

Middleware Support for Non-repudiable Business-to-Business Interactions

Thesis by
Nick Cook

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy



University of Newcastle upon Tyne
Newcastle upon Tyne, UK

2006

To Lesley, Joseph and Eve.

Acknowledgements

I am fortunate to have worked in the School of Computing Science at Newcastle since 1989 (not all of this time as a PhD student!). Many people in the School have helped me over the years and I have received valuable feedback on my research work, especially from other members of the Distributed Systems Group. I do not have the space to acknowledge them all individually. Four people deserve special mention with respect to the work in this dissertation. First, I thank my supervisor, Professor Santosh Shrivastava, for his support, guidance and insight. It has been a privilege to work closely with Santosh, and with Stuart Wheeler and Paul Robinson. Stuart has had a significant influence on the work on non-repudiable information sharing. Paul and I have worked together closely on a series of middleware projects over the past few years, which has been a rewarding experience. I must also thank Paul Ezhilchelvan for useful discussions about fundamental concepts. He is not responsible for any misconceptions that may persist. In addition, as internal examiner, he has suggested a number of important improvements to this dissertation. The external examiner, Professor Luc Moreau of Southampton University, was equally helpful and rigorous.

My research has been funded by the UK EPSRC under grant GR/N35953/01: "Information Coordination and Sharing in Virtual Environments", and under e-Science Pilot Project: "GOLD (Grid-based Information Models to Support the Rapid Innovation of High Value Added Chemicals)"; by the European Union under Project IST-2001-34069: "TAPAS (Trusted and QoS-Aware Provision of Application Services)"; and, for a short but vital period, by the School from its own stretched reserves. I am very grateful to each of these sponsors. I would not have been able to study for a PhD without their support.

Finally, and most importantly, I thank Lesley, Joseph and Eve. I owe them an incalculable debt. This has been a long journey for me. I know it must have seemed longer to them.

Abstract

The wide variety of services and resources available over the Internet presents new opportunities for organisations to collaborate to reach common goals. For example, business partners wish to access each other's services and share information along the supply chain in order to compete more successfully in the delivery of goods or services to the ultimate customer. This can lead to the investment of significant resources by business partners in the resulting collaboration. In the context of such high value business-to-business (B2B) interactions it is desirable to regulate (monitor and control) the behaviour of business partners to ensure that they comply with agreements that govern their interactions. Achieving this regulation is challenging because, while wishing to collaborate, organisations remain autonomous and may not unguardedly trust each other. Two aspects must be addressed: (i) the need for high-level mechanisms to encode agreements (contracts) between the interacting parties such that they can be used for run-time monitoring and enforcement, and (ii) systematic support to monitor a given interaction for conformance with contract and to ensure accountability. This dissertation concerns the latter aspect — the definition, design and implementation of underlying middleware support for the regulation of B2B interactions. To this end, two non-repudiation services are identified — non-repudiable service invocation and non-repudiable information sharing. A flexible non-repudiation protocol execution framework supports the delivery of the identified services. It is shown how the services can be used to regulate B2B interactions. The non-repudiation services provide for the accountability of the actions of participants; including the acknowledgement of actions, their run-time validation with respect to application-level constraints and logging for audit. The framework is realised in the context of interactions with and between components of a J2EE application server platform. However, the design is sufficiently flexible to apply to other common middleware platforms.

Contents

Acknowledgements	v
Abstract	vii
1 Introduction	1
1.1 Example applications	3
1.1.1 A B2B auction application	4
1.1.2 Negotiation of a tender to supply chemicals	7
1.2 Summary of requirements and scope of work	10
1.3 Overview of proposed non-repudiation services	15
1.3.1 Service invocation	16
1.3.2 Information sharing	18
1.4 Thesis contributions and overview of dissertation	20
2 Background	27
2.1 Definitions	27
2.1.1 Non-repudiation	28
2.1.2 Cryptographic primitives	30
2.1.3 Trusted third party services	31
2.1.4 Well-behaved parties	33
2.1.5 Perfect cryptography assumption and protocol subversion model	34
2.1.6 Notation	35
2.1.7 Generation of protocol run identifiers	35
2.2 Non-repudiation protocols	36
2.2.1 Exchange of non-repudiation evidence	37
2.2.2 Non-repudiation and fairness	38
2.2.2.1 Coffey-Saidha protocol with in-line TTP	39
2.2.2.2 Optimistic fair exchange	42
2.2.2.3 Gradual and probabilistic approaches	46
2.2.2.4 Fault tolerance and fair exchange	49
2.2.3 Non-repudiation and the role of TTPs	50
2.2.4 Repeat business and the relaxation of fairness guarantees	54
2.3 Contract-mediated interaction	56
2.3.1 The Clark-Wilson security model	56
2.3.2 Law Governed Interaction	58
2.3.3 Work at the DSTC, Brisbane	61
2.3.4 Work at Newcastle University	63
2.4 Middleware support for non-repudiation and accountability	66

2.4.1	Research systems	66
2.4.1.1	CORBA and non-repudiation	67
2.4.1.2	The FIDES project	69
2.4.1.3	Provenance services	72
2.4.2	Commercial systems	74
2.5	Discussion	76
3	Definition of non-repudiation services	79
3.1	Interceptor-mediated interaction	80
3.2	Non-repudiable service invocation	82
3.2.1	Example exchange protocols for service invocation	83
3.2.1.1	Direct exchange for service invocation	84
3.2.1.2	Fair exchange for service invocation	87
3.3	Non-repudiable information sharing	103
3.3.1	Information control state	105
3.3.2	Coordination protocols	107
3.3.2.1	Definition of protocol notation and information control state	107
3.3.2.2	Protocol assumptions and invariants	110
3.3.2.3	Coordination of state changes	111
3.3.2.4	Coordination of membership changes	116
3.3.3	Concurrency control	125
3.4	Supporting infrastructure	128
3.5	Evaluation of non-repudiation services	130
4	Generic protocol execution framework	135
4.1	Overview of framework	136
4.2	Framework APIs and implementation	138
4.2.1	Coordinator service and protocol handlers	138
4.2.2	Representation of protocol messages	141
4.2.3	Agreed representation of evidence	143
4.2.4	Application-level validation and protocol events	145
4.3	Supporting services	147
4.3.1	CertificateService	148
4.3.2	Cryptographic services	149
4.3.3	TimeStampService	152
4.3.4	NonRepudiationLog	153
4.3.5	MessageLog	154
4.4	Recovery from temporary local failure	154
4.5	Summary	156
5	Implementation of non-repudiation services	157
5.1	J2EE application server platform	158
5.2	Implementation of non-repudiable service invocation	161
5.2.1	Application programmer responsibilities	166
5.2.2	Regulated interaction with a B2B auction application	166
5.2.2.1	Auction application set-up	167
5.2.2.2	Auction demonstration	169
5.2.2.3	Evaluation of demonstration	174

5.3	Implementation of non-repudiable information sharing	175
5.3.1	B2BObjects middleware	176
5.3.2	J2EE implementation and application programmer responsibilities . . .	182
5.3.3	Transactional information sharing	185
5.3.4	Validated negotiation of a tender to supply chemicals	195
5.3.4.1	Tender application set-up	195
5.3.4.2	Demonstration of tender negotiation	196
5.3.4.3	Evaluation of demonstration	198
5.4	Summary	199
6	Summary of contributions and future work	201
	References	211

List of Figures

1.1	B2B auction application	5
1.2	Tender application	8
1.3	Transactional access to shared information	14
1.4	Service invocation	16
1.5	Exchange of service invocation evidence	17
1.6	Information sharing	19
1.7	Exchange of information sharing evidence	19
2.1	Protocol subversion model	34
2.2	A voluntary non-repudiation protocol	37
2.3	Coffey-Saidha in-line TTP fair exchange protocol	40
2.4	Wang's optimistic non-repudiation protocol	43
2.5	Security module-based <i>virtual</i> TTP	48
2.6	Signed messages and time-stamping	52
2.7	LGI message processing	59
2.8	The DSTC business contract architecture	62
2.9	x-contract deployment models	65
2.10	CORBA non-repudiation service: mandatory and optional components	67
2.11	Wichert et al's CORBA-based transparent non-repudiation	69
2.12	FIDES architecture	70
2.13	PASOA provenance architecture	73
2.14	BEA WebLogic Trading Partner Integration Engine	75
3.1	Interceptor-mediated interaction	81
3.2	Non-repudiable service invocation	82
3.3	Simple non-repudiation protocol	84
3.4	Direct non-repudiable service invocation	85
3.5	Fair non-repudiable service invocation	88
3.6	Request phase abort sub-protocol	93
3.7	Request phase resolve sub-protocol	94
3.8	Fair exchange of request with TTP as TSA	100
3.9	Optimistic fair exchange for request phase of service invocation	101
3.10	Non-repudiable information sharing	103
3.11	Agreed control state transitions	105
3.12	Proposed and agreed control state transitions	106
3.13	State coordination protocol	112
3.14	Member connection control state transition	119
3.15	Connection protocol	120

3.16	Voluntary disconnection protocol	122
3.17	Control state transitions with locking	127
4.1	Interceptor-based protocol execution with NRCoordinator services	136
4.2	Non-repudiation protocol execution components	137
4.3	Non-repudiation protocol execution API	139
4.4	NRProtocolMessage API	141
4.5	DataBinding interface	144
4.6	Example data-bindings for an Offer object	145
4.7	NRValidationListener interface	146
4.8	NREventListener interface	147
4.9	CertificateService interface	148
4.10	Cryptographic services API	150
4.11	TimeStampService interface	152
4.12	TimeStamp interface	152
4.13	NonRepudiationLog interface	153
4.14	MessageLog interface	154
5.1	J2EE-based component architecture with non-repudiation	159
5.2	Non-repudiable service invocation components	161
5.3	JBoss/J2EE-based implementation of non-repudiable invocation	162
5.4	JBoss NRInterceptor invoke operation	163
5.5	JBoss interceptor and NRInvocation API	163
5.6	Regulated B2B auction application	167
5.7	Auctioneer registering bidder	169
5.8	Request to register a bidder with NRO	170
5.9	NRR/NRV of request to register a bidder	171
5.10	Bidder placing bid	172
5.11	placeBid request with NRO	173
5.12	Client-side exception due to invalid bid	173
5.13	Evidence of invalidity of placeBid request	174
5.14	B2BObjects-based information sharing	176
5.15	Non-repudiable information sharing components	177
5.16	B2BObjects API	178
5.17	Application Offer implementation	178
5.18	Extract of B2BOffer implementation	179
5.19	Non-reentrant B2BOffer	182
5.20	JBoss/J2EE-based implementation of non-repudiable information sharing	183
5.21	Shared Offer entity bean	185
5.22	Middleware supported distributed transaction	186
5.23	Pseudo-code for a B2BObjects transaction	188
5.24	Transaction commit control state transitions	189
5.25	Transaction abort control state transitions	189
5.26	TXB2BObjectController interface	191
5.27	B2BOffer transaction sequence diagram	192
5.28	Pseudo-code for a transaction with deferred coordination	194
5.29	B2BOffer tender application	195
5.30	Application Offer interface	196

5.31	ChemicalsRUs editing and saving offer	197
5.32	ChemProc Ltd viewing and editing offer	197
5.33	Invalid proposed update from ChemProc Ltd	198
6.1	Generic protocol execution framework	203
6.2	WS-NRExchange architecture	206
6.3	Message exchange pattern composition	207

List of Tables

2.1	Notation	35
2.2	Example generation of a protocol run identifier	36
2.3	Tokens used in voluntary non-repudiation protocol	37
2.4	Tokens used in Coffey-Saidha in-line TTP protocol	39
2.5	Tokens used in Wang's optimistic non-repudiation protocol	43
3.1	Tokens used in direct non-repudiable service invocation	85
3.2	Modified tokens for fair non-repudiable service invocation	87
3.3	Tokens for request phase exception handling	93
3.4	Tokens for response phase exception handling	95
3.5	Tokens used in optimistic non-repudiable service invocation	101
3.6	Control state transition table	106
3.7	Notation for coordination protocols	108
3.8	Basic elements of control state	109
3.9	State coordination protocol control state and response tokens	111
3.10	Connection protocol tokens	120
3.11	Voluntary disconnection protocol tokens	122
3.12	Communication costs for request phase non-repudiation with validation	131
3.13	Communication costs for request phase non-repudiation without validation	132
4.1	Extended Coffey-Saidha protocol names	140
4.2	Pre-defined data bindings	144
5.1	Permitted state transitions	188

Chapter 1

Introduction

The wide variety of services and resources available over the Internet presents new opportunities for organisations to collaborate to achieve common goals. For example, business partners wish to access each other's services and share information along the supply chain in order to compete more successfully in the delivery of goods or services to the ultimate customer. This can lead to the investment of significant resources by business partners in the resulting collaboration. In the context of such high value business-to-business interactions (B2B) interactions it is desirable to regulate (monitor and control) the behaviour of business partners to ensure that they comply with agreements that govern their interactions. While cooperating on agreed undertakings, organisations will continue to act autonomously and are likely to privilege their own interests over those of their business partners. This can lead to tension between the need for cooperation and the need of each business partner to protect their own interests. In this context, we assume that each business partner has a set of local policies that they wish to enforce and that there are one or more agreements between partners for the conduct of their interactions. Examples of agreements include natural language contracts that govern business collaborations, agreements that specify acceptable service provision and usage, and the specification of the correct execution of cross-organisational processes or workflows to achieve some common goal. To resolve the tension between cooperation and autonomy, it is desirable to regulate B2B interactions so that they comply with the agreements between business partners. At the same time businesses should be able to enforce their own policies. Such enforcement should include accountability for actions performed. That is, it should not be possible to subsequently deny having requested some service or having modified shared information. If actors

are not accountable for their actions, then agreements are unenforceable. So, there are two aspects to achieving regulation in the context of high-value B2B interactions:

1. high level mechanisms to specify contractual rights and obligations and to derive an electronic embodiment of contractual conditions for the validation of observed behaviour, and
2. lower level mechanisms that record observed behaviour — generating a non-repudiable audit trail of an interaction and verifying that the observed behaviour is valid with respect to the electronic embodiment of interaction agreements.

I aim to address the latter aspect of regulation. The work presented in this dissertation concerns the run-time validation and non-repudiation of B2B interactions, where an interaction is non-repudiable if the parties to the interaction cannot subsequently deny their participation. A related concern is that well-behaved parties should not suffer disadvantage as a result of the misbehaviour, or non-cooperation, of their business partners. Thus accountability can extend to supporting fair outcomes to parties who comply with agreements.

The main contribution of my thesis is the design and implementation of flexible middleware to support the non-repudiation and validation of B2B interactions. I define two non-repudiation services — non-repudiable service invocation and non-repudiable information sharing — that address requirements in two domains for action (over private and shared resources). I show how the non-repudiation services can be implemented in a flexible framework for protocol execution. This results in abstractions that are familiar from the enterprise context and provide regulated interaction in the B2B context. For example, non-repudiable service invocation can be used to audit requests between organisations to access or modify each other's internal information, or for transfer of control over shared information. Non-repudiable information sharing regulates access to and updates of shared information. An example of shared, or jointly owned, information is a negotiated agreement that governs a B2B interaction.

The non-repudiation of a service interaction is achieved by the execution of a security protocol for the exchange of evidence of the interaction. Different non-repudiation protocols can be used to meet different application requirements. For example, in Chapter 3, I present a

simple, direct exchange for interaction with a trusted service and I contrast this with protocols for fair, non-repudiable service invocation. The middleware described in this dissertation also supports application-specific validation of service interactions. Thus, proposed service invocations and updates to shared information can be validated with respect to agreements and local policies to ensure compliance with the regulatory regime in force. In Section 1.1 of this chapter I describe two example applications that serve to motivate the work presented in this dissertation and provide requirements for that work. Section 1.2 summarises the derived requirements and the scope of my work. Section 1.3 provides an overview of proposed services for non-repudiable service invocation and information sharing that are intended to address the identified requirements. Section 1.4 concludes the chapter with the assertion of thesis contributions and an overview of the remainder of the dissertation.

1.1 Example applications

This section describes two applications that motivate the work in this dissertation. These applications are in part used to derive requirements for my work. In Chapter 5 the applications also serve as proof-of-concepts for the design and implementation of the non-repudiation middleware services that address the requirements.

The first application is a B2B car parts auction. An Application Service Provider (ASP) hosts the auction application and is responsible for enforcing the auction rules and for ensuring that auctioneers and bidders are held to account for their actions as auction participants. The work on the auction application was part of the final deliverable for the EU TAPAS project on the Trusted and QoS-Aware Provision of Application Services [CRS04a, BS04].

The second application concerns the negotiation of a tender to supply chemicals as part of a chemical development supply chain. The tender, or offer, is shared by the supply chain business partners who must approve any changes to it. This application is derived from the real-life requirements capture exercise conducted as part of the UK e-Science GOLD Project [PCC⁺06].

1.1.1 A B2B auction application

TAPAS [SMJ05] was an EU-funded project to address the problems faced by ASPs that host distributed applications that involve interaction across organisational boundaries. The project addressed two main areas of regulation in this context: (i) the definition, monitoring and enforcement of service level agreements (SLAs); and (ii) the monitoring of service requests to ensure compliance with contractual terms and conditions. The first area concerns the collection of statistical measures of the performance of a service to determine whether the service provider is delivering the quality of service (QoS) that its service consumers can legitimately expect. The second area concerns the monitoring of interactions to ensure that requests (business messages) are syntactically and semantically correct as specified by business agreements. The ASP is an intermediary that facilitates cooperation between the organisations in order to achieve a shared business goal. The ASP hosts an application on behalf of the organisations and ensures that their interactions with the application adhere to the agreements that are in force.

In the auction scenario a car manufacturer uses a sealed reverse bid auction to buy car parts from car part suppliers. The sealed reverse bid auction proceeds through a finite number of bid rounds. A bidder is allowed to place at most one bid in each round. The bid is known only to the bidder and the auctioneer. The winner of each round is the bidder with the lowest bid. The value of this bid is announced before the next round is opened. A bidder can only place a bid in a round if they placed a bid in the previous round. Their bid must be lower than the winning bid from the previous round. The winner of the auction is the winner of the last round. The auctioneer determines the number of rounds at the outset of the auction. The auctioneer has the right to abandon the auction at any time during the bidding process.

Figure 1.1 shows the basic structure of the auction application. In effect, the ASP is a trusted third party (TTP) that regulates the interaction between the auction clients — an auctioneer plus bidders from two or more part suppliers. The auction clients participate in an auction by submitting requests to invoke operations on the auction application. The auctioneer may invoke application client operations that:

1. register/create potential bidders for an auction (the `registerBidder` operation in Fig-

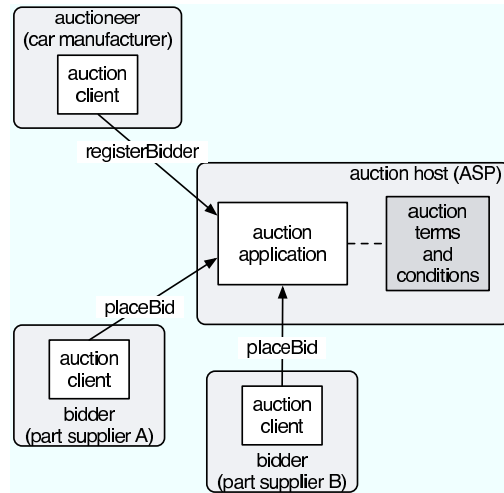


Figure 1.1: B2B auction application

ure 1.1),

2. create an auction, specifying parts to supply, auction deadlines, number of bid rounds, opening price etc.,
3. invite bidders to an auction instance, and
4. select an auction winner or declare an auction abandoned.

A bidder may invoke application client operations that:

1. accept, or reject, an invitation to participate in an auction, and
2. place a bid in an auction round (the `placeBid` operation in Figure 1.1).

Each application client operation must be validated with respect to the auction terms and conditions. These terms and conditions are the general rules for conduct of the auction outlined previously and may also include extensions, or modifications, to those rules that apply to specific auctioneers and/or bidders. In addition, client operations must be logged for audit and, to ensure accountability, evidence must be generated to bind the invoker of an operation to its invocation. That is, an auction client must not be able to subsequently deny the invocation an operation — the invocation of the operation must be non-repudiable. To protect auction client interests, the ASP must provide an irrefutable acknowledgement (receipt) of each operation

that a client invokes. This evidence must include an indication of the validity, or otherwise, of the requested operation with respect to the terms and conditions in force. Further, only valid operations should be invoked on the auction application. We can summarise the non-repudiation requirements as follows.

1. Both the origin and receipt of requests must be non-repudiable; that is, evidence must be generated to irrefutably bind a request to an originating client and, given evidence of origin, the auction service must provide irrefutable evidence of receipt of a request. This evidence must be recorded in a persistent log to provide an audit trail on behalf of the clients and the ASP.
2. Requests must be validated with respect to business contract terms and conditions that govern the behaviour of the clients participating in an auction. The collection of evidence of the validity of client requests must be integrated with the collection of non-repudiation evidence.

Now consider that the ASP may host many different instances of a given auction application and, as suggested previously, certain terms and conditions may be specific to a particular set of clients (auctioneer and bidders). That is, the ASP may need to customise the behaviour of the application, as perceived by the application clients, to meet the requirements of a specific set of business relationships. To manage such customisation, the ASP must be able to impose agreed constraints on the behaviour of auction participants without having to re-implement the application. Similarly, the ASP requires that the generation and collection of non-repudiation evidence does not require modification of the application.

The B2B auction progresses by the invocation of operations on the ASP-hosted application. There is no direct communication between auctioneer and bidders. An extension to this scenario could involve bidders offering part specification services to the auctioneer. Then the auctioneer could clarify the details of a bid made to the auction service by using a bidder's part specification service to obtain supplementary information about the parts that are the subject of the bid. This supplementary information may be commercially sensitive. So, the bidder requires evidence that the request for the parts information originated at the auctioneer and that

the auctioneer received the information in response to the request. In effect, relevant bid now includes the supplementary information provided by the bidder. The auctioneer requires evidence that their request for clarification was received by the bidder's service. In order to hold the bidder accountable for the supplementary information, the auctioneer also requires evidence that the response originated from the bidder's service. As with the auction application hosted by the ASP, the auctioneer uses service invocations to interact with bidder services. The accountability requirements are also essentially the same — the non-repudiation and validation of invocation request and response. However, it is more challenging to meet these requirements for direct interactions between auctioneer and bidder organisations than for the interaction with the trusted ASP. This is because one of the ASP functions, as a trusted service provider, is to ensure that none of its clients suffers disadvantage as a result of misbehaviour, intentional or otherwise, by another client. Further, the ASP guarantees delivery of appropriate interaction evidence to clients that are entitled to receive that evidence. In contrast, in the direct interaction between an auctioneer and a bidder service, either party may seek to gain advantage over the other party.

The preceding discussion highlights the need for flexibility when meeting accountability requirements. The auction house may need to deploy the same application under with different contractual constraints. Different relationships between interacting parties may mean that different underlying mechanisms are needed to ensure accountability. Section 1.2 takes account of the need for flexibility when summarising requirements for generic services for non-repudiable service invocation.

1.1.2 Negotiation of a tender to supply chemicals

The GOLD project concerns the provision of middleware services to support business collaboration. The project focuses on the formation of consortia for chemical development. Commercial pressures, such as time-to-market, are driving a trend to closer collaboration in the sector. In Europe, the pressure for collaboration is not just commercial. Recent European legislation for the registration, evaluation, authorisation and restriction of chemicals [Com05] mandates the sharing of commercially sensitive information along the supply chain. The formation of

consortia to share the results of chemical safety tests and to trace the use and supply of chemicals is becoming imperative. A key requirement in this context is that shared information should not be modified without the agreement of all parties that share, or jointly own, the information. In this section, the negotiation of a tender agreement to supply chemicals is used to derive requirements for information sharing. The shared information is the tender agreement that will be used to regulate subsequent interactions for the actual delivery of chemicals.

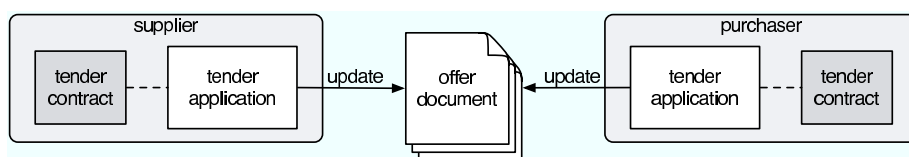


Figure 1.2: Tender application

The scenario is that a manufacturer adopting some new chemical development process wishes to negotiate a tender for the supply of chemicals. To manage the tendering process, the supplier and manufacturer (purchaser) *share* an offer to supply chemicals. Both the supplier and the purchaser own the offer document. Figure 1.2 depicts a conceptual view of the interaction in which the offer is outside the absolute control of either supplier or purchaser and they share an agreed view of the offer. The document, or documents, that represent the offer may actually reside in some shared repository or there may be copies at supplier and purchaser. The significance of the notion of shared information is that updates to the offer must be visible to both parties and must be agreed by both parties. The manufacturer and supplier are primarily concerned with the integrity of the offer — that it accurately represents their agreement. The negotiation of changes is an iterative peer-to-peer process. Each business partner (supplier or purchaser) updates the parts of an offer that they are permitted to update and the other party validates the proposed updates. Only valid updates result in an actual change to the agreed (shared) view of the offer. When the supplier proposes a change to the offer, the purchaser can decide whether to accept or reject the offer terms. The purchaser may also propose amendments to the offer.

As shown in Figure 1.2, a tender contract governs the offer negotiation process. As a simple example, the purchaser wishes to enforce the following rules with respect to a supplier's updates to an offer: (i) the offer must specify a pre-defined contract period during which it is

valid; (ii) each offer must have a unique identifier; (iii) a supplier must complete the following offer details: contract period, offer identifier, supplier and purchaser names, offer expiry date, description of chemical, unit of supply, unit price and quantity offered at the unit price; (iv) the expiry date must be within the contract period; (v) a supplier can only complete their part of a uniquely identified offer once; and (vi) a supplier must not indicate the acceptance or otherwise of the offer. A supplier wishes to enforce the following rules with respect to a purchaser's updates to an offer: (i) a purchaser can only update a uniquely identified offer with their acceptance decision once, (ii) a purchaser must update the offer with their acceptance decision before the offer expiry date, and (iii) a purchaser must not alter any of the terms of the offer that were set by the supplier.

