Implementing Fair Non-Repudiable Interactions with Web Services

Paul Robinson, Nick Cook and Santosh Shrivastava
School of Computing Science, University of Newcastle, UK
Email: {p.robinson, nick.cook, santosh.shrivastava}@ncl.ac.uk

Abstract

The use of open Internet-based communications for business-to-business (B2B) interactions requires accountability for and acknowledgment of the actions of participants. Accountability and acknowledgment can be achieved by the systematic maintenance of an irrefutable audit trail to render the interaction non repudiable. To safeguard the interests of each party, the mechanisms used to meet this requirement should ensure fairness. That is, misbehaviour should not disadvantage well-behaved parties. Despite the fact that Web services are increasingly used for enabling B2B interactions, there is currently no systematic support to deliver such guarantees. This paper introduces a flexible framework to support fair non-repudiable B2B interactions based on a trusted delivery agent. A Web services implementation is presented. The role of the delivery agent can be adapted to different end user capabilities and to meet different application requirements.

Keywords: System FT; FT Architecture/Middleware Software Engineering; System Security; Networks/Networking; Performance, dependability and security issues in the administration of large computer systems; Non-repudiation; Web Services; Fair exchange

1 Introduction

The increasing use of open internet-based communications for business-to-business (B2B) interactions adds urgency to the requirements for security and regulation to safeguard the interests of participants. These requirements include: accountability for and acknowledgement of the actions of participants; and the monitoring of interactions for compliance with business contract. Accountability and acknowledgement can be achieved by the systematic maintenance of an irrefutable audit trail to render B2B interactions non-repudiable. Regulation entails the monitoring of interactions to ensure that messages exchanged are consistent with the business contracts that govern the interaction.

The above requirements are particularly important in high-value B2B relationships, such as in a virtual organisation (VO). In a VO a number of autonomous organisations collaborate to achieve some mutually beneficial goal. Each organisation requires that their interests are protected in the context of the VO. Specifically, that partner organisations comply with contracts governing the VO; that their own legitimate actions (such as delivery of work, commission of service) are recognised; and that partner organisations are accountable for their actions. This implies the recording of activity for audit and the monitoring of activity for compliance with the regulatory regime. Further, to protect the interests of well-behaved members of a VO, the interaction should be non-repudiable (no party should be able to deny their participation) and the auditing and monitoring functions must be fair (misbehaviour should not disadvantage well-behaved parties).

It is increasingly common to standardise B2B interactions in terms of message-exchange patterns. The work of the RosettaNet Consortium [22] is a case in point. RosettaNet define the externally observable aspects of a B2B interaction through a set of Partner Interface Processes (PIPs). PIPs standardise the XML-based business messages that should be exchanged between partners to execute some function (such as order processing). Figure 1 shows the delivery of a business message and associated acknowledgements in such

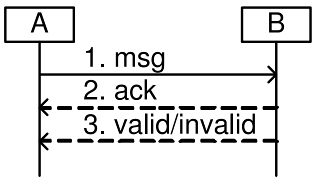

Figure 1: Business message delivery with acknowledgements

an interaction. Typically, for each business message, there should be an immediate acknowledgement of receipt — indicating successful delivery of the message. Eventually a second acknowledgement indicates whether the business message is valid (or invalid) in the context of the given interaction. This validation (performed at B) can be arbitrarily complex. For example, it may simply involve verification that a message is syntactically valid and in correct sequence with respect to a single PIP. Alternatively, a message may require validation with respect to more complex contractual conditions or with respect to local application state. Triggering validation at the level of business message delivery has the potential to allow specialization of an application to meet the constraints of different regulatory regimes. A PIP, or composition of PIPs, may specify the general form of a B2B process that is then validated at run-time in the context of a specific business relationship. Web services are increasingly used to enable B2B interactions of this kind. However there is currently no systematic support to make the exchange of the business message for acknowledgements both fair and non-repudiable. For example, there is no systematic support for preventing a customer from denying they submitted a purchase order and at the same time preventing the supplier from denying its receipt.

The main contribution of this paper is the introduction of a flexible framework for fair, non-repudiable message delivery and its implementation using Web services technologies. The implementation comprises a set of services that are invoked at the middleware level and, therefore, enable the Web services developer to concentrate on business functions. Our middleware renders the exchange of business messages fair and non-repudiable. Arbitrarily complex, application-level validation is supported through the registration of message validators to validate messages upon receipt.

Section 2 provides an overview of the underlying concepts of non-repudiation and fairness, and of our approach to achieving these properties for an interaction of the type described above. Section 3 provides a detailed discussion of fair exchange protocols chosen for the implementation. The implementation is based on a third party delivery agent (DA). The agent can either take on most of the responsibilities of evidence verification and storage and, thereby, simplify the tasks for its users; or greater responsibility can be transferred to the users to reduce the demands on the delivery agent. Section 4 describes the implementation based on Web Services technologies. Section 5 discusses related work. Section 6 concludes the work.

2 Basic concepts and approach

In this section we introduce some basic concepts that will be used throughout the paper. We then provide an overview of the approach taken to render the interaction described in Section 1 both fair and non-repudiable.

2.1 Basic concepts

Non-repudiation is the inability to subsequently deny an action or vent. It is one of the key properties of secure systems as defined in [11]. In the context of distributed systems, non-repudiation is applied to the sending and receiving of messages. For example, for the delivery of a message from *A* to *B*: (i) *B* may require non-repudiation of origin of the message (NRO) — irrefutable evidence that the message originated at *A*; and (ii) *A* may require non-repudiation of receipt of the message (NRR) — irrefutable evidence that *B* received the message

Non-repudiation is usually achieved using public key cryptography. If *A* signs a message with their private key, *B* can confirm the origin of the message by verifying the signature using *A's* public key. Similarly, given *B's* signature on the message, *A* can confirm receipt by verifying the signature using *B's* public key. To support the assertion that a key used to sign evidence was not compromised at time of use, and for audit trail logs, signed evidence should be timestamped by a mutually trusted third party timestamping service [27].

