School of Computing Science,
University of Newcastle upon Tyne

Implementing Fair Non-repudiable Interactions with Web Services

Paul Robinson, Nick Cook and Santosh Shrivastava

Technical Report Series

CS-TR-913

June 2005

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

September 13, 2005

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 to enable 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: Inter-enterprise collaboration and virtual enterprises; Middleware standards and systems; Enterprise computing; Security; Non-repudiation; Fair exchange; Web services

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 [1] 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

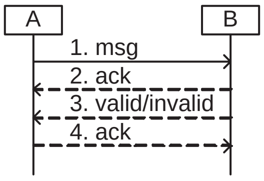

Figure 1: Business message delivery with acknowledgements

message and associated acknowledgements in such 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. Finally, the validation message is acknowledged in return.

Validation of the original business message (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 and associated acknowledgements both fair and non-repudiable. For example, there is no systematic support to prevent a customer from denying that they submitted a purchase order and at the same time to prevent 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. The paper describes two fair exchange protocols that have been chosen for initial implementation. However, our framework is sufficiently flexible to adapt to different application requirements and, in particular, to execute different protocols.

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 the initial fair exchange protocols chosen. Their implementation is based on a trusted third party (TTP) delivery agent. 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 service technologies. Section 5 discusses related work. Section 6 concludes the paper.

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 event. It is one of the key properties of secure systems as defined in [2]. 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.

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 [3] 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.”.

Most practical fair non-repudiation protocols require the involvement of a TTP to some extent. The level of intervention can vary depending on the protocol and the requirements of the end users. Kremer et al [4] identify three main types of TTP: inline, online and offline. An inline TTP (sometimes called a delivery agent) is involved in transmission of each protocol message. An online TTP is involved in each session of a protocol but not in every message transmission. An offline TTP is involved in a protocol only in case of incorrect behaviour of a dishonest entity or in case of 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 introduces a delivery agent (*DA*),

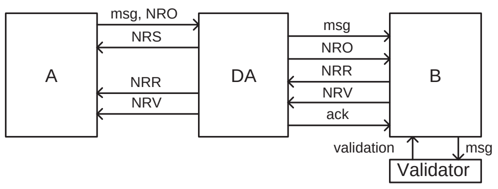

Figure 2: Executing a business interaction through a delivery agent

or inline TTP, to the interaction shown in Figure 1. Four types of evidence are generated: (i) non-repudiation of origin (NRO) that *msg* originated at *A;* (ii) non-repudiation of submission (NRS) to the *DA* of *msg* and NRO; (iii) non-repudiation of receipt (NRR) of *msg* by *B*; (iv) non-repudiation of validation (*NRV*) — valid or otherwise, as determined by validation of *msg* by *B*. 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*). Then *DA* provides *NRR* to *A* — equivalent to message 2 in Figure 1. 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* and provides acknowledgement of *NRV* to *B*. Note, the exact sequence of the exchange will be dictated by the actual protocol used and should not be inferred from Figure 2.

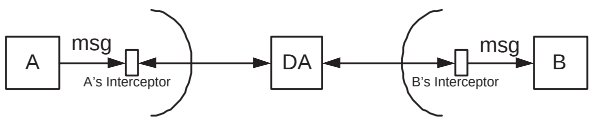

Figure 3: Interceptor approach

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 TTP 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 TTP — *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. 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 place is as shown in Figure 2. 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.

From *DA's* point of view there is no distinction between A and *A's* interceptor, or B and B's interceptor (similarly, for A with respect to B and B with respect to A). Therefore, we only distinguish between an end user and their interceptor when necessary — for example, when discussing the implementation.

3 Delivery-agent based fair exchange

This section discusses the first two fair exchange protocols we have chosen to implement. 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. In addition to a main protocol there are sub-protocols to support exception handling. Since the delivery agent is trusted, these sub-protocols can be used for timely, and fair, termination of a protocol despite non-cooperation of one of the other participants in the main protocol.

3.1 Protocol overview

Coffey and Saidha developed a fair non-repudiation protocol utilising an in-line TTP [5]. This was later improved by Zhou and Gollman in [6]. The protocols presented here are derived from the improved protocol. Both protocols include modifications: (i) to provide A with proof of submission to *DA*; (ii) to support the non-repudiation of the extra validation message described in Section 2 (adding 3 protocol steps); and (iii) to support exception handling through execution of sub-protocols. The first protocol is a further modification to support light-weight end users. The second protocol uses a light-weight delivery agent and is closer to the original Coffey and Saidha protocol.

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 only need to verify evidence produced by the delivery agent and, in consequence, only need access to the information necessary to verify a known third party's signature and credentials. This means that the end users may only require credential management at the level used by the typical Web browser (as opposed to access to the more comprehensive public key infrastructure required to verify evidence from all parties with whom they may interact). End user logging requirements are also reduced. Minimally, they only need to maintain the information necessary to link an interaction to the evidence held by the delivery agent (such as a protocol run identifier).