The negotiation proceeds by the supplier proposing amendments to a shared offer template by adding the information specified in the contract. The purchaser continues the process by validating the proposed changes with respect to the contract. If the changes are valid, the purchaser updates the offer with their decision whether to accept the offer or not. The supplier then, in turn, checks that the purchaser did not breach the contract conditions when declaring their decision. Assuming the purchaser behaved correctly, the amended offer becomes the agreed version of the offer. If the decision was to reject the offer, then the offer provides evidence that the proposed terms of tender were unacceptable to the purchaser. If the purchaser's decision was to accept the supplier's offer, then the offer becomes the contract that governs the supply of the chemicals. In this way the supplier and purchaser build an agreed, shared view of their interaction and can reach a negotiated agreement to govern continued interaction.

The preceding simple offer and negotiation process could be extended to more complex scenarios. For example, it could be the purchaser's responsibility to suggest both the quantity of chemicals required at the offered price and a deadline for the delivery. The supplier may then have the option whether to commit to these terms of the offer. Another scenario may involve a third party to fulfil a part of the offer and this party must agree to the delivery terms. Another extension may involve agreement on the outsourcing of testing to meet the legal obligations of both supplier and purchaser. In this case, the test provider must agree the work to be outsourced. In all cases, the parties sharing the offer are interested in reaching agreement on changes to the

offer and at the same time maintaining the integrity of the offer with respect to the contract that governs the process to update it. In particular, no party should be able to claim that an offer is valid unless the changes to the offer were agreed by all that other parties that share the offer. Similarly, no party should be able to repudiate their agreement to changes proposed by other parties.

Chapter 5 uses the B2B auction and the negotiation of an offer as proof-of-concepts applications for the non-repudiation middleware services. I now derive requirements for the services and define the scope of the dissertation.

1.2 Summary of requirements and scope of work

For the purposes of this dissertation, a B2B interaction involves the electronic exchange of messages between computing entities at different organisations. These entities are variously referred to as clients, servers, applications, nodes and protocol participants. In all cases, the actors in the interaction are computing entities and messages are exchanged electronically. Clearly, human actors are also involved in B2B interactions but they are only within the scope of this dissertation in so far as they are represented by electronic entities — such as the representation of a user by their digital certificate.

From the example applications described in Section 1.1, we can identify two domains for action in B2B interactions: (i) actions on private information or resources that are wholly owned by a single organisation and (i) actions on shared information that is jointly owned by two or more organisations. An organisation has complete control over access to its private resources. To facilitate business, organisations expose services that act upon their private resources. There are then the following two necessary forms of basic B2B interaction.

1. A client from one organisation invoking a service that is hosted by another organisation.
A service invocation may involve the sending of a one-way request or the exchange of a client request for a service response.
2. Access and/or update to shared information by an entity from one organisation that is subject to the agreement of one or more other organisations.

Business partners will engage in other forms of interaction. However, the identified basic forms can serve as useful building blocks for more complex interaction. For example, enactment of a cross-organisational workflow typically involves the composition of service invocations. Similarly, processes specified by standard B2B protocols such as RosettaNet Partner Interface Processes (PIPs) [Ros05] and ebXML [GEN⁺01] can be realised through service invocations between partners. The chemical development example is part of a more complex ongoing process that combines information sharing to reach agreement on a tender with subsequent service invocations to fulfil the tender, subject to the negotiated agreement. When considering consortia of collaborating organisations, we can adopt a recursive view of organisations. That is, organisations that are not members of a given consortium view the consortium as a single organisation. Then the consortium may provide services to external clients. These services may, in turn, involve operations on the consortium's private information. The information that is private to the consortium from the viewpoint of its external clients may be shared from the viewpoint of the members of the consortium.

From the example applications we can also identify two broad categories of interaction that depend on the relationships between the participants: (i) interaction between and with trusted entities and (ii) interaction between business partners that do not unguardedly trust each other. Examples of the former are the auctioneer and bidder client interactions with the ASP hosted auction service. Examples of the latter are the auctioneer's use of bidder part specification services. A distinguishing feature of trusted entities in an interaction is that they will not seek to gain unfair advantage over other participants. In contrast, business partners may obtain unfair advantage over other parties, either through deliberate action or through omission. Business partners reserve the right for autonomous action and will privilege their own interests over others. A requirement in this context is that the legitimate interests of all parties are safeguarded. The mechanisms required to achieve this will depend on the relationships between the parties, the risk of disadvantage in a given context and the domain of action. For example, when a service invocation between business partners involves the disclosure of sensitive information it may be imperative that the information is only disclosed in return for a guaranteed acknowledgement of receipt, even if the recipient ceases to cooperate in the invocation. In contrast,

when sharing information a lack of progress due to non-cooperation may be acceptable as long as information integrity is preserved.

The categorisation of interactions by action domain and by participant relationship leads to the overall challenge to regulate both service invocations and information sharing when participants may not unguardedly trust each other. An important part of meeting this challenge is to provide regulatory mechanisms that apply in many different contexts yet can be adapted to the specific requirements of a given context. To achieve regulation, a given action must be attributable to the party who performed the action and commitments made must be attributable to the committing party. For example, it should not be possible for a client to subsequently deny the request and consumption of a service. Similarly, it should not be possible for the service provider to subsequently deny having delivered a service. Parties that share information should be able to validate proposed updates to the information, any update should be attributable to its proposer and the other parties who share the information should not be able to repudiate their decisions with respect to the validity of the update. That is, to achieve regulation we require attribution, validation and audit of B2B interactions. Non-repudiable attribution binds an action to the party performing the action. Validation determines the legality of an action with respect to interaction agreements. Audit ensures that evidence is available in case of dispute and to inform future interactions.

From the preceding discussion, I highlight the following specific requirements for systematic support for regulation:

Non-repudiation Non-repudiation is fundamental to the achievement of accountability in B2B interactions. It must not be possible to deny the initiation or receipt of either a service request or the associated response. Similarly, it must not be possible to deny a proposed change to shared information or to deny participation in the validation of a proposed change.

No unfair advantage Actors must be accountable for their actions in B2B interactions yet they may have an interest in the subverting the non-repudiation mechanisms intended to achieve accountability. The middleware should ensure that a party that attempts such subversion can gain no unfair advantage over other parties. As indicated previously,

the characterisation of unfair advantage may depend on the domain of action. For a service invocation, acquiring a service request without providing a receipt represents unfair advantage. For information sharing, the ability to misrepresent the agreed state of the information is an unfair advantage.

Application-specific validation The validity of actions in a B2B interaction is determined by the regulatory context. This context is a combination of the local policies of participants, the agreements between participants and legal constraints on participant behaviour. The determination of the validity of an action is application-specific. Therefore, the requirement is to support the integration of application-specific validation when collecting evidence for the accountability of actions.

Flexibility The requirement for flexibility arises mainly from the need to provide support for regulation in different application contexts and with respect to different relationships between B2B interaction participants (whether trusted entities or business partners). For example, in the auction scenario it was suggested that the ASP may wish to deploy instances of the application under different business contexts — customising the rules that govern auction client operations. To apply regulation in this case it should be possible to use the same basic application and, without modification to the application logic, provide non-repudiable validation of client operations where the rules that are applied vary according to the regulatory regime. That is the mechanisms to enforce accountability and to invoke application-specific validation should not require modification to existing application business logic. Another example is that the ability to guarantee no unfair advantage is dependent on the relationship between parties. The requirement for flexibility is that the support for regulation should adapt to these different relationship by providing underlying non-repudiation mechanisms that are appropriate to a given relationship.

The preceding requirements apply to both service invocation and information sharing. In addition, in the case of information sharing we require support for transactional access to the shared information in order to perform a set of related changes, and for the extension of transaction context to span both shared and private information. Shared information does not exist in iso-

lation. For example, in the chemical development example, a change to the offer proposed by the supplier may depend both on validation of the change by the manufacturer and on the commitment of changes to related local resources such as delivery schedules. As depicted in

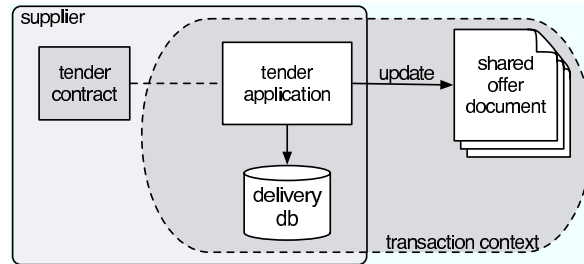


Figure 1.3: Transactional access to shared information

Figure 1.3, to manage such dependencies, the supplier needs to be able to complete an update to shared information and to their local database as an atomic action. Thus we need to extend transaction context to support the atomic action.

We now have a challenging set of requirements that are not met by any existing system. For example, no existing non-repudiation system addresses requirements both to avoid unfair advantage and at the same time to provide the flexibility of application independence. I now provide a brief overview of the scope of this dissertation. The overall aim is to design and implement a flexible system to support the regulation of B2B interactions. The approach is to identify, apply and, where necessary, extend fundamental work on security protocols for non-repudiation and fairness in the design of the system. To arrive at a practical system, the following two restrictions apply to the scope of work.

Exchange is not fault-tolerant. There is a considerable body of work on non-repudiation and fair exchange (see Section 2.2). As discussed in Section 2.2.2.4, less attention has been paid to fault tolerance and fair exchange. The fault tolerance problem is to avoid unfair disadvantage to an honest party despite local failures of an assumed type. It is beyond the scope of this dissertation to deploy fault-tolerant exchange protocols in the system that is developed. Section 4.4 does address middleware support for recovery from local failure. The exploitation of this support by fault-tolerant exchange protocols is the subject of future work. The proposed protocol execution framework is sufficiently flexible for the

subsequent incorporation of such protocols.

Update to shared information is blocking. I assume that the overriding concern when sharing information is the maintenance of the integrity of shared information. Therefore, my work concentrates on maintaining a consistent view of shared information that cannot be repudiated by the parties sharing the information. Applications that access shared information progress through a series of agreed updates to that information. The system presented in this dissertation does not address lack of application progress due to non-cooperation of one or more participants. However, cooperation in application progress may be considered rational behaviour when establishing agreements to govern subsequent, mutually beneficial interaction. Section 2.2.4 discusses incentives for such rational behaviour. Section 3.5 discusses approaches for guaranteeing termination to well-behaved parties. Again, the proposed protocol execution framework is sufficiently flexible to incorporate guaranteed termination as future work.

As suggested, the work presented in this dissertation (see Chapters 3, 4 and 5) will demonstrate the inherent flexibility of the system design and the ability to extend it to address the preceding limitations. Essentially, the system can be extended by the deployment of new protocols that will address requirements for fault-tolerance in exchange and non-blocking update to shared information.

1.3 Overview of proposed non-repudiation services

This section proposes two services to address the preceding requirements. The service descriptions generalise support for regulation in the two domains for action — over private and shared resources — and further clarify the scope of work. The core of the dissertation, from Chapter 3 to Chapter 5, concerns the full definition of these services and the demonstration that their flexible design and implementation can meet all of the requirements set out in Section 1.2. The first service is for non-repudiable and validated service invocation. The second is for non-repudiable and validated information sharing.

1.3.1 Service invocation

Figure 1.4 shows a typical two-party, client-server interaction. The client invokes a service by

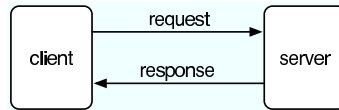


Figure 1.4: Service invocation

sending a request to the server. The server then issues a response. From the point of view of client and server, a service invocation corresponds to the correlation of two business messages — one for the request and the other for any associated response. In certain cases there will be no response¹ and only the request is of interest. For non-repudiable service invocation we wish to generate irrefutable evidence that the interaction took place and to validate both request and response with respect to contract (however defined). After an attempt to submit a request to a server, the following assurances should hold for the client:

1. that either: (i) the submission failed and the server did not receive the request; or (ii) the submission succeeded and there is proof that the request is available to the server; **and**
2. that if a response is received, there is proof that the server generated the response.

The corresponding assurances for the server are:

1. that there is proof that the client generated the request; **and**
2. that following an attempt to deliver any response to the client, either (i) the delivery failed and the client did not receive the response; or (ii) the delivery succeeded and there is proof that the response is available to the client.

Further, to ensure that the server only processes valid requests and that the client only processes valid responses, both request and response should be subject to validation with respect to contract. Thus, evidence should be generated with respect to the validity of the request and the response.

¹As for a remote invocation with a void return type.

To provide the preceding assurances, client and server must execute a non-repudiation protocol to exchange evidence of the interaction and to ensure that:

1. a request is only passed to the server if: (i) the client provides evidence for the non-repudiation of origin of the request (NROreq), (ii) the request is valid with respect to contract, as indicated by non-repudiation of validation of the request (NRVreq), and (iii) the server provides evidence for the non-repudiation of receipt of the request (NRRreq), **and**
2. a response is only passed to the client if: (i) the server provides evidence for the non-repudiation of origin of the response (NROresp), (ii) the response is valid with respect to contract, as indicated by non-repudiation of validation of the response (NRVresp), and (iii) the client provides evidence for the non-repudiation of receipt of the response (NRRresp).

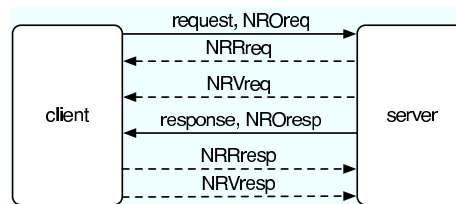


Figure 1.5: Exchange of service invocation evidence

Figure 1.4 is a logical view of the exchange of evidence that is achieved by the execution of an appropriate non-repudiation protocol. It shows the minimum requirements for the evidence that must be generated and exchanged during a request/response service invocation. As shown, the client initiates a request for some service. The client generates an NROreq token and sends the request and the token to the server. The server generates a NRRreq token and returns it to the client. The server then subjects the request to application-level validation and generates a NRVreq token based on the result of the validation, which they send to the client. If the request is valid, it is subject to server-side processing. The server then sends the result of this processing and the associated NROresp token to the client. The client generates a NRRresp token and returns it to the server. The response is then subject to application-level validation and a NRVresp token generated from the result of validation. The client sends the NRVresp

to the server. Finally, if the response is valid, the client application consumes the response. The precise meaning of generation of non-repudiation tokens will be dependent on the actual protocol used to execute the exchange. For example, a local signing service may be used to sign evidence or a token may be generated with the assistance of a TTP. Similarly, the precise sequence of token exchange is protocol-specific, as is the level of involvement of TTPs.

The aim of the proposed service for non-repudiable and validated service invocation is to preserve the abstraction shown in Figure 1.4 and at the same time generate and exchange the evidence shown in Figure 1.5. That is, the mechanism used to achieve the exchange of evidence should not violate the abstraction presented in Figure 1.4. Further, as indicated in Section 1.2, the business/application context should determine the choice of mechanism.

1.3.2 Information sharing

The primary concern with respect to information sharing is the maintenance of the integrity information that is shared between application processes at two or more collaborating organisations. As noted in Section 1.2, we wish to hold organisations to account for their actions with respect to shared information. As with service invocation, this leads to the need to collect non-repudiation evidence of those actions and their validity with respect to agreements that determine which actions are legitimate. This section proposes a non-repudiation service that facilitates information sharing by enforcing the observance of agreements to preserve the integrity of the information.

There are a number of alternatives for the physical realisation of information sharing. For example, shared information may be hosted by some centralised service that the information sharing group can access. Alternatively, the information may be distributed, in which case each member of the sharing group hosts a replica of the information. Whatever the physical realisation, changes to shared information should be agreed by all the members of the group that share the information. Through this agreement, the members of the group share a consistent view of the valid (agreed) state of the information. The challenge for the proposed service for non-repudiable information sharing is to maintain this irrefutable consistent view. Conceptually, shared information resides in a shared space. The members of the sharing group act upon

the information in the shared space. They can observe each other's actions and subject those actions to validation. Thus, the shared space is an abstraction that presents the agreed view of the information.

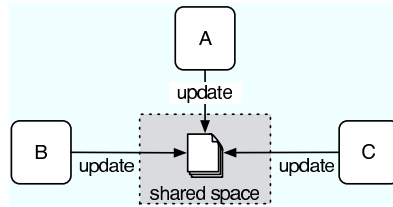


Figure 1.6: Information sharing

Figure 1.6 shows three organisations (A, B and C) that have access to and can update information in a shared space. If A wishes to update the information, then, to meet regulatory requirements, they must reach agreement with B and C on the validity of their proposed update. For the agreement to be non-repudiable:

1. B and C require evidence that the update originated at A, and
2. A, B and C require evidence that, after reaching a decision on the update, all parties have the same (agreed) view of the state of the shared information.

The latter condition implies that there must be evidence that all parties received the update and that they all agreed to the update being applied to the information.

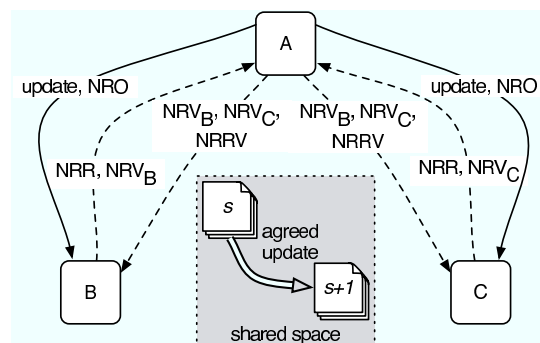


Figure 1.7: Exchange of information sharing evidence

Figure 1.7 shows the exchange of evidence required to achieve the irrefutable, agreed transition from information state s to information state $s + 1$. First A proposes an update to both B

and C. As shown, A also provides evidence of the origin of the update (NRO). Then B and C each provide evidence of receipt of the update (NRR) and their non-repudiable decision on the validity of the update (NRV_B and NRV_C , respectively). Finally, A relays the group decision to B and C by sending them the collected validation tokens and A's non-repudiation of receipt of the validation evidence (NRRV). If the validation tokens represent unanimous agreement to the change, then the consistent shared view of the information after the exchange of evidence is that it is now in state $s + 1$ (as shown in Figure 1.7). Otherwise the update is considered invalid and the shared view of the information is that it is in state s .

From the preceding discussion it is apparent that both the application state of information and the membership of the group that shares the information are fundamental to the agreed view of shared information. Consider, the offer that the supplier and purchaser share as part of the tendering process in Section 1.1.2. The state of the shared information includes the contract period, offer identifier, supplier and purchaser names, description of chemical, unit of supply etc. The group sharing the offer comprises the supplier and the purchaser. An example transition in the state of an offer is a change to the unit of supply. An example transition in the membership of the sharing group is an extension of the group to include a test provider. Either type of change, whether to the application state of the offer and to the sharing group, must be subject to the agreement of the existing members of the sharing group. In both cases, group members should be able to apply the rules of the tender contract, and of any relevant local policy, to validate a proposed change.

In summary, the overall aim of the proposed service for non-repudiable and validated information sharing is similar to that for service invocation: to preserve the abstraction shown in Figure 1.6 and at the same time generate and exchange the evidence shown in Figure 1.7. The mechanism used to achieve the exchange of evidence should not violate the abstraction and the business/application context should determine the choice of mechanism.

1.4 Thesis contributions and overview of dissertation

Section 1.2 identified two domains — the private and the shared — that provide broad coverage of B2B interactions and a challenging set of requirements to support the regulation of actions

in each domain. Section 1.3 proposed two non-repudiation services to meet the identified requirements. My thesis is that:

1. a set of middleware services can provide a flexible framework for protocol-based interaction,
2. that the framework can deliver non-repudiation services that are appropriate to the two domains for action — the private and the shared, and
3. that the services can be used to regulate B2B interactions and at the same time preserve application-level semantics.

This statement will be justified by the design, implementation and deployment of novel middleware. The principal novelty of the approach is the combination of fundamental work on non-repudiation with the flexibility to adapt to different business contexts to meet different application-level requirements. As noted in Section 1.2, flexibility is required because the relationships between business partners vary from partner to partner and over time. There will be variations in the agreements and regulatory regimes that constrain interactions. Yet, there is a consistent need to enforce agreements and provide accountability for B2B interactions that comprise service invocations and/or information sharing. In this dissertation I show how the novel middleware can adapt to use different underlying regulatory mechanisms (non-repudiation protocols) that offer different security guarantees to be applied in different contexts. There is a careful separation of run-time enforcement mechanism from the higher-level process that determines the validity of actions. This separation allows the middleware to enforce arbitrary application-specific validation and at the same time provide accountability for validation decisions. In Chapter 2 I review fundamental work on non-repudiation. This work is applied and extended in the development of the middleware services that are the subject of the remainder of the dissertation. Also in Chapter 2, I survey other middleware systems for accountability and non-repudiation. The remainder of the dissertation substantiates the claim of novelty with respect to these systems. Before providing an overview of the dissertation, I elaborate on novel contributions in three areas.

1. *The development of flexible middleware that is based on fundamental work on non-repudiation and fairness.* Most of the existing middleware support for non-repudiation takes no account of the substantial theoretical work on the development and verification of protocols. Existing middleware systems, whether commercial or academic, tend to provide for the signing of outgoing requests and for verification of signatures on incoming request; that is, voluntary non-repudiation of origin. Typically, there is no attempt to guarantee fairness. An exception is the FIDES project (see Section 2.4.1.2). However, the FIDES system is customised to a specific sub-set of their own protocols for fair document exchange. Moreover, they do not provide an API that could be used to adapt the system to provide general support for non-repudiable service interactions or for validation with respect to contract. The middleware presented in this dissertation addresses both these problems. The middleware is also adaptable to different protocol implementations, and can provide both voluntary non-repudiation and non-repudiation with fairness.
2. *Systematic support for regulation.* Section 2.3 describes various approaches to the problem of ensuring that an interaction complies with business agreements (or contracts). In general, this work concentrates on how to express contracts in a form that is amenable to integrity checking and that can potentially be used in the regulation of interactions at run-time. The problem that I address is how to provide systematic support for this run-time monitoring and enforcement. No other middleware that supports non-repudiation addresses this problem, except in a very limited sense (see the BEA WebLogic system in Section 2.4.2). In addition, work on electronic contracts does not address the binding of actions, and the binding of decisions on the validity of actions, to actors. A novel aspect of the mechanisms that I develop is, then, the combination of the irrefutable binding of interacting parties to their actions and the run-time validation of those actions. Furthermore, the validation mechanism can be used to prevent illegal actions. This approach links work on non-repudiation to work on the electronic contracts. It provides an irrefutable audit trail of the actions that were permitted and the actions that were forbidden during an interaction. A contribution in this area is the extension of fair exchange

protocols to provide non-repudiation of (application-level) validation of actions (see Section 3.2.1.2). An important consideration when designing the enforcement mechanisms was the separation of application concerns from regulatory mechanisms. The ability to specialise validation of interactions to specific (application-level) regulatory requirements and, yet, trigger the validation from non-repudiation services executing at the middleware level is another novel contribution. This approach allows regulation to be applied as an aspect to interactions. BEA provide non-repudiable regulation of certain types of interaction that is based on a tight-coupling of the non-repudiation mechanism with the business protocol that provides some enforcement of contract². My contribution could be seen as breaking this tight-coupling and, thereby, providing a more general approach to support for the regulation of B2B interactions.

3. *Inter-organisational information sharing*. In this dissertation I develop the notion of regulated, inter-organisational information sharing. This provides a convenient programmer abstraction for the manipulation of shared information. I present a coherent model for agreed transitions in shared information that covers updates to information state, changes to the membership of the group who share the information, and transactional access to the information. The middleware ensures non-repudiable coordination of the meta-information that describes shared information, and, in consequence, agreed (safe) update and access to the information. Work on security models and policy from Clark-Wilson³ onwards typically concerns the regulation of information that is under the control of a single organisation. A novel contribution in this area is the combination of the abstraction of shared information with agreement to changes and access to the information in the context of B2B collaborations in which no single organisation “owns” the information.

Overview of dissertation

The work presented in this dissertation addresses the requirements identified in Section 1.2 by providing middleware to support the regulation of B2B interactions. I develop two services: (i) non-repudiable and validated service invocation; and (ii) non-repudiable, validated and trans-

²See 2.4.2 for further discussion.

³See Section 2.3.1.

actional information sharing. The approach is to provide a coherent definition of the identified non-repudiation services. These definitions include the non-repudiation protocols that underpin the services. Given the service definitions, I present the design of a flexible framework for their implementation at the middleware level that supports the systematic regulation of service interactions. The design also identifies the supporting infrastructure necessary to provide the non-repudiation services. The inherent flexibility of the design ensures that its implementation can adapt to application-specific requirements. Given the design, I develop prototype implementations of the non-repudiation services. Proof-of-concepts applications demonstrate the utility of the service implementations. The dissertation has the following structure.

Chapter 2 defines concepts and notation that are necessary to understand the remainder of the dissertation and provides an overview of related work. The chapter concentrates on work on non-repudiation protocols that is fundamental to the services I develop. I also survey work on contract-mediated interaction, and on middleware support for non-repudiation and accountability.

Chapter 3 defines the non-repudiation services that, along with their implementation, are the novel contribution. The services are based on an interceptor-mediated view of interaction that: (i) allows application programmers to view the provision of non-repudiation as orthogonal to application concerns, (ii) supports the design of adaptable services, and (iii) provides a coherent framework for consideration of both non-repudiable service invocation and non-repudiation information sharing. This chapter develops work first reported in [CSW02, CRS04b]. Section 3.2 extends the description in [CRS04b] to present concrete examples of protocols for non-repudiable service invocation. Section 3.3 introduces a new approach to modelling transitions in shared information that is a significant extension to the work in [CSW02].

Chapter 4 presents the design and implementation of a generic protocol-execution framework and related services to support the non-repudiation services. The design is flexible enough to meet the specific requirements of different interactions and, at the same time, remain application-independent.

Chapter 5 describes the implementation of the two identified non-repudiation services based on the framework introduced in Chapter 4. Proof-of-concepts applications based on the examples in Section 1.1 demonstrate the utility of the implementations. Section 5.2 describes the implementation of non-repudiable service invocation and is based on work first reported in [CRS04b]. The implementation of non-repudiable information sharing in Section 5.3 extends work first reported in [CSW02, CSW03, CRS04b]. In particular, the support for transactional information sharing in Section 5.3.3 has been revised in the light of the new model for transitions in shared information. The chosen demonstrator platform is a Java J2EE [Sun03] application server. Preliminary work [RCS05, CRS06] on a re-implementation for Web services illustrates the flexibility of the approach.

Chapter 6 concludes the dissertation with a summary of contributions, including an evaluation with respect to the requirements set out in this chapter, and an overview of future work.