Exchanging non-repudiation evidence is essential to be able to subsequently demonstrate what happened during an interaction. An additional requirement is that at the end of the interaction no well-behaved party is disadvantaged. For example, consider the situation where the sender provides proof of origin but does not obtain the corresponding proof of receipt. This is unfair because the receiver can choose to deny receipt of a message. Fairness can be achieved by executing a fair exchange protocol. Markowitch et al [15] provide the following definition of fairness: "... The communication channel's quality being fixed, at the end of the protocol run, either all involved parties obtain their expected items or none (even a part) of the information to be exchanged with respect to the missing items is received.".

All practical fair non-repudiation protocols require involvement of a trusted third party (TTP) to some extent. The level of intervention can vary depending on the protocol and the requirements of the end users. Kremer et al [12] categorise TTPs into three main types, inline, online and offline. An inline TTP (sometimes called a delivery agent) is involved in each message's transmission during the protocol. An online TTP is involved during each session of the protocol but not during each message's transmission. An offline TTP is involved in a protocol only in case of incorrect behaviour of a dishonest entity or in case network failures.

2.2 Overview of approach

To illustrate our approach we take the business interaction described in Section 1 and make it fair and non-repudiable. Figure 2 shows the addition of a delivery agent (inline TTP) to the interaction in Figure 1. Four types of evidence are generated; (i) the proof of submission (NRS) that the *DA* received *msg* and is able to continue the protocol; (ii) Proof of origin (NRO) that *msg* originated at *A*; (iii) Proof of receipt (*NRR*) that *B* has received *msg*; (iv) Proof of validation result (*NRV*) regarding the outcome of *B*'s validation of *msg.* As shown, *A* starts an exchange by sending a message, with proof of origin, to *DA*. This is the equivalent of message 1 in Figure 1 with the *NRO* appended. *DA* exchanges *msg* and *NRO* for *NRR* with *B* (before application-level validation of *msg*). The *DA* provides *NRR* to *A* — equivalent to message 2 in Figure 1.

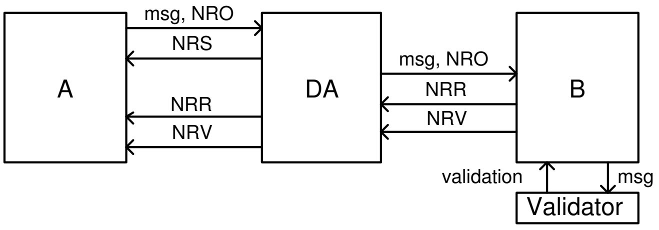

Figure 2: Executing a business interaction through a delivery agent

Subsequently, *B* performs application-level validation of *msg* (as in message 3 of Figure 1) and provides *NRV* to *DA*. The *DA*, in turn, provides *NRV* to *A*. Note the exact sequence of message exchange will be dictated by the actual protocol used and should not be inferred from Figure 2.

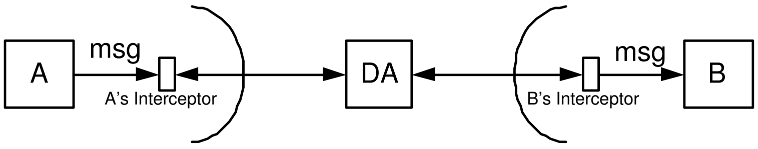

Figure 3: Interceptor approach

As shown in Figure 3, our approach is to deploy interceptors that act on behalf of the end users in an interaction. An interceptor has two main functions: (i) to protect the interests of the party on whose behalf it acts by executing appropriate protocols and accessing appropriate services, including trusted third party services; and (ii) to abstract away the detail of the mechanisms used to render an interaction safe and reliable for its end user. In this case, the mechanism used is to communicate through a trusted third party — *DA*. It is the responsibility of the *DA* to ensure fairness and liveness for well-behaved parties in interactions that the *DA* supports. Further, the *DA*'s fairness and liveness guarantees hold for well-behaved parties in spite of any misbehaviour by any other party involved in an interaction (including misbehaviour by interceptors). Since the cooperation of misbehaving parties cannot be guaranteed, *in extremis* the *DA* will ensure that any disputes that arise can be resolved in favour of well-behaved parties.

For an interaction of the type described in Section 1, the introduction of interceptors means that the end users, *A* and *B*, experience the message exchange shown in Figure 1. However, the exchange that actually takes places is as shown in Figure2. That is, as far as possible, *A* and *B* are free to concentrate on application level concerns while their interaction is rendered fair and non-repudiable.

Since from *DA's* point of view there is no distinction between *A* and *A's* interceptor (and likewise for *A* with respect to *B* and *B* with respect to *A*), in the remainder of the paper we only distinguish between an end user and its interceptor when necessary. For example, when discussing the implementation.

3 Delivery-agent based fair exchange

This section discusses the initial set of two fair exchange protocols that have been implemented. First we provide an overview of the chosen protocols and discuss the motivation behind the choice. Then we state protocol assumptions and notation. The section concludes with a detailed description of each protocol. For brevity, in each case only the main (normal operation) protocol is presented. In addition to the main protocol there are sub-protocols for abnormal termination. Since the delivery agent is trusted these sub-protocols can be used to deliver fairness and liveness guarantees in the event of failure of the main protocol. The interested reader is referred to our related technical report for details of the sub-protocols [21].

3.1 Protocol overview

Coffey and Saidha developed a fair non-repudiation protocol utilising an in-line TTP [2]. This was later improved by Zhou and Gollman in [27]. We discuss two protocols, both of which are derived from the improved version. Both protocols include an extension to support the extra validation message described in Section 2. The first protocol is a further modification to support light weight end uses. The second is intended for use with a more light weight delivery agent, as envisaged by Coffey and Saidha.