In the second protocol, the responsibilities for evidence verification and long-term storage are transferred to the end users, reducing the delivery agent responsibilities. In this case, the delivery agent may discard information after termination of a protocol run and need only sign the non-repudiation of submission evidence (as opposed to all non-repudiation tokens seen by the other participants). A lighter-weight delivery agent inevitably leads to increased direct end-user reliance on supporting public key infrastructure and to increased end-user responsibility for long-term storage of evidence.

3.2 Assumptions

We make the standard perfect cryptography assumptions [7], including: (i) that message digests (secure hashes) 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 of 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. 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 time-stamped by a TTP time-stamping service [6]. For brevity, time-stamping is not shown in protocol descriptions.

The following assumptions are made with respect to well-behaved parties to a non-repudiable interaction:

- The communication channel between well-behaved parties provides eventual message delivery (there is a bounded number of temporary network and computer related failures).
- 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).
- 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* will ensure protocol resolution in favour of well-behaved parties.

3.3 Notation

In the protocols, participant A, wishes to send a business message, msg, to participant B. All communications between A and B take place through delivery agent DA. Table 1 provides the notation used for basic protocol elements. To

Notation	Description
rn	secure pseudo-random number
$h(x)$	secure hash of x
id	unique protocol run identifier
i, j	concatenation of items i and j
$P \rightarrow Q : m$	P sends m to Q
$sig_P(x)$	P's digital signature on x
$enc_P(x)$	encryption of x with P's public key
$VAL \mid INV$	signifies msg validity or invalidity
$NVAL$	signifies msg not validated

Table 1: **Notation for protocol elements**

simplify protocol descriptions, and without loss of generality, it is assumed that the signature scheme is recoverable. That is, if necessary, x (and any items that are concatenated to construct x) may be recovered from $sig_P(x)$[1]. To allow verification of rn as a protocol authenticator, it is also assumed that id contains $h(rn)$. Table 2 defines the non-repudiation evidence exchanged during

Non-repudiation token	Description
$NRS_{DA} = sig_{DA}(id, A, B)$	DA's NRS of initial protocol message
$NRO_A = sig_A(id, A, B, h(msg))$	A's NRO of msg
$NRR_B = sig_B(id, A, B, h(msg))$	B's NRR of msg
$NRV_B = sig_B(id, VAL \mid INV)$	B's NRV of msg
$NRO_{DA} = sig_{DA}(id, A, B, msg)$	DA's NRO of msg for B
$NRR_{DA} = NRO_{DA}$	DA's NRR of msg for A
$NRV_{DA} = sig_{DA}(id, VAL \mid INV)$	DA's NRV of msg
$NRV'_{DA} = sig_{DA}(id, NVAL)$	DA's substitute NRV of msg

Table 2: **Definition of non-repudiation tokens**

a protocol run. DA associates a termination state with each exchange. The state is $SUCCEEDED$ if the exchange is successfully completed and $ABORTED$ if the exchange is cancelled.

[1]If the signature scheme is non-recoverable, then any necessary items are sent with the associated signature.

3.4 Fair exchange for light-weight end users

This section first discusses normal execution to successful completion of the main protocol for light-weight end users. We then present abort and resolve sub-protocols for exception handling.

Normal protocol execution

Normal execution of the main protocol is shown below, followed by a commentary on each step.

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

Step 1: A sends a business message (msg), a secure pseudo-random number (rn) and its non-repudiation of origin token (NRO_A) to DA. At this step, if DA finds that the id included in NRO_A is not unique, an appropriate response will be generated to prompt A to restart the protocol with a newly generated id. Otherwise, the protocol will proceed to step 2. The rn provided by A is used in step 8 as the acknowledgement of receipt of *B's NRV* token. All items are encrypted to guarantee that B does not obtain the items before providing non-repudiation of receipt.

Step 2: DA provides proof of submission to A to signal willingness to proceed with protocol execution. This step may be executed in parallel with step 3.

Step 3: To enable B to construct NRR_B: DA sends the id, the participant identifiers A and B (recovered from NRO_A) and a hash of msg to B.

Step 4: B responds with NRR_B. It is safe for B to send the receipt to DA before obtaining msg because DA, as TTP, can and will provide msg in return. NRR_B is encrypted to guarantee that A can only obtain the receipt if the exchange runs to some form of successful completion.

Step 5: DA sends NRR_{DA} to A. This is DA's receipt for msg and assurance that it has received and verified NRR_B. This step may be executed in parallel with step 6.

Step 6: DA sends msg and associated NRO_{DA} to B.

Step 7: B performs application-level validation of msg. The outcome of this validation is signed along with id to form NRV_B that is sent to DA.