Chapter 2

Background

This chapter first defines concepts and notation that are necessary to understand the remainder of the dissertation. I then provide an overview of related work. Section 2.2 focuses on fundamental work on non-repudiation protocols that can be used to achieve accountability when exchanging information between business partners. Cryptography is used to provide an irrefutable binding between the information, its originator and its recipient(s). I pay particular attention to fairness in the exchange of non-repudiation evidence and to the role of TTPs. I also discuss circumstances when the business context may provide incentives for cooperation. In Section 2.3 I survey work on contract-mediated interaction that concerns validation of actions for compliance with business agreements. Section 2.4 presents existing middleware support for non-repudiation and accountability. Section 2.5 concludes the chapter with a discussion of the relationship between the related work and this dissertation.

2.1 Definitions

This section defines the following key concepts: non-repudiation, various cryptographic primitives used in non-repudiation services, various trusted third party services, and the notion of well-behaved parties. I then present the perfect cryptography assumption and the protocol subversion model that holds for protocols described in this dissertation. The section concludes with the definition of common protocol notation and a note on the generation of unique protocol run identifiers.

2.1.1 Non-repudiation

Non-repudiation is the inability to subsequently deny an action or event. In the context of network security, ISO 7498-2 [ISO89] identifies non-repudiation as one of five main security services. The others are authentication, access control, confidentiality and data integrity. The ISO definition states that a non-repudiation service provides protection against false denial of involvement in a communication. In distributed systems, therefore, non-repudiation typically applies to evidence of the interaction between communicating nodes. In the case of component-based middleware we are interested in non-repudiation of interaction (communication) with and between distributed components.

Non-repudiation services support the generation, collection, maintenance and validation of evidence of an interaction. The evidence collected can be used to sustain or refute claims with respect to the occurrence or non-occurrence of an event or action and, therefore, to resolve disputes between parties to an interaction. As an example, consider sending a message as part of a transaction between the sender of the message and its recipients. To be able to subsequently prove that the interaction took place — that the message was sent and was received and by whom — we are typically interested in two types of non-repudiation:

- *non-repudiation of origin (NRO)*: irrefutable evidence that the message originated at the sender (generated by the sender or by a trusted third party (TTP) on their behalf); and
- *non-repudiation of receipt (NRR)*: irrefutable evidence that the message was received by the recipient(s) (generated by the recipient(s) or by a TTP on their behalf).

NRO and NRR protect against false denial of origin and receipt respectively. Other types of non-repudiation have been defined to meet the requirements of different interaction scenarios [Zho01]. For example, when a delivery agent is involved in transmission of a message from sender to recipient(s), the following may also be of interest:

- *non-repudiation of submission (NRS)*: irrefutable evidence that a message was submitted to a delivery agent for onward delivery to the message recipient(s) (generated by the delivery agent or by a TTP on their behalf); and

- *non-repudiation of delivery (NRD)*: irrefutable evidence that the message was delivered to the intended recipient(s) (generated by the delivery agent or by a TTP on their behalf).

In addition, I introduce the following notion:

non-repudiation of validation (NRV): irrefutable evidence of the validity or otherwise of application-specific message content as perceived by the recipient(s) (generated by the recipient(s) or by a TTP on their behalf). NRV can also be seen as the NRO of a decision on the application-level validity of a message.

Given the preceding definitions, I intend to develop the following non-repudiation services:

- *non-repudiable service invocation* that comprises: NRO, NRR and NRV of service request **and**, for any response, correlated NRO, NRR and NRV of service response; and
- *non-repudiable information sharing* that comprises: NRO, NRR and NRV of both access to and changes to the state of shared information.

Non-repudiation evidence can be generated either by using TTP-generated secure envelopes or using digital signatures. A TTP generates a secure envelope using a secret key known only to the TTP. In this case, only the TTP can generate and verify evidence. Other parties must unconditionally trust the TTP's assurance of the validity of evidence. In contrast, asymmetric (public key) cryptography is used to generate digital signatures. So, any party may use their private key to generate signed evidence. Any relying party may use the signing party's public key to verify the evidence. The symmetric key cryptography techniques used for secure envelopes are computationally more efficient than asymmetric cryptography. A significant disadvantage of secure envelopes is the unconditional reliance on a TTP for the generation, verification and maintenance of all evidence for as long as that evidence is required. Consequently, for non-repudiation, digital signatures are generally preferred to secure envelopes. In the remainder of this dissertation, I assume the use of a digital signature scheme of the type defined in Section 2.1.2.

2.1.2 Cryptographic primitives

This section defines the following cryptographic primitives: a secure hash function, a digital signature scheme, and a secure pseudo-random sequence generator [Gol99, Sch96]; and the notation for these primitives used throughout the dissertation.

A **secure hash function** has the following basic properties:

- *ease of computation*: given x , it is easy to compute $hash(x)$;
- *compression*: given a string of arbitrary length, the function produces a fixed length string as output (often called a message digest);
- *preimage resistance (one-wayness)*: given h , it is computationally infeasible to find x such that $h = hash(x)$;
- *2nd preimage resistance*: given x and $hash(x)$, it is computationally infeasible to find $y \neq x$ such that $hash(x) = hash(y)$; and
- *collision resistance*: if $hash(x) = hash(y)$, then with an effective probability of 1 $x = y$.

A **digital signature scheme** cryptographically authenticates digital information and is analogous to a physical signature on paper. The work presented here assumes a digital signature scheme based on public key cryptography. In public key cryptography, each user has a pair of keys: one public and one private. The public key can be distributed freely. The private key should only be known to its user. Key pairs have the following properties:

- it is computationally infeasible to compute a private key from its corresponding public key,
- a ciphertext generated using a user's public key can only be decrypted using the corresponding private key, and
- a ciphertext generated using a user's private key can only be decrypted using the corresponding public key.

It is this last property that supports digital signatures. Given a key pair and some input x , a digital signature scheme consists of two algorithms:

- the *signing algorithm*, where a signing party encrypts x using their private key to generate a signature s , and
- the *verification algorithm*, where a relying party uses the signing party's public key to decrypt s and obtain x .

When the relying party retrieves x from s they can confirm that s can only have been generated using the signing party's private key, verifying the digital signature. The verifiability property of a digital signature scheme ensures that a third party can resolve disputes about the validity of a digital signature without knowledge of the signing party's private key.

Unless stated otherwise, and without loss of generality, I assume that signature schemes are non-recoverable; that is, given a signature s over x , a relying party can only determine that s is a valid signature over x . They cannot learn any other useful information about x from the signature alone.

Throughout this dissertation, unless stated otherwise, the term signature means digital signature and signing means digitally signing.

A **secure pseudo-random sequence generator** generates a sequence of bits with the following properties:

- *pseudo-randomness*: the sequence is statistically random; and
- *unpredictability*: even with complete knowledge of the algorithmic or hardware generator and all previous bits of a sequence, it is computationally infeasible to predict the next bit of the sequence¹.

2.1.3 Trusted third party services

A TTP is a security authority trusted by other entities with respect to security related activities [ISO96]. Here, I use the classification in [Zho01] to give a brief overview of the following three types of TTP:

1. a certification authority (CA) that issues public key certificates,

¹There is a distinction between a secure random number and a "nonce" (number used only once). It is common to use nonces in security protocols to ensure the freshness of messages but a nonce is not necessarily random and it may be possible to guess the next value of a nonce.

2. a time-stamping authority (TSA) that provides trusted time-stamps on evidence, and
3. a guarantor TTP that supports, and may be directly involved in, the generation, exchange and verification of non-repudiation evidence.

A public key certificate is a digital credential that binds the subject of the certificate to their public verification key and thereby to their use of the corresponding private key. A certificate typically includes: the subject's public key, information related to the subject's identity, the period of validity of the certificate, and the algorithm to use with the verification key. A CA issues certificates to guarantee the authenticity of the associated public key and the binding of the key to the subject of the certificate. CAs sign certificates with their own public verification key to guarantee certificate integrity. Since private keys may be compromised, a CA is also responsible for maintaining certificate revocation information. X509 [HFPS99] is a widely used standard for CA-issued public key certificates.

A TSA is used to obtain a signed time-stamp on evidence as proof of the time of generation of the evidence. For example, given a signature $s = sig_A(x)$ by principal A over data x , the TSA could provide the following time-stamp: $\{T_g, sig_{TSA}(s, T_g)\}$. This time-stamp is the TSA's counter-signature over s and the time T_g — the time of generation of the counter-signature. The time-stamp serves as proof of the generation (or existence) of s at time T_g [ZG97]. As discussed in Section 2.2, trusted time-stamps can be critical to the subsequent verification of non-repudiation evidence. TSAs may also be used to certify the time of occurrence of some event or, more precisely, the time the TSA gained knowledge of the occurrence of some event.

Guarantor TTPs, or simply TTPs for short, are used to safeguard the interests of honest parties to an interaction. For example, a trusted delivery agent may be used to guarantee delivery of messages and to directly exchange non-repudiation evidence. A third party adjudicator may use non-repudiation evidence to resolve disputes between parties.

TTPs of whatever type may also be categorised by the extent of their involvement in an interaction.

- An *in-line TTP*, such as a message delivery agent, acts as an intermediary for all communication between the parties to an interaction.

- An *on-line TTP* is actively involved in every instance of a non-repudiable exchange though not necessarily in every communication. A TSA is an example of a TTP that should be on-line to time-stamp evidence but is not necessarily in-line with respect to the exchange of evidence between parties.
- An *off-line TTP* supports non-repudiation but is not involved in each exchange. For example, a CA is off-line for issuance of certificates.

I return to the role of TTPs in Section 2.2.

2.1.4 Well-behaved parties

Non-repudiation protocols specify the correct behaviour of the protocol participants. *Well-behaved* (or honest) parties adhere to the protocol specification. I make the following assumptions about communications between well-behaved parties and their cooperation in protocol execution.

- The communication channel between well-behaved parties provides eventual message delivery. That is, there is a known bound on the number of temporary network and computer related failures experienced by well-behaved parties.
- Each well-behaved party has persistent storage for messages. Minimally, well-behaved parties will ensure that messages are available for as long as is necessary to meet their obligations to other well-behaved parties.
- Well-behaved parties only send messages that comply with the specification of the protocol being executed. Similarly, they only process messages that are correct with respect to the protocol specification.

In this dissertation, “trusted”, “honest” and “correctly behaving” are synonyms for well-behaved. Participants that are not well-behaved — do not satisfy the above assumptions — are considered to be misbehaving. Such misbehaviour may be through intentional or accidental fault. TTPs are guaranteed to be well-behaved by definition.

2.1.5 Perfect cryptography assumption and protocol subversion model

I make the standard perfect cryptography assumption: that there exist implementations of the cryptographic primitives described in Section 2.1.2 that are secure against attack even if adversaries have complete knowledge of the algorithms used. Cryptographic primitives are not susceptible to cryptanalysis or brute force attack. For example, a ciphertext can only be deciphered with the appropriate decryption key.

Security protocols are traditionally analysed under the Dolev-Yao intruder model [DY83]. The Dolev-Yao intruder has control of the network between protocol participants and can attempt to subvert a protocol using the knowledge obtained from the messages that are exchanged. As shown in Figure 2.1a, in effect a protocol executes through the intruder. The

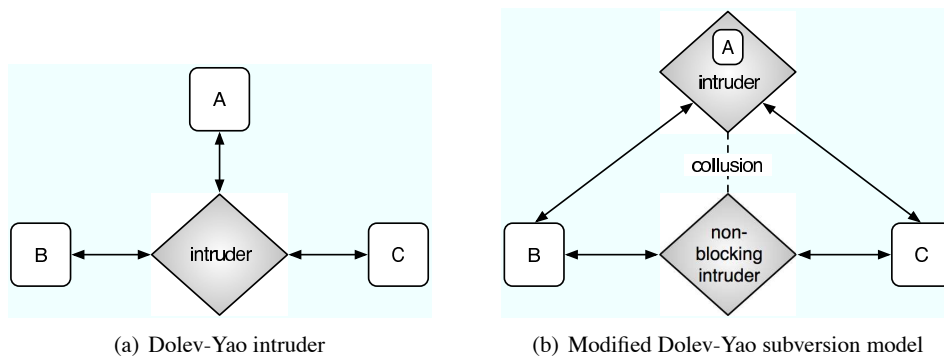


Figure 2.1: Protocol subversion model

intruder can remember all messages transmitted over the network, they can compose new messages from their knowledge, and they can remove or delay messages. The perfect cryptography assumption holds for the Dolev-Yao intruder and, therefore, the intruder cannot use cryptanalysis to subvert a protocol. However, they can decrypt and sign messages if they know the relevant key or keys.

Unless stated otherwise, I adopt the model of protocol subversion formalised by Cederquist and Dashti [CD04] that is based on a modification to the Dolev-Yao intruder. In this case, to model misbehaviour, a protocol participant may act as an intruder and may collude with a *non-blocking* Dolev-Yao intruder. The non-blocking intruder cannot block messages between well-behaved parties forever. Thus the assumption of eventual message delivery between well-

behaved parties holds in the presence of the non-blocking intruder. In particular, communications with and between TTPs will eventually succeed. Of course, the non-blocking intruder may still delay messages and relay messages out of order, etc. Figure 2.1b illustrates this model, where A misbehaves and both B and C are well-behaved. Note, apart from the well-behaved assumption with respect to TTPs, there is no restriction on the number of misbehaving parties in the model.

2.1.6 Notation

Table 2.1 shows the notation used throughout this dissertation for various cryptographic primitives and other protocol elements.

Notation	Definition
$h(x)$	a secure hash of x
$sig_P(x)$	principal P's signature over x (using P's private key)
$enc_P(x)$	encryption of x with P's public key
$rn_{[P]}$	a secure pseudo-random number/sequence with optional indication of the principal P who generated rn
$nrId_{[i]} [= f()]$	a unique identifier of a non-repudiation protocol run with optional index, i , to differentiate between 2 or more identifiers from related runs and the specification of an optional generating function, $f()$
T_g	time of generation of some information
$t_{TSA}(x) = \{T_g, sig_{TSA}(x, T_g)\}$	time-stamp on x generated by TSA as witness to the time of generation, T_g , of x
$P \rightarrow Q : x$	principal P sends x to principal Q

Table 2.1: Notation

2.1.7 Generation of protocol run identifiers

I assume that a protocol run identifier, $nrId$, is unique for an interaction context. That is, an $nrId$ need not be globally unique but must be sufficiently unique to unambiguously distinguish between different protocol runs in a given context. In non-repudiation protocols it is common to use a secure random number as an authenticator. A secure hash of the random number is released first as a commitment to the random number that is revealed later in the protocol. In this case, it is convenient to use the hash of the random number as the basis for the unique identifier. The hash could be contextualised by prepending some base URI to a URL-safe

Base64 [Jos03] encoding of the hash. Table 2.2 shows an example of the generation of a

Element	Value
rn	okUDiHMXVES0aAU5PTpBDW75Lo7pGmisnaWUAm7X+ck=
$h(rn)$	UrvmUq2Bs2a08d5ae7v3yZV398/elfFcRV0JxuuC37Q=
$nrId$	http://cs.nc1.ac.uk/nick.cook/UrvmUq2Bs2a08d5ae7v3yZV398_elfFcRV0JxuuC37Q=

Table 2.2: Example generation of a protocol run identifier

$nrId$ of this form given the Base64 representations of random number and hash. Note, the “/” character in $h(rn)$ is converted to a “_” in the URL-safe encoding for the $nrId$.

In this dissertation, I assume that the protocol initiator is responsible for generation of the $nrId$. I also assume that if a $nrId$ is not unique, then it will be rejected by well-behaved recipients and this rejection triggers re-start of a protocol with a newly-generated identifier.

2.2 Non-repudiation protocols

Chapter 1 established the need for accountability in high-value B2B interactions. As suggested in Section 1.3, non-repudiation protocols meet this requirement by exchanging evidence that prevents the subsequent denial of participation in an interaction. Chapter 1 also identified the need for flexibility in the choice of mechanism, or protocol, to achieve non-repudiation in order to adapt to a given business context. In this section I survey fundamental work on non-repudiation that is the basis for meeting these requirements. The range of available protocols — from non-repudiation without fairness, through fair non-repudiation with TTP, to probabilistic fairness — can be seen as an armory of mechanisms to exploit in an adaptable framework for non-repudiation services. First, I discuss the problem of the exchange of non-repudiation evidence. Then I review the various approaches to fair non-repudiation. A discussion of the role of TTPs is followed by consideration of scenarios in which fairness guarantees may be relaxed.

In Chapter 3 I show how the fundamental work discussed in this section can be used to provide flexible services for non-repudiable service invocation and non-repudiable information sharing. For example, I show how to extend the non-repudiation protocols presented in this section in order to integrate application-level validation of service invocations.

2.2.1 Exchange of non-repudiation evidence

To illustrate the non-repudiation problem, consider the exchange of a message with the sender's non-repudiation of origin evidence (NRO token) for the recipient's non-repudiation of receipt evidence (NRR token). Here we present a simple, voluntary non-repudiation protocol for the direct exchange of evidence between sender A and recipient B. Table 2.3 defines the non-

Token	Definition
<i>NRO</i>	$sig_A(nrId, A, B, m)$
<i>NRR</i>	$sig_B(nrId, A, B, NRO)$

Table 2.3: Tokens used in voluntary non-repudiation protocol

repudiation tokens used in the protocol. Figure 2.2 defines the protocol.

1 $A \rightarrow B : nrId, A, B, m, NRO, t_{TSA}(NRO)$
 2 $B \rightarrow A : nrId, A, B, NRR, t_{TSA}(NRR)$

Figure 2.2: A voluntary non-repudiation protocol

In step 1, A sends B a protocol run identifier, the participant identifiers, the message m , the NRO token and a time-stamp over the token. The NRO token consists of A's signature over the identifiers and over message m . The NRO token binds the identifiers and m to A. It constitutes proof of origin of m and shows that m was sent in the protocol run identified by $nrId$. In step 2, B replies with the identifiers, the NRR token and a time-stamp over the token. The NRR token consists of B's signature over the identifiers and over A's NRO token. The NRR token binds the identifiers, the NRO token and, therefore, m to B. It constitutes proof of receipt by B of m and A's NRO token and shows that this information was received in the protocol run identified by $nrId$. Assuming that B behaves correctly and we have eventual message delivery between A and B, the voluntary protocol in Figure 2.2 is sufficient to guarantee the exchange of non-repudiation evidence. For example, the protocol can be used for non-repudiable delivery of a message m from any sender to a TTP recipient. The sender may misbehave but they cannot disadvantage the recipient because their misbehaviour simply results in the loss of entitlement to a receipt. In contrast, if B fails to cooperate in some way, then A may provide m and the NRO token without obtaining B's receipt. This is the *selective receipt problem*. It is characteristic

of the general problem of achieving the fair exchange of electronic items when parties do not necessarily trust each other.

2.2.2 Non-repudiation and fairness

Asokan [Aso98] provides one of the most widely accepted informal definitions of fairness: that a system is fair if it does not discriminate against correctly behaving parties. More recently, Markowitch et al [MGK02] provide an overview of the evolution of the notion of fairness in exchange and the various definitions that have been proposed. To clarify the position, they propose the following three mandatory properties that must be satisfied for an exchange protocol to be considered secure and fair.

Viability is the property that, if both parties are well-behaved, there exists an execution of the protocol that results in successful exchange.

Timeliness is the property that, with the quality of communication channels fixed, there is always a point in the protocol that can be reached in a finite amount of time when parties can stop the protocol without compromising fairness.

Fairness is the property that: “the communication channel’s quality being fixed, at the end of the exchange 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.”

In addition they define non-repudiability as the optional property of a secure exchange protocol that:

“it is impossible for a single entity, after execution of the protocol, to deny having participated in a part or the whole of the communication.”

There is an extensive literature on the problem of fair exchange and its solution. Kremer et al [KMZ02] provide a detailed survey of fair non-repudiation protocols. Pagnia et al [PVG03] provide a wider review of fair exchange in general. A significant distinction between the various proposed protocols is the extent of involvement of a guarantor TTP. For protocols that

provide a deterministic guarantee of fairness, there must be some TTP involvement. Different protocols have been proposed with either in-line, on-line or off-line TTP involvement. In non-deterministic approaches, there is no TTP involvement but the fairness guarantee is either probabilistic or relies on assumptions about the relative computing power of protocol participants. I discuss the role of TTPs further in Section 2.2.3. In this section I use examples from the literature to illustrate the different approaches to fair non-repudiation. The first protocol relies on an in-line TTP to solve the selective receipt problem. The second example is an optimistic protocol to solve the same problem using an off-line TTP. An overview of the alternative (non-deterministic) gradual and probabilistic approaches follows the protocol descriptions. Finally, I briefly describe work on the relatively neglected area of fault-tolerance and fair exchange.

2.2.2.1 Coffey-Saidha protocol with in-line TTP

The Coffey-Saidha protocol [CS96] uses an in-line TTP to solve the selective receipt problem. The protocol exemplifies the problem of guaranteeing a receipt to a message sender without disadvantage to a message recipient. Section 3.2 presents my extensions to this protocol to provide fair and validated non-repudiable service invocation, and to support exception handling for timely termination.

As in Section 2.2.1, A wishes to send a message, m , to B. B requires NRO of the message and A requires NRR. In the protocol, A and B do not communicate directly but send all messages via the in-line TTP. Here I present a version of the protocol that takes account of improvements suggested by Zhou and Gollman [ZG97]. Table 2.4 defines the tokens used in

Token	Definition
NRO	$sig_A(nrId, A, B, m)$
NRR	$sig_B(nrId, A, B, h(NRO))$

Table 2.4: Tokens used in Coffey-Saidha in-line TTP protocol

protocol. Figure 2.3 shows the protocol. There follows a step-by-step description.

In step 1: A sends the TTP the following items: a unique identifier, $nrId$; the participant identifiers, A and B; the message, m ; A's NRO token; and a time-stamp over the NRO token witnessed by a TSA. All items are encrypted with the TTP's public key. This prevents

1	A	\rightarrow	TTP	$:$	$enc_{TTP}(nrId, A, B, m, NRO, ts_{TSA}(NRO))$
2	TTP	\rightarrow	B	:	$nrId, A, B, h(NRO)$
3	B	\rightarrow	TTP	:	$enc_{TTP}(nrId, A, B, NRR, ts_{TSA}(NRR))$
4	TTP	\rightarrow	B	:	$nrId, A, B, m, NRO, ts_{TSA}(NRO)$
5	TTP	\rightarrow	A	:	$nrId, A, B, NRR, ts_{TSA}(NRR)$

Figure 2.3: Coffey-Saidha in-line TTP fair exchange protocol

B from obtaining access to any of the information before they provide NRR of the message. If the TTP finds that $nrId$ is not unique, an appropriate response will be generated to prompt A to restart the protocol with a newly generated identifier. Otherwise, the protocol proceeds to step 2.

In step 2: the TTP sends $nrId$, the participant identifiers, and a hash of A's NRO token. The hash on the NRO token is the TTP's commitment to provide B with the evidence from which the hash and the NRO token were generated. That is, the TTP will deliver m and NRO to B in step 4. B is now able to generate a signature over the items provided. This signature serves as NRR evidence because $sig_B(nrId, A, B, h(NRO))$ is cryptographically bound to NRO and NRO is, in turn, cryptographically bound to the message, m .

In step 3: B responds with: $nrId$, the participant identifiers, their NRR token, and a timestamp over the NRR token witnessed by a TSA. All items are encrypted with the TTP's public key. This ensures that A only obtains NRR if the protocol runs to some form of successful termination. It is safe for B to send the receipt to the TTP before obtaining m because the TTP is, by definition, committed to providing m .

In step 4: the TTP sends B m , the NRO token and associated information provided by A in step 1.

In step 5: the TTP sends A the NRR token and associated information provided by B in step 3.

Steps 4 and 5 may execute in parallel without loss of fairness to either A or B.

On completion of the protocol in Figure 2.3, A has NRR for m . In return, B has m and NRO for m . Fairness is guaranteed because the TTP controls the release of evidence to A

and B, and because the TTP can support exception handling. Exception handling can provide timely termination of an exchange because the TTP can guarantee to well-behaved parties that they can obtain the information they expect from an exchange or that the exchange can be aborted without loss of fairness. Coffey and Saidha did not specify the exception handling sub-protocols, which is one of the extensions to this work that I present in Section 3.2. To illustrate some of the issues involved in exception handling, we observe that fairness can be guaranteed to A and B if:

1. the protocol in Figure 2.3 completes normally, or
2. B chooses not to engage in the main protocol by not responding to the message sent by the TTP in step 2 because B does not receive any useful information about m or the NRO token in step 2, or
3. the exchange is aborted when the main protocol has progressed no further than step 3 because up to and including execution of step 3 neither A nor B has received any useful information about their respective items², or
4. the exchange is completed successfully after execution of step 3.

Execution of step 3 is the pivotal point in execution of the protocol. Once the TTP has received the receipt from B in step 3, the TTP has all the information necessary to complete the exchange. Thus the TTP has the ability to complete the exchange for both A and B. The TTP may also satisfy requests to abort the exchange because the TTP has not yet released the critical information that either A or B expect from the exchange (A's message for B or B's receipt for A). After execution of step 4, the TTP can and must guarantee that all expected items are available to both A and B.

The encryption of NRR evidence in step 3 of the protocol illustrates the care that must be taken to achieve fair exchange. For example, without encryption of the protocol message at step 3, A could mount the following attack: (i) intercept the protocol message to acquire NRR, and (ii) delay delivery of the message to the TTP for long enough to ensure that the TTP agrees

²As an example, A may wish to abort an exchange because B fails to respond to the message sent by the TTP in step 2.

to a request to abort the exchange. This would allow A to obtain a receipt but deny B access to m and NRO.

2.2.2.2 Optimistic fair exchange

Optimistic fair exchange protocols use an off-line guarantor TTP that only intervenes to handle exceptions. Initially, the participants behave optimistically by attempting the direct exchange of items between themselves. If this direct exchange completes successfully, then the TTP is not involved. In case of delay or suspected misbehaviour, a request can be made to the TTP to abort or successfully complete the exchange and, thereby, guarantee fairness to well-behaved parties.

The Asokan, Shoup and Waidner (ASW) contract signing protocol [ASW98] is one of the earliest examples of an optimistic fair exchange protocol. Their direct exchange protocol requires four steps for the successful fair exchange of signatures on an agreed text between two parties. If the direct exchange does not complete, then, upon request, a TTP can issue an abort token or a replacement for the signed text. If the replacement is issued, then the involvement of the TTP is apparent because they counter-sign the agreed text. That is the replacement is equivalent but not identical to the expected signed text. Markowitch and Kremer [MK01] subsequently proposed an alternative fair non-repudiation protocol that uses a novel, transparently recoverable signature scheme to render any TTP involvement transparent with respect to the evidence generated. That is, the TTP's involvement is not apparent from the evidence generated regardless of how the exchange concludes. The Markowitch and Kremer protocol also requires four steps to complete the direct exchange between two parties. More recently Wang [Wan05] published a generic fair non-repudiation protocol with transparent TTP. Wang's protocol is generic because any signature scheme can be used to generate non-repudiation evidence. Further, the protocol is the most communication-efficient of known fair non-repudiation protocol. It requires just three steps for the direct exchange between two parties. I use this protocol as an illustration of the optimistic approach to fair exchange. Section 3.2 shows how to combine executions of the protocol to provide fair, non-repudiable service invocation.

