A Fair Non-repudiation Service in a Web Services Peer-to-Peer Environment

Berthold Agreiter, Michael Hafner and Ruth Breu

University of Innsbruck
Institute of Computer Science
A-6020 Innsbruck

Abstract. “Non-repudiation”, a well known concept in security engineering, provides measures to ensure that participants in a communication process cannot later on deny of having participated in it. Such a concept is even more important in service oriented architectures (e.g. electronic billing). However, there is no sophisticated standard implementing fair Non-repudiation in such an environment.
In this paper, we will introduce a framework providing fair Non-repudiation for Web service messages. It executes a previously specified protocol using Web services technology itself, but completely hides the protocol execution from the target Web services. To allow the integration of such security requirements already in an early phase of development, a model-driven configuration approach is used. Furthermore, the procedure is not tied to Non-repudiation protocols only, which means that a broad range of protocols can be integrated in a similar way. The framework presented in this paper leverages existing standards and protocols for an efficient adoption in service oriented architectures.

1 Introduction

Service Oriented Architectures have matured to a widely accepted architectural paradigm for the realization of B2B cooperation. One of the key characteristics of these systems is the peer-to-peer style communication with no central control. Security aspects in B2B communication are heavily regulated by law, which implies that it is also a potential subject to change. Although sophisticated solutions and standards already exist for the enforcement of the basic security requirements – like confidentiality or integrity – this is not the case for more advanced security requirements like Non-repudiation.

So far, there is actually no standard for a protocol enforcing *fair* Non-repudiation in these environments. Available solutions only consider some kind of voluntary, so-called *naïve* Non-repudiation, which cannot guarantee that one communicating party is not gaining any advantage over the other. This means that cooperating parties have to trust each other to behave in a legal way. Nevertheless, the enforcement of protocol based security requirements is crucial in many B2B scenarios, e.g. electronic tendering, electronic auctions, electronic procurement, etc.

In this paper we present an approach which supports the enforcement of a fair Non-repudiation protocol between two parties without even touching the endpoint-services themselves. The communicating parties are forced to behave correctly, otherwise the misuse is detected and no one of the parties gains advantage over the other. We directly apply the methods described herein to the SECTET-framework – a tool-suite for the correct realization of decentralized, security-critical workflows – and deploy it with an example protocol using a trusted third party. The solution presented in this paper is flexible enough to instantiate other protocol-based security requirements as well, e.g., transactional security.

The paper is structured as follows: Section 2 sketches some background to our previous work and related approaches. A motivating example is given in section 3 followed by our solution and partly its theoretical background. We close the paper with a conclusion and a brief outlook in the last section.

2 Background

2.1 The Sectet Framework

The SECTET-framework supports business partners during the realization and distributed management of a common Global Workflow – a decentralized, security-critical collaboration across domain boundaries. SECTET realizes a domain architecture aiming at the correct technical implementation of domain-level security patterns. Security requirements are integrated into the specification of a Global Workflow as UML 2.0 model artifacts. The models form the input for a chain of integrated tools that transform the models into artefacts configuring security components of a Web services-based target architecture. The framework consists of three core components.

The ***Modeling Component*** supports the collaborative definition of a Global Workflow and related security requirements at the abstract level in a platform independent context. It implements an intuitive domain specific language (DSL), which is rendered in a visual language and is currently implemented as a UML 2.0 profile for MAGICDRAW [1].

The ***Reference Architecture*** represents a Web services-based target runtime environment for the Local Workflows and back-end services at the partner node. The workflow and security components implement a set of workflow and security technologies based on XML and Web services technologies and standards.

The ***Transformation Component*** translates the models into executable configuration artifacts for the Reference Architecture. Source and target models can easily be defined or adapted by importing the respective metamodels. This supports domain experts in rapidly developing and adapting a domain specific language in an agile way and visualizes the transformation process.

For an in-depth account on the conceptual foundations of the SECTET framework, please refer to [1-3].

2.2 Non-repudiation

The security requirement *Non-repudiation* is needed in a communication process where two participants want to assure that the other communication partner cannot deny having participated. There are two kinds of Non-repudiation: Non-repudiation of origin, where the sender cannot deny he is the originator of a message, and Non-repudiation of receipt, where the receiver is not able to deny having received a message. Throughout this paper we always want to achieve both kinds of Non-repudiation at once.