Step 8: rn, hitherto known only to A and DA, is sent to B as acknowledgement of receipt for NRV_B. This step may be executed in parallel with step 9.

Step 9: DA sends NRV_{DA} to A — non-repudiation of the outcome of validation of msg.

At the end of execution of the main protocol, A has acquired the following evidence: NRS_{DA}, NRR_{DA} and NRV_{DA} — non-repudiation of submission, receipt and validation of msg. In return, B has acquired: msg, NRO_{DA} and rn — the business message with non-repudiation of origin and acknowledgement of validation of msg. DA has the complete set of evidence, including: NRO_A, NRR_B and NRV_B.

Fairness is guaranteed because DA controls the release of the evidence to A and B. Furthermore, DA's signature on the evidence provided to A and B: (i) serves as a guarantee that DA has seen, verified and will store the evidence for future reference; and (ii) reduces the verification work of A and B to that of verifying the signature and associated credentials of a well-known TTP.

If B does not wish to perform application-level validation of msg, then the protocol can terminate at step 6. In this case, at step 4, B sends DA both NRR_B and a default NRV_B token that confirms the validity of msg. At step 5, DA sends A both NRR_{DA} and NRV_{DA}. At step 6, DA sends rn to B with msg and NRO_{DA}.

On successful completion of the main protocol, DA sets termination state to $SUCCEEDED$.

Exception handling

In exceptional circumstances A or B may request that DA terminate the main protocol before completion. Such requests typically occur because A or B is concerned about the liveness of protocol execution (whether as a result of the non-cooperation of a participant or extraneous factors such as network delays). There are two types of request:

abort: where the requesting party wishes to terminate the protocol as if no exchange had taken place. That is, neither A nor B receive any useful information about the exchange.

resolve: where the requesting party seeks DA's assistance in securing normal termination. That is, all expected items (or their equivalent) are available to well-behaved parties.

These requests are, in effect, the statement of a preference for how the exchange should complete. Irrespective of the type of request, it is the responsibility of DA to ensure that fairness guarantees hold for all honest parties. Depending on the progress of the main protocol and whether the exchange termination state has already been set, DA must determine whether the exchange should terminate in $ABORTED$ state (no exchange has taken place) or $SUCCEEDED$ state (exchange has taken place). An exchange can terminate in $SUCCEEDED$ state if and only if: (i) A is entitled to NRS_{DA}, NRR_{DA} and NRV_{DA} (or an equivalent substitution); **and** (ii) B is entitled to msg, rn and NRO_{DA}.

DA is empowered to issue the substitute non-repudiation of validation, NRV'_{DA}, in place of NRV_{DA}. NRV'_{DA} is DA's signed confirmation that B has not validated msg. Once DA has produced NRV'_{DA} no validation of msg by B will be accepted. NRV'_{DA} is equivalent to invalidation of msg with the supplementary information that B did not cooperate in the decision. At first sight, this places A at a disadvantage, since B can receive *msg* and simply decide not to cooperate in its validation. However: (i) in any case, B may autonomously decide that a message is not valid (and such invalidation may be subject to *extra-protocol* dispute resolution); and (ii) A obtains evidence of B's lack of participation in validation. Thus, in terms of evidence exchanged, the substitution of NRV'_{DA} for NRV_{DA} is fair.

From the above, we observe that fairness is guaranteed to both A and B if:

1. the main protocol completes normally; or
2. B chooses not to engage in the main protocol by not responding to step 3 (up to and including step 3, A has only received NRS_{DA} and B has no useful information about *msg*); or
3. the exchange is aborted when the main protocol has progressed no further than step 4 (at step 4, B sends NRR_B to DA but the protocol can still be aborted because A does not have NRR_{DA} and B is yet to receive *msg* or NRO_{DA}); or
4. the exchange is completed successfully after execution of step 4 (at step 4, DA has all the information necessary to complete the exchange; after execution of step 5, DA must guarantee that all expected items are available to both A and B).

The pivotal point in the main protocol is step 4. Before step 4, DA can only respond to either type of termination request by aborting the exchange. Upon execution of step 4, DA has *rn*, *msg*, NRO_A and NRR_B but is yet to complete the release of information to either A or B. Thus, at this point, they can satisfy whichever type of termination request they receive first. Once DA releases critical information in step 5, they must respond to a termination request by successfully resolving the exchange.