As with the Coffey-Saidha protocol, the aim of Wang's protocol is for a sender A to ex-

change a message m , and its NRO, for NRR from B. A begins optimistic direct exchange by sending input information to B with which B can generate a receipt for m . B then responds by sending the generated receipt to A. Finally, A sends B the information necessary to recover m . So, if successful, the exchange completes in three steps. The key insights behind the protocol are that:

1. A's interests are safeguarded because they obtain a receipt for m before providing B with the information they require to recover m , reversing the selective receipt problem, and
2. B's interests can be safeguarded if the input for the receipt provided by A is cryptographically bound to m through A's evidence of origin and, in the event of A not providing the information that B requires, a TTP, and only the TTP, can enable B to recover m from A's input.

Token	Definition
kA	a secret key generated by A to encrypt m
$mCipher$	$enc_{kA}(m)$ — the encryption of m using kA
$nrId$	$h(A, B, TTP, h(mCipher), h(kA))$ — $nrId$ generating function
$kACipher$	$enc_{TTP}(nrId, kA, rn_A)$
NRO	$sig_A(nrId, kACipher)$
NRR	$sig_B(nrId, kACipher)$

Table 2.5: Tokens used in Wang's optimistic non-repudiation protocol

Table 2.5 defines the tokens used in the protocol. Figure 2.4 shows the main direct exchange

1 $A \rightarrow B : nrId, A, B, TTP, mCipher, h(kA), kACipher, NRO$
 2 $B \rightarrow A : nrId, NRR$
 3 $A \rightarrow B : nrId, kA, rn_A$

Figure 2.4: Wang's optimistic non-repudiation protocol

between between A and B. For brevity, time-stamps are omitted from the protocol description.

To start the protocol A generates the two encrypted tokens: $mCypher$ and $kACipher$. As shown in Table 2.5, $mCypher$ is the encryption of m using the secret key kA that is newly generated by and known only to A. A generates $kACipher$ by using the TTP's public key to encrypt the secret key, the protocol run identifier and a newly generated random number.

The identifier $nrId$ is cryptographically bound to m through the hash of $mCipher$ used in the generating function for $nrId$. Only the TTP can decrypt $kACipher$ but any other party can verify that it is an encryption of $nrId$ and kA with some random number. The protocol now proceeds as follows.

In step 1: A sends B the protocol run identifier, the participant identifiers (including the TTP's identifier), the encrypted tokens, a hash of the private key and the NRO token. The NRO token cryptographically binds $kACipher$ to $nrId$ and, therefore, to $mCipher$ and m . Given these items, B can generate a receipt by signing the same items as A to produce the NRR token. At this point in the protocol, B knows that either:

1. A prepared $kACipher$ correctly and, therefore, that A can produce the corresponding kA and rn_A or that the TTP can recover these items from $kACipher$ for B to enable B both to verify the NRO token and obtain m ; or
2. A has provided an invalid $kACipher$, in which case the NRR token that B provides by signing $nrId$ and $kACipher$ will not form a valid receipt for A.

In step 2: B sends the NRR token to A. It is safe for B to do this before receiving m because, as stated previously, the NRR token is only valid and useful to A if the input provided by A is also valid. So, if the receipt provided to A is valid, then A must have provided B with the information necessary to complete the exchange through the TTP, should A cease to cooperate.

In step 3: A sends B kA and rn_A . These items enable B to verify the integrity of all the evidence provided by A. Assuming A's evidence is valid, B can now decrypt $mCipher$ and obtain m . If any evidence is invalid, then the cryptographic linkage between evidence is such that the receipt provided to A is also invalid and A has gained no advantage over B.

Wang [Wan05] defines exception handling sub-protocols for the abort of an exchange or the recovery of items through the TTP. A is entitled to request that an exchange is aborted after step 1 of the optimistic exchange. A may initiate an abort request because B fails to initiate

step 2 in a timely fashion. The TTP satisfies an abort request from A by issuing a signed abort token for the identified exchange. Similarly, B may seek timely termination of the exchange by requesting that the TTP recover the secret key k_A and random number rn_A from the $kACipher$ token provided by A in step 1. B can use the recovered items to obtain m and A's NRO token. In either case, the request for abort or recovery must contain the evidence necessary for the TTP to determine whether the request should be fulfilled. For example, the TTP will only comply with a recovery request from B if the request includes a valid receipt for A and the exchange has not already been aborted. Following the first valid request for abort or recovery, the TTP records the status of the identified exchange as either *aborted* or *recovered*. Once the TTP has set the status of an exchange they will forever respond in the same way to any subsequent request from A to abort the exchange or from B to recover items. That is, the TTP will not recover items for B if an exchange has been aborted as the result of a prior request from A. In this case, the TTP will issue an abort token to B. Similarly, the TTP will not satisfy a request from A to abort an exchange that has been recovered. In this case, the TTP will send B's receipt to A.

Wang [Wan05] provides a detailed description of the security properties of the protocol. Here I briefly describe the protection against attempts to cheat by either A or B. A can attempt to cheat in two ways: (i) they can provide B with an invalid $mCipher$ or invalid $kACipher$, or (ii) they can decline to send the final message. As previously discussed, invalid input from A will lead to the generation of a useless receipt from B. A only achieves non-repudiation of receipt if they provide valid input to B in the first step of the protocol. If A does not send the final message, then B can obtain k_A and rn_A from the TTP in return for their NRR token. The TTP will provide k_A and rn_A unless A has requested that the exchange be aborted. If the exchange has been aborted, then the TTP provides B with a signed abort token that can be used to repudiate receipt of m . B can only attempt to cheat by failing to provide A with a valid receipt and requesting that the TTP provide k_A and rn_A . However, the TTP only provides these items to B if B provides the TTP with a valid NRR token and all other items that are necessary for the TTP to verify their consistency and integrity. That is, the TTP will only allow B to recover the message m if B provides the information necessary to construct a valid receipt for A and the exchange has not already been aborted. In conclusion, neither A or B can gain advantage

by attempting to cheat. The optimistic exchange is only likely to fail because of timeliness constraints. In the normal case, the exchange will complete as a direct exchange between A and B without the need to involve the TTP. Further, if the TTP is required to successfully complete an exchange, the evidence provided by the TTP to either A or B is exactly the same as if the TTP had not been involved — the transparency property.

2.2.2.3 Gradual and probabilistic approaches

Prior to the development of fair non-repudiation protocols with guarantor TTP, gradual exchange protocols were proposed for the fair exchange of secrets and contract signing [Ted85, EGL85, BCDv88]. In gradual exchange, parties release bits of items to ensure that at any given point in the exchange the amount of knowledge on each side is approximately the same. The exchange continues until each party has sufficient information to construct the secret or one of the parties ceases participation. If each party has the same or equivalent computing power, then the exchange should be fair. The main drawbacks of this approach are the large number of communication steps typically required to successfully complete an exchange and the impractical assumption of equivalent computing power.

An alternative approach to fair exchange without guarantor TTP is to adopt a probabilistic notion of fairness: that, at the end of an exchange, there is an overwhelming probability that either all involved parties obtain their expected information or none of them does.

Markowitch and Roggeman developed the first probabilistic non-repudiation protocol without TTP [MR99]. The aim of the protocol is for a message sender, A, to exchange their message m and NRO for NRR from the recipient, B. It is an iterative protocol with the characteristic that, except at the last iteration, neither A nor B is more privileged than the other. The message m is decomposed into n pieces, where n is a secret random number known only to A. The protocol proceeds through n rounds. In each round A sends a piece of m and B responds with an acknowledgement of receipt. The cryptographic decomposition of m is such that B needs all n pieces to reconstruct m . B's acknowledgement of receipt for the n th piece is the NRR token that A requires. Therefore, there is no incentive for A to stop the protocol before it completes (unless they suspect B of cheating). The problem is to reduce the probability that B can obtain

the message and NRO without providing NRR. A and B agree a time limit t for completion of each protocol round. This time limit is shorter than the computation time needed for B to reconstruct m from the information provided by A. That is, it takes longer than time t for B to determine that A has provided the n th piece of m . After initiating a round, A waits for time t to receive B's acknowledgement. If A does not receive an acknowledgement within time t , they abandon the protocol. Therefore, B must continue to send acknowledgements within time t to prompt A to provide the next piece of n . Since the last iteration is when B gains significant advantage, there is no benefit to B to stop the protocol before completion of the penultimate iteration. B can cheat if they guess n and fail to provide the final acknowledgement. The only way for B to guess n is to compute m from the pieces received to date, which takes longer than time t . Therefore, there is an incentive for B to send the final acknowledgement and to be sure that they have reached the n th round of the protocol. If A does not initiate a further round within time t , then B knows that either A has stopped the protocol prematurely, the last round has not been reached, or that they have reached the n th round and can now compute m . B's interests are safeguarded, regardless of A's behaviour, because B only provides a valid receipt to A if A provides all n pieces of m . The probability that B can gain advantage over A is inversely proportional to the number of protocol rounds chosen by A — the probability that the protocol is fair for A is $1 - \frac{1}{n}$. For example, there must be 10 rounds (20 messages) for a 90% guarantee that the protocol is fair for A.

The probabilistic approach has the advantage that there is no need for a guarantor TTP and that the risk of loss of fairness is known, and parameterizable by A. Disadvantages when compared to TTP-based protocols are the increased communication that is typically required and an increase in protocol complexity.

Avoine et al [AGGV05] have taken a hybrid approach to multi-party fair exchange that provides the equivalent of a deterministic guarantee of fairness with an honest majority of participants. If there is no honest majority, their protocol ensures an arbitrarily low probability of loss of fairness. As shown in Figure 2.5, the protocol assumes that a *virtual* TTP can be constructed from tamper-resistant³ hardware security modules hosted by untrusted hosts at

³Avoine et al use the term “tamper-proof”.

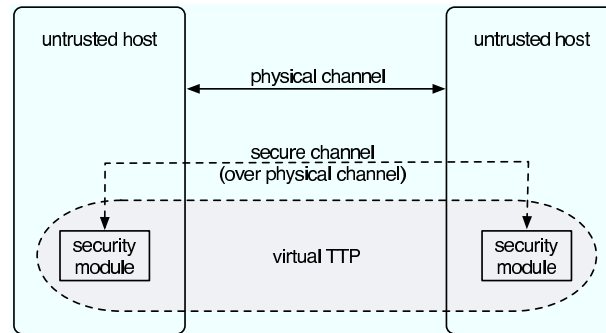


Figure 2.5: Security module-based *virtual TTP*

protocol participants. Security modules can be trusted by other security modules and by any participant host. Hosts are untrusted. A malicious host can behave as a Dolev-Yao intruder with respect to their security module. An underlying assumption is that the security modules are perfectly tamper-resistant — any attempt by a host to tamper with a security module can be detected and will result in a security module that ceases to communicate. The security modules are assumed to be black-box cryptographic engines with cryptographic keys that are not available to their hosts. Hosts are said to be honest if they participate correctly in the protocol and do not disrupt communication with their security module. Dishonest hosts misbehave in some way. For example, they may not follow the protocol correctly or they may disrupt communication with their security module. Given these assumptions, a virtual TTP can be constructed if there is an honest majority of hosts and, therefore, the protocol provides a guarantee of fairness that is equivalent to that provided by protocols with a genuine TTP.

The security of Avoine et al’s protocol and, in particular, the honest majority fairness guarantee is founded on an extension of the perfect cryptography assumption to include perfect tamper resistance of hardware security modules. Under this assumption, it is impossible for a dishonest host to undetectably compromise their security module and masquerade as an honest host. However, Anderson et al [ABCS05] have demonstrated both physical and software-based attacks on such modules. A relatively new class of practical attacks involves the exploitation of vulnerabilities in device application programmer interfaces (APIs) [CB02]. API vulnerabilities are analogous to security protocol vulnerabilities, can be mounted at the interface of the device to its hosting environment (or an intruder into that environment), and are arguably easier to

exploit than the physical protection of the hardware devices. In practice, therefore, tamper-resistance is a probabilistic notion — there is a non-zero probability that tamper-resistant hardware can be compromised without detection. Thus, apparently honest hosts may have compromised security modules and the fairness guarantee with honest majority is probabilistic.

In practice, genuine TTPs are also vulnerable to attack. However, a distinction can be drawn between reliance on security measures adopted by a genuine TTP that is liable for the failure of those measures and the need to rely on the invulnerability of security modules on (undetectably) dishonest hosts. Issues such as this highlight the need to consider the wider context when adopting solutions to non-repudiation problems. Ultimately, non-repudiation evidence is collected in order to prove to some dispute resolution authority that an action or event occurred. The choice of mechanism to achieve non-repudiation will not only be determined by technical trade-offs between different approaches but also by the regulatory environment. This underlines the need for flexible non-repudiation services that can adapt to different environments.

The decision to use an irrevocable key signature scheme will depend, in part, on the availability of implementations and their compliance with the security policies of the interacting parties. The fact that there is a choice reinforces the need for flexible middleware support for non-repudiation.

2.2.2.4 Fault tolerance and fair exchange

In most of the work on fair exchange and non-repudiation no distinction is made between local system failure and non-cooperation in a protocol. While the literature often states communication model assumptions, the fault-tolerance of an exchange protocol is not generally considered. Notable exceptions are the work of Liu et al [L NJ00] and Ezhilchelvan and Shrivastava [ES05].

Ezhilchelvan and Shrivastava define a fault-tolerant fair exchange protocol as a protocol that preserves fairness for honest participants despite failures of the participant's node of the assumed type. They provide a systematic treatment of two-party fair exchange with guarantor TTP under a variety of assumptions with respect to user misbehaviour, node failures and communication delays. There is also a critique of work by Liu et al and, in particular, the subtleties

of message logging for crash tolerance.

In addition to consideration of fair exchange under standard malicious user assumptions, Ezhilchelvan and Shrivastava introduce the restricted abuser model. Restriction on abuse is achieved by relying on a trusted (tamper-resistant) subsystem in the user's host. This is equivalent to Avoine et al's security module that resides on a dishonest host. The previous remarks with respect to tamper-resistance also apply here and, in practice, the notion of restricted abuser is probabilistic. There is a non-zero probability that abuse by a user cannot be restricted in such a way as to prevent undetected access to the cryptographic keys of tamper-resistant hardware.

The implementation of fault tolerant fair exchange protocols is beyond the scope of this dissertation and will be the subject of future work (see Chapter 6).

2.2.3 Non-repudiation and the role of TTPs

The role of TTP as guarantor of fairness is central to much of the work on fair exchange. Even and Yacobi [EY80] provide the fundamental insight that deterministic fair exchange is impossible without a guarantor TTP.⁴ Their proof is by contradiction. Two communicating parties, A and B, wish to compute a mutual signature on an agreement ($sig_{A,B}$). In order to compute $sig_{A,B}$, A and B must engage in the iterative exchange of information. Assume that there exists a system such that it never happens that one party can compute $sig_{A,B}$, while the other cannot. Once A has sufficient information to compute $sig_{A,B}$, B must also have sufficient information and vice versa. After n communications, but not before, assume A has sufficient information to efficiently compute $sig_{A,B}$. By definition of the system, B transmits the n^{th} communication and, therefore, the first time B has sufficient information is after n' communications where $n' \neq n$. However, this contradicts the system definition. This is similar to an instance of the iterative Prisoner's Dilemma when there is insufficient incentive to cooperate (see Section 2.2.4). The impossibility result for deterministic fair exchange is also supported by arguments that relate the problem to the FLP impossibility result for distributed consensus [FLP85, PG99, ES05].

As seen in Section 2.2.2, the impossibility result and the desire to eliminate the potential bottleneck of a guarantor TTP have led protocol developers to concentrate on the reduction

⁴Even and Yacobi's definition of a Public Key Agreement Scheme, in which they use the term "judicator" to mean "a third honest party", is essentially an expression of the fair contract signing problem.

of the involvement of the TTP. The main motivation for the optimistic approach to deterministic fair exchange is to significantly reduce active involvement (to zero in the normal case). Probabilistic approaches eliminate the need for a guarantor TTP altogether.

The focus on guarantor TTP is understandable. However, in the case of non-repudiation (with or without fairness guarantees), the involvement of a TSA can be as, if not more, significant. This is because non-repudiation relies on the long-term integrity of signed evidence.⁵ If signatures cannot be verified, then the signing party can subsequently deny an action or event. Verification can be compromised if a digital signature scheme allows for revocation of signing keys, as is the case with common schemes such as those based on X509 public key infrastructure.

It should be noted that not all fair exchange protocols provide non-repudiation. There are circumstances when fair exchange is a requirement but it is not necessary to prevent subsequent deniability of the exchange. In this case, digital signatures may be used to ensure the integrity and, usually, the authenticity of information. Therefore, the relying party only needs to determine the validity of a signature, and its associated signing key, at the time of verification — during protocol execution. The problem of revocation can be dealt with by reference to CA-provided certificate revocation lists, copies of which may be available locally.

To illustrate the problem of key revocation in non-repudiation, consider the Coffey-Saidha protocol without trusted time-stamps on the signed evidence. The protocol could complete successfully with both A and B having received the evidence and items they expect. Then, some time after protocol completion, a misbehaving party, for example A, could claim that their signing key had been compromised and revoke the key and associated certificate by communication with the relevant CA. Without evidence of the time of use of A's signing key, B is unable to prove that A signed their NRO token before revocation and A can repudiate the evidence. As suggested in Section 2.1.2, and as shown in the protocol description in Section 2.2.2, the traditional solution to the key revocation problem is to use a TSA to apply trusted time-stamps to signed evidence. If a trusted time-stamp is used, then B can prove that A's key was valid at time of use (as evidenced by the time-stamp). The evidence remains non-repudiable even if A

⁵Where long-term means beyond the execution of the given exchange, though typically time-limited by some non-repudiation policy and/or interaction agreement.

subsequently revokes their signing key.

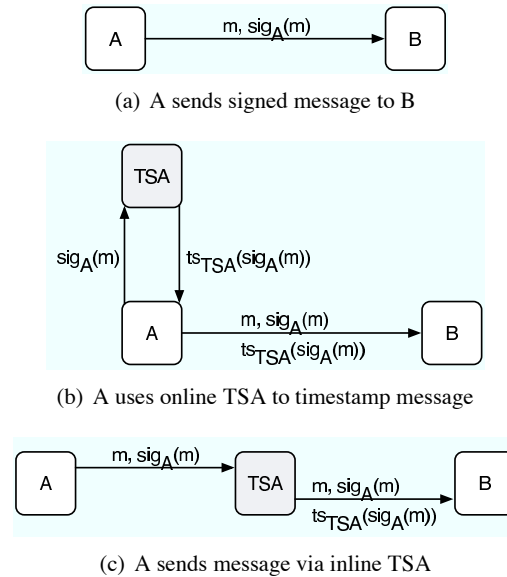


Figure 2.6: Signed messages and time-stamping

To summarise, Figure 2.6a shows A sending a signed message to B. To ensure non-repudiation, if A's signing key is revocable, the message must include a time-stamp signed by a TSA. Figure 2.6b shows how this can be achieved using an on-line TSA. Alternatively, for fair non-repudiation protocols with in-line TTP, the guarantor TTP may also act as TSA and apply a time-stamp before relaying the message from A to B, as shown in Figure 2.6c.

The requirement for an on-line TSA can be expressed as an additional impossibility result:

If signing keys are revocable without reference to the relying party,⁶ then non-repudiation is impossible without an on-line TSA to time-stamp signed evidence.

This impossibility is a consequence of key revocation in signature schemes and the requirement to prevent the subsequent deniability of signed evidence. It therefore holds irrespective of exchange protocol, communication model etc.

The problem of revocable keys has led to some work on novel signature schemes that make signing keys irrevocable. Zhou and Lam [ZL99] propose a temporary certificate scheme that uses two types of signing keys: a long-term, revocable key certified by a CA; and short-term,

⁶For example, A may revoke their key by contacting their CA without reference to B (the relying party).

irrevocable, self-certified keys. A temporary certificate assigns a lifetime to the associated short-term signing key. The temporary certificate is signed using the user's long-term key and counter-signed by a TSA. Any signatures generated using a short-term key are therefore non-repudiable for the TSA-witnessed lifetime of the temporary certificate. The signing party incurs the risk that their short-term key may be compromised. However, because the key cannot be revoked, the relying party is assured that signed evidence is valid. The scheme eliminates the need for an on-line TSA because temporary certificates can be generated off-line before transactions take place. The scheme is suitable for high-volume, relatively low value transactions for which the signing party will accept an adjustable degree of risk of compromise (see [ZL99]).

One-way sequential link signatures can be used to ensure that signing keys are only revocable with reference to the relying party and, therefore, maintain the validity of non-repudiation evidence without TSA involvement [Zho02]. One-way hash functions are used to link signatures. Any change to the chain of links is detectable. The signing party may revoke a signing key by sending the first and last signature in the link to the relying party for counter signature. Thus, the signing party can only deny signatures that have been generated with their revoked key but that are not in the counter-signed link.

Another scheme combines *forward secure signatures* for signature validation without trusted time-stamps and *refreshable certificates* to avoid reference to a CA for certificate revocation [ZBD03]. Forward secure signature schemes divide the lifetime of a key pair into time periods. At the end of a time period, the signing party derives a new signing key from the current key using a one-way function and then erases the current key. The keys are cryptographically bound to a given time period. Therefore, the relying party does not need a trusted time-stamp to verify their validity. Refreshable certificates use a one-way hash chain to maintain the validity of public-key certificates without CA involvement for revocation in a similar way to sequential link signatures.

2.2.4 Repeat business and the relaxation of fairness guarantees

The work discussed in Section 2.2.2 demonstrates the problems of guaranteeing fair exchange of non-repudiation evidence. It is therefore of interest to consider circumstances in which it may be possible to relax fairness guarantees. For example, the business context may provide strong incentives for cooperation, providing a form of *extra*-protocol probabilistic fairness as opposed to guaranteeing fairness at the level of each business message exchange.

A single instance of fair exchange is comparable to one-shot of the iterative Prisoner's Dilemma game [Axe90]. In the iterative Prisoner's Dilemma, two players have the choice of whether to cooperate or defect on each move. If both cooperate, they each receive the *reward for mutual cooperation*. If both defect, they each receive the *punishment for mutual defection*. If one defects and the other attempts to cooperate, the defector receives the *temptation to defect* and the other player receives the *sucker's payoff*. The payoffs may be different for each player but the following relationships hold for each player's payoffs:

$$\text{temptation } (T) > \text{reward } (R) > \text{punishment } (P) > \text{sucker's payoff } (S); \quad \text{and } R > \frac{T + S}{2}$$

The best outcome for an individual player is to obtain the temptation for defection. The worst outcome is to receive the sucker's payoff. The reward for mutual cooperation is greater than the punishment for mutual defection and mutual cooperation is better than taking turns to exploit each other — the even chance of exploitation and being exploited. The only communication between players is through the sequence of their behaviour. Thus the influences on the game are the history of an interaction and the prospect of future reward.

It has been shown that there is no incentive to cooperate in the one-shot Prisoner's Dilemma because there is no future to influence. The worst accumulated outcome of a defection in the one-shot game is punishment, which is still better than the sucker's payoff. In fact, there is no incentive to cooperate during any interaction of a known finite number of rounds. In the last round there is no future to influence and the incentive is to defect. In the penultimate round the incentive is to defect in anticipation of an opponent's defection in the final round. This reasoning leads to an unravelling of a game of known finite length back to mutual defection

in the first round.⁷ However, Axelrod has shown that in a game of indeterminate length there may be good reason to cooperate [Axe90]. Further, the incentive to cooperate can hold with no other influence on the game than the prospect of mutual cooperation. The future influences the present and cooperation emerges. The requirements for cooperation are that the players are sufficiently likely to interact again to have a stake in future interaction and that the future is a significant factor in accumulated payoffs. Axelrod provides many examples of real-life Prisoner Dilemma interactions where cooperation emerges. In business, the continuing relationship can be regarded as an “enforcer” that prevents defection and thereby avoids the need to resort to costly litigation. That is, B2B collaborations are typically based on continuing relationships that favour cooperation. However, even in the presence of cooperation, the subsequent undeniability property of non-repudiation may be required. In this case, it may be sufficient to assume cooperation in the collection of non-repudiation evidence and to resort to dispute resolution if cooperation ceases.

Work on the Prisoner’s Dilemma may provide a formal basis to explore the influence of repeat business and reputation on the behaviour of parties to an interaction. Issues that arise include: how to reliably attach reputations to parties, how to make decisions about the mechanisms to deploy based on reputations, and how to ensure that the prospect of repeat business influences the current interaction. It is beyond the scope of this dissertation to determine how judgements of this kind can be made. The problem is to determine whether the risk of non-cooperation is sufficient to incur the costs of guaranteeing fairness. In any case, the realisation that there are incentives to cooperation reinforces the need for a flexible non-repudiation framework. It should be possible to elect to execute the equivalent of the simple non-repudiation protocol shown in Figure 2.2 when the risk of selective receipt is low and, equally, to choose a protocol with appropriate fairness guarantees when there are insufficient incentives to cooperation.

⁷As noted previously, this is similar reasoning to Even and Yacobi’s proof of the impossibility of deterministic fair exchange without guarantor TTP.

2.3 Contract-mediated interaction

My work addresses the need for systematic support for non-repudiation in B2B interactions. As discussed in Section 1.2, there is also a need for higher-level services to determine whether business partner actions comply with business agreements. This compliance must be confirmed at run-time and the validation of actions must also be non-repudiable. Validators should be accountable for the outcome of their validation processes. As previously stated, it is therefore a requirement on the proposed non-repudiation middleware that validation with respect to agreements can be triggered during an interaction and that the outcome of validation integrated with the generation of non-repudiation evidence. This section considers higher-level validation services for which the middleware can provide a supporting non-repudiation infrastructure.

There has been considerable work recently on the use of business contracts to derive rules to mediate B2B interactions. The basic premise is that real-world business partnerships are governed by natural language business contracts. Therefore, the corresponding electronic B2B interactions should be regulated by an electronic contract that is somehow equivalent to or derived from the natural language contract. Much of the literature concentrates on formalisms for expressing contract clauses. The intention is that these formalisms can be used to verify properties of an electronic contract such as its internal consistency. Ideally, an executable contract can be derived from its formal representation and this executable contract can then be used to regulate interactions at run-time. A typical model is that, conceptually, the executable contract sits between interacting parties and, by some mechanism, enforces the clauses of the contract. Variations in the work range from the formalisms developed to the general practical applicability of a given approach. To give a flavour of the research area, I review relatively mature work by groups at Rutgers University, at the Distributed Systems Technology Centre (DSTC) in Brisbane and by colleagues at Newcastle University. First, I discuss foundational work by Clark and Wilson on enterprise security policy.