In the first protocol, the delivery agent is responsible for much of the evidence verification and for the long-term storage of evidence for audit. The end users are only required to verify evidence produced by the delivery agent. They only require long-term storage for the information necessary to link an interaction to the evidence held by the delivery agent (such as a run protocol identifier). This approach means the end users do not have to provide as much infrastructure support for fair exchange. For example, they only need access to the certificate and public key of the delivery agent for verification and not the information for all parties they will potentially interact with.

In the second protocol, the responsibilities for evidence verification and long-term storage are transferred to the end users. In this case the delivery agent may discard information after completion of a protocol run. Furthermore, the light weight delivery agent only signs the NRS as oppose to all evidence generated.

3.2 Assumptions

We make the standard perfect cryptography assumptions [23], including: (i) that message digests are one-way, collision resistant; (ii) that it is computationally infeasible to predict the next bit of a secure pseudo-random sequence even with complete knowledge of the algorithmic or hardware generator and all of the previous bits in the sequence; (iii) that digital signatures cannot be forged; and (iv) that encrypted data cannot be decrypted except with the appropriate decryption key.

The following assumptions are made with respect to well-behaved parties to a non-repudiable interaction: (i) the communication channel between well-behaved parties provides eventual message delivery (there is a bounded number of temporary network and computer related failures); (ii) each party has persistent storage for messages. More precisely, well-behaved parties will ensure that messages are available for as long as is necessary to meet their obligations to other parties. Longer term storage may be required for their own purposes and (iii) well-behaved parties only exchange messages that are well-constructed with respect to the protocol being executed. For example: messages exchanged are either tamper-resistant (encrypted), or tampering is detectable and well-behaved parties will cooperate to ensure a well-constructed message is eventually delivered.

To guarantee fairness, we make the same assumptions with respect to the DA as in existing fair exchange protocols, namely: (i) that the DA is well-behaved; (ii) that, given the perfect cryptography assumptions, the DA can detect the misbehaviour of other parties and (iii) that the DA ensures protocol resolution in favour of well-behaved parties.

3.3 Notation

In both protocols an originator, A, wants to send a message to a recipient, B, and obtain proof of receipt. All communications between A and B take place through the delivery agent DA. The following tables present the notation used in protocol descriptions. Table 1 provides the notation for some basic protocol elements. Table 2 defines the non-repudiation tokens generated during a protocol run.

Notation	Description
rn	A secure pseudo random number.
$h(X)$	A secure hash of *X*.
id	unique identifier, for convenience this is assumed to contain $h(rn)$.
$i,\ j$	concatenation of two items *i* and *j*.
$P \rightarrow Q :\ msg$	principal *P* sends message *msg* to principal *Q*.
$sig_P(Y)$	principal *P*'s digital signature on Y.
$enc_P(msg)$	message *msg* encrypted with principal *P*'s public key.

Table 1: Notation for protocol elements

Non-repudiation token	Description
$NRS_{DA} = sig_{DA}(id)$	An unforgeable acknowledgment by *DA* of the submission of *msg* by *A*.
$NRO_A = sig_A(id,\ A, B, msg)$	Non-repudiation of origin for *msg* by *A*.
$NRR_B = sig_B(id,\ A, B, h(msg))$	Non-repudiation of receipt of *msg* by *B*.
$NRV_B = sig_B(id, valid\|invalid)$	Non-repudiation of the validity of *msg* by *B*.
$NRO_{DA} = sig_{DA}(id,\ A, B, msg)$	Non-repudiation of origin for *msg* vouched for by *DA*.
$NRR_{DA} = sig_{DA}(id,\ A, B, msg)$	Non-repudiation of receipt for *msg* vouched by *DA*.
$NRV_{DA} = sig_{DA}(id, valid\|invalid)$	Non-repudiation of the validity of *msg* vouched by *DA*.

Table 2: Notation and definitions of non-repudiation tokens

3.4 Fair exchange for lightweight end users

The protocol for light weight end users is given below, followed by a step-by-step explanation.

$$\begin{array}{lllll}
1 & A \rightarrow DA & : & id,\ enc_{DA}(msg, rn),\ A,\ B,\ NRO_A \\
2 & DA \rightarrow A & : & NRS_{DA} \\
3 & DA \rightarrow B & : & id,\ A,\ B,\ h(msg) \\
4 & B \rightarrow DA & : & NRR_B \\
5 & DA \rightarrow A & : & NRR_{DA} \\
6 & DA \rightarrow B & : & NRO_{DA},\ msg \\
7 & B \rightarrow DA & : & NRV_B \\
8 & DA \rightarrow B & : & id,\ rn \\
9 & DA \rightarrow A & : & NRV_{DA}
\end{array}$$

1. The protocol begins with *A* sending message containing the following elements to *DA*: the business message, the identities A and B, the protocol run id and the random number used to generate the id. The NRO_A described in the notation is also provided. The random number *rn* will be used in step 8. If *DA* finds that the *id* used is not unique, an appropriate response will be generated to restart the protocol with a newly generated identifier.

2. In this step, *DA* provides proof of submission to *A* signaling that the *DA* has received the message and will continue with protocol execution.

3. *DA* sends a digest of the *msg* and *id* along with the two identifiers *A* and *B* to *B* to enable *B* to construct NRR_B.

4. *B* responds with the NRR_B. Since *DA* is trusted, it is safe for *B* to provide the receipt before obtaining the *msg* as *DA* can and will provide *msg* in return. The proof of receipt, NRR_B consists of *B*'s signature on the hash of the message along with the identifiers representing *A*, *B* and *id*. At this point *DA* has both NRO_A and NRR_B.

5. DA sends NRR_{DA} to *A*. This is DA's assurance that it has received and verified NRR_B and that the piece of evidence will be stored until needed.