A request from an end user, $U \in \{A, B\}$, to DA to abort an exchange results in execution of the following abort sub-protocol:

```
1    $U \rightarrow DA : sig_U(sid, ABORT, id)$
     if $ABORTED$
     or (not $SUCCEEDED$ and $lastStep < 5$) then:
2.1  $DA \rightarrow U : sig_{DA}(sid, ABORTED, id)$
     else if $lastStep < 7$ then:
2.2  $DA \rightarrow U : SUCC_{DA}, res_U, NRV'_{DA}$
     else:
2.3  $DA \rightarrow U : SUCC_{DA}, res_U, NRV_{DA}$
```

where:

sid	$=$	unique sub-protocol identifier
$SUCC_{DA}$	$=$	$sig_{DA}(sid, SUCCEEDED)$
res_A	$=$	NRS_{DA}, NRR_{DA}
res_B	$=$	rn, msg, NRO_{DA}

In step 1, U submits a signed request to abort the exchange identified by *id*. DA then checks the state of the exchange. If termination state is not already $SUCCEEDED$ and the main protocol has not progressed beyond step 4 ($lastStep < 5$), then DA sets termination state to $ABORTED$. DA provides U with non-repudiation of aborted exchange in step 2.1. If the exchange cannot be aborted, DA checks whether the main protocol has progressed beyond step 6. If not, then step 2.2 above is executed to complete a successful exchange with substitute NRV'_{DA}. Otherwise step 2.3 is executed to complete successful exchange of the evidence that would have been provided during normal execution. As shown above, res_U is the resolution evidence provided to A or B, as appropriate. If either step 2.2 or 2.3 is executed, DA sets termination state to $SUCCEEDED$.

The corresponding resolve sub-protocol is:

$$
\begin{array}{lllll}
1 & U & \rightarrow & DA & : \; sig_U\,(sid,\, RESOLVE,\, id) \\
 & \multicolumn{4}{l}{\text{if } ABORTED \text{ or } lastStep < 4 \text{ then:}} \\
2.1 & DA & \rightarrow & U & : \; sig_{DA}\,(sid,\, ABORTED,\, id) \\
 & \multicolumn{4}{l}{\text{else if } lastStep < 7 \text{ then:}} \\
2.2 & DA & \rightarrow & U & : \; SUCC_{DA},\, res_U,\, NRV'_{DA} \\
 & & & & \\
2.3 & DA & \rightarrow & U & : \; SUCC_{DA},\, res_U,\, NRV_{DA}
\end{array}
$$

Apart from the signed request that initiates the resolve sub-protocol, the only significant difference to the abort sub-protocol is that execution of step 2.1 is triggered if either termination state is $ABORTED$ or the main protocol has not progressed beyond step 3.

Once the termination state of an exchange has been set (whether after execution of the main protocol or one of the above sub-protocols), DA will forever respond in the same way to any subsequent request to abort or resolve the identified exchange (with appropriate abort token or resolution evidence). DA also responds in the same way to any subsequent message of the main protocol. That is, once the termination state has been set, DA suspends the main protocol at $lastStep$.

Termination may also be triggered by the *a priori* indication of deadlines for the acknowledgements provided in the main protocol (*NRS, NRR* and *NRV).* In this case, in step 1 of the main protocol A can indicate deadline(s) for delivery that they wish to be observed (on a best effort basis). During protocol execution, DA determines locally whether a delivery deadline is achievable. If not, DA will pro-actively terminate the exchange and issue appropriate abort or resolve tokens to A and B depending on the state of the main protocol at the time of termination.

3.5 Fair exchange with light-weight delivery agent

Normal protocol execution

The protocol for fair exchange with light-weight DA is:

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

This protocol is closer to the Coffey-Saidha protocol with the addition of steps 7 to 9 for NRV of msg. The difference between this protocol and the light-weight end user protocol is that A and B are now responsible for verification of each other's evidence and for its long-term storage. Thus, in steps 5, 6 and 9, DA relays the tokens provided by A and B rather than generating new signed tokens.

Exception handling

The abort and resolve sub-protocols are basically as defined in Section 3.4 except that the resolution evidence provided in steps 2.2 and 2.3 is now:

$$\begin{array}{lcl}
res_A & = & NRS_{DA},\, NRR_B \\
res_B & = & rn,\, msg,\, NRO_A
\end{array}$$

For successful termination before step 7, the DA provides NRV'_{DA} (as described in Section 3.4). For successful termination after step 6, DA provides NRV_B (as opposed to NRV_{DA}). The abort token is identical to that defined in Section 3.4.

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 participant's node experiences failures of the assumed type. In other words, an honest user does not suffer a loss of fairness because of their 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 [8] 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 associated message schema that underpin WS-NRExchange.

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 inline, online or offline TTP.

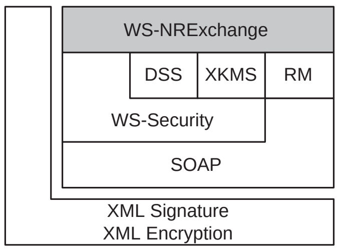

Figure 4: WS-NRExchange and Web service standards

To place the following discussion in context, Figure 4 shows how various XML and Web service standards support WS-NRExchange.