2.3.1 The Clark-Wilson security model

The Clark-Wilson model [CW87] was the first rigorous treatment of security policy in the commerce (enterprise/business) domain. Hitherto, work on security policy had been driven by

requirements from the military domain and concentrated on the problem of regulating information disclosure, which was considered of paramount concern to the military and to government departments. Such work addressed the classification of information according to its sensitivity and concentrated on mechanisms to enforce policies with respect to the disclosure and declassification of information. Data integrity was a secondary concern, if considered at all. The key insight of Clark-Wilson was that in the commercial environment, or within the enterprise, preventing disclosure may be important but the paramount concern is to prevent unauthorised modification to information and, thereby, ensure data integrity. For example, the prevention of fraud and errors is often of greater importance than preventing accidental disclosure of information. In their view, enterprise security policy should emphasise who can do what to some data as opposed to who has access to it. From this distinction they concluded that new mechanisms were needed to enforce enterprise security in a computerised system. They identified the two key notions of the well-formed transaction and the separation of duties. In a well-formed transaction, users are not allowed to manipulate information arbitrarily and changes to information should be logged for the subsequent audit of actions. To address the separation of duties, they suggested that the computer system that provides access to data should separate operations on the data into sub-parts that must be executed by different users. In consequence, the different actors are accountable for their actions.

Clark-Wilson distinguish between traditional security policy in the military domain that was primarily concerned with granting of access to information and the need to place greater emphasis on preserving integrity when operating on information in the commercial domain. In the military domain, information is associated with a security level, the user demonstrates a right to access information at that level, and the user is then granted permission to read or write the information. In the commercial domain, they suggested that information should be associated with a set of programs (or operations) to manipulate the information. Then the user is granted permission to execute certain programs. Thus in the traditional military-inspired policy, the user is constrained by the data items that they can read or write. Given write permissions, they can modify the data in any way they wish. In contrast, Clark-Wilson proposed that users be constrained by the programs that they are allowed to execute. The actual

manipulation of data is implicit in the actions of these programs. They formalised this model with the notion of verified transformation procedures that enforce well-formed transactions. The transformations to information respect an organisation's integrity rules and are logged for audit.

The Clark-Wilson model concerns enforcement of policy within organisations. However, it is of relevance to the other work presented here. It is both foundational work on enterprise security policy and introduces the notion of mediators to enforce policy. Verified transformation procedures mediate the actions of users on an organisation's information. This is analogous to the use of some form of electronic contract to mediate interactions between organisations, or between the services offered by organisations, and to enforce agreed policies that govern the interaction. Their notion of well-formed transactions is particularly relevant to access and update to shared information. In essence, the service for non-repudiable information sharing defined in Section 3.3 ensures that actions are well-formed from the viewpoint of the business partners that share the information.

2.3.2 Law Governed Interaction

Work at Rutgers University on Law Governed Interaction (LGI) [MU00] represents one of the earliest attempts to provide regulated coordination between autonomous organisations. In LGI a law, expressed in a restricted form of Prolog, determines the validity of messages that can be sent between independent agents (computer or human) that form the system to be coordinated. The law is the coordination policy that the agents agree to abide by to achieve some common goal. Trusted controllers enforce a well-formed law by intercepting messages sent between agents. The implementation of LGI, called Moses, ensures that all communication between agents in the system is routed through controllers. The placement and operation of controllers is flexible. A controller may be co-located with an agent or may execute on a server on the Internet. One or more agents may be assigned to a single controller. A given controller may enforce one or more laws with respect to the agents that are assigned to it. The controller decides what to do with a message according to event-condition-action (ECA) rules encoded in the law.

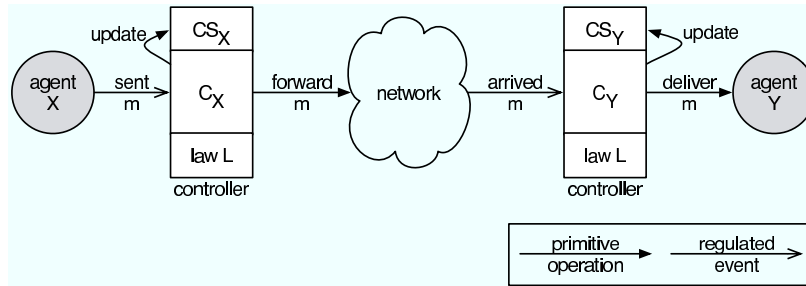


Figure 2.7: LGI message processing

Figure 2.7 illustrates how LGI handles the sending of a single, valid message, m , from agent X to agent Y. Conceptually, a controller is assigned to each agent — C_X for agent X and C_Y for agent Y. Each controller maintains its own control state (CS_X and CS_Y , respectively) and carries the same law, L . Control state CS_X is a bag of Prolog-like terms that represent the attributes of agent X. An example attribute of X may be X's role under L . The semantics of the control state are defined by the prevailing law. CS_X is not directly accessible to agent X, or any other agent, and can only be updated by operations that controller C_X invokes in response to events that occur under law L .

LGI distinguishes between regulated events and primitive operations. Regulated events occur at controllers and are subject to the prevailing law, triggering actions for conditions specified in the law. Figure 2.7 shows two regulated events. The sent event at C_X occurs when message m arrives from C_X 's agent X. The arrived event at C_Y occurs when message m arrives from agent X via C_X . Other events include notification that an obligation is due and notification that an exceptional condition has occurred. The law specifies primitive operations for a controller to perform for a given regulated event. As shown, these include operations to update control state and to forward and deliver messages. Given these elements of LGI, law L specifies a set of ECA rules that controllers C_X and C_Y are trusted to observe. The arrival of message m at C_X triggers the sent event. When C_X detects this event, it evaluates L with respect to the event and its current control state CS_X (the condition) and performs any actions that are specified by L . In Figure 2.7, the obligation on C_X is to forward valid messages to C_Y . Forwarding m results in an arrived event at C_Y . In turn, C_Y evaluates this event with respect to L and its control state CS_Y . The action taken by C_Y is to deliver m to agent Y. It is

possible for L to specify a variety of obligations on C_X and C_Y that are fulfilled when a message arrives. For example, they may be required to log the message or they may be required to take some corrective action if the message is invalid with respect to L . Controllers sign all messages they send along with a digest of the prevailing law. C_X appends a signature over $\{m, L\}$ to m before they forward it to C_Y . The signature cryptographically binds m to L and represents C_X 's non-repudiable assertion that m is valid with respect to L .

LGI is not restricted to interaction between two agents. The interaction in Figure 2.7 can be extended to communication between multiple agents. LGI also caters for events other than the arrival of message at controllers.

LGI has been applied to a number of application scenarios. A recent example is the regulation of distributed coalitions [AM03] where it was shown how a hierarchy of policies governing the coalition can be enforced by LGI and how LGI-based policies can be federated between coalitions.

Assuming all messages are mediated by controllers that interpret the same law and that all the controllers behave correctly, agents will only receive messages that comply with the law. Controllers are similar to Clark-Wilson's verified transformation procedures. The controllers ensure that agents only exchange valid messages and, in consequence, comply with agreed interaction policy.

There is no systematic support for audit or non-repudiation with respect to the actions of agents in the coordinated system. Furthermore, there is no systematic provision for acknowledgement, or NRR, of messages. To meet these requirements within LGI, the law in force would have to explicitly impose the necessary obligations on controllers. For example, controllers would be obliged to drop messages that had not been signed by an agent and would be obliged to provide some form of NRR for messages before forwarding them to agents. This implies a considerable amount of trust in the correct behaviour of controllers. To guarantee fairness, the law would oblige the controllers collectively to perform the function of the in-line TTP in the Coffey-Saidha protocol (see Section 2.2.2.1).

LGI relies on the interpretation of messages with respect to a law. However, it is not clear what form the messages take and, therefore, how or whether LGI could interoperate with pre-

existing service offerings.

2.3.3 Work at the DSTC, Brisbane

A common theme in research on electronic contracts has been the development of logical notations that aim to completely specify business contracts. Such specifications include temporal constraints, role players and their relationship to policies that are internal to an organisation. One of the more influential examples is the work of Milosevic et al at the DSTC in Brisbane [MD02, MGL⁺04]. They define a contract as a set of policy statements that specify constraints in terms of permissions, prohibitions and obligations for roles involved in the contract. A role is an entity that can perform an action. In [MD02], the transformation of natural language contracts into forms suitable for verification for consistency and for run-time monitoring is addressed in two phases.

In the first phase, they specify the expected behaviour of contracting parties using a formal behaviour tree notation. Their methodology is inspired by research on Genetic Software Engineering. The natural language contract is treated as an informal set of requirements. These requirements are translated into the formal structure and notation of a behaviour tree. The nodes of the tree identify contract components (including actors and users), the states they realise, the events and decisions they are associated with, the data that is exchanged, and causal, logical and temporal dependencies associated with component interactions. Nodes of the tree are tagged for traceability to clauses in the original contract. They can also be marked to indicate behaviour that is missing from, or only implied by, the original contract. Advantages claimed for behaviour trees include their expressive power, their notational simplicity, their ease of composition and their ability to expose behavioural defects. The methodical refinement of a behaviour tree model, and the identification of integration points between sub-trees of the model, can lead to a complete and defect-free formal specification of a contract. The model can also be extended to incorporate policy external to a contract, such as an organisation's internal policy. Although there is some tool support for the use of behaviour trees in software engineering, there is currently none for their application to formal contract specification. Therefore, building the behaviour tree model and detecting defects is essentially a manual process. The

authors liken this to solving a jigsaw, where the solution is revealed by the pieces and the behaviour tree identifies the correct position of the pieces. By inspection of the model as it is built, defects can be detected. New pieces can be added both to model behaviour that was anticipated in the original contract and to remedy behavioural incompleteness, adding behaviour that was unforeseen but is necessary for correctness. Thus, a solution to the jigsaw is found.

The second phase of the transformation is to derive a form of contract that can be used for run-time monitoring of business interactions. For this phase, Milosevic et al have defined a Business Contract Language (BCL) that expresses policy statements (contract clauses) in terms of deontic logic constraints. Deontic logic concerns reasoning about notions such as obligation, permission, prohibition and authority. BCL describes constraints on the behaviour of contracting parties. It allows the specification of what parties or role-players are permitted, prohibited or obligated to do under various temporal and other constraints. The BCL policy statements can be interpreted at run-time to monitor the actual behaviour of participants with respect to contract. There is no tool support to derive a BCL contract that is amenable to run-time monitoring from a behaviour tree specification. This is a manual process in which the behaviour tree specification acts as a guide to programming in BCL.

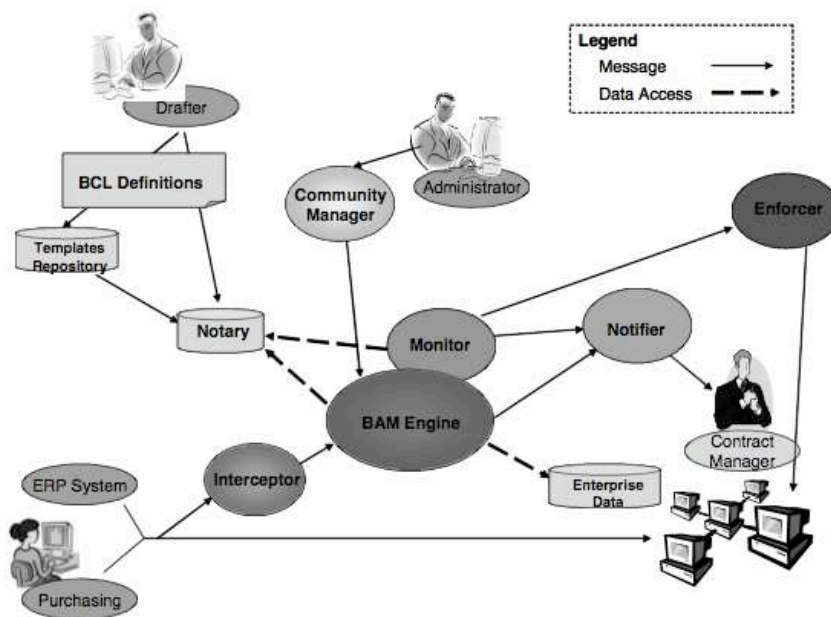


Figure 2.8: The DSTC business contract architecture

Figure 2.8⁸ shows the business contract architecture proposed in [MGL⁺04]. This architecture is intended to support the full life-cycle of contract management. The core components are: (i) a Contract Repository that stores standard contract templates and clauses to construct contracts, (ii) a Notary that stores evidence of agreed (signed) contract instances, (iii) an Interceptor to (unobtrusively) intercept business messages that are subject to contract monitoring, (iv) a Business Activity Monitor (BAM) that processes events generated by the Interceptor and that accesses application state that is required for policy evaluation by the Contract Monitor, (v) a Contract Monitor that evaluates BCL-encoded policies with respect to event and application state processing by the BAM component, (vi) a Notifier that sends notification messages such as reminders of tasks to be performed and violation detections to contract managers, and (vii) a Community Manager that allows contract administrators to update roles, policies etc. Contract Manager operations are also checked for validity by the contract monitor and BAM component. Additional, optional, components are envisaged to enforce contract clauses (as opposed to monitoring for their violation) and to validate contract specifications.

The unobtrusive interception of business messages and the processing of messages by the BAM component prior to evaluation suggests that the system could be adapted to provide systematic monitoring in different execution environments. The DSTC work does not consider accountability for actions.

2.3.4 Work at Newcastle University

In contrast to research on formal notations that attempt to fully specify contracts and detect a large range of inconsistencies, colleagues at Newcastle University adopt a modular approach [MJSSW04, MJSW05]. Complex contracts are divided into individual sub-contracts that can be formally described and verified to detect ambiguities. If necessary, their composition can also be verified. Two independent sources of inconsistency are identified: (i) internal enterprise policies that conflict with contractual clauses, and (ii) inconsistencies within and between contract clauses. In the Newcastle work, enterprises are considered black boxes and, therefore, the former source of inconsistency is considered the concern of each autonomous enterprise.

⁸Taken from [MGL⁺04].

Instead they concentrate on the latter source of inconsistency and on the contract compliance of the cooperative behaviour of enterprises as opposed to their internal structure.

The widely-used Promela modelling language [Hol91] is used to model the clauses of a natural language contract. In [MJSW05], Molina et al show how Promela can represent permissions, obligations, prohibitions, agents, time constraints and message type checking. It is not as expressive as languages such as BCL. However, a significant advantage is that a Promela representation of a contract can be verified using the associated SPIN model-checking tool [Hol04]. Correctness properties that can be checked include: that there are no unreachable states in a contract, that a contract is free of deadlocks, that contracting parties do not receive unsolicited responses, and that deadlines are observable with appropriate corrective action. The Promela model of a contract can be refined until it is free of ambiguities and all desired correctness properties are satisfied. This is assumed to be an iterative process where the natural language contract is updated in step with the Promela model. This provides a correct, implementation-neutral version of the contract that can be further refined to produce an implementation-specific version of the contract. The implementation-specific version of the contract is also amenable to model-checking but includes details such as additional acknowledgement and synchronisation messages that are specific to the implementation environment and standards selected. Example implementation standards include the RosettaNet PIPs specification of business conversations [Ros05].

From the Promela models it is straightforward to derive one or more finite state machine (FSM) models of the contract (see [MJSSW04]). These FSMs encode the run-time constraints on the observable behaviour of the contracting parties. Implementations of the FSMs then monitor and enforce contract clauses at run-time. In essence, the FSM implementations perform the same function as controllers in LGI. They react to events such as the interception of a message sent between contracting parties or the expiry of deadlines. Depending on the state of an FSM, the event may result in the sending of a new message and/or transition to a new FSM state (equivalent to the update of LGI control state).

In [MJSW05], Molina et al present four models for deployment of executable contracts (x-contracts). Figure 2.9⁹ illustrates three of these for a contract that governs the interaction

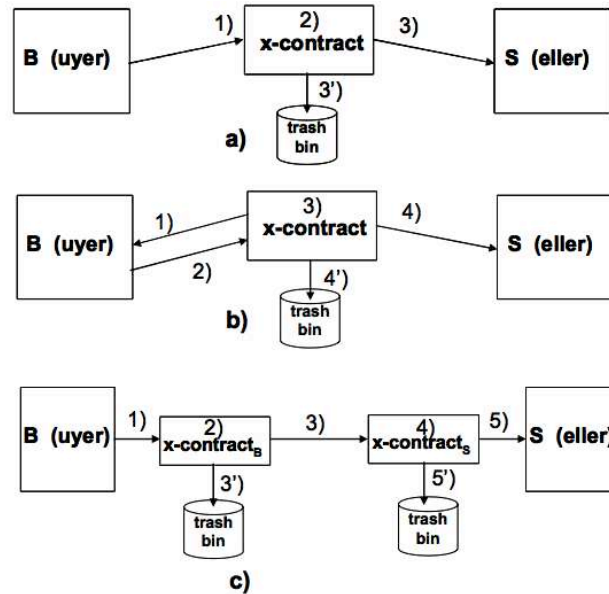


Figure 2.9: x-contract deployment models

between a buyer (B) and seller (S). The x-contract is an implementation of an FSM derived from a verified contract model. As shown, for a two-party interaction there is either a single x-contract that sits between the parties or the x-contract is split into parts that monitor the interaction at each party. The four deployment models follow.

- *Reactive central* (see Figure 2.9a) where the x-contract is deployed centrally. It is reactive because the x-contract (1) intercepts messages, (2) analyses the messages with respect to contract, and then either (3) forwards legal messages or (3') drops illegal messages.¹⁰
- *Proactive central* (see Figure 2.9b) where the x-contract is also deployed centrally but proactively coordinates the conversation between contracting parties. The x-contract (1) sends an invitation to a business party, (2) receives the response, (3) analyses the response with respect to contract, and then either (4) forwards legal responses or (4') drops illegal responses.
- *Reactive distributed* (see Figure 2.9c) where the x-contract is split into parts that can be

⁹Taken from [MJSW05].

¹⁰Other actions may be taken in response to illegal messages, but illegal messages are not forwarded.

deployed at each participant to reactively regulate the interaction from the point of view that participant.

- *Proactive distributed* where the x -contract is split into parts that can be deployed at each participant to proactively regulate the interaction from the point of view that participant.

Molina et al address the high-level aspect of regulation identified in Chapter 1. Aspects such as accountability, audit and the mechanisms to invoke validation with respect to contract at run-time are the concern of the work presented in this dissertation.

2.4 Middleware support for non-repudiation and accountability

In this section I present the most significant examples of existing middleware support for non-repudiation and accountability. Apart from the FIDES project (see Section 2.4.1.2), as far as I am aware existing middleware support is restricted to various forms of voluntary non-repudiation — it does not address the selective receipt problem discussed in Section 2.2. Only the work by BEA (see Section 2.4.2) provides any form of systematic support for accountability and audit in contract-mediated interactions. Section 2.4.1 discusses academic research. Section 2.4.2 discusses commercial systems. The emergence of commercial systems demonstrates an increasing awareness beyond academia of the need for non-repudiation services to support B2B interactions.

2.4.1 Research systems

In this section I describe the two existing research systems that provide middleware, or middleware-based, support for non-repudiation. Section 2.4.1.1 presents early work that provides non-repudiation of origin for remote method invocation in CORBA. Section 2.4.1.2 describes the FIDES system for fair, non-repudiable document exchange. I conclude this section with a discussion of provenance and provenance services and their relationship to non-repudiation.

2.4.1.1 CORBA and non-repudiation

The CORBA security specification [OMG98] specifies an optional non-repudiation service that is under control of applications, is not used automatically at object invocation, and, therefore, is only available to applications that are aware of the service. The application must directly invoke the non-repudiation service to generate and verify evidence, and must append non-repudiation tokens to application-level messages. The specification identifies the following components of non-repudiation evidence: the policy (or policies) applicable to the evidence, the type of action or event, and the parameters related to the type of action or event (including the date and time). Non-repudiation of origin and non-repudiation of receipt are identified as common types of evidence required. The specification also mentions the use of a delivery authority to provide third-party proof of origin and proof of delivery. The only mandatory part of a

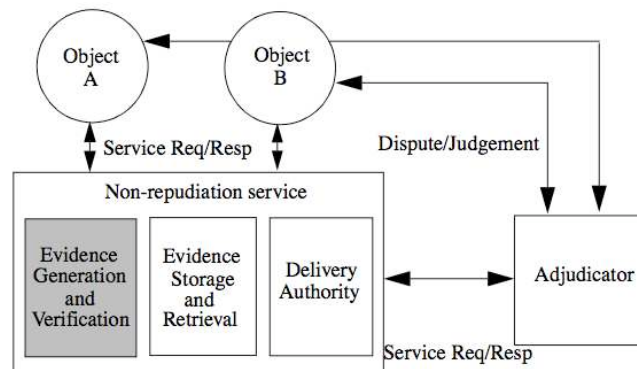


Figure 2.10: CORBA non-repudiation service: mandatory and optional components

CORBA-compliant non-repudiation service is a component for evidence generation and verification — the shaded box in Figure 2.10¹¹. However, as shown in Figure 2.10, other service components for evidence storage and retrieval, for third-party delivery, and for adjudication may be deployed depending on the choice of mechanism or policy. Note that Figure 2.10 also shows application objects interacting directly with the non-repudiation services — the application objects are “non-repudiation aware”. The specification is silent on how mechanisms are chosen or how policy is specified.¹² Other than noting that a delivery authority may be used to

¹¹Taken from [OMG98].

¹²In this dissertation I assume that business partners agree a non-repudiation policy. The non-repudiation services described in Chapter 3 are designed to be able to adapt to use mechanisms that are appropriate to the policy in force.

protect against false denial, the specification does not address fairness in the exchange of evidence. As Alireza et al [ALP⁺00] observe, the specification is incomplete and unsatisfactory in many other respects. For example, there is no agreed representation of evidence, which leads to interoperability problems. They also note that there are no known implementations of the CORBA non-repudiation service, mainly because it relies on CORBA security level 2 and is an optional extension to the specification.¹³ It should be noted that no other common middleware platform (J2EE, .NET or Web services) even provides a specification for non-repudiation services.

Wichert et al [WIC99] take a different approach to non-repudiation in CORBA. They present a service for transparent¹⁴ non-repudiation of remote invocation of CORBA objects. They share some of the same design goals that have driven my work. For example, their system aims to place as little additional burden on application programmers as possible by systematically generating non-repudiation evidence at the middleware level. In the Wichert system, application objects, such as a purchase order, are responsible for defining the format of non-repudiation evidence and application programmers tag object interfaces to indicate the requirement to generate non-repudiation evidence. The non-repudiation service then automatically generates evidence at run-time, transfers the evidence with an object invocation, and verifies and stores the evidence. CORBA filters (invocation path interceptors) are used to achieve this transparency.

Figure 2.11¹⁵ uses the example of non-repudiation of submission of an order to illustrate the Wichert service. In the standard application a client simply invokes the `process_order` method of a shop object that is hosted by a Marketplace service. The parameter to the method call is an offer object that the Marketplace service had previously provided to the client for their acceptance. The client signals their acceptance of the offer by invoking the `process_order` method. It is this acceptance that should be rendered non-repudiable. To achieve this, an outgoing filter

¹³The MICO ORB <<http://www.mico.org/>> is level 2 compliant and there is a claim at <http://www.ito.tu-darmstadt.de/projects/corbasec/index_en_html> that it also provides a CORBA-compliant non-repudiation service. However, I can find no evidence of this support in either the MICO documentation or the latest version of the downloadable source code. In 2005, Goovaerts et al [GDWJ05] also reported that they know of no existing implementation.

¹⁴Transparent in this context means transparent to the application objects.

¹⁵Taken from [WIC99].

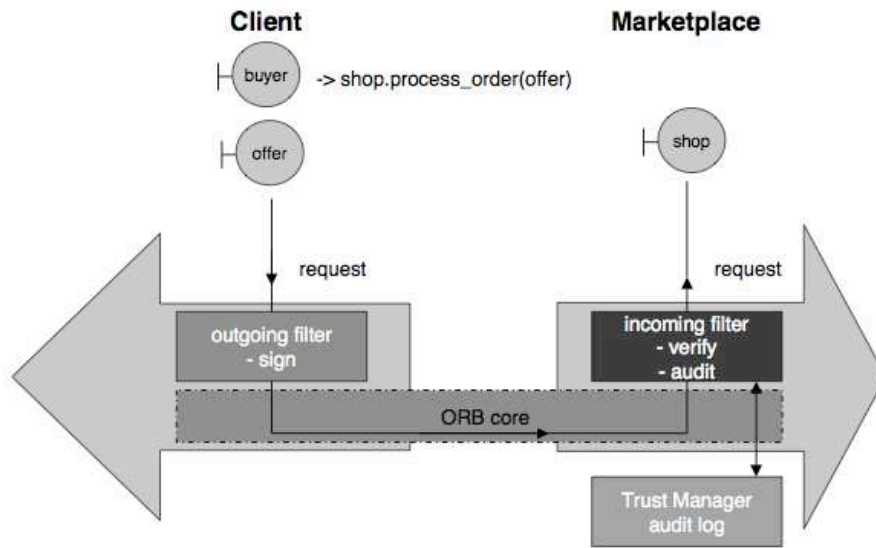


Figure 2.11: Wichert et al's CORBA-based transparent non-repudiation

intercepts the method invocation and generates a non-repudiation token, including a signature over the XML representation of the offer. The token is appended to the request and both are transmitted to the Marketplace service. An incoming filter at the Marketplace service verifies the evidence in the non-repudiation token and, if valid, sends the token to a TrustManager service. The TrustManager is responsible for secure storage and retrieval of evidence, as defined in the CORBA security specification. The verified request is then passed to the host shop object for processing.

The approach taken by Wichert et al is asymmetric. The client provides non-repudiation of origin of a service invocation. There is no exchange to provide corresponding evidence of receipt to the client. Nevertheless, this is the first known middleware implementation of a non-repudiation service. Furthermore, the authors make the key insight that there must be an agreed representation of non-repudiation evidence for the evidence to be useful. To address this, they settled on an XML representation of object state and non-repudiation evidence.