There exist a number of Non-repudiation protocols for different purposes and with different properties (e.g. fair, reliable channel, etc.). Some of them, including the one discussed in this paper, use a trusted third party to guarantee Non-repudiation.

2.3 Related Work

The integration of Non-repudiation services for Web services was also studied before. The simplest way – the implementation of a naïve protocol – which was also partly integrated in the SECTET-framework until now, uses digital signatures [4] within the messages exchanged through SOAP [5]. Such an approach is also discussed in [6]. Here, digital signatures on the document are used to ensure that the documents creator really is who he claims to be. For authentication and to prevent replay attacks SSL/TLS [7] is used. Without the addition of SSL the creator and the originator of a message could be different, which would open the door to replay attacks. However, the significant shortcoming with this approach is that Non-repudiation is only voluntary. This means that the recipient is not forced to return a confirmation of receipt. In such cases, the sender is in a weaker position, because he could not present evidence to the judge in case of dispute.

Another approach, but very similar to the one just described, is *WSS: Non-Repudiation* [8]. It is also using digital signatures defines a schema SOAP header elements to transport the Non-repudiation requests and signatures. The different types of receipt requests range from timestamp-only to digital signatures of specific elements. Although this approach supports Non-repudiation of receipt, this form of Non-repudiation is only voluntary. It does not define any mechanism to prove receipt of a message by a non-conformant implementation.

3 Fair Non-Repudiation in a P2P Environment

3.1 A Security Incident

As a motivating example, we present an electronic procurement scenario. A company purchases goods from a supplier. The buyer sends the purchase order to the supplier, which then returns the invoice. The buyer does not receive the ordered goods until he transfers the money stated on the invoice (cf. **Fig. 1**).

Considering this example, a number of security incidents could happen, each of them leading to a conflict almost impossible to resolve:

- The supplier sends an invoice to a buyer without an actual purchase order.
- The supplier does not acknowledge the payment.
- The supplier claims he did not receive a purchase order/money transfer.
- The buyer claims he did not send a purchase order.
- The buyer claims he paid the goods without doing so.

One can see that both participants are able to cheat and cannot present valid evidence to a judge in case of dispute. Annoyingly enough, the respective other participant cannot do anything about it until the business partner changes his mind and complies with his duties.

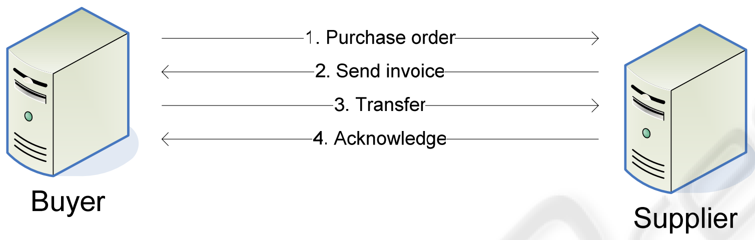


Fig. 1. B2B purchasing process.

3.2 Voluntary Non-Repudiation

The security incident of the example discussed above happens because a naïve protocol was used. Non-repudiation happens to be enforced on a voluntary basis, which means that the recipient is not forced to return a confirmation of receipt. Figure 2 shows the simplest naïve protocol possible:

- *A* sends a signed message to *B*.
- *B* returns a signed acknowledgement to *A*.

The messages are signed with the respective private key of the originator. After *B* received the message *M* from *A*, he can decide by itself whether to send an acknowledgement or not. *B* already has a confirmation that the originator of the message is *A*. This means it is voluntary to *B* to return an acknowledgement to *A*.

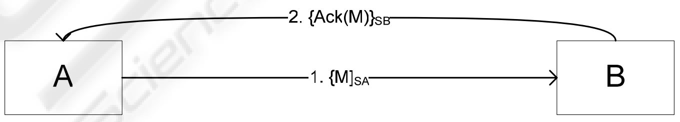


Figure 2: A naive Non-repudiation protocol

3.3 Fair Non-Repudiation

The shortcomings of voluntary Non-repudiation can be counteracted by using different protocols which are aware of this situation. This kind of Non-repudiation protocols is called *fair*. Fair protocols provide the originator and the recipient with valid and irrefutable evidence after completion of the protocol, without giving a party an advantage over the other at any stage of the protocol run [9]. That is, none of the participants is able to cheat.