4.1 Overview of Web services and supporting standards

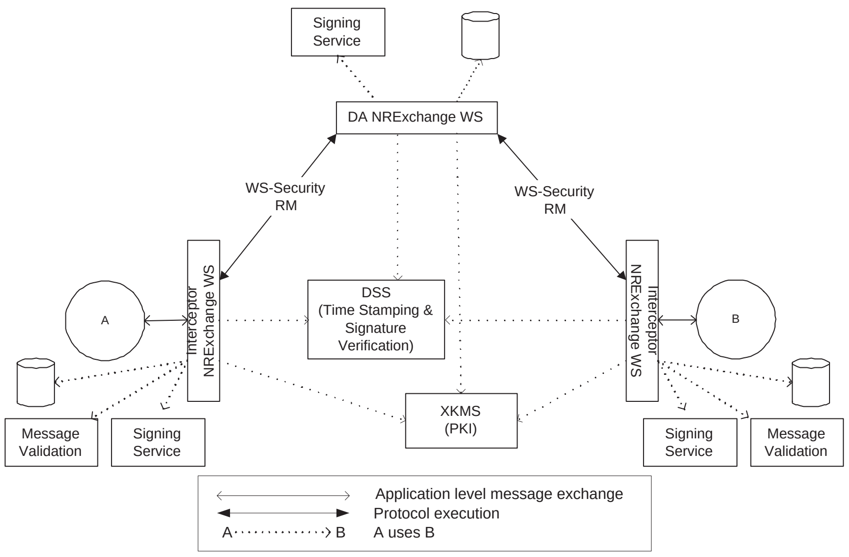

Figure 5: WS-NRExchange architecture

A Web service can be described, published, located and invoked over the Web. Web services are based on open standards and are designed to interoperate independent of implementation platform. Web services typically communicate using the SOAP messaging protocol [9]. A SOAP message is an XML document that comprises a message envelope with a header and a body. The header contains message processing information. The body contains the application payload. 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) [10] .

The WS-Security standard [11] covers the creation of self-protecting messages for Web services. WS-Security applies XML technologies, such as XML Signature [12] and XML Encryption [13], 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 specifications that use WS-Security. DSS specifies a service for the verification and the application of signatures to XML; and for trusted time-stamping of signed information. XKMS concerns public key lifecycle management. It specifies how to register, locate, verify and revoke the digital credentials that are associated with public keys. XKMS and DSS may be offered as TTP 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.

Reliable messaging (RM) 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 message delivery between well-behaved parties. Currently, there are competing standards proposed for Web service reliable messaging: WS-Reliability [14] and WS-ReliableMessaging [15]. There is overlap between the two proposals and WS-NRExchange can 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. DA, 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 and B this service is deployed as an interceptor that mediates Web service interactions that require 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 end-users, A and B, may themselves be Web services or Web service clients or both. 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[2] (as dictated by the protocol being executed). This service may be an implementation of DSS or some other mechanism for obtaining private keys

[2]It is possible for signatures to have been applied at the application level, in which case the NRExchange service does not need to countersign.

to apply signatures as defined by WS-Security. Persistence is required to meet fault tolerance requirements and also for audit. The NRExchange services also access trusted time-stamping services and public key management services (for example, DSS and XKMS services provided by third parties). For protocols that use an inline TTP, trusted time-stamps may optionally be applied by the *DA* Web service.

As described in Section 4.3, a WSDL interface has been defined for the interaction between NRExchange services. The SOAP messages exchanged comply with the WS-Security specification.

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. If no validation listener is registered, then the NRExchange service assumes that a message is valid with respect to business contract. Messages that are found to be invalid with respect to contract are logged but are not passed to the target application for processing. 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 Section 3.

4.3 Generic NRExchange interface and message schema

Figure 6 is an extract of the WSDL of an NRExchange Web service that shows the operations that are exposed to other NRExchange services for protocol execution.

```
<wsdl:portType name="NRExchange">
 <operation name="processMessage">
  <input message="ProtocolMessage"/>
 </operation>
 <operation name="processRequest">
  <input message="ProtocolMessage"/>
  <output message="ProtocolMessage"/>
 </operation>
 <operation name="abort">
  <input message="ProtocolStateMessage"/>
 </operation>
 <operation name="resolve">
  <input message="ProtocolStateMessage"/>
 </operation>
 <wsdl:operation name="getProtocolState">
  <input message="GetProtocolStateMessage"/>
 </operation>
 <wsdl:operation name="setProtocolState">
  <input message="ProtocolStateMessage"/>
 </operation>
</wsdl:portType>
```

Figure 6: Extract of NRExchange WSDL

NRExchange services use the `processMessage` operation to exchange non-repudiation 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. The schema is sufficiently general and extensible for the `processMessage` operation to be used to execute any protocol that participant services support. The `processRequest` convenience operation allows send and receipt of protocol messages as request/response pairs.