2.4.1.2 The FIDES project

The FIDES project [NZB04] provided services, including TTP services, and an associated application for fair exchange of business items (documents). The project addressed two prob-

lems: (i) the development of a family of efficient, deterministic fair exchange protocols, and (ii) the design and implementation of a system for execution of the protocols. I will provide an overview of the characteristics of the protocols that have been developed and then concentrate on the FIDES system for protocol execution.

The FIDES protocols apply to the exchange of two types of business item: (i) confidential, certified e-goods, where the content and/or quality of the goods has been certified by some independent authority, and (ii) digital signatures on documents such as contracts. The protocols have a common structure: a normal exchange protocol for the optimistic exchange of items, and a recovery protocol that involves a transparent, off-line, semi-trusted third party (STTP) to allow a well-behaved party to recover from the misbehaviour of another party. The TTP is considered semi-trusted because it is assumed that they may misbehave by attempting to decrypt encrypted items but they do not collude with the other parties to an exchange. The following requirements are satisfied by each protocol in the family: fairness; non-repudiation; confidentiality of exchanged items — no party external to the exchange, including the STTP, will gain any knowledge of the items; content and/or quality assurance of e-goods; reduction in the involvement of the STTP; and transparency of the STTP. Further details of the protocols can be found in the project publications cited in [NZB04].

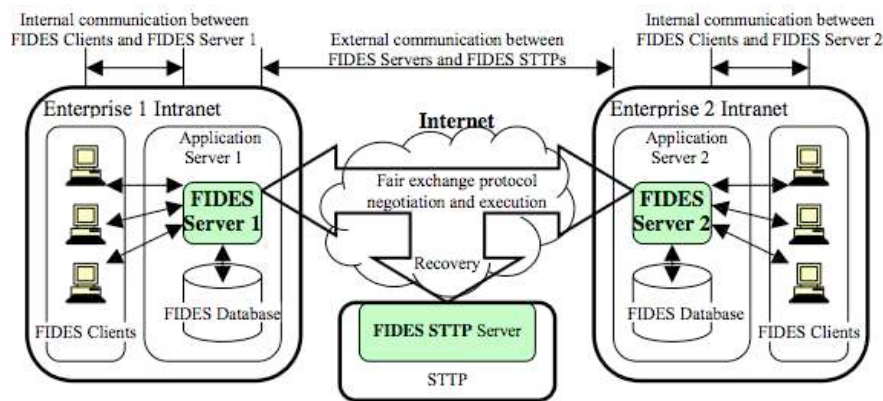


Figure 2.12: FIDES architecture

Figure 2.12¹⁶ provides an overview of the FIDES system architecture. Each enterprise hosts a FIDES server and a set of one or more FIDES clients. A FIDES STTP server is available

¹⁶Taken from [NZB04].

for dispute resolution. Within an enterprise, users interact with the FIDES system through GUI-based clients that provide secure access to the services hosted by their local FIDES server. As shown, the FIDES server at one enterprise executes fair exchange protocols with a peer server at another enterprise. FIDES servers also execute recovery protocols with the STTP server. Items for exchange are stored in a secure database. FIDES services are implemented on a J2EE application server. Java Messaging Service (JMS) is used for both client-to-server and server-to-server communication.

A typical FIDES exchange proceeds as follows. A user at Enterprise 1 uses a FIDES client to specify both a previously stored item to be sent to Enterprise 2 and the item they expect in return from Enterprise 2. They may also specify a transaction timeout. The user's transaction specification is sent to FIDES server 1 that, in turn, initiates negotiation of the exchange parameters with FIDES server 2 at Enterprise 2. Parameters to negotiate include: exchange protocol to use, a mutually trusted STTP, a transaction identifier, and an agreed timeout value. FIDES server 2 then notifies an appropriate Enterprise 2 user of the specified transaction request. The user at Enterprise 2 uses a FIDES client to browse transaction requests and decide whether to confirm acceptance of any pending transaction specifications. If a transaction is accepted then the specified items are exchanged between the FIDES servers using the agreed exchange protocol. The servers deploy the protocol implementation from a protocol library.

As far as I am aware, FIDES and our work at Newcastle are the only service-based implementations of fair exchange. A drawback of the FIDES approach is that the only interaction with the exchange protocol service is through FIDES application clients. There is no published API for client-side interaction with a FIDES server or for the server-to-server execution of protocols. Users are restricted to an application-specific mechanism for the exchange of items. Other than user agreement to involvement in an exchange, there is no support for run-time validation of the interaction with respect to contract. Any contractual constraints that should be imposed must be dealt with during negotiation of the items to be exchanged, and of the parameters of the exchange.

2.4.1.3 Provenance services

Data provenance can be defined as information, or meta-information, "... that helps determine the derivation of a data product, starting from its original source" [SPG05]. Alternatively, "... the provenance of a data item ... " is the "... process that produced the data" [MGBM06]. That is, the recorded provenance of data tells us how the data came into being. The advent of Grid computing, particularly in support of e-Science, has led to considerable interest in data provenance and provenance services. An example application in e-Science is the recording of the experimental processes leading to a set of scientific results. The collected provenance information can be used to verify that a correct process took place and, potentially, to replay an experiment. The provenance of data is also of interest in other domains, such as the long-term maintenance of accounting information from which a financial audit report was derived. Simmhan et al [SPG05] provide a useful survey of data provenance in e-Science. This includes a taxonomy of provenance services by: (i) the use of provenance information — data quality, audit, replication, attribution, informational; (ii) the provenance subject — process or data, and granularity; (iii) its representation; (iv) the storage method; and (v) dissemination or access method. They also characterise a number of provenance research systems for e-Science according to the taxonomy.

Provenance services and non-repudiation services address some of the same requirements. Both are concerned with the attribution, accountability and audit of actions that, electronically, are represented by the creation of data. Provenance services address a number of additional, higher-level concerns. For example, the PASOA project gathered requirements from various use cases to arrive at the proposed provenance architecture shown in Figure 2.13¹⁷. There are functions to record, process, query and present provenance information. There are services to support re-enactment. There is a need to attach domain-specific meta-information (semantics) to data as it is collected.

Simmhan et al identify the federation of provenance information and support for the assertion of its truthfulness as open research questions with respect to the usability of provenance information across organisations. This suggests that actors must be held to account for the in-

¹⁷Taken from [MGBM06].

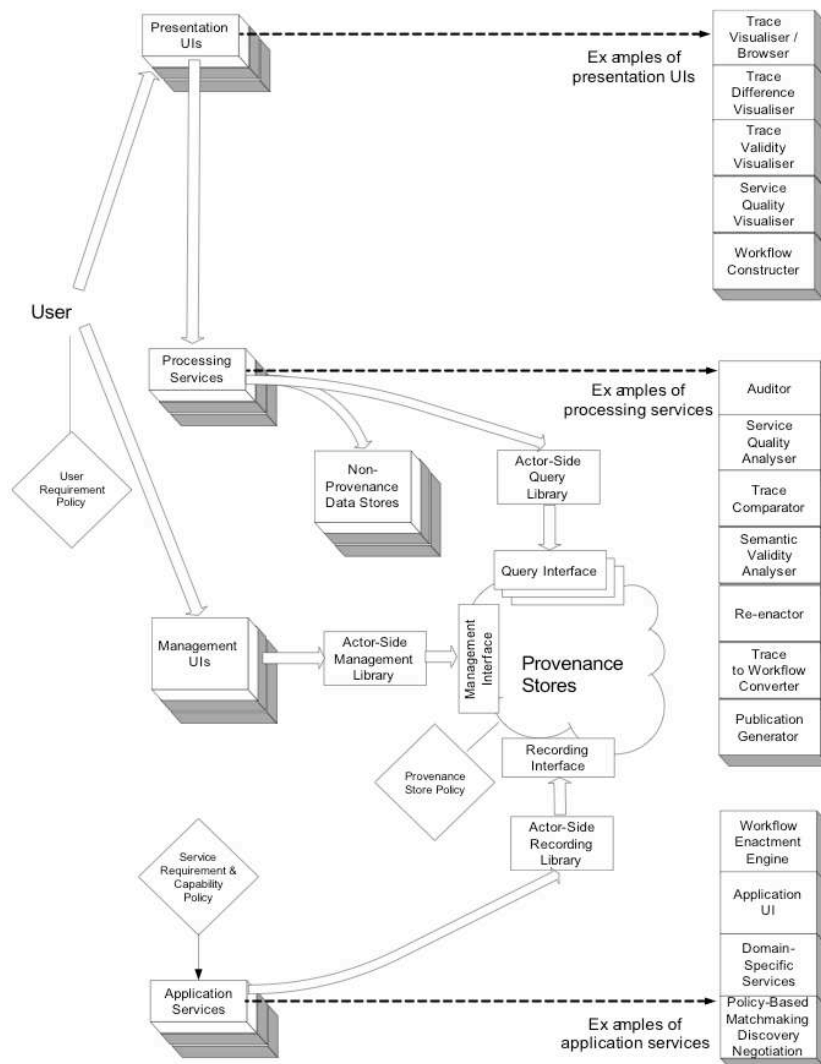


Figure 2.13: PASOA provenance architecture

formation that they generate. The PASOA project [MGBM06] also identifies non-repudiation as an important requirement. For example, they highlight the need for irrefutable evidence to support patent applications. Also, in [GLM04], they present a protocol for post-hoc agreement on provenance information that has been recorded with TTP provenance store(s) by different parties to an interaction. In [MJSW05], colleagues propose the use of non-repudiable information sharing, as defined in Section 3.3, in order to periodically synchronise the agreed view that participants have of an interaction. This is very similar to the purpose of the protocol used in the PASOA project. In both cases the intended outcome is that interacting parties have a

consistent view of interaction state. In the PASOA project, that view is arrived at, or imposed, by one or more trusted provenance stores. It is not clear how non-cooperation of interacting parties is handled in the PASOA project nor do they specify underlying mechanisms for non-repudiation. The work I present in this dissertation, and proposed future work, is of relevance here. An avenue for investigation is how non-repudiation services that generate and record irrefutable evidence of interactions can be used to generate the meaningful audit trails that are at the heart of provenance services. I will return to this topic when I discuss future work in Chapter 6.

2.4.2 Commercial systems

Various companies are beginning to offer XML firewall solutions that perform various security functions such as applying encryption and signatures to outgoing messages, and the decryption and verification of incoming messages. For example, DataPower [Dat04] offer XML processing in hardware that can apply cryptography to XML documents at "wire-speed". The Verisign Trust Gateway [Ver04] is a server-based organisational gateway to secure SOAP-based message exchanges. Enterprise applications are then deployed behind the gateway and requests to the application can be encrypted and decrypted at the gateway. Signatures can also be applied and verified at the gateway. Optionally, a business partner can also deploy the Verisign Partner Gateway and facilitate the exchange of non-repudiation of origin of a client request for non-repudiation of its receipt. This approach could also be used to exchange evidence of the enterprise application response. This corresponds to the simple exchange shown in Figure 2.2 on page 37. The gateway also provides configurable message logging. It is not clear whether trusted time-stamping is provided or whether client request and server response are correlated in the non-repudiation evidence. There are other similar offerings in the marketplace. All of them provide some form of voluntary non-repudiation. They are essentially variations on a Wichert style non-repudiation service. The variations are whether non-repudiation of receipt is provided and on the provision of supporting services such as access to a PKI, trusted time-stamping and message logging. None address the selective receipt problem described in Section 2.2.

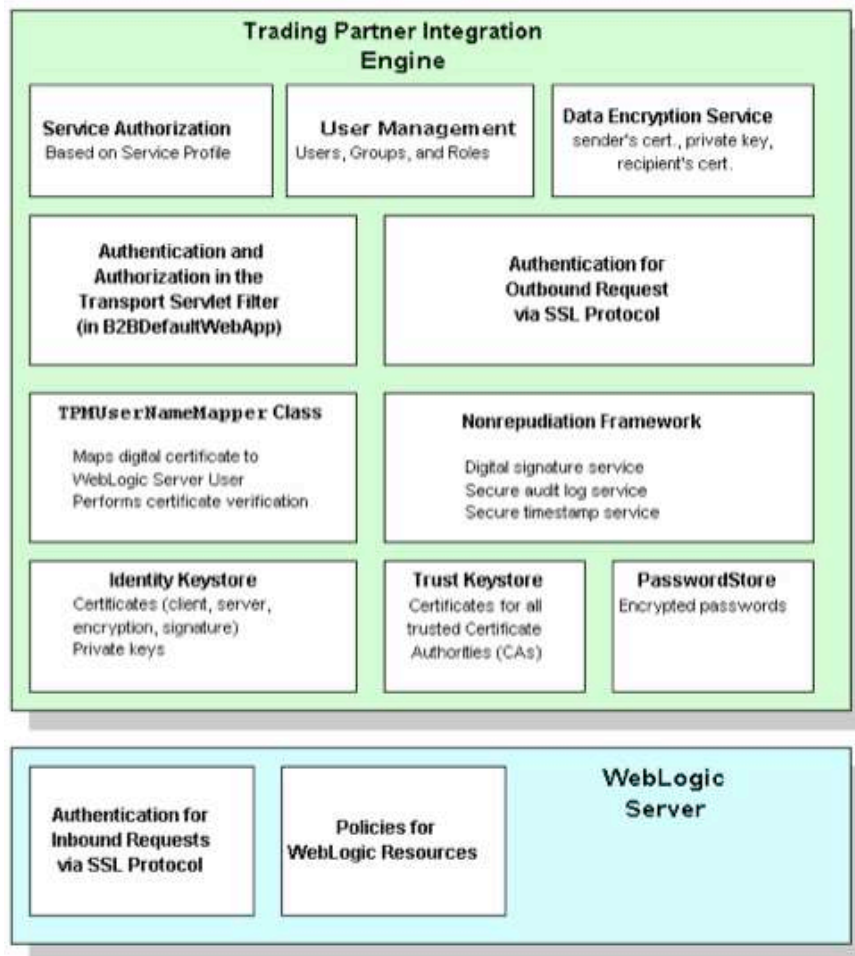


Figure 2.14: BEA WebLogic Trading Partner Integration Engine

The BEA WebLogic application server [BEA05] offers an alternative approach to firewall solutions. As shown in Figure 2.14¹⁸, their Trading Partner Integration Engine (TPIE) is an optional service that is integrated with the application server. Non-repudiation is provided as a component of the TPIE. This is similar to the provision of non-repudiation as a container-level service that I describe in Chapter 5. The BEA approach is for trading partners to agree protocol bindings for their business conversations. Currently they provide bindings for the RosettaNet PIPs [Ros05] and ebXML [GEN⁺01] standards. The protocol binding to use at run-time is then configured in the B2BDefaultWebApp component of each partner's TPIE. The trading partners have the option to apply signatures to the messages of the given business

¹⁸Taken from [BEA05].

protocol. The non-repudiation service handles application and verification of signatures at run-time. The service also provides audit and time-stamping services. BEA have defined interfaces to these support services and allow the plug-in of third-party services that comply with these interfaces. This is similar to the approach taken to the definition of interfaces for supporting services in Section 4.3. The BEA system is interesting because they provide non-repudiation of business message exchanges that are validated with respect to the business protocol binding in use. This is form of regulated B2B interaction. The non-repudiation is voluntary. There is no support for fair exchange. Both non-repudiation and validation with respect to contract are only available for the supported business protocol bindings. They cannot be applied to more general interactions. For example, it is not possible to verify that a composition of PIPs is adhered to correctly (see Chapter 6). Nevertheless the TPIE does represent an example of systematic non-repudiation and validation with respect to a form of contract.

The Universal Postal Union have proposed a Global Electronic Postmark (EPM) standard [UPU02]. This is a TTP service provided by postal administrations for the generation, verification, time-stamping and storage of non-repudiation evidence. The service supports linking of evidence under a unique transaction identifier to allow the binding together of business transaction events. There is no support for exchange of evidence between end users. The EPM service can be seen as an on-line backend TTP service that non-repudiable exchange services can use for secure time-stamping, storage and linkage of evidence. If widely adopted, EPM services could offer the assurance that evidence will be accepted by postal administrations worldwide and, therefore, the likelihood of the acceptance of evidence by arbitration authorities.

2.5 Discussion

The work surveyed in Section 2.2, and the protocols developed in the FIDES project, can be viewed as the basis for a toolkit of mechanisms to achieve non-repudiable exchange with and without a fairness guarantee. The surveyed work illustrates the need to be able choose mechanisms that are appropriate to a given application context. An aim of my work is to provide a flexible framework for deployment of mechanisms that underpin the non-repudiation services proposed in Section 1.3.

The FIDES project adopted a flexible approach for the family of protocols that they have developed in support of the exchange services that they provide. Unfortunately, their system cannot be used for the systematic regulation of B2B service interactions because it is not possible to use FIDES servers as interceptors in the B2B service invocation path. To use their system as middleware-level interceptors, we require an API for the FIDES server functionality. This functionality is currently only accessible through FIDES client applications. Similarly, the lack of an API for protocol execution between FIDES servers limits their solution to their family of protocols. If APIs were provided, then it is conceivable that the FIDES system could be used to provide the services I present in this dissertation. Similarly, it should be possible to build a FIDES-like document exchange application using my non-repudiation services.

Wichert et al did use invocation path interception. Their approach is closer to my work on non-repudiable service invocation than to the FIDES work. However, Wichert et al did not address the issues raised in Section 2.2, such as how to guarantee fairness and the need for a choice of exchange mechanism. A significant insight of their work is the need for an agreed representation of the information exchanged. In my work the exact representation is assumed to be a matter for agreement between interacting parties. The important requirement is that the representation can be subsequently rendered meaningful and irrefutable. I therefore allow configuration for different agreed data-bindings (see Section 4.2.3).

The commercial systems surveyed in Section 2.4.2 are of interest because, apart from the BEA system, they could be used as the support infrastructure for an exchange protocol execution framework. The commercial system could provide delivery and verification of signed messages, message logging etc. The protocol execution framework could drive the exchange of evidence according to application requirements. The BEA system is problematic in this respect because of the tight-coupling of non-repudiation to high-level business protocols. This coupling makes it difficult to insert an exchange protocol layer between the business protocol and signed message delivery.

The work on contract-mediated interaction is complementary to my work. All of the approaches surveyed could benefit from the systematic generation of an irrefutable audit trail of an interaction to hold actors to account for their actions. The Newcastle work explicitly de-

fers this support to the services I have developed. I assume that the interceptors in the DSTC business contract architecture could be adapted to use my services for non-repudiation. LGI provides non-repudiation of messages between trusted controllers but not of messages from the agents that are assigned to a controller. Since there may be a one-to-many relationship between a controller and an agent, it cannot be assumed that the controller's non-repudiation evidence can be used to hold an agent to account. Therefore, either the LGI law in force must oblige controllers to guarantee the exchange of appropriate evidence or the LGI system must be adapted to include an additional interception layer between agent and controller in order to provide systematic non-repudiation of agent actions.

To conclude, Section 2.2 provides an overview of the substantial literature on security protocols for non-repudiation and fairness. Section 2.3 samples the active research area of electronic contracting that is, for the most part, complementary to the work presented in this dissertation. There has been much less work on systematic support for non-repudiation. There is still less that takes account of the fundamental work in Section 2.2. As far as I am aware, there is none that also addresses the requirement for the validation of B2B interactions with respect to contract that is evident from the work in Section 2.3 or that meets the requirement to adapt to different application contexts. None of the related work provides the non-repudiable information sharing service defined in Section 3.3. The remainder of this dissertation addresses these gaps with a flexible approach to the provision of non-repudiation services, thereby meeting the requirements identified in Chapter 1.

Chapter 3

Definition of non-repudiation services

Chapter 1 identified two domains for B2B interactions — one in which business partners access, or act upon, each other’s private (autonomously owned) resources or information, and the other in which business partners act upon shared resources that they jointly own. In the first domain, business partners, and third parties, provide services that allow controlled access to their resources. The resources remain wholly under the control of the organisation providing the service(s). Business partner clients use service invocations to perform actions in this domain. Agreements between business partners may determine the obligations and rights of both the client and the service provider with respect to service usage. In the second domain, actions are performed on information that is shared by a group of business partners. Each member of the group has the right to access and update the shared information. However, these rights should only be exercised with the agreement of the other members of the group who jointly own the information. As established in Section 1.2, there is an overarching requirement to enforce agreements that constrain the actions that business partners may perform in each domain. To meet this requirement, I identified the need for support to hold actors to account for their participation in B2B interactions. Section 1.3 proposed two services intended to meet the various requirements identified in Section 1.2, one for each of the domains of action. The first, non-repudiable service invocation, aims to ensure that both parties to a client-server interaction obtain non-repudiable evidence of both the request phase and any associated response phase of the invocation. The second, non-repudiable information sharing, ensures that all parties sharing information have the same, irrefutable view of that information and of the membership of the group that shares the information. Further challenges when providing these services are

that they should both be able to adapt to use different underlying mechanisms (non-repudiation protocols) to meet different requirements and that they should not violate application-level abstractions. In this chapter I provide detailed definitions of these services. Their design and implementation is novel. As discussed in Chapter 2, there is no other work that combines fundamental work on non-repudiation with the flexibility to apply that work as middleware services to regulate B2B interactions at run-time and at the same time maintain application-level abstractions. The coverage of this work, across the two domains for B2B interaction, is also unique.

Section 3.1 introduces the abstraction of interceptor-mediated interaction that supports the two non-repudiation services. Mediation is fundamental to the flexibility of the services and the preservation of application-level abstractions. Section 3.2 defines non-repudiable service invocation. This is based on the execution of non-repudiation protocols of the type described in Section 2.2. Section 3.3 defines non-repudiable information sharing. This includes the definition of the meta-information used to describe transitions in shared information and a detailed description of the protocols used to coordinate transitions. Section 3.4 provides an overview of supporting infrastructure for the two non-repudiation services. The chapter concludes with an evaluation of the non-repudiation services. Then Chapters 4 presents a generic framework for their implementation as middleware that is described in Chapter 5.

3.1 Interceptor-mediated interaction

This section introduces the abstraction of interceptor-mediated interaction. Each participant in a multi-party interaction has an interceptor that acts on their behalf to meet regulatory requirements. For the purposes of the non-repudiation services presented in this chapter, a participant's interceptor has four main functions:

1. to manage the participant's involvement in regulation,
2. where appropriate, to subject the actions of other participants to local, application-level validation,
3. to insulate the participant from the incorrect behaviour of other parties and, in particular,

from invalid actions proposed by other parties, and

4. to preserve the application-level semantics of a given interaction at the same time as addressing regulatory requirements.

The interceptors protect the interests of the party on whose behalf they act by executing appropriate protocols and accessing appropriate services, including TTP services, while abstracting away the details of the mechanisms used. This is similar to the well-known proxy design pattern [GHJV93]. In this case, interceptors can be seen as as proxies for the participants that handle the regulatory aspects of their interaction. In contrast to LGI controllers, interceptors are owned by the party on whose behalf they act and are not necessarily trusted by any other party. That is, for two business partners, A and B, B will have no more trust in the correct behaviour A's interceptor than they do in the correct behaviour of A. In fact, from the point of view of A and B, the involvement of the interceptors that mediate their interaction is transparent. As an example, Figure 3.1a shows A sending a business message to B. To regulate delivery

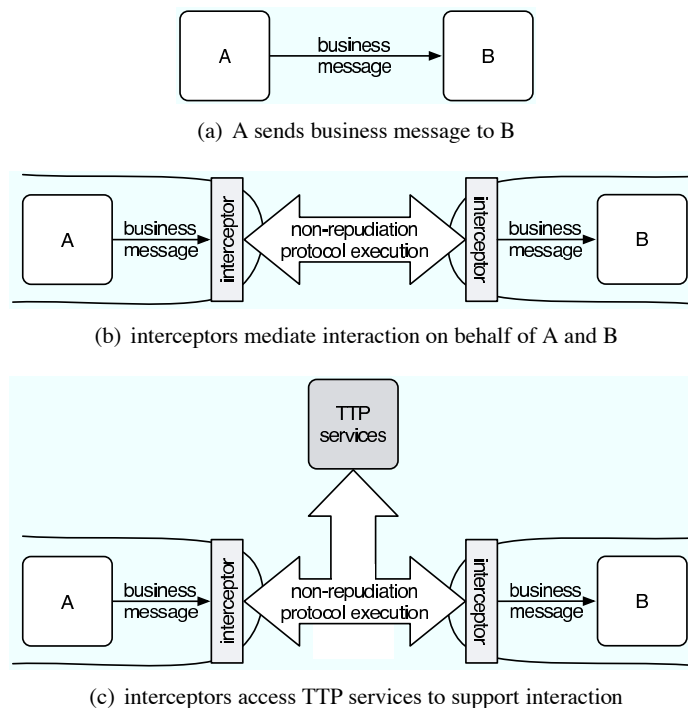


Figure 3.1: Interceptor-mediated interaction

of the business message, interceptors act on behalf of A and B to execute some non-repudiation

protocol. As shown in Figure 3.1b, the interceptors maintain the abstraction of A sending the original business message to B. Figure 3.1c shows the case where the interceptors access TTP services during protocol execution. As we shall see from the service definitions, the non-repudiation protocol generates and exchanges evidence of the participation of A and B and can also support the validation of the business message with respect to agreements governing the interaction between A and B. The exact protocol executed will depend on the type of service being provided — non-repudiable service invocation or non-repudiable information sharing — and also on application-specific factors such as the relationship between participants.

3.2 Non-repudiable service invocation

The purpose of non-repudiable service invocation is provide non-repudiation and validation of the request and response phase of service invocation. Figure 3.2a recalls the logical view of

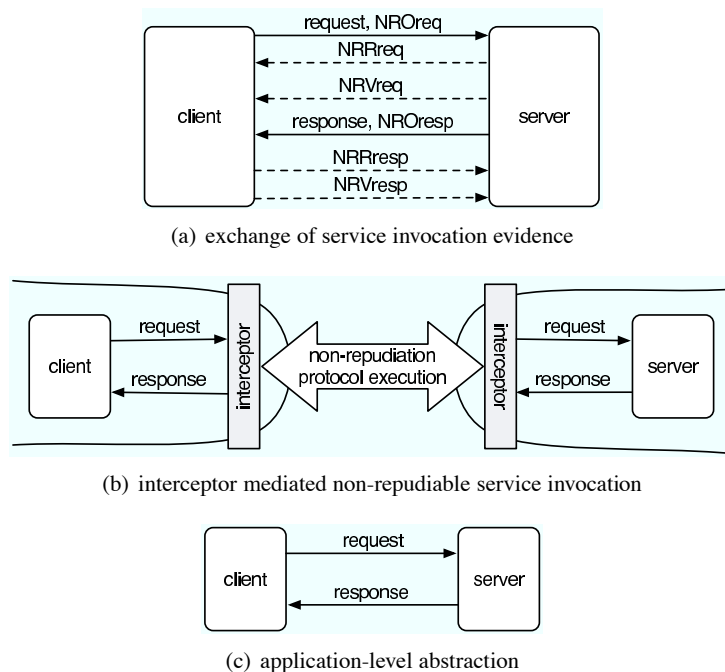


Figure 3.2: Non-repudiable service invocation

the evidence that the client and server need to exchange from Section 1.3.1. Given a request and evidence of its origin (NROreq) from the client, the server provides non-repudiation of its receipt and of its validation (NRRreq and NRVreq, respectively). Similarly, in the response

phase, the server provides the application-level response to the request with evidence of its origin (NROresp). In return, the client provides non-repudiation of receipt and of validation of the response (NRRresp and NRVresp). The preceding description specifies the evidence to be exchanged. As noted in Section 1.3.1, the exact details of the exchange will depend on the protocol that is executed.