One such fair Non-repudiation protocol is presented in [9] (cf. **Fig. 3**). This protocol uses a trusted third party (TTP). However, there also exist fair Non-repudiation protocols without the need for TTPs (e.g. [10]). Moreover, different protocols make use of the TTP in different ways (cf. [11]).

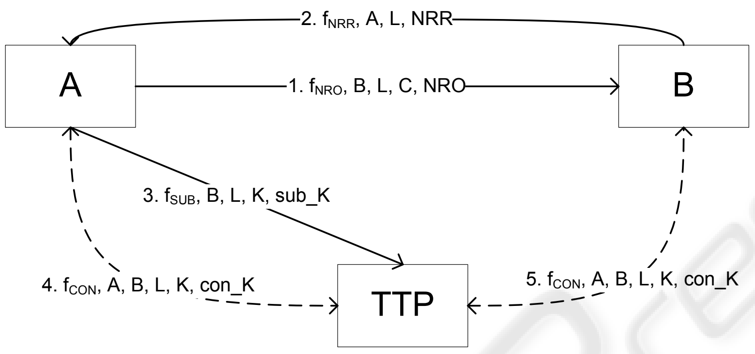

2. f_{NRR}, A, L, NRR
A
1. f_{NRO}, B, L, C, NRO
B
3. f_{SUB}, B, L, K, sub_K
4. f_{CON}, A, B, L, K, con_K
TTP
5. f_{CON}, A, B, L, K, con_K

Fig. 3. The Zhou and Gollmann Non-repudiation protocol (ZG1).

The ZG1 protocol works as follows (cf. **Fig. 3**):

- *A* sends a message to *B* with the Non-repudiation of origin flag (f_{NRO}) set and containing a commitment *C*.
- *B* returns a Non-repudiation of receipt message to *A*, but is still unable to read *C,* because it is encrypted with key *K*.
- *A* submits the key needed to decrypt *C* to the *TTP*.
- After this operation, *A* and *B* are able to retrieve *K* and the confirmation of *K* issued by *TTP* (*con_K*) from *TTP*.

Every message exchanged in this protocol contains a protocol run identifier *L* to correlate sets of messages, and every message is always signed with the private key of the sender. Otherwise it would not be valid evidence. These signatures are denoted by *NRO*, *NRR*, *con_K* and *sub_K*.

In every step of this protocol, the following is valid: if one of the participants tries to cheat, he cannot get any valid evidence supporting its potential fraudulent claims. The evidence needed in case of dispute is either *NRO* and *con_K* or *NRR* and *con_K*.

In this paper we use the ZG1 protocol to demonstrate the functionality of our architecture.

3.4 Modeling of Non-repudiation in the Global Workflow Model

When modeling the inter-organizational workflow of two partners, one has to configure which part of the communication process should be Non-repudiatable. We do this in the Global Workflow Model (cf. **Fig. 4**). Out of this security configuration, the necessary entries in the XACML policy (see section Protocol Execution) are generated. This has two consequences:

- The triggering of the protocol-execution when sending a message.

- The enforcement of a check whether a protocol has been executed when receiving a message.

There is even a possibility to configure the kind of protocol to be executed on the Global Workflow Model.

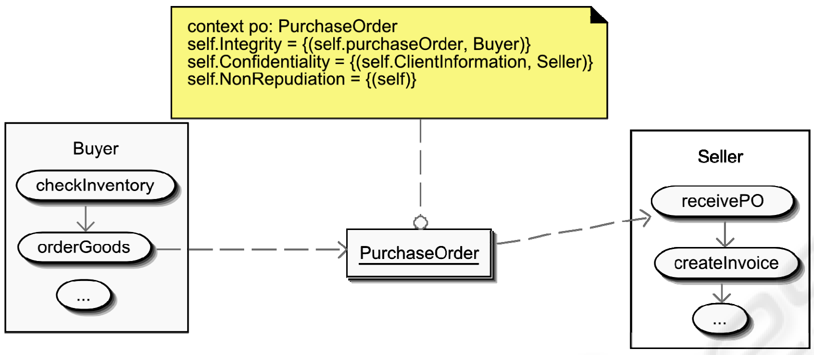
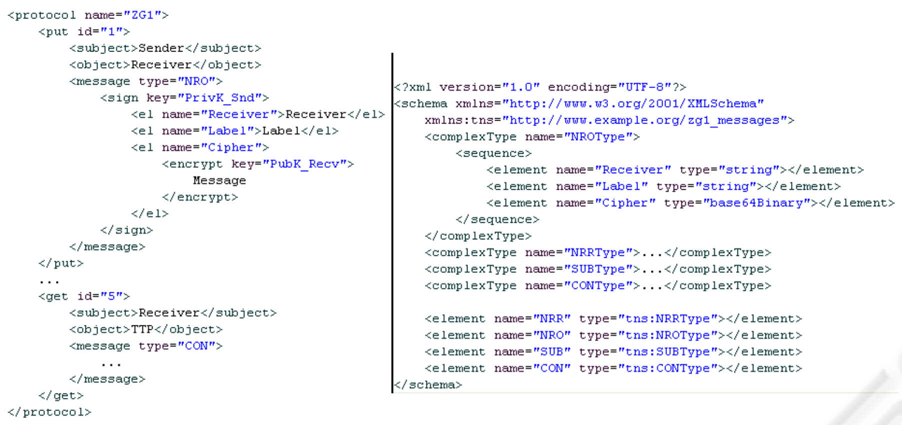