6. *DA* gives *B* the NRO_{DA} and the associated *msg* for validation.

7. *B* performs application-level validation of *msg*. The outcome of this validation is signed along with *id* — the NRV_B that is sent to *DA*. *B* only receives the random number authenticator for the protocol run if they provide NRV_B. Hence the protocol cannot complete as far as *B* is concerned until step 8.

8. The unforgeable acknowledgement for NRV_B, in the form of the random number, is sent by *DA* to *B*. Since, hitherto, only *A* and *DA* knew the *rn*, *B* cannot have obtained by any means other than one of them providing it.

9. The final step consists of DA sending NRV_{DA} to A. NRV_{DA} is *DA*'s proof that it has the validation result and vouches that it will be stored for audit.

We have made some modifications to the protocol in [27] to reduce the verification requirements of the end users and to provide non-repudiable validation of messages. Given that *DA* is trusted, the burden of verification and long-term storage can be relieved from *A* and *B*. Thus, in out modified protocol evidence distributed by *DA* is signed by *DA* (as opposed to *DA* simply passing on the evidence generated by *A* or *B*). The *DA* provides long-term storage of all evidence and therefore can produce the evidence signed by *A* or *B*, if required. The benefit is that neither *A* nor *B* need to be able to verify or store the other's evidence. This allows lighter weight end users at the cost of a more heavy weight delivery agent. To provide for exchange of non-repudiation of validation of the business message, the additional steps 7-9 have been added. If it is known *a priori* that such validation is not required, a shorter version of the protocol can be executed, terminating at step 6.

3.5 Fair exchange with lightweight delivery agent

This protocol is closer to the improved version of [2], using a lighter weight delivery agent. The protocol is shown below.

$$\begin{array}{llcl}
1 & A \rightarrow DA & : & id,\, enc_{DA}(msg),\, A,\, B,\, NRO_A \\
2 & DA \rightarrow A & : & NRS_{DA} \\
3 & DA \rightarrow B & : & id,\, A,\, B,\, h(msg) \\
4 & B \rightarrow DA & : & NRR_B \\
5 & DA \rightarrow A & : & NRR_B \\
6 & DA \rightarrow B & : & NRO_A \\
7 & B \rightarrow DA & : & NRV_B \\
8 & DA \rightarrow B & : & id,\, rn \\
9 & DA \rightarrow A & : & NRV_B
\end{array}$$

Apart from the improvements suggested in [27], the main modification to the original protocol are the addition of steps 7 to 9 for non-repudiation of business message validation. The remarks made in Section 3.4 about optional termination at step 6 apply here also. The main differences between this protocol and the light weight end user protocol are that the end users are now responsible for the verification of each other's evidence and for its long-term storage.

3.6 Fault tolerance in fair exchange

A fair exchange protocol is fault-tolerant if it ensures no loss of fairness to an honest participant even if the participants node experiences failures of the assumed type. In other words, an honest user does not suffer a loss of fairness because of his node failure. This is not the case with most of the fair exchange protocols studied in the literature, including the ones presented here. Ezhilchelvan and Shrivastava (in [9]) describe various approaches to preserving fairness in the presence of node crashes and recovery; application of these to the protocols presented here is left as a direction for future work.

4 Implementation based on Web services

In this section we present the Web services based implementation of a framework for non-repudiation protocol execution (WS-NRExchange). First we provide an overview of Web services and the standards used to support WS-NRExchange. Then we provide a high-level view of a WS-NRExchange based non-repudiable interaction. Section 4.3 describes the generic interface to protocol execution and the message schema developed that underpin WS-NRExchange. This section concludes with examples of SOAP messages exchanged by the Web services during a protocol run.

The combination of an interceptor based approach and a generic interface to protocol execution allows the infrastructure to adapt to different application requirements, including the execution of different protocols, without disturbing application-level logic. For example, the infrastructure presented here could be used to execute protocols with either on-line delivery agent or off-line delivery agent (where the DA is only involved for abnormal termination).

To place the following discussion in context, Figure 4 shows how various XML and Web service standards

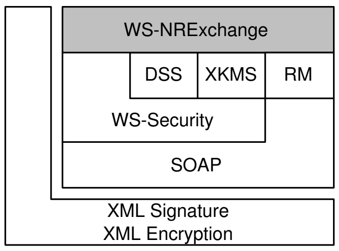

Figure 4: WS-NRExchange and Web service standards

support WS-NRExchange.

4.1 Overview of Web services and supporting standards

A Web service is a service that can be described, published, located and invoked over the Web. Web services are based on open standards and are designed to be platform-neutral. Web services typically communicate using the SOAP [10] messaging protocol. A SOAP message is an XML document with a defined body and header. The body contains the message payload and the header contains information regarding how the message should be processed. The interface that a Web service exposes, in terms of messages that can be processed and operations that can be invoked, can be described using the Web Services Description Language (WSDL) [1] .

Various Web service standards have been proposed to implement inter-operable secure systems. The WS-Security[18] standard covers the creation of self-protecting SOAP messages. WS-Security describes how to apply XML technologies, such as XML signature [8] and XML Encryption [7], to SOAP messages. XML-Signature specifies how to attach signatures, and related information, to XML documents, or parts of documents, and related material. XML-Encryption is the corresponding standard for encryption.

Digital Signature Service (DSS) and XML Key Management Specification (XKMS) are higher level services that use WS-Security. DSS specifies a service for the verification and application signatures to XML; and for trusted timestamping of signed information. XKMS is concerned with public key life-cycle management. It specifies how to register, locate, verify and revoke the digital certificates that are associated with public keys. XKMS and DSS may be offered as trusted third party services to support secure Web service interactions, thereby reducing the security infrastructure requirements of users. Organisations may also provide a sub-set of the services in-house as part of their own security infrastructure. For example, an in-house DSS service can be used to apply corporate signatures to XML messages.