The `abort` and `resolve` operations are for pro-active termination (see Section 3.4). These operations are typically used by a *DA* to inform another participant service that an identified exchange has been aborted or resolved with the given termination state. Invocation of these operations may result in execution of a new sub-protocol using the protocol execution operations.

The `getProtocolState` operation 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 two types of message:

1. protocol messages that are exchanged during execution of a main protocol or of related sub-protocols using `processMessage` and, optionally, `processRequest`; and

2. protocol state (housekeeping) messages that convey information about the state of an identified protocol run or request update about protocol state (exchanged using `abort`, `resolve`, `getProtocolState` and `setProtocolState`).

Both types of message use a WS-Security header to carry security tokens such as: signatures over evidence; time-stamps; credential and key information; security context and access control information.

```
<soapenv:Envelope>
 <soapenv:Header>
  <wsse:Security />
  <nrex:NRExchangeProtocol name runId
        messageNumber purpose?  runSequence?  id?>
   <nrex:Acknowledgement />*
   <nrex:AcknowledgementsRequired />?
   <nrex:Participant />*
   <nrex:Receipt />*
   <nrex:ReceiptsRequired />?
   <nrex:RelatedRun />*
   <nrex:RandomDigest />*
   (<nrex:RunIdGenerator runId baseURI?>
     (<nrex:RandomDigest algorithm value>
       <RandomNumber />?
       <InputURI />*
     </nrex:RandomDigest>)?
    </nrex:RunIdGenerator>)?
    </nrex:NRExchangeProtocol>
  </soapenv:Header>
  <soapenv:Body />?
</soapenv:Envelope>
Key:
?  = 0 or 1 occurrences
* = 0 or n occurrences
```

Figure 7: General form of ProtocolMessage

As shown in Figure 7, protocol messages must have a NRExchangeProtocol header. This 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 header must have protocol name, runId and messageNumber attributes. The protocol name is a URI that serves to uniquely identify the protocol, or sub-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 secure pseudo-random number and other associated input). The inputs to runId generation can be specified using a RunIdGenerator element. The messageNumber is a positive, non-zero double value that corresponds to the step of the protocol being executed. Depending on the protocol being executed or the step of the protocol, the following optional items may be included in the NRExchangeProtocol header: a run sequence number; the purpose of the protocol message (NRR, NRO etc.); the participants in the protocol; the runIds of any related protocol or sub-protocol runs; and information related to acknowledgements and receipts required. The message body, if present, contains the application data originally intended to be conveyed from sender to receiver as the body of the business message. Section 4.4 provides examples of messages exchanged during execution of the protocol described in Section 3.4.

Housekeeping messages do not have an NRExchangeProtocol header. They carry protocol state or state request information intended for another NRExchange service in the message body. The general form of a protocol state mes-

```
<soapenv:Envelope>
 <soapenv:Header>
  <wsse:Security />
 </soapenv:Header>
 <soapenv:Body>
  <nrex:ProtocolState name runId />
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 8: General form of ProtocolStateMessage

sage is shown in Figure 8. In addition to identifying the protocol and run to which a message relates, a ProtocolState element may include information such as: the protocol run status; if terminated, the termination state; and the message numbers of any messages seen by the recipient. Protocol messages may be provided as attachments to a protocol state message. Figure 9 presents the

```
<soapenv:Envelope>
 <soapenv:Header>
  <wsse:Security />
 </soapenv:Header>
 <soapenv:Body>
  <nrex:ProtocolStateRequest name runId />
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 9: General form of GetProtocolStateMessage

general form of a message requesting information on the state of a protocol run. The ProtocolStateRequest element again identifies the protocol name and run to which it relates. In addition, the element may identify messages seen by the sender and may include a list of the protocol messages that the sender wishes to receive as attachments to any response. Housekeeping messages may result in execution of a sub-protocol to confirm a participant's entitlement to information. Also, security tokens included in the WS-Security header may serve to assert entitlement to information.

4.4 Example protocol messages

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 1.0 — 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, which is just one of the mechanisms available. The actual mechanism used is not relevant to the annotation of a business message described here.