context po: PurchaseOrder
self.Integrity = {(self.purchaseOrder, Buyer)}
self.Confidentiality = {(self.ClientInformation, Seller)}
self.NonRepudiation = {(self)}
Buyer
checkInventory
orderGoods
...
PurchaseOrder
Seller
receivePO
createInvoice
...

Fig. 4. Configuration of Non Repudiation in Global Workflow Model.

3.5 Protocol Specification

First of all, the protocol itself is described in a machine-readable way. XML offers great possibilities for doing this, not least due to its interoperability and ease of use. An XML-based definition is also necessary for the messages which are part of the protocol. There are a number of primitives which have to be replaced by the respective sender of a message (e.g. `Receiver`, `Label`, `Message`, etc.). The protocol gives a description about which elements are used at which place in a protocol run. All this information configures the Target Architecture by storing it in the Protocol Storage.

The potential of this approach lies in the fact that this protocol configuration could also be specified in a model-driven way. This would result in an even more intuitive way for the specification of various protocol-based security requirements (e.g. fair Non-repudiation without TTP, transactional security, etc.) while preserving high flexibility. For the time being, we assume a specific protocol for Non-repudiation – namely the ZG1 – graphically modeled for a specific transaction in a Global Workflow and automatically translated for the protocol engines of the target architectures.



```xml
<protocol name="ZG1">
    <put id="1">
        <subject>Sender</subject>
        <object>Receiver</object>
        <message type="NRO">
            <sign key="PrivK_Snd">
                <el name="Receiver">Receiver</el>
                <el name="Label">Label</el>
                <el name="Cipher">
                    <encrypt key="PubK_Recv">
                        Message
                    </encrypt>
                </el>
            </sign>
        </message>
    </put>
    ...
    <get id="5">
        <subject>Receiver</subject>
        <object>TTP</object>
        <message type="CON">
            ...
        </message>
    </get>
</protocol>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.example.org/zg1_messages">
    <complexType name="NROType">
        <sequence>
            <element name="Receiver" type="string"></element>
            <element name="Label" type="string"></element>
            <element name="Cipher" type="base64Binary"></element>
        </sequence>
    </complexType>
    <complexType name="NRRType">...</complexType>
    <complexType name="SUBType">...</complexType>
    <complexType name="CONType">...</complexType>

    <element name="NRR" type="tns:NRRType"></element>
    <element name="NRO" type="tns:NROType"></element>
    <element name="SUB" type="tns:SUBType"></element>
    <element name="CON" type="tns:CONType"></element>
</schema>
```

Fig. 5. Sample protocol and schema definition.

3.6 Protocol Execution

The execution of the protocol happens in a transparent way, which means that the Web service client and server are not actively taking part in it. Instead, the protocol is executed by the Non-Repudiation Engine, which is part of a Security Gateway wrapping the services. This engine executes the protocol (cf. **Fig. 5**) stored in the Protocol Storage. In short, the protocol is logically executed on a lower layer than the Web services calls. The messages exchanged by the Non-Repudiation Engine are not part of the Web service call from an application point of view. They build a signaling channel which is again realized through Web services technology (cf. Figure 6).

