </RecipientInfo>
   <Attributes>
    <Attribute type='signing-time'
               critical='true'>
     <Date value='1999-12-12T03:11+0100'>
    </Attribute>
   </Attributes>
   <SignatureAlgorithm>
    (algorithm information block)
   </SignatureAlgorithm>
  </Manifest>
  <Value encoding="base64">
   uSDdfa2sSD82fAS4FD52dfaDsdf3=
  </Value>
 </Signature>
</NR-token>
```

The non-repudiation token (*NR-token block*) consists of the non-repudiation token data (*Token-data* block) and the digital signature referring to the token-data (*Signature* block). The *Token-data* block describes the type (non-repudiation with proof of origin, non-repudiation with proof of delivery) , method name and parameters of a method invocation. The signature block refers to the *token-data* block with the *Locator* markup and contains

the digital signature, information on the used algorithms, the originator, the recipient, date and time. This or the like could be an evidence for non-repudiation for CORBA using XML and a standard digital signature format.

4. Conclusions

The specification of the CORBA non-repudiation service describes the functionality of such a security service but lacks interface details and interoperability. This paper describes an implementation of a non-repudiation service for CORBA that has minimum impact on application programmers. Evidence management is performed automatically, relieving programmer burden in creating, transporting, validating and storing evidence. Non-repudiation tokens are digitally signed and contain date, time, non-repudiation type, method name and method parameter information. Our approach uses the same data for both non-repudiation evidence and application processing improving system integrity. It is proposed to encode the non-repudiation tokens using digitally signed XML documents specified by the proposed Internet standard *Digital Signatures for XML*. It is believed that this approach could form the basis for a proposal for interoperable non-repudiation tokens.

In the EU funded project MultiPLECX we are implementing the generic non-repudiation service described in this paper. The service will be used to provide security for the created infrastructure required to support multi-party Electronic Commerce. The project will run pilots of commercial applications which will demonstrably enable multi-party business-to-business e-commerce transactions over the Internet in a fashion which is secure, robust and scaleable.

5. References

[1] Microsoft Corp., Distributed Component Object Model, http://www.microsoft.com/com/tech/dcom.asp.

[2] Richard D. Brown: *Digital Signatures for XML*. Proposed Internet Standard, January 1999, http://www.ietf.org/internet-drafts/draft-brown-xml-dsig-00.txt, 42 pages.

[3] Forrester Research, Inc., http://www.forrester.com .

[4] ISO 7498-2: *Information processing systems -- Part2: Security Architecture.* International Organization for Standardization, 1989.

[5] ISO 10181-4*: Information technology – Security frameworks for open systems: Non-repudiation framework*. International Organization for Standardization, 1997.

[6] ISO 13888-1: *Information technology – Security techniques – non-repudiation – Part 1: General.* International Organization for Standardization, 1997.

[7] ISO 8879: *Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*. International Organization for Standardization, 1986.

[8] MultiPLECX: *Multi-party Processes for Large-scale Electronic Commerce Transactions*. EU funded ESPRIT project. http://www.multiplecx.org, 1999.

[9] OMG (Object Management Group): *CORBA security specification*. Http://www.omg.org, 1998.

[10] OMG (Object Management Group): *Common Object Request Broker Architecture (CORBA)*. http://www.omg.org, 1999.

[11] Robert Orfali, Dan Harkey: *Client/Server Programming with Java and CORBA*. John Wiley & Sons Inc., New York, 1997

[12] Alan Pope: *The CORBA Reference Guide*. Addison Wesley, Reading, Massachusetts, 1998.

[13] Tim Bray et al.: *Extensible Markup Language (XML)* 1.0. W3C Recommendation., http://www.w3.org/TR/1998/REC-xml-19980210.html, February 1998.