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

Figure 10: Example SOAP purchase order

Figure 10 represents a purchase order message in an application without non-repudiation. The 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 WS-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.

```
<soap:Envelope>
 <soap:Header>
 <!-- WS-Security header -->
  <wsse:Security soap:actor="..." soap:mustUnderstand="1">
   <!-- sender certificate -->
   <wsse:BinaryToken ValueType="wsse:X509v3">
    <!-- token value -->
   </wsse:BinaryToken>
   <ds:Signature>
    <ds:SignedInfo>
     <!-- reference to nrex_header -->
     <ds:Reference URI="#nrex_header">
      <ds:Transforms>
       <!-- any transforms -->
      </ds:Transforms>
     <ds:DigestMethod Algorithm="AlgorithmName"/>
      <ds:DigestValue>FallRGr8/...</ds:DigestValue>
     </ds:Reference>
     <!-- reference to body -->
     <ds:Reference URI="#message_body">
      <ds:Transforms>...</ds:Transforms>
      <ds:DigestMethod Algorithm="..."/>
      <ds:DigestValue>tribUt4//Peel...</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>lsPiZkngCb4uDi/BsPear...</ds:SignatureValue>
   </ds:Signature>
   <!-- other security tokens -->
  </wsse:Security>
  <!-- WS-NRExchange header -->
  <nrex:NRExchangeProtocol
        xmlns:nrex="http://www.cs.ncl.ac.uk/ws/nrex/v1"
        name="http://www.cs.ncl.ac.uk/ws/nrex/protocols/coffsaid-lwuser/main"
        runId="BL8jdK12.."
        messageNumber="1.0" purpose="nrex.purpose.NRO"
        soap:actor="..." soap:mustUnderstand="1" id="nrex_header">
   <nrex:Participant role="nrex.role.TTP">
    http://www.ttp.com/
   </nrex:Participant>
   <nrex:Participant role="nrex.role.SENDER">
    http://www.purchaser.com/
   </nrex:Participant>
   <nrex:Participant role="nrex.role.RECEIVER">
    http://www.supplier.com/
   </nrex:Participant>
   <nrex:RunIdGenerator runId="BL8jdK12..">
   <nrex:RandomDigest
         algorithm="http://www.w3.org/2000/09/xmldsig#sha-256"
         value="BL8jdK12..">
    <nrex:RandomNumber>eUr0TapAS04...</nrex:RandomNumber>
   </nrex:RandomDigest>
   </nrex:RunIdGenerator>
   <nrex:ReceiptsRequired>
    <nrex:ReceiptSpecification URI="#nrex_header"/>
    <nrex:ReceiptSpecification URI="#message_body"/>
   </nrex:ReceiptsRequired>
   <nrex:AcknowledgmentsRequired>
    <nrex:AckSpecification type="nr.ack.TTP"/>
    <nrex:AckSpecification type="nr.ack.RECEIVER"/>
    <nrex:AckSpecification type="nr.ack.VALIDITOR"
          expires="2005-02-28T17:00:00Z"/>
    </nrex:AcknowledgmentsRequired>
   </nrex:NRExchangeProtocol>
   <!-- other soap header elements -->
  </soap:Header>
 <soap:Body Id="message_body">
  <po:PurchaseOrder>
   <Quantity>1000000</Quantity>
   <itemID>234233</itemID>
  </po:PurchaseOrder>
 </soap:Body>
</soap:Envelope>
```

Figure 11: Example of Coffey-Saidha protocol message 1.0

As shown in Figure 11, a WS-Security header and a NRExchangeProtocol header are added to the purchase order message to generate a protocol message suitable for submission to a *DA*. The WS-Security header includes signature blocks that reference the message body (the application-level data) and the NRExchangeProtocol header (the protocol meta-information). The meta-information identifies the protocol name, protocol run, message with the run, message purpose (NRO in this case), participants etc. The RunIdGenerator block specifies how the protocol runId was generated (including any base URI and random number used as input). The ReceiptsRequired block identifies all the message parts for which a receipt is required.

On receipt of message 1.0, the *DA* responds with message 2.0 shown in Figure 12. As with message 1.0, message 2.0 includes a WS-Security header and a NRExchangeProtocol header. The WS-Security header includes a signature over the whole of the NRExchangeProtocol header. The NRExchangeProtocol header again identifies the purpose of the message (non-repudiation of submission) and includes an acknowledgement and a receipt. Both acknowledgement and receipt reference message 1.0. In addition, the receipt includes a digest of the NRExchangeProtocol header from message 1.0 in order to cryptographically link both messages.

```
<soap:Envelope>
 <soap:Header>
  <!-- WS-Security header -->
  <wsse:Security soap:actor="..." soap:mustUnderstand="1">
   <!-- sender certificate -->
   <wsse:BinaryToken ValueType="wsse:X509v3">
    <!-- token value -->
   </wsse:BinaryToken>
   <!-- signature on nrexchange header -->
   <ds:Signature>
    <ds:SignedInfo>
     <!-- xml signature info -->
      <ds:Reference URI="#nrex_header">
      <ds:Transforms>
       <!-- any transforms -->
      </ds:Transforms>
      <ds:DigestMethod Algorithm="AlgorithmName"/>
      <ds:DigestValue>888TUpMM...</ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>lsPiZkngCb4uDi/BsPear...</ds:SignatureValue>
   </ds:Signature>
   <!-- other security tokens -->
  </wsse:Security>
  <!-- WS-NRExchange header -->
  <nrex:NRExchangeProtocol
        xmlns:nrex="http://www.cs.ncl.ac.uk/ws/nrex/v1"
        name="http://www.cs.ncl.ac.uk/ws/nrex/protocols/coffsaid-lwuser/main"
        runId="BL8jdK12.."
        messageNumber="2.0" purpose="nrex.purpose.NRS"
        soap:actor="..." soap:mustUnderstand="1" id="nrex_header">
   <nrex:Acknowledgement Type="nrex.ack.TTP" ForMessage="1.0"/>
   <nrex:Participant role="nrex.role.TTP">
    http://www.ttp.com/
   </nrex:Participant>
   <nrex:Participant role="nrex.role.SENDER">
    http://www.purchaser.com/
   </nrex:Participant>
   <nrex:Participant role="nrex.role.RECEIVER">
    http://www.supplier.com/
   </nrex:Participant>
   <nrex:Receipt runId="BL8jdK12..." forMessage="1.0">
    <ds:Reference URI="#nrex_header">
    <ds:Transforms>...</ds:Transforms>
     <ds:DigestMethod Algorithm="..."/>
     <ds:DigestValue>FallRGr8/...</ds:DigestValue>
    </ds:Reference>
   </nrex:Receipt>
  </nrex:NRExchangeProtocol>
 <!-- other soap header elements -->
 </soap:Header>
 <soap:Body/>
</soap:Envelope>
```