As documents are transferred, the Non-Repudiation Engine somehow has to know two important things:

1. Whether to execute the protocol or not.
2. At which stage of protocol execution it is.

The first issue is implemented through the use of policies. As documents are sent, the Security Gateway always checks for the security requirements and enforces them. In case of Non-repudiation, this triggers the execution of the protocol. The entry in the policy has the nice implication that the receiving endpoint can also check whether a Non-repudiation protocol was executed. Thus, starting to execute the protocol and observe its enforcement is solved via XACML policies.

The solution to the second issue, the identification of the protocol execution step, does not seem trivial in the first place. The point is that Web services should be stateless to retain the independency of different clients. This can be solved by avoiding to store the service-state in the service themselves, and to outsource this storage to the execution engine. We then only need a link to the pertaining state every time a message arrives. A protocol run identifier (i.e. the label *L* in ZG1) would fit perfectly for this purpose. This means by submitting a unique identifier along with each message in a protocol run, every endpoint is able to determine and modify its current state in this run. A comparable standard tackling these kind of problems is described in [12].

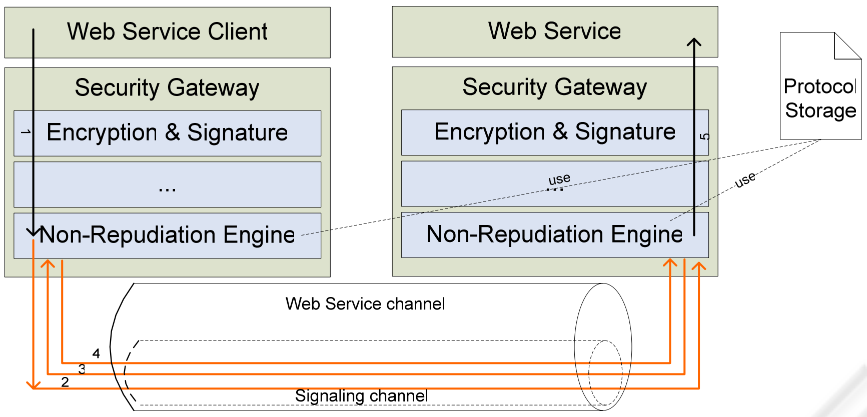


Fig. 6. Layer architecture.

Depending on the protocol the TTP, if any, takes part differently in the process. In the case of ZG1, it stores the proof of submission and lets clients fetch the confirmation of the key K.

3.7 Dispute Resolution

One case which has to be considered by any Non-repudiation protocol is the case of dispute. If one of the communication partners denies having participated in a communication, a kind of independent party – a “judge” - will evaluate the evidence held by the participants and determine whether recipient or origin are as claimed. Therefore, one thing we further integrated in our architecture is the Evidence Store. The Non-Repudiation Engine has the knowledge of how protocols have to be executed (through the usage of the protocol storage) and simply forwards evidences to this store. So in case of dispute the participants can present them to a judge.

3.8 Target Architecture Integration

The affected components in the target architecture are the Protocol Engine, Evidence Store and Non-Repudiation Engine which acts as a handler inside the Security Gateway (cf. Fig. 7).

The Non-Repudiation Engine is the interface to the Protocol Engine. Every message passes this lightweight handler, which triggers certain protocol steps when necessary. The Protocol Engine themselves contains the protocol definitions (Protocol Storage) and different execution states. This component is executing the protocols and also has an interface to the Evidence Store. One can see that the definition of a protocol is in fact the configuration of the Protocol Engine.

3.9 Benefits

Our framework implements a comprehensive approach for the model-driven realization of security requirements in a Web service environment. The use of the SECTET-framework comes with a number of benefits:

1. High and flexible configurability through the use of a model-driven approach.
2. Evidence is bound to Web service messages. This would not be achievable that easy for a realization on the transport layer.
3. The Web services itself remain untouched. The whole protocol is executed before the endpoint receives the message. This uncoupled execution renders the Protocol Engine and Configuration exchangeable.