A reliable messaging (RM) service specifies the message content, protocols and persistence requirements necessary for Web services to implement various forms of reliable message delivery — we require at least once delivery guarantee. Currently, there are competing standards proposed for Web service RM: WS-Reliability [5] and WS-ReliableMessaging [20], for reliable messaging. There is overlap between the two proposals and WS-NRExchange should be able to adapt to both.

Our contribution is to provide the NRExchange Web service that uses the standards outlined above.

4.2 WS-NRExchange based interaction

Figure 5 shows the interactions between the various components and services that make up our implementation. The delivery agent, A and B each provide an NRExchange Web service that manages their participation in non-repudiation protocols. Each Web service exposes the same interface for protocol execution. At A

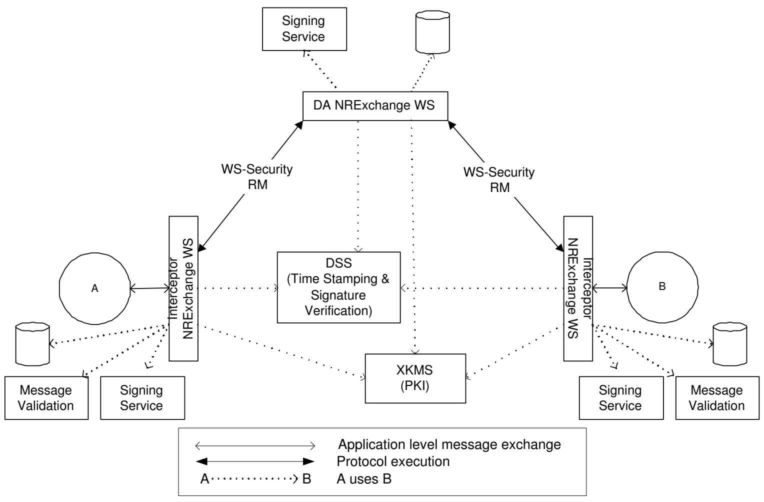

Figure 5: WS-NRExchange architecture

and B this service is deployed as an interceptor that mediates Web service interactions that require fair, non-repudiation. This interceptor may be co-located with the local application that uses it or may, for example, be part of a corporate firewall service. The NRExchange services access additional local services for signing evidence, message persistence and application-level validation. The signing service is required to apply signatures to the parts of messages that have not already been signed[1], as dictated by the protocol being executed. This service may be an implementation of DSS or some other mechanism for obtaining private keys to apply signatures as defined by WS-Security. Persistence is required to meet fault tolerance requirements and also for audit. The NRExchange services also require access to trusted timestamping services and public key management services (DSS and XKMS services provided by third parties).

As described in Section 4.3, a WSDL interface has been defined for the interaction between NRExchange services. The messages exchanged comply with the WS-Security specification. The services are written to the DSS and XKMS specifications for access to trusted timestamping, signature verification, public key life-cycle management etc.

The NRExchange Web service also provides a local interface to allow registration of application-specific listeners for message validation and other events. A message validation listener may trigger arbitrarily complex validation of a business message, as discussed in Section 2. If no validation listener is registered, then the NRExchange service assumes that a message is valid with respect to business contract. Registration of event listeners allows notification of protocol-related events. For example, an application can register to receive notification of zero or more of the acknowledgements generated by the protocols described in

[1]It is possible, for example, that the message body or documents attached to a message have been signed at the application level in which case the NRExchange service does not need to countersign.

Section 3.

4.3 Generic NRExchange interface and message schema

The following extract from the WSDL definition of an NRExchange Web service shows the operations that are exposed to other NRExchange services for protocol execution:

```
<wsdl:definitions>
  ...
  <wsdl:portType name="NRExchange">
     <!-- protocol message exchange interface -->
     <wsdl:operation name="processMessage">
       <wsdl:input message="wsnrex:ProtocolMessage"/>
     </wsdl:operation>
     <!-- abnormal termination interface -->
     <wsdl:operation name="terminate">
       <wsdl:input message="wsnrex:TerminationRequestMessage"/>
     </wsdl:operation>
     <wsdl:operation name="abort">
       <wsdl:input message="wsnrex:ProtocolStateMessage"/>
     </wsdl:operation>
     <wsdl:operation name="resolve">
       <wsdl:input message="wsnrex:ProtocolStateMessage"/>
     </wsdl:operation>
     <!-- protocol state query interface -->
     <wsdl:operation name="getProtocolState">
        <wsdl:input message="wsnrex:GetProtocolStateMessage"/>
     </wsdl:operation>
     <wsdl:operation name="setProtocolState">
       <wsdl:input message="wsnrex:ProtocolStateMessage"/>
     </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

Services that participate in a non-repudiation protocol use the `processMessage` operation to exchange protocol messages with each other. The sender provides a protocol message for the receiver to process according to a specified non-repudiation protocol. Message elements are defined in a related XML Schema and are sufficiently general and extensible to allow all interaction between NRExchange services to be conducted using the `processMessage` operation. However, for abnormal operation, and to query protocol state, a small set of convenience operations are also defined.

The `terminate` operation allows a participant to request termination of a protocol run. Termination is typically requested when timely completion of a protocol run is not possible. For example, because a party to a protocol ceases to cooperate — requiring intervention of the DA to resolve or abort the protocol run while maintaining fairness guarantees. A `terminate` request leads to execution of a sub-protocol to ensure that the appropriate non-repudiation tokens are available to all parties.

The `abort` operation is an instruction, typically invoked by the DA on one or more participant services, to abort a specified protocol run with the given protocol state. This operation will normally only be invoked during or after execution of a termination sub-protocol.

The `resolve` operation is an instruction to resolve a specified protocol run with the given protocol state. This operation is typically invoked by the DA on one or more protocol participants to effect successful resolution of a protocol run during or after execution of a termination sub-protocol.

The `getProtocolState` is a request for information concerning the known state of a protocol run as viewed by the service on which the operation is invoked. This operation may lead to execution of a state request sub-protocol to determine how much of the current state should be revealed to the invokee. The protocol state may be provided during execution of this sub-protocol or by invocation of the `setProtocolState` operation on the service that originally invoked the `getProtocolState` operation.

The SOAP binding for the NRExchange service specifies that any message submitted using the any of the above operations must contain an NRExchangeProtocol header block. The NRExchangeProtocol block is an extensible container for non-repudiation protocol data items that are defined in the NRExchange XML schema or may be defined in derivations of that schema or in schemas that are specific to a given non-repudiation protocol. The NRExchange schema specifies that any NRExchangeProtocol block must contain a ProtocolDescriptor element that completely identifies a protocol run and the message within the protocol run. The ProtocolDescriptor type is defined as follows:

```
<xsd:element name="ProtocolDescriptor" type="ProtocolDescriptorType"/>
<xsd:complexType name="ProtocolDescriptorType">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:anyURI"/>
    <xsd:element name="RunId" type="wsnrex:RunIdType"/>
    <xsd:element name="RunSequenceNumber" type="xsd:unsignedLong"
      minOccurs="0"/>
    <xsd:element ref="MessageNumber"/>
    <xsd:element name="Purpose" type="wsnrex:PurposeType" minOccurs="0"/>
    <xsd:element name="Participants"type="wsnrex:ParticipantsType"
      minOccurs="0"/>
    <xsd:element name="RelatesTo" type="wsnrex:RunIdType"
      minOccurs="0"maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:ID"/>
</xsd:complexType>
```

As shown, a ProtocolDescriptor must give the protocol name, a protocol run identifier and a message number. The protocol name URI serves to uniquely identify the protocol being executed and may also provide access to protocol documentation including schema that specialise the NRExchange schema. The RunId is a unique identifier that is normally generated from some base URI and a random digest (a hash of a random number and other associated input). The message number is a non-negative integer that corresponds to the step of the protocol being executed. For a given protocol, specific message number ranges will be reserved for the main and any related sub-protocols. Depending on the protocol being executed or the step of the protocol, the following optional items may be included in the descriptor: a run sequence number; the purpose of the protocol message (NRR, NRO etc.), the participants in the protocol; any related protocol, or sub-protocol, RunIds.

In addition to the protocol descriptor, and other information related to a given protocol, the schema specifies that the NRExchangeProtocol block may contain: at most one acknowledgements required specification; at most one receipts required specification; and at most one acknowledgement block. These elements are discussed in Section 4.4.

A protocol message will also include a WS-Security header block that contains security tokens such as: signatures over evidence; timestamps; certificate and key information; security context and access control information. The general form of an NRExchange SOAP message is:

```
<SOAP:Envelope>
  <SOAP:Header>
    ...
    <wsse:Security>
      <!-- WS-Security tokens -->
      ...
    </wsse:Security>
    <wsnrex:NRExchangeProtocol>
      <wsnrex:ProtocolDescriptor>
        ...
      </wsnrex:ProtocolDescriptor>
      <!-- other non-repudiation protocol data -->
      ...
    </wsnrex:NRExchangeProtocol>
  </SOAP:Header>
  <SOAP:Body>
     <!-- application specific content -->
     ...
  </SOAP:Body>
</SOAP:Envelope>
```

In normal operation the message body will either contain the application data to be conveyed from sender to receiver as the body of the original business message or will be empty. In the case of abnormal termination messages or protocol state messages the body will be as specified by the TerminationRequestMessage, ProtocolStateMessage or GetProtocolMessage in the NRExchange WSDL.

4.4 Example non-repudiation protocol message

This section uses a purchase order as a simple example of the type of business message discussed in Section 1. We show a subset of business message content and the most significant annotations to the message for non-repudiation protocol execution. Timestamps are omitted and so is encryption. It should be noted that the specific confidentiality requirement for message 1 — that content be kept secret from the ultimate receiver — can be met using a variety of mechanisms. In Section 3.4, the requirement is expressed by the use of public cryptography. In practice, this is just one of the mechanisms available and the actual mechanism used is not relevant to the annotation of a business message described below.

The following SOAP message represents a purchase order in an application that does not use NRExchange services.

```
<SOAP:Envelope>
  <SOAP:Header>
    ...
  </SOAP:Header>
  <SOAP:Body>
    <po:PurchaseOrder>
      <Quantity>1000000</Quantity>
      <itemID>234233</itemID>
    </po:PurchaseOrder>
    ...
  </SOAP:Body>
</SOAP:Envelope>
```

The above message is deliberately kept simple. In practice, business messages may be quite complex and include numerous header and body elements along with mixed media attachments and references to external information. Given the NRExchange infrastructure described in Section 4.2, it is possible to ensure that fair exchange can be applied to messages regardless of their content, attachments or references.

To transform the purchase order message into an NRExchange protocol message for submission to a *DA* various WS-Security and WS-NRExchange blocks must be added to the message header. There follows an overview of this transformation:

```
<SOAP:Envelope>
  <SOAP:Header>
    <!-- WS-Security header -->
    <wsse:Security SOAP:actor="..." SOAP:mustUnderstand="1"/>
      <!-- sender certificate -->
      <!-- signature on nrexchange header -->
      <ds:Signature>
          ...
          <ds:Reference URI="#wsnrex_header">...</ds:Reference>
          ...
      </ds:Signature>
      <!-- signature on body -->
      <ds:Signature>
          ...
          <ds:Reference URI="#message_body">...</ds:Reference>
          ...
      </ds:Signature>
      <!-- other security tokens -->
```