The proposed middleware for non-repudiable service invocation provides interceptors at client and server that execute appropriate non-repudiation protocols to achieve the preceding exchange. If protocol execution is successful, regulatory requirements are met because:

1. a request is only passed to the server for processing if: (i) the client-side interceptor provides valid evidence of origin (NROreq); (ii) the request is valid with respect to contract, as indicated by NRVreq provided by the server-side interceptor; and (iii) the server-side interceptor provides evidence of receipt of the request (NRRreq); **and**
2. a response is only passed to the client for processing if: (i) the server-side interceptor provides valid evidence of origin of the response (NROresp), (ii) the response is valid with respect to contract, as indicated by NRVresp provided by the client-side interceptor, and (iii) the client-side interceptor provides evidence of receipt of the response (NRRresp).

Figure 3.2b shows the execution of some non-repudiation protocol between client and server interceptors. As shown, in contrast to Figure 3.2a, the interceptors hide the details of the exchange from the client and server. Thus the introduction of interceptors, maintains the service invocation abstraction shown in Figure 3.2c. Chapter 4 describes a generic framework for the execution of the protocols that provides the services necessary to realise the interceptor-based interaction. I now discuss how to apply and extend fundamental work on non-repudiation and fairness to the exchange of evidence between the client and server interceptors.

3.2.1 Example exchange protocols for service invocation

There are two phases to non-repudiable service invocation. The first phase concerns the exchange of the request, and evidence of its origin, for evidence of its receipt and validation. The second phase concerns the exchange of the response, and evidence of its origin, for evidence

of its receipt and validation. Appropriate non-repudiation protocols must be chosen for the exchange of evidence in each phase. In addition, the execution of exchange protocols in each phase must be correlated in order to bind the service request to the response. This section presents example protocols to achieve the exchange of non-repudiation evidence. The first example, in Section 3.2.1.1, uses a simple direct exchange protocol. An exchange of this type is potentially unfair to the initiator of each phase of the service invocation. Nevertheless, there are circumstances in which a direct exchange is appropriate. A discussion of these circumstances follows the protocol description. The other examples, in Section 3.2.1.2, involve the correlated execution of fair exchange for each phase of a service invocation.

The protocols described here rely on the assumptions stated in Sections 2.1.4 and 2.1.5 with respect to well-behaved parties, cryptographic primitives and the intruder model. The protocol execution framework described in Chapter 4 can also be used to execute other protocols and, for example, to combine different underlying protocols for request and response phases. Section 5.2 provides details of the implementation of non-repudiable service invocation that is based on the protocol execution framework.

To simplify the descriptions that follow, the protocol participants A and B correspond to the interceptors shown in Figure 3.2b that act on behalf of client and server, respectively. However, it should be noted that, from the point of view of each entity involved in the service invocation (client, server or TTP), an interceptor is synonymous with the party on whose behalf it acts.

3.2.1.1 Direct exchange for service invocation

Non-repudiable service invocation by direct exchange is based on the correlation of two runs of the simple non-repudiation protocol shown in Figure 3.3 (recalled from Figure 2.2 on page 37).

$$\begin{array}{l}
 A \rightarrow B : A, B, m, sig_A(A, B, m), ts_{TSA}(sig_A(A, B, m)) \\
 B \rightarrow A : sig_B(sig_A(A, B, m)), ts_{TSA}(sig_B(A, B, m))
 \end{array}$$

Figure 3.3: Simple non-repudiation protocol

Table 3.1 defines the non-repudiation tokens used for service invocation. Figure 3.4 shows the two phases of the direct exchange.

Token	Definition
$NROreq$	$sig_A(nrId_1, A, B, req)$
$NRRreq$	$sig_B(nrId_1, A, B, NROreq)$
$NRVreq$	$sig_B(nrId_1, VAL INVALID)$
$NROresp$	$sig_B(nrId_2, A, B, nrId_1, resp)$
$NRRresp$	$sig_A(nrId_2, A, B, NROresp)$
$NRVresp$	$sig_A(nrId_2, VAL INVALID)$

Table 3.1: Tokens used in direct non-repudiable service invocation

Phase 1: A submits request to B :

- 1.1 $A \rightarrow B : nrId_1, A, B, req, NROreq, ts_{TSA}(NROreq)$
- 1.2 $B \rightarrow A : nrId_1, A, B, VAL | INVALID, NRRreq, NRVreq, ts_{TSA}(NRRreq, NRVreq)$

Phase 2: if request is valid, B issues response to A :

- 2.1 $B \rightarrow A : nrId_2, A, B, nrId_2, resp, NROresp, ts_{TSA}(NROresp)$
 - 2.2 $A \rightarrow B : nrId_2, A, B, VAL | INVALID, NRRresp, NRVresp, ts_{TSA}(NRRresp, NRVresp)$
-

Figure 3.4: Direct non-repudiable service invocation

In step 1.1, A sends B the service request (req) along with a protocol run identifier, $nrId_1$, the participant identifiers and the NRO of the request and time-stamp over the NRO of request. The service request is some agreed (serialized) representation of the service invocation as intended for processing by the server¹. Assuming B successfully verifies the integrity of the information provided by A , they may then subject the service request to application-specific validation. The result of this validation is essentially a boolean: VAL if the request is considered valid, $INVALID$ otherwise. It may also include diagnostic information. In step 1.2, B returns the validation result with time-stamped NRR and NRV of the request. If B indicates that the request is valid, then A can assume that the request will be passed to the target service for processing and that the second phase of the service invocation will ensue. If B indicates that the request is invalid in step 1.2, then the client will be notified of its invalidity, typically through an exception, and the service invocation will terminate.

The response phase is another execution of the simple non-repudiation protocol, this time initiated by B . In step 2.1 of this phase, B provides the service response, in some agreed repre-

¹See Section 4.2.3 for discussion of agreed representations of information.

sentation, and also includes $nrId_1$ from the first phase in order to correlate request and response. If A determines that the response is valid at step 2.2, then it is passed to the client and B receives non-repudiation evidence of its validity. If the response is deemed invalid, then the client will be notified of its invalidity and evidence of its invalidity will be logged by both A and B.

In Figure 3.4 application-level validation is effectively a part of the evidence of receipt provided in the second step of each of phase 1 and 2. It is possible to combine receipting and validation in this way because the initiator of each phase discloses all of their input to the exchange in the first step — including the service request in step 1.1 and the service response in step 2.1. As discussed in Section 3.2.1.2, this is not possible when fair exchange is used for the exchange of evidence and further steps are required for the fair exchange of validation evidence.

Direct non-repudiable service invocation is appropriate when both parties are well-behaved and will therefore cooperate to exchange the information they each expect (see Section 2.2.4). Circumstances may arise where both parties trust each other but nevertheless they require proof that an exchange took place to exercise rights they have acquired with respect to some other party not involved in the exchange. A real-life example is the purchase of goods where buyer and seller trust each other, yet the buyer requires proof of the purchase in order to exercise their rights under a guarantee with respect to the manufacturer of the goods purchased. A direct exchange is also appropriate if, for example, a client wishes to submit a request to a well-behaved (trusted) server and there is no response phase to the invocation. In this case, the client will receive the NRR and NRV in step 1.2 if the client proves that they are well-behaved by providing valid NRO in step 1.1. The interaction with the ASP-hosted auction service described in Section 1.1.1 consists of one-way requests of this type.

To summarise, the direct approach is appropriate if the party expected to provide a receipt is well-behaved. That is, in the request phase, the direct exchange is appropriate if B can be trusted. In the response phase, the direct exchange is appropriate if A can be trusted. Otherwise a fair exchange is required to safeguard the interests of the initiator of the relevant phase of the service invocation, A for the request and B for the response.

3.2.1.2 Fair exchange for service invocation

Fair, non-repudiable service invocation is achieved by two runs of a fair exchange protocol — the first for the request phase and the second for the response phase. In this section I present modifications to the Coffey-Saidha in-line TTP fair exchange protocol described in Section 2.2.2.1 to provide non-repudiation of validation and to correlate the two phases of service invocation. In addition, I specify exception handling sub-protocols to allow timely termination of the exchange in either phase of an invocation. The protocol descriptions are followed by a protocol analysis that substantiates the claim that the exchange of evidence is fair. I then present an optimisation of the main exchange protocol that uses the in-line TTP as a time-stamping authority. The section concludes with a further example that uses an off-line TTP to guarantee fair exchange.

In the following, the request phase protocol run identifier is used as a correlation identifier for the response phase. As in the original Coffey-Saidha protocol, to guarantee fairness, the request and associated $NROreq$, and the response and associated $NROresp$, are not revealed to their intended recipients until receipts have been provided to the TTP. Therefore, as shown in

Token	Definition
$NRRreq$	$sig_B(nrId_1, A, B, h(NROreq))$
$NRRresp$	$sig_A(nrId_2, A, B, h(NROresp))$

Table 3.2: Modified tokens for fair non-repudiable service invocation

Table 3.2, the $NRRreq$ and $NRRresp$ tokens used in the direct exchange are modified to include a digest of the relevant NRO token, as opposed to the NRO token itself. The remaining tokens are as defined for the direct exchange in Table 3.1.

In the direct exchanges described in Section 3.2.1.1, application-level validation is provided as part of the receipt for a service request or service response. This is not possible in a fair exchange because, as previously noted, relevant receipts must be provided before the service request or response is revealed. That is, the receipts are generated before the information to be validated is available for validation. A consequence of the separation of receipting from validation is that, in order to maintain fairness to both parties, some form of receipt must be provided to the validating party for their application-level validation of request or response.

Phase 1: *A* submits request to *B*:

- 1.1 $A \rightarrow TTP : enc_{TTP}(nrId_1, A, B, rn_A, req, NROreq, ts_{TSA}(NROreq))$
- 1.2 $TTP \rightarrow B : nrId_1, A, B, h(NROreq)$
- 1.3 $B \rightarrow TTP : enc_{TTP}(nrId_1, A, B, NRRreq, ts_{TSA}(NRRreq))$
- 1.4 $TTP \rightarrow B : nrId_1, A, B, req, NROreq, ts_{TSA}(NROreq)$
- 1.5 $TTP \rightarrow A : nrId_1, A, B, NRRreq, ts_{TSA}(NRRreq)$
- 1.6 $B \rightarrow TTP : enc_{TTP}(nrId_1, A, B, VAL | INVALID, NRVreq, ts_{TSA}(NRVreq))$
- 1.7 $TTP \rightarrow B : nrId_1, rn_A$
- 1.8 $TTP \rightarrow A : nrId_1, A, B, VAL | INVALID, NRVreq, ts_{TSA}(NRVreq)$

Phase 2: if request is valid, *B* issues response to *A*:

- 2.1 $B \rightarrow TTP : enc_{TTP} \left(\begin{array}{c} nrId_2, A, B, nrId_1, rn_B, resp, NROresp, \\ ts_{TSA}(NROresp) \end{array} \right)$
- 2.2 $TTP \rightarrow A : nrId_2, A, B, nrId_1, h(NROresp)$
- 2.3 $A \rightarrow TTP : enc_{TTP}(nrId_2, A, B, NRRresp, ts_{TSA}(NRRresp))$
- 2.4 $TTP \rightarrow A : nrId_2, A, B, resp, NROresp, ts_{TSA}(NROresp)$
- 2.5 $TTP \rightarrow B : nrId_2, A, B, NRRresp, ts_{TSA}(NRRresp)$
- 2.6 $A \rightarrow TTP : enc_{TTP}(nrId_2, A, B, VAL | INVALID, NRVresp, ts_{TSA}(NRVresp))$
- 2.7 $TTP \rightarrow A : nrId_2, rn_B$
- 2.8 $TTP \rightarrow B : nrId_2, A, B, VAL | INVALID, NRVresp, ts_{TSA}(NRVresp)$

Figure 3.5: Fair non-repudiable service invocation

There are now two stages to each phase of service invocation. The first stage involves the exchange of application-level input (request or response) for a receipt. The second stage involves the exchange of application-level validation of the input for a receipt for its validation.

The extension of the Coffey-Saidha protocol to integrate application-level validation involves the exchange of a secure random authenticator for the NRV token. In the request phase, *A* generates rn_A that the TTP will exchange with *B* for both their validation decision and the $NRVreq$ token. In the response phase *B* generates rn_B that the TTP will exchange with *A* for both their validation decision and the $NRVresp$ token. To allow verification of an authenticator, and to bind it to the other evidence, a digest of the authenticator is included in the relevant protocol run identifier. That is, $nrId_1$ includes a digest of rn_A , $h(rn_A)$, so that both TTP and *B* can verify that rn_A is a valid authenticator. Similarly, $nrId_2$ includes a digest of rn_B . Section 2.1.7 describes the generation of identifiers that include such bindings to secure random numbers.

Figure 3.5 presents the two phases of fair non-repudiable service invocation based on my

modifications to the Coffey-Saidha protocol. A detailed description of the basic protocol is provided in Section 2.2.2.1. Here I emphasise the following points with respect to the modifications to the main protocols for each phase.

1. In the request phase, steps 1.6 to 1.8 are added for application-level validation. In particular, note the provision of rn_A to B in step 1.7 as a receipt for the validation decision provided in step 1.6. As previously stated, $nrId_1$ includes $h(rn_A)$. So, rn_A is cryptographically bound to each step of the request phase protocol and can be used as an authenticated receipt. B encrypts the validation message in step 1.6 to ensure that A cannot both obtain B's validation of their request and prevent B from receiving rn_A . Since rn_A is cryptographically bound to the other messages and was provided to the TTP by A in step 1.1, there is no need to sign or encrypt the message in step 1.7. Corresponding additions, steps 2.6 to 2.8, are made to the response phase. A obtains rn_B in step 2.7 as a receipt for its validation of the response in step 2.6. For this phase, $nrId_2$ includes $h(rn_B)$. So, rn_B is cryptographically bound to each step of the response phase protocol and can be used as an authenticated receipt.
2. To correlate request and response phases, B includes $nrId_1$ in step 2.1 of the response phase and the TTP relays $nrId_1$ to A in step 2.2. Both $NROresp$ and $NRRresp$ provide a cryptographic binding of the correlation.
3. Step 1.8 of the request phase corresponds to step 1.2 of the direct exchange in Figure 3.4. If the request is considered valid by B, step 1.8 is the point at which A knows that the request will be passed to the service for processing. Similarly, step 2.8 corresponds to step 2.2 of the direct exchange.
4. To support exception handling, the TTP associates a termination state with each phase of an invocation or, more precisely, with each execution of a main exchange protocol. The termination state is *SUCCEEDED* if the exchange completed successfully and *ABORTED* if the exchange is cancelled.

In exceptional circumstances either A or B may request that the TTP terminate a phase of an invocation before it completes. Such requests typically occur because either A or B is

concerned about the liveness of protocol execution, whether as a result of non-cooperation of a participant or extraneous factors such as network delays. Next, I define exception handling sub-protocols for the TTP to satisfy termination requests from either A or B. I then provide an analysis of this Coffey-Saidha-based approach to non-repudiable service invocation, followed by a discussion of protocol optimisation where the in-line TTP also performs trusted time-stamping and then an overview of how to achieve non-repudiable service invocation using optimistic fair exchange protocols.

Exception handling

There are two types of termination request:

1. **Abort:** where the requesting party wishes to terminate a phase of the invocation as if that phase had not taken place and neither A nor B receives any useful information about the exchange.
2. **Resolve:** where the requesting party seeks the TTP's assistance to secure normal termination of a phase of the invocation and all expected items for that phase (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. The preference for abort or resolve is determined by local policy. For example, a participant may always seek to abort an exchange if it has not completed within some pre-determined time limit. Alternatively, if the request phase has completed and the response phase has commenced, A's local policy may be to always seek successful completion of the response phase. Irrespective of the type of request, it is the TTP's responsibility to ensure that fairness guarantees hold for all well-behaved parties. Depending on the progress of the main protocol for the given phase of an invocation, and whether the termination state of the phase has already been set, the TTP must determine whether to terminate the phase in the *ABORTED* state or the *SUCCEDED* state.² The request phase can terminate in *SUCCEDED* state if and only if:

1. A is entitled to *NRRreq* and some form of decision on the validity of the request, **and**

²Successful termination relates to the exchange of items in Figure 3.5 for the given phase and not to whether the request or response was determined to be valid as a result of application-specific validation.

2. B is entitled to *req* and *NROreq*.

Similarly, the response phase can terminate in *SUCCEDED* state if and only if:

1. A is entitled to *resp* and *NROresp*, **and**
2. B is entitled to *NRRresp* and some form of decision on the validity of the response.

In the request phase, the TTP is empowered to issue the substitute non-repudiation of validation of request, *NRVreq_{TTP}* defined in Table 3.3, in place of *NRVreq*. *NRVreq_{TTP}* is the TTP's signed confirmation that B has not validated *req*. Once the TTP has issued *NRVreq_{TTP}*, no validation of *req* by B will be accepted. *NRVreq_{TTP}* is equivalent to invalidation of *req* with the supplementary information that B did not cooperate in the decision. In effect, *NRVreq_{TTP}* is an abort token for the request validation stage. The TTP's form of invalidation is indicated by the inclusion of *NVAL*, as opposed to *VAL* or *INVAL*, in the relevant message of exception handling sub-protocols. At first sight the substitution places A at a disadvantage, since B can receive *req* and simply decide not to cooperate in its validation. But, B always has the option to autonomously decide that any request is invalid³. Furthermore, A obtains evidence of the fact that B did not provide a decision on the validity of the request. Thus, in terms of the evidence exchanged, the substitution of *NRVreq_{TTP}* for *NRVreq* is fair. Given this substitution, and recalling the observations from Section 2.2.2.1, fairness can be guaranteed to both A and B in the request phase if:

1. the protocol in phase 1 of Figure 3.5 completes normally; or
2. B chooses not to engage in the main protocol by not responding to step 1.2 (at this point B has no useful information about *req* or *NROreq*); or
3. the exchange is aborted when the main protocol has progressed no further than step 1.3 (at this point A and B still have no useful information about each other's items); or
4. the exchange is completed successfully after execution of step 1.3.

³A decision that may trigger *extra*-protocol dispute resolution.

On execution of step 1.3, the TTP has all the information necessary to complete the exchange. At step 1.4 the TTP releases *req* and *NROreq* to B. Therefore, from step 1.4 onwards the TTP must guarantee that all expected items are available to both A and B. The pivotal point in the main protocol for the request phase is step 1.3. Before step 1.3, the TTP can only respond to either type of termination request by aborting the exchange. Upon execution of step 1.3, the TTP has *rn_A*, *req*, *NROreq* and *NRRreq* but is yet to complete the release of information to either A or B. Thus, at this point, the TTP can satisfy whichever type of termination request they receive first. Once the TTP releases critical information, in step 1.4, they must respond to a termination request by successfully resolving the exchange. Similar observations hold with respect to the response phase of an invocation.

In summary, in the request phase, there are two key points in the main exchange:

1. the pivot point at step 1.3 when B provides a receipt for the request, before which the exchange can only be aborted and after which it must run to some form of normal termination, and
2. step 1.6 when B provides their validation of the request.

As stated previously, the TTP records the termination state of the exchange — either *SUCCEEDED* or *ABORTED*. In order to recover non-repudiation evidence for either A or B, the TTP also maintains a log of protocol messages from which the TTP can determine the progress of the main exchange. So, the TTP decides whether the request phase exchange is *abortable* or *resolvable* or *verifiable*, where

1. an exchange is *abortable* if the TTP has set the termination state to *ABORTED* or the main exchange protocol has not progressed beyond step 1.3 (step 1.4 has not been executed),
2. an exchange is *resolvable* if the TTP has not set the termination state to *ABORTED* and the main exchange protocol has progressed beyond step 1.2 (at least step 1.3 has been executed), and
3. an exchange is *verifiable* if B provides their validation decision (in step 1.6 of the main exchange protocol) before the TTP has set the exchange termination state.

Token	Definition
<i>abortRequest</i>	$nrId_1, A, B, ABORT$
<i>resolveRequest</i>	$nrId_1, A, B, RESOLVE$
<i>abortToken</i>	$nrId_1, A, B, ABORTED$
<i>resolveToken_A</i>	$nrId_1, A, B, NRRreq, ts_{TSA}(NRRreq)$
<i>resolveToken_B</i>	$nrId_1, A, B, req, NROreq, ts_{TSA}(NROreq)$
<i>NRVreq_{TTP}</i>	$sig_{TTP}(nrId_1, NVAL)$

Table 3.3: Tokens for request phase exception handling

Table 3.3 defines the tokens used in exception handling for the request phase of service invocation. A request from $P \in \{A, B\}$ to the TTP to abort the request phase of Figure 3.5 results

A1	$P \rightarrow TTP$:	$nrId_3, abortRequest, sig_P(nrId_3, abortRequest),$ $ts_{TSA}(sig_P(nrId_3, abortRequest))$
			if <i>abortable</i> :
	A2.1	$TTP \rightarrow P$: $nrId_3, abortToken, sig_{TTP}(nrId_3, abortToken),$ $ts_{TSA}(sig_{TTP}(nrId_3, abortToken))$ – TTP sets termination state to <i>ABORTED</i>
			else if not <i>verifiable</i> :
	A2.2	$TTP \rightarrow P$: $nrId_3, resolveToken_P, NVAL, NRVreq_{TTP},$ $ts_{TSA}(NRVreq_{TTP})$ – TTP sets termination state to <i>SUCCEEDED</i>
			else:
	A2.3	$TTP \rightarrow P$: $nrId_3, rn_A, resolveToken_P, VAL INVALID, NRVreq,$ $ts_{TSA}(NRVreq)$ – TTP sets termination state to <i>SUCCEEDED</i>

Figure 3.6: Request phase abort sub-protocol

in execution of the abort sub-protocol shown in Figure 3.6. In step A1, P submits a signed request, *abortRequest*, to abort the exchange identified by $nrId_1$. The abort sub-protocol is uniquely identified by $nrId_3$. P 's signature over both $nrId_3$ and *abortRequest*, which includes $nrId_1$, serves to bind the sub-protocol to the main exchange. The TTP then checks the state of the exchange. If the exchange is *abortable*, then in step A2.1 the TTP provides P with non-repudiable evidence of abort of the exchange; consisting of *abortToken* and the TTP's signature over *abortToken*. On execution of step A2.1, the TTP sets the exchange termination state to *ABORTED*. If the exchange cannot be aborted then the TTP checks whether the exchange is *verifiable*. If the exchange is not *verifiable*, the TTP executes step A2.2 to complete

a successful exchange with substitute $NRVreq_{TTP}$. Otherwise, the TTP executes step A2.3. If either step A2.2 or A2.3 is executed, then the TTP sets the termination state to $SUCCEDED$ and provides P with the information that they expect to obtain from a successful exchange: $resolveToken_A$ for $P = A$ and $resolveToken_B$ for $P = B$. In addition, to authenticate B's validation of req , the TTP provides rn_A in step A2.3.

R1	$P \rightarrow TTP$:	$nrId_4, resolveRequest, sig_P(nrId_4, resolveRequest),$ $ts_{TSA}(sig_P(nrId_4, resolveRequest))$
if not <i>resolvable</i> :			
R2.1	$TTP \rightarrow P$:	$nrId_4, abortToken, sig_{TTP}(nrId_4, abortToken),$ $ts_{TSA}(sig_{TTP}(nrId_4, abortToken))$ — TTP sets termination state to $ABORTED$
else if not <i>verifiable</i> :			
A2.2	$TTP \rightarrow P$:	$nrId_4, resolveToken_P, NVAL, NRVreq_{TTP},$ $ts_{TSA}(NRVreq_{TTP})$ — TTP sets termination state to $SUCCEDED$
else:			
A2.3	$TTP \rightarrow P$:	$nrId_4, rn_A, resolveToken_P, VAL INVALID, NRVreq,$ $ts_{TSA}(NRVreq)$ — TTP sets termination state to $SUCCEDED$

Figure 3.7: Request phase resolve sub-protocol

A request to resolve the request phase results in execution of the resolve sub-protocol shown in Figure 3.7. A signed request to resolve the main protocol ($resolveRequest$) starts the protocol. The sub-protocol is uniquely identified by $nrId_4$, which is bound to the main exchange protocol by P 's signature over $resolveRequest$ and $nrId_4$. Apart from these changes, the only significant difference to the abort sub-protocol is that the TTP executes step R2.1 if the exchange is not *resolvable*. That is, step R2.1 is executed if the termination state has already been set to $ABORTED$ or the main exchange has not progressed beyond step 1.2. Otherwise, the TTP terminates the exchange successfully by executing either step R2.2 or step R2.3, as appropriate.

The exception-handling sub-protocols for the response phase are essentially the same as those for the request phase. The differences are:

1. that the definition of *abortable*, *resolvable* and *verifiable* refer to the relevant steps of

Token	Definition
<i>abortRequest</i>	$nrId_2, A, B, nrId_1, ABORT$
<i>resolveRequest</i>	$nrId_2, A, B, nrId_1, RESOLVE$
<i>abortToken</i>	$nrId_2, A, B, ABORTED$
<i>resolveToken_A</i>	$nrId_2, A, B, resp, NROresp, t_{TSA}(NROreq)$
<i>resolveToken_B</i>	$nrId_2, A, B, NRRresp, t_{TSA}(NRRresp)$
<i>NRVresp_{TTP}</i>	$sig_{TTP}(nrId_2, NVAL)$

Table 3.4: Tokens for response phase exception handling

the main response phase protocol (2.3, 2.2 and 2.6, respectively),

2. the tokens in Table 3.4 replace the corresponding tokens from Table 3.3, in particular *NRVresp_{TTP}* is the TTP's substitution for *NRVresp*, and
3. in the final step of each sub-protocol *NRVresp* replaces *NRVreq* and *rn_B* replaces *rn_A*.