Figure 12: Example of Coffey-Saidha protocol message 2.0

5 Related work

In our earlier work [16] we described how component middleware (such as J2EE application server) can be enhanced to support non-repudiation. The work presented in this paper is a significant extension to provide 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 [17] (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.

Wichert et al [18] represents an early implementation of a non-repudiation service that uses filters in CORBA to provide non-repudiable invocation on a remote object. However, their 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.

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 [19] offer XML processing in hardware that can apply cryptography to XML documents at "wire-speed". Verisign [20] offer an organisational gateway service to secure SOAP-based message exchange. Earlier, Lee et al [21] 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 of verifying signatures and security tokens attached to incoming messages. They are essentially providing a Wichert-type service for SOAP messaging. They do not provide systematic non-repudiation of receipt and there is no support for fair exchange. Neither is there support for systematic binding of non-repudiation evidence to application-level message validation. 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 [4] 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 [22, 8]. This body of work can be brought to bear on the choice of protocols that the NRExchange services use to meet application requirements.

The FIDES system [23] 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 if 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) [24] 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 [25] that 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 to support 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 the addition of extra steps for message validation. We then described how the protocol could be modified to facilitate a more light-weight delivery agent. In Section 4 we described our Web services based implementation. This involved a general overview of the architecture showing how our services interact with existing Web service standards and services. We then described our Web services interface to protocol execution.

Future work will 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 [16] but will apply in the Web services context and will operate for different request/response semantics (for example: asynchronous, deferred synchronous and synchronous). We also intend to use the NRExchange infrastructure to provide Web services based non-repudiable information sharing [26].

Our implementation operates a layer above 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 tighter integration of the two services. Our approach will be modular and configurable. Essentially, either the fair exchange service will provide reliable messaging directly (for greater performance) or, as now, it will rely on an existing standards-based implementation to provide a reliable channel.

We also intend to investigate systematic support for protocol development from verification and modelling through a derived implementation to deployment as part of the infrastructure described in this paper.

Acknowledgements

This work is part-funded by the UK EPSRC under e-Science Pilot Project: "GOLD (Grid-based Information Models to Support the Rapid Innovation of High Value Added Chemicals)" and grant GR/S63199: "Trusted Coordination in Dynamic Virtual Organisations"; and by the European Union under Project IST-2001-34069: "TAPAS (Trusted and QoS-Aware Provision of Application Services)".

References

[1] RosettaNet, *eBusiness Standards for the Global Supply Chain.* RosettaNet, http://www.rosettanet.org/RosettaNet/, 2005.

[2] ISO, *Information Processing Systems - Open Systems Interconnection - Security Frameworks for Open Systems - Basic Reference Model - Part 2: Security Architecture.* ISO 7498-2, 1989.

[3] 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.

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

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

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

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

[8] 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.

[9] 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.

[10] 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.

[11] A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo, *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004).* OASIS Standard 200401, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, 2004.

[12] D. Eastlake, J. Reagle, D. Solo, M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon, "XML-Signature Syntax and Processing," IETF RFC 3275 and W3C Recommendation, IETF/W3C, http://www.w3.org/TR/xmldsig-core/, 2002.

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

[20] Verisign, *Simplifying Application and Web Services Security: Verisign Trust Gateway.* Verisign White Paper, http://www.verisign.com/resources/wp/trustgateway/Gateway_wp1.pdf, 2004.

[21] 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.

[22] 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.

[23] 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)*, (London, UK), 2004.

[24] 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.

[25] 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.

[26] 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.