```
        </wsse:Security>
        <!-- WS-NRExchange header -->
        <wsnrex:NRExchangeProtocol SOAP:actor="..." SOAP:mustUnderstand="1"
          Id="wsnrex_header"/>
          <wsnrex:ProtocolDescriptor Name="http://.../coffey-saidha/lightuser/">
             <wsnrex:RunId>urn://.../BL8jdK12...</wsnrex:RunId>
             <wsnrex:MessageNumber>1</wsnrex:MessageNumber>
             <wsnrex:Purpose>nrex.purpose.NRO</wsnrex:Purpose>
            <wsnrex:Participants>
               <TTP>http://www.ttp.com/...</TTP>
               <Sender>http://www.purchaserr.com/...</Sender>
               <Receiver>http://www.supplier.com/...</Receiver>
             </wsnrex:Participants>
           </wsnrex:ProtocolDescriptor>
           <wsnrex:RunIdGenerator>
             <BaseURI>urn://.../</BaseURI>
             <wsnrex:RandomDigestInput>
               <ds:Transforms>...</ds:Transforms>
               <ds:DigestMethod Algorithm="..."/>
               <wsnrex:RandomNumber>eUr0TapAS04...</wsnrex:RandomNumber>
               <!-- other elements to digest as part of RunId -->
             </wsnrex:RandomDigestInput>
            </wsnrex:RunIdGenerator>
            <wsnrex:ReceiptsRequired>
              <wsnrex:ReceiptSpecification URI="#wsnrex_header"/>
              <wsnrex:ReceiptSpecification URI="#message_body"/>
                ...
            </wsnrex:ReceiptsRequired>
            ...
         </wsnrex:NRExchangeProtocol>
         ...
      </SOAP:Header>
      <SOAP:Body Id="message_body">
        <po:PurchaseOrder>
          <Quantity>1000000</Quantity>
          <itemID>234233</itemID>
        </po:PurchaseOrder>
        ...
      </SOAP:Body>
    </SOAP:Envelope>
```

The WS-Security header includes signature blocks that reference the message body and the WS-NRExchange header. These are the minimum requirements for signatures on message content. The WS-NRExchange header contains all the protocol meta-information required to identify the protocol, protocol run, message within a run, participants and message purpose (NRO in this case). The RunIdGenerator specifies how the protocol RunId was generated, including the base URI and the random number, if any, used as input[2]. The ReceiptsRequired block identifies all the message parts for which a receipt is required.

[2]If a RunId is generated from a base URI and a message digest, the message digest will be URI-encoded.

5 Related work

In our earlier work [3] we described how component middleware (such as J2EE application server) can be enhanced to support non-repudiation. This work is a significant extension to provide third party services for fair, validated, non-repudiable message delivery in the sort of asynchronous and long-running interactions that Web services are intended to support.

The Universal Postal Union has proposed the Global Electronic Postmark [24] (EPM) standard. This is a TTP service for generation, verification, time-stamping and storage of non-repudiation evidence. It does not provide support for the fair exchange of evidence.

Various companies are beginning to offer XML firewall solutions that perform various security functions such as signing, encryption and verification of cryptographic information. Notably, DataPower [6] offer XML processing in hardware that can apply cryptography to XML documents at "wire-speed". Verisign [25] offer a solution based on a server-based organisational gateway to secure SOAP-based message exchange. Earlier, Lee et al [13] proposed an interceptor-based solution that provided similar facilities. All of these approaches can offer non-repudiation of origin by applying signatures to outgoing messages. They are also capable to verifying signatures and security tokens attached to incoming messages. However, none are able to provide for fair exchange of non-repudiation evidence. The NRExchange service we have presented could make use of these solutions for its basic cryptographic and verification services.

There has been much work on fair exchange and fair non-repudiation, and on the formal verification of protocols. Kremer et al [12] summarise the state of the art and provide a useful classification of protocols according to types of fairness and the role of TTPs in protocols. There have also been contributions on the transformation of fair exchange to meet fault tolerance requirements [14, 9]. This body of work can be brought to bear on the choice of protocols that the NRExchange services use to meet application requirements.

Wichert et al [26] represents an early implementation of a non-repudiation service that used filters in CORBA to provide non-repudiable invocation on a remote object. However, there approach was asymmetric — the client provides the server with non-repudiation of origin of a request but there is no exchange to provide corresponding evidence to the client.

The FIDES [19] system provides services, including TTP services, and an associated application for fair exchange of documents. Application clients submit documents to the FIDES system for fair exchange with partners who also have a FIDES client for verifying and receipting documents received. In effect FIDES offers a standalone service for fair exchange. Such a system could be built on our NRExchange middleware. Similarly, if the FIDES system were to expose appropriate service interfaces and their protocol execution engines could be instrumented to trigger application-level validation, the NRExchange service could be implemented using FIDES services. Without such interfaces, it appears difficult to adapt the FIDES approach to arbitrary service interactions. Apart from FIDES, we know of no other work on service-based implementation of fair exchange.

With respect to application-level validation of business messages, the work of Minsky et al on Law Governed Interaction (LGI) [16] represents one of the earliest attempts to provide coordination between autonomous organisations. Trusted agents act as mediators that comply with a global policy. LGI does not address systematic non-repudiation, but does support automatic validation techniques. This is similar to work by colleagues [17] which aims to automate, as far as possible, the derivation and verification of the validation process from business contracts.

6 Conclusions and future work