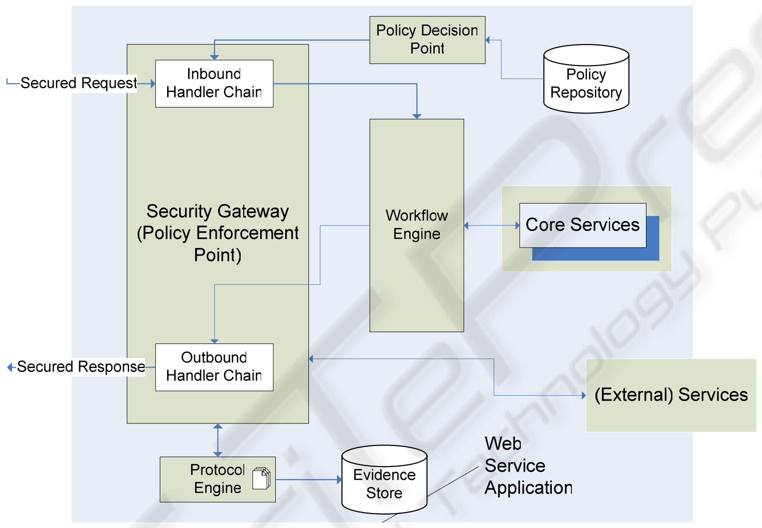


Fig. 7. SECTET Target Architecture.

4 Conclusions and Outlook

Fair Non-repudiation is a security requirement imposed by the business logic. However, the simple application of crypto-functions like digital signature and encryption is not sufficient. A protocol, which has a set of non-trivial implications, is needed.

In this paper, we presented how such a fair Non-repudiation protocol can be applied to the world of Web services without ever touching the Web services itself. The whole protocol-related traffic runs over a signaling channel which ends at each participant's Security Gateway. On the other side, we remain technically on the Web services layer since the messages we exchange in the protocol are still SOAP messages. This also makes sense with respect to the evidence collection.

In the future we will have a deeper look into the modeling of protocols. The Protocol Engine should follow the same principles as the rest of the SECTET-framework, i.e. the configuration takes place in a model-driven way from which executable artifacts are produced. This leaves the engine open to any kind of protocols, not only Non-repudiation protocols.

A first step into this direction could be to integrate a Non-repudiation protocol without TTP. Different kinds of protocols need different kinds of primitives. A kind of plug-in mechanism would render the architecture more flexible.

Another case not discussed in this paper is the dispute resolution. Here, a judge has to collect evidence from the participants and probably from the TTP.

Overall, we can say that the model-driven configuration of security requirements and protocols in this case offers many possibilities in rendering a system more flexible and considering security aspects in an early phase of development. The evaluation of the profitability of for such a Protocol Engine with its configurability of a lower layer is an interesting open problem for future research.

References

1. Hafner, M., M.M. Alam, and R. Breu. *Towards a MOF/QVT-based Domain Architecture for Model Driven Security*. in *Models 2006*. 2006. Genova, Italy.
2. Hafner, M., et al. *Realizing Advanced Security Requirements for Inter-organizational Workflows*. in *eChallenges 2006*. 2006. Bacelona, Spain.
3. Hafner, M., et al. *Sectet - An Extensible Framework for the Realization of Secure Inter-Organizational Workflows*. in *WOSIS 2006*. 2006. Paphos, Cyprus: INSTICC Press.
4. Bartel, M., et al. *XML-Signature Syntax and Processing*. 2002 [cited 2007 19.01.]; Available from: http://www.w3.org/TR/xmldsig-core/.
5. *SOAP Version 1.2 Part 1: Messaging Framework*. 2003 [cited 2007 19.01.]; Available from: http://www.w3.org/TR/soap/.
6. Hada, S. *SOAP security extensions: digital signature*. 2001 [cited 2007 18.01.]; Available from: http://www-128.ibm.com/developerworks/library/ws-soapsec/.
7. Freier, A., P. Karlton, and P. Kocher, *The SSL Protocol Version 3.0*. 1996.
8. Gravengaard, E., *Web Services Security: Non-Repudiation*. 2003.
9. Zhou, J. and D. Gollmann, *A Fair Non-repudiation Protocol*. 1995: University of London, Royal Holloway, Department of Computer Science.
10. Markowitch, O. and Y. Roggeman, *Probabilistic non-repudiation without trusted third party.* Second Conference on Security in Communication Networks, 1999. 99.
11. Kremer, S., O. Markowitch, and J. Zhou, *An intensive survey of fair non-repudiation protocols.* Computer Communications, 2002. 25(17): p. 1606-1621.
12. Graham, S., et al. *Web Services Resource 1.2*. 2006 [cited 2007 19.01.]; Available from: http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.