In this paper we have developed an approach and a reference implementation for supporting fair non-repudiable interactions. We showed how an existing delivery agent based protocol could be augmented to render a common business message exchange non-repudiable. This involved adding extra steps for non-repudiable message validation. We then described how the protocol could be modified to facilitate a more light weight delivery agent. In the final section we described our Web services based implementation. This involved a general overview of the architecture showing how we interact with existing Web service standards and services. We then described the schema for our protocol messages and presented an example message.

Future work will be to provide end-to-end fair non-repudiation of application level request/response message exchange. Essentially, the message delivery primitive will be used to provide fair exchange of both the request and the response message along with non-repudiation of the correlation between the application level messages. This will be similar to the non-repudiable service invocation described in [3] but will apply in the Web services context and will operate for different application-level request/response semantics (for example: asynchronous, deferred synchronous and synchronous). We also intend to use the NRExchange infrastructure to provide a Web services based non-repudiable information sharing [4].

Our implementation operates a layer above the reliable messaging. However, there is duplication of effort between fair exchange and reliable messaging both in terms of acknowledgements generated and message persistence. We intend to investigate the tighter integration between the two services. Our approach would be modular and configurable. Essentially, either the fair exchange service would provide reliable messaging directly (for greater performance) or, as now, it would rely on an existing standards-based implementation to provide a reliable channel.

References

[1] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*. W3C Note, http://www.w3.org/TR/wsdl, March 2001.

[2] T. Coffey and P. Saidha, “Non-repudiation with mandatory proof of receipt,” *Computer Communications Review*, vol. 26, no. 1, 1996.

[3] N. Cook, P. Robinson, and S. Shrivastava, “Component Middleware to Support Non-repudiable Service Interactions,” in *Proc. IEEE Int. Conf. on Dependable Syst. and Networks (DSN)*, Florence, Italy, 2004.

[4] N. Cook, S. Shrivastava, and S. Wheater, “Distributed Object Middleware to Support Dependable Information Sharing between Organisations,” in *Proc. IEEE Int. Conf. on Dependable Syst. and Networks (DSN)*, Washington DC, USA, 2002.

[5] D. Bunting, J. Durand et al, *Web Services Reliable Messaging TC WS-Reliability 1.1*. OASIS Committee Working Draft, http://www.oasis-open.org/committees/wsrm/, August 2004.

[6] DataPower, *XML Security Gateway*. http://www.datapower.com/products/xs40.html, 2004.

[7] D. Eastlake, J. Reagle, T. Imamura, B. Dillaway, and E. Simon, *XML Encryption Syntax and Processing*. W3C Recommendation, http://www.w3.org/TR/xmlenc-core/, 2002.

[8] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, *XML-Signature Syntax and Processing*. W3C Recommendation, http://www.oasis-open.org/committees/wss/, 2004.

[9] P. Ezhilchelvan and S. Shrivastava, “Systematic Development of a Family of Fair Exchange Protocols,” in *Proc. 17th IFIP WG 11.3 Working Conf. on Database and Applications Security*, Colorado, USA, 2003.

[10] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. Nielsen, *SOAP Version 1.2 Part 1: Messaging Framework*. W3C, W3C Recommendation, http://www.w3.org/TR/soap12-part1/, June 2003.

[11] ISO, “Information processing systems - open systems interconnection - basic reference model - part 2: security architecture, iso 7498-1,” 1989.

[12] S. Kremer, O. Markowitch, and J. Zhou, “An Intensive Survey of Fair Non-repudiation Protocols,” *Computer Communications*, vol. 25, pp. 1601–1621, 2002.

[13] S. Lee, O. Kwon, J. Lee, C. Oh, and S. Ko, “TY*SecureWS: An Integrated Web Service Security Solution Based on Java,” *in Proc E-Commerce and Web Technologies: 4th International Conference, EC-Web, Springer LNCS 2738*, pp. 186–195, 2003.

[14] P. Liu, P. Ning, and S. Jajodia, “Avoiding Loss of Fairness Owing to Process Crashes in Fair Data Exchange Protocols,” in *Proc. IEEE Int. Conf. on Dependable Syst. and Networks (DSN)*, New York, USA, 2000.

[15] O. Markowitch, D. Gollmann, and S. Kremer, “On Fairness in Exchange Protocols,” in *Proc. 5th Int. Conf. on Information Security and Cryptology (ISISC 2002)*, Springer LNCS 2587, 2002.

[16] N. Minsky and V. Ungureanu, “Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems,” *ACM Trans. Softw. Eng. and Methodology*, vol. 9, no. 3, pp. 273–305, 2000.

[17] C. Molina-Jimenez, S. Shrivastava, E. Solaiman, and J. Warne, “Contract Representation for Run-time Monitoring and Enforcement,” in *Proc. IEEE Int. Conf. on E-Commerce (CEC)*, Newport Beach, USA, pp. 103–110, 2003.

[18] A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo, *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*. W3C Recommendation, http://www.w3.org/TR/xmldsig-core/, 2002.

[19] A. Nenadic, N. Zhang, and S. Barton, “FIDES - A Middleware E-Commerce Security Solution,” *in Proc 3rd European Conf on Inf Warfare and Security (ECIW 2004)*, 2004.

[20] R. Bilorusets, A. Bosworth et al, *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*. BEA, Microsoft, IBM and TIBCO Software, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-reliablemessaging.asp, March 2004.

[21] P. Robinson, N. Cook, and S. Shrivastava, “Implementing Fair Non-Repudiable Interactions with Web Services,” tech. rep., School of Computing Science, Univ. Newcastle, 2004.

[22] Rosettanet. http://www.rosettanet.org/, 2004.

[23] B. Schneier, *Applied Cryptography*. John Wiley and Sons, 2nd ed., 1996.

[24] UPU, *Global EPM Non-repudiation Service Definition and the Electronic Postmark 1.1*. Universal Postal Union, http://www.globalepost.com/prodinfo.htm, Oct 2002.

[25] Verisign, *Simplifying Application and Web Services Security: Verisign Trust Gateway*. Verisign White Paper, http://www.xmltrustcenter.org/index.htm, 2004.

[26] M. Wichert, D. Ingham, and S. Caughey, “Non-repudiation Evidence Generation for CORBA using XML,” in *Proc. IEEE Annual Comp. Security Applications Conf.*, Phoenix, USA, 1999.

[27] J. Zhou and D. Gollmann, “Evidence and non-repudiation,” *J. Network and Comp. Applications*, vol. 20, no. 3, pp. 267–281, 1997.