Once the termination state of a service invocation phase has been set, the TTP will forever respond in the same way to any subsequent request to abort or resolve the identified phase. In effect, the TTP suspends the main exchange protocol once a corresponding exception handling sub-protocol has been initiated. Thus, the TTP guarantees to provide A and B with an identical view of the success or failure of a service invocation phase irrespective of whether normal or exceptional termination occurred, and irrespective of who requested termination.

If the request phase of a service invocation is aborted then:

- from the server's viewpoint it will be as if no request was made by the client — the target service has no knowledge of the aborted request and the response phase of the invocation does not ensue; and
- from the client's viewpoint it will be as if their request was not delivered to the server and an appropriate failure notification is provided to the client.

If the response phase of a service invocation is aborted then:

- from the server's viewpoint it will be as if the service response could not be delivered to the client, and

- from the client's viewpoint it will be as if no response was generated and an appropriate failure notification is provided to the client.

In the event of abort of the response phase of an invocation, both client and server will have irrefutable evidence that the request phase had completed successfully. There will also be irrefutable evidence of who requested abort of the response phase. Thus a well-behaved party will be able to demonstrate their correct behaviour should a dispute arise over the missing service response.

The resolution of either request or response phase is treated as successful completion of the phase. In steps A2.2 and R2.2 of request phase exception handling the phase is resolved without application-level validation of the request. In this case, the phase is deemed to have terminated as if the request had been invalidated, and it is not processed by the server. Likewise for resolution of the response phase without application-level validation, in which case the response will not be passed to the client for processing.

Protocol analysis

The informal analysis that follows builds on the analysis of the original Coffey-Saidha protocol in Section 2.2.2.1 to cover my extensions for application-level validation of messages, exception handling and the correlation of exchanges for the two phases of service invocation. The analysis supports the assertion that the TTP can guarantee fairness to both A and B for the exchange of irrefutable evidence of validated service invocation. The basis of the guarantee is that the TTP controls the exchange of information between A and B. In particular, the encryption of items in the first, third and sixth steps of each phase ensures that neither A nor B can obtain useful additional information about an exchange until the TTP has all of the information necessary to deliver the fairness guarantee⁴.

The use of protocol run identifiers in each protocol step, the binding of identifiers to the random authenticators, the correlation of phases and sub-protocols through their identifiers, and the signatures over the various items, ensures that all protocol steps and service invocation phases are cryptographically bound to each other. That is, given any protocol message, a

⁴Including the random authenticator for the relevant phase (rn_A for request, rn_B for response).

relying party can verify message integrity and its relationship to other messages in the same protocol and to other non-repudiation protocols that are related to the same service invocation. It is possible to verify that the signed parts of protocol messages are consistent with the unsigned parts and, therefore, to detect internally inconsistent messages. It is possible to detect inconsistency between messages because of the cryptographic link of each message to its uniquely identified protocol run and to correlated runs. Any attempt to subvert an exchange by generating inconsistent message content can be detected as misbehaviour — a special case of non-cooperation by the misbehaving party.

I now show how fairness can be guaranteed in the request phase⁵ in the face of attempted subversion by either A or B, or in case of their non-cooperation. In the following, A and B can independently specify timeouts for forward progress of a given exchange. Upon expiry of a timeout, for whatever reason, they can independently request that the TTP terminate an exchange by executing one of the exception handling sub-protocols.

Attempted request phase subversion by A A gains unfair advantage if they are able to obtain B's *NRReq* and prevent B from receiving the corresponding *req*. To this end, A can intercept the message from B to the TTP in step 1.3 and then initiate abort of the exchange before the TTP sends *req* to B in step 1.4. However, the message in step 1.3 is encrypted with the TTP's public key. Therefore, A can gain no useful information from this attempted subversion. The TTP will simply abort the exchange and, upon request, will provide the same termination evidence to B. Similarly, encryption of B's message in step 1.6 ensures that A cannot access B's validation decision before B is entitled to receive the authenticator rn_A from the TTP. After execution of step 1.4, the TTP will not satisfy any request from A to abort an exchange. Between execution of step 1.3 and 1.6, A can persuade the TTP to terminate the exchange without application-level validation by B. However, A does not obtain any advantage over B because the request is not processed by B's server unless it is valid and the authenticator rn_A has been provided by the TTP as a receipt for request validation. That is, the request is not considered valid for processing by B's server until the exchange has completed successfully for B with the

⁵Shown in Figure 3.5 on page 88.

receipt of rn_A .

Attempted request phase subversion by B B does not receive any useful information until after they provide the encrypted receipt in step 1.3 of the main protocol. The TTP will not satisfy a request from B to abort the exchange after execution of step 1.4. Therefore B cannot obtain A's request without providing a valid receipt that is verifiable by the TTP. B may cease cooperation after step 1.4. In this case, B will have obtained the request in return for their receipt. This cannot compromise the fairness guarantee to A. A can use an exception handling sub-protocol to obtain the receipt from the TTP and they will also obtain the TTP's certification that B did not provide application-level validation of the request. In this case, from the application client's viewpoint at A, the service invocation failed. The TTP's guarantee in these circumstances is that they will not provide B with the authenticator rn_A as a receipt for any validation evidence that B may subsequently provide. Instead, B will receive the *NVAL* and *NRVreq_{TTP}* that was provided to A. So, B will not be able to show that their validation was made available to A. B can only demonstrate that they have made validation evidence available to A, through the TTP, if they can also produce the authenticator rn_A .

The response phase of service invocation is essentially a repeat of the request phase with the roles of A and B reversed. Thus the preceding observations about the request phase apply to the response phase. In addition, there is the possibility B may not start the response phase despite having received and validated A's request. Alternatively, B's response may be delayed. In these circumstances, there will be an application-level time-out at A and A will not cooperate in any subsequent response phase exchange that B may attempt to initiate. Should B commence the response phase and A not engage in the exchange, B will eventually seek abort of the exchange through the TTP. Thus, both A and B can determine that the invocation failed in spite of successful completion of the request phase.

Taken together, the main protocol and exception handling protocols for each phase enable the TTP to guarantee fairness to A and B. There are no circumstances in which B receives A's request and *NROreq* without B providing a corresponding receipt. B does not obtain the request phase authenticator, rn_A , unless they provide non-repudiation of validation of the request. A

cannot obtain B's validation decision unless rn_A is available to B. Similarly, A cannot obtain B's response without provision of a receipt. The response authenticator, rn_B , is not available to A unless their validation of the response is available to B. B cannot obtain A's validation decision unless rn_B is available to A.

On successful completion of request and response phases, A and B both have the non-repudiation evidence that they expect. If an exchange is aborted, A and B can obtain TTP-certified evidence of the abort of the request and/or response phase of an invocation. That is, we have confirmed that interceptors at client and server can execute fair exchange to deliver the following guarantees (recalled from the introduction to Section 3.2):

1. a request is only passed to the server for processing if: (i) the client-side interceptor provides valid evidence of origin (NROreq); (ii) the request is valid with respect to contract, as indicated by NRVreq provided by the server-side interceptor; and (iii) the server-side interceptor provides evidence of receipt of the request (NRRreq); **and**
2. a response is only passed to the client for processing if: (i) the server-side interceptor provides valid evidence of origin of the response (NROresp), (ii) the response is valid with respect to contract, as indicated by NRVresp provided by the client-side interceptor, and (iii) the client-side interceptor provides evidence of receipt of the response (NRRresp).

Protocol time-stamping optimisation

The preceding protocol descriptions assume that signature verification requires a trusted time-stamp for all signed evidence (see the time-stamps in Figures 3.5, 3.6 and 3.7). That is, a time-stamp must be obtained from a TSA before sending any message that contains new signed evidence. One advantage of in-line TTP protocols is that it may be possible to combine the role of guarantor TTP and TSA. In this case, neither A nor B submit data to a TSA for time-stamping but rely on time-stamps applied by the guarantor TTP (as shown in Figure 2.6 on page 52).

Figure 3.8 shows the time-stamping optimisation for the request phase of service invocation. On receipt of the first message in step 1.1, the TTP generates a time-stamp on *NROreq*. The TTP also generates a time-stamp on *NRRreq* at step 1.3. The TTP sends both of these

-
- 1.1 $A \rightarrow TTP : enc_{TTP}(nrId_5, A, B, rn_A, req, NROreq)$
 - 1.2 $TTP \rightarrow B : nrId_5, A, B, h(NROreq)$
 - 1.3 $B \rightarrow TTP : enc_{TTP}(nrId_A, A, B, NRRreq)$
 - 1.4 $TTP \rightarrow B : nrId_5, A, B, req, NROreq, ts_{TTP}(NROreq), ts_{TTP}(NRRreq)$
 - 1.5 $TTP \rightarrow A : nrId_5, A, B, NRRreq, ts_{TTP}(NRRreq), ts_{TTP}(NROreq)$
 - 1.6 $B \rightarrow TTP : enc_{TTP}(nrId_5, A, B, VAL | INVALID, NRVreq)$
 - 1.7 $TTP \rightarrow B : nrId_5, rn_A, ts_{TTP}(NRVreq)$
 - 1.8 $TTP \rightarrow A : nrId_5, A, B, VAL | INVALID, NRVreq, ts_{TTP}(NRVreq)$
-

Figure 3.8: Fair exchange of request with TTP as TSA

time-stamps to B in step 1.4 and to A in step 1.5. Finally, on receipt of B's validation message in step 1.6, the TTP generates a time-stamp on *NRVreq*. The TTP sends this time-stamp to B in step 1.7 and to A in step 1.8. These optimisations eliminate the six message communication overhead to obtain time-stamps from a TSA for *NROreq*, *NRRreq* and *NRVreq*. Similarly, the use of the TTP as TSA in the response phase eliminates the communication overhead to time-stamp *NROresp*, *NRRresp* and *NRVresp*. Application of the optimisation to the exception handling sub-protocols, removes the need for a time-stamp on the first message of each sub-protocol.

Optimistic fair exchange for service invocation

A disadvantage of the use of an in-line TTP for fair exchange is the additional communication overheads imposed by relaying every message through the TTP. So, in some circumstances it is desirable to use optimistic fair exchange, supported by an off-line TTP, to avoid these overheads. As with in-line TTP fair exchange, the problem is to integrate application-level validation and preserve fairness. In the extension to the Coffey-Saidha protocol it was shown how an initiator of an invocation phase, whether request or response, can provide an authenticator for the TTP to subsequently exchange for evidence of application-level validation. Clearly, this is not possible when the TTP is off-line. Therefore, to guarantee fairness, the initiator of an invocation phase must provide a receipt for application-level validation that is directly bound to the relevant evidence. To achieve this, we correlate two executions of an optimistic exchange protocol in each phase — one for the receipt stage and the other for the validation stage. For

example, in the request phase of an invocation, there is one protocol run to exchange the request for its receipt and then a second protocol run to exchange application-level validation of the request for a receipt for the validation evidence. Protocol run identifiers correlate the runs for each stage. I now use Wang's protocol, described in Section 2.2.2.2, to illustrate how to achieve this correlation. Table 3.5 defines the tokens to use in the request phase of a service

Token	Definition
k_A	a secret key generated by A to encrypt their request req
$reqCipher$	$enc_{k_A}(req)$ — the encryption of req using k_A
$nrId_1$	$h(A, B, TTP, h(reqCipher), h(k_A))$ — $nrId_1$ generating function
$kACipher$	$enc_{TTP}(nrId_1, k_A, rn_A)$
$NROreq$	$sig_A(nrId_1, kACipher)$
$NRRreq$	$sig_B(nrId_1, kACipher)$
k_B	a secret key generated by B to encrypt their validation decision val
$valCipher$	$enc_{k_B}(val)$ — the encryption of val using k_B
$nrId_2$	$h(A, B, TTP, h(valCipher), h(k_B))$ — $nrId_2$ generating function
$kBCipher$	$enc_{TTP}(nrId_2, k_B, rn_B)$
$NROval$	$sig_B(nrId_2, nrId_1, kBCipher)$ — non-repudiation of B's validation decision
$NRRval$	$sig_A(nrId_2, nrId_1, kBCipher)$ — A's receipt for B's validation decision

Table 3.5: Tokens used in optimistic non-repudiable service invocation

invocation. These correspond to the tokens presented in Table 2.5 for Wang's protocol. The only difference is that the non-repudiation evidence for the second stage, $NROval$ and $NRRval$, cryptographically binds the identifiers of the two protocol runs together. Figure 3.9 shows the

-
- 1.1 $A \rightarrow B : nrId_1, A, B, TTP, reqCipher, h(k_A), kACipher, NROreq, ts_{TSA}(NROreq)$
 - 1.2 $B \rightarrow A : nrId_1, NRRreq, ts_{TSA}(NRRreq)$
 - 1.3 $A \rightarrow B : nrId_1, k_A, rn_A$
 - 1.4 $B \rightarrow A : nrId_2, A, B, TTP, nrId_1, valCipher, h(k_B), kBCipher, NROval, ts_{TSA}(NROval)$
 - 1.5 $A \rightarrow B : nrId_2, NRRval, ts_{TSA}(NRRval)$
 - 1.6 $B \rightarrow A : nrId_2, k_B, rn_B$
-

Figure 3.9: Optimistic fair exchange for request phase of service invocation

complete optimistic protocol for the request phase of an invocation. Steps 1.1 to 1.3 show the first protocol run in which A's request, req , and $NROreq$ are exchanged for B's $NRRreq$. Steps 1.4 to 1.6 show the correlated protocol run to exchange B's validation decision, val , and

NROval (non-repudiation of the validation decision) for A's *NRRval*. These two correlated protocol runs ensure that:

1. B only obtains the request if they provide A with a receipt and that A cannot deny having submitted the request, and
2. A only obtains the validation decision if they provide B with a receipt and that B cannot deny having generated the validation decision.

The validation decision is cryptographically bound to the request through the signatures over the correlation of protocol run identifiers in *NROval* and *NRRval*. As shown in Table 3.5, the identifier *nrId₁* is cryptographically bound to the request and *nrId₂* is cryptographically bound to the validation decision. So, the non-repudiation evidence includes the correlation between the protocols runs that combine to provide both non-repudiation and application-level validation of the request phase. A corresponding correlation of protocols runs is used to achieve non-repudiation and validation in the response phase. Finally, as for in-line TTP fair exchange, correlation of identifiers is used to bind response phase and request phase of a service invocation. Thus, a complete request/response service invocation involves four runs of the optimistic exchange protocol — two for receipting and validation of the request and another two for receipting and validation of the response. It is possible that A or B may cease cooperation at any point during an invocation. In the event of non-cooperation Wang's exception handling sub-protocols, described in Section 2.2.2.2, can be used to guarantee fair termination through the identified TTP of any one of the four protocol runs. Termination of one of the protocol runs may lead to failure of the service invocation. However, both A and B can still recover the evidence they are entitled to through the TTP. That is, the application-level failure of the invocation does not signify a loss of fairness to either A or B. Other optimistic protocols can be combined in a similar way.

As for the other examples, it is assumed that trusted time-stamps are required for signature verification. Figure 3.9 shows time-stamps over *NROreq*, *NRRreq*, *NROval* and *NRRval* in steps 1.1, 1.2, 1.4 and 1.6 of the request phase. As there is no in-line TTP, it is not possible to combine TTP and TSA roles. Therefore, given the signature verification assumption, there is

a fixed eight message overhead to obtain time-stamps in each phase of the service invocation. This compares with the six message overhead in each phase for the un-optimised in-line TTP exchange. Section 3.5 concludes the chapter with further discussion of the relative merits of different approaches to non-repudiable service invocation. A novel aspect of my approach is that the interceptor mediation introduced in Section 3.1, as realised using the framework described in Chapter 4, will support the deployment of any of the mechanism discussed so far, and others, to meet different application requirements. I now define the service for non-repudiable information sharing that was proposed in Section 1.3.2.

3.3 Non-repudiable information sharing

The primary purpose of non-repudiable information sharing is to maintain the integrity of information that is shared by two or more collaborating organisations. We preserve the integrity of the information by ensuring that changes to the information are legitimate — changes are subject to the agreement of the organisations that share, or jointly own, the information. To ensure that members of the sharing group are accountable for their actions, irrefutable evidence of the proposal of and agreement to changes must be generated and logged.

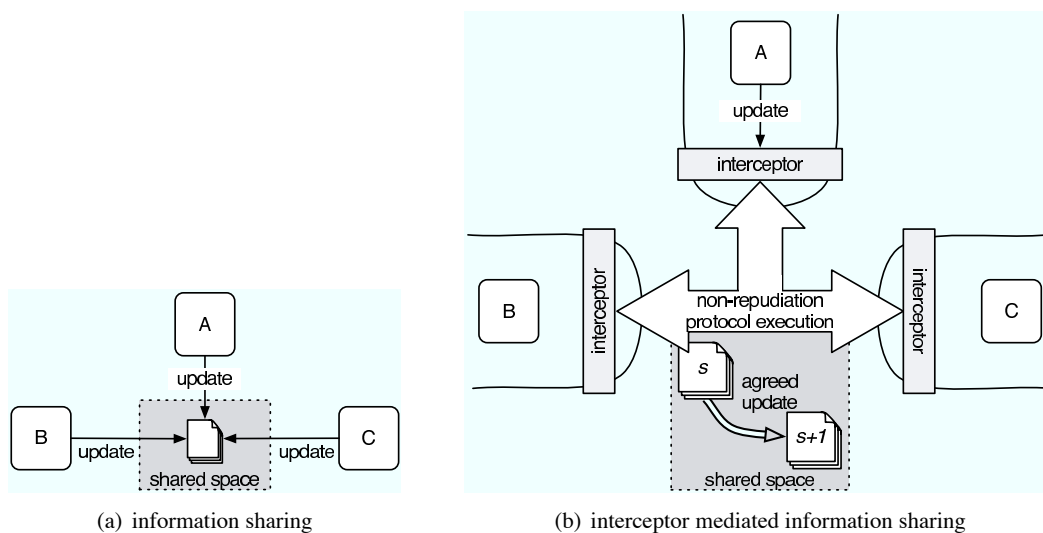


Figure 3.10: Non-repudiable information sharing

Figure 3.10a recalls the logical view of information sharing from Section 1.3.2. Conceptu-

ally, shared information resides in a shared space that can be accessed by the members of the sharing group: A, B and C. For example, if A wishes to update the shared information, then they must seek the agreement of B and C. Further, A's proposed update must be irrefutably bound to A and there must be irrefutable evidence of the decisions of B and C with respect to the proposed update. Assuming B and C agree to the update, then each party shares an agreed view of the new state of the shared information and no party can misrepresent this view. The sharing group execute a non-repudiation protocol to establish the agreed view. As with non-repudiable service invocation, interceptors mediate the interaction in order to allow flexibility in the approach taken to achieve non-repudiation and to maintain the application-level abstraction presented to sharing group — A, B and C in the example.

Figure 3.10b shows the use of interceptors to execute a non-repudiation protocol for an agreed update to shared information from state s to state $s+I$. This corresponds to the change illustrated in Figure 1.7 on page 19. The interceptors ensure that A, B and C have a consistent view of the agreed state of shared information and of the group who have access to the information. That is, A, B and C have common knowledge of the state of the shared information and irrefutable evidence that it is the agreed view of information state. They also know, and have irrefutable evidence, that they comprise the sharing group that owns the information. Their ownership of the information means that each of A, B and C must agree to any changes both to the state of the information and to the membership of their group. The interceptors also present a local representation of the information for application-level access at A, B and C. As far as possible, the participants are free to concentrate on application-level concerns while maintaining a non-repudiable, agreed view of the information they share.

Section 3.3.1 introduces the notion of information control state to describe transitions in the state of the shared information and in the membership of the sharing group. In Section 3.3.2 I describe the non-repudiation protocols that have been implemented to maintain the consistent view of shared information by reaching agreement on control state transitions. The multi-party protocols coordinate both state and membership changes between a sharing group of two or more members. Section 5.3 describes how this interceptor-mediated information sharing is realised using distributed object middleware and how the middleware can be extended to

support transactional access to shared information.

3.3.1 Information control state

Information control state is meta-information that uniquely identifies a view of shared information. The control state includes references to the previously agreed control state, to the state of the shared information and to the sharing group. Figure 3.11 depicts a sequence of agreed

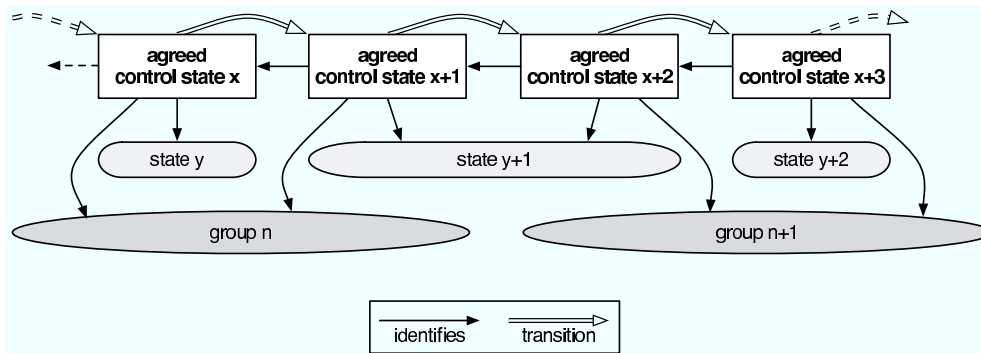


Figure 3.11: Agreed control state transitions

transitions between control states from control state x through to control state $x + 3$. In the figure, x , $x + 1$, $x + 2$ and $x + 3$ are simply labels to distinguish between the different control states. Similarly, y , $y + 1$ and $y + 2$ label different states of the shared information. Labels n and $n + 1$ distinguish between different sharing groups. The agreed transitions progress from left to right (see the double-lined arrows in Figure 3.11). Shared information is defined by a tuple that consists of the application state of the information and the membership of the sharing group. For example, in the tender processing application from Section 1.1.2, the information (application) state consists of the current values of the various attributes of the tender offer and the members of the sharing group are the processes at supplier and purchaser that share the offer (or the supplier and purchaser, for short). As shown in Figure 3.11, a control state identifies the current information state and sharing group. The control state transitions correspond to changes in the shared information. The new control state identifies the corresponding new information state or new sharing group, and also identifies the previously agreed control state. Table 3.6 summarises the changes to shared information represented by transitions from control state x to $x + 3$. So, the transition from control state x to control state $x + 1$ corresponds

Control state	Shared information
x	$\langle \text{state } y, \text{group } n \rangle$
$x+1$	$\langle \text{state } y+1, \text{group } n \rangle$
$x+2$	$\langle \text{state } y+1, \text{group } n+1 \rangle$
$x+3$	$\langle \text{state } y+2, \text{group } n+1 \rangle$

Table 3.6: Control state transition table

to a change in shared information from state y to state $y+1$ that was agreed by group n . Transition from control state $x+1$ to control state $x+2$ corresponds to a membership change agreed by group n in which a participant leaves or joins the group to form new group $n+1$. Transition from control state $x+2$ to $x+3$ represents a further state change agreed by group $n+1$. Figure 3.12 shows the general case of both agreed transitions to control state and unsuccessful

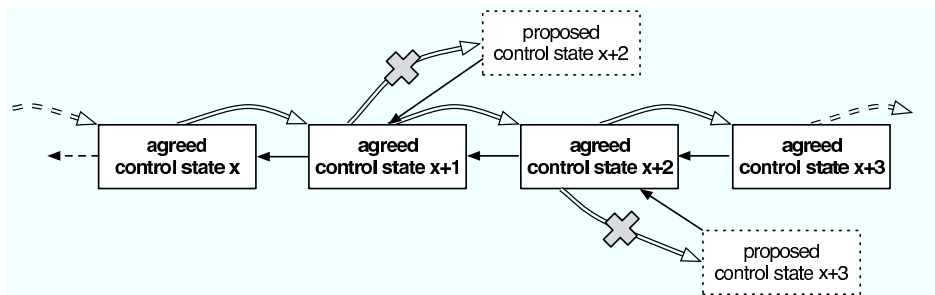


Figure 3.12: Proposed and agreed control state transitions

ful proposed transition. In this case, proposed control state $x+2$ and proposed control state $x+3$, in dashed boxes, represent invalid branches of the shared information. The references to previously agreed control state, illustrated by the back arrows between control states, link a sequence of control states together. Thus a history of the agreed versions of the shared information can be constructed even if the associated state and sharing group do not convey any version information. Section 3.3.2.1 defines the control state meta-information that identifies successive control states and associated shared information.

The model of transitions in control state completely describes the installation of successive changes to the referenced shared information, including the identification of the group that decided upon the transitions. In this dissertation I show how all changes to shared information can be described in terms of control state transitions. In particular, Section 3.3.2.1 shows how the references to information state and to sharing group cryptographically bind control state

to its associated shared information. Coordinated, non-repudiable transitions between control states can support: (i) agreement on state changes, (ii) agreement on membership changes, and, with appropriate extensions, (iii) concurrency control and transactional access.

3.3.2 Coordination protocols

Each member of a sharing group has a view of shared information that is fully identified by their view of the currently agreed control state. The coordination protocols I present in this section ensure that this view remains identical for all well-behaved members of the group. There are protocols to regulate changes to the state of shared information and to the sharing group. The aim of the proposer of a change is that all parties should install (agree to) a new view of the shared information: the new control state and the associated information state and group membership. Whether for state changes or for membership changes, agreement is achieved by non-repudiable transitions in control state. Each of the protocols is based on the execution of signed two-phase commit.

A proposed change to shared information is valid if there is unanimous agreement to the transition to the new view of the shared information. A proposed change is invalid if any party vetoes the transition. The requirement for unanimous agreement to a proposed change is fundamental to the following discussion. It is imposed because, in the context of mutually mistrusting parties, it cannot be assumed that majority agreement to a change will suffice. For example, any majority decision on the validity of a change to an offer that is shared by a purchaser, a supplier and a delivery agent is potentially disadvantageous to the minority party. The unanimity restriction ensures that an invalid control state transition cannot be misrepresented as being valid and vice versa.

3.3.2.1 Definition of protocol notation and information control state

Table 3.7 defines the notation used for coordination protocol elements. Where appropriate, the member index, i , is used as shorthand for generation of an element by party P_i , where $i \in 1 : n$ for a group with n members. For example, rn_k is a random number generated by P_k .

Participant identifiers, such as P_k , are typically URIs that uniquely identify a participant

Notation	Definition
G_n	group n sharing information, $G_n = \{P_i i \in 1 : n\}$
$P_k \in G_n$	the identifier of the proposer of a new control state
$RG_k = G_n - P_k$	the recipient group of a proposal from member P_k
$P_j \in RG_k$	the identifier of a member of RG_k , $j \neq k$
$D_{k,j}$	member P_j 's decision on the validity of P_k 's proposal
$newCS_k$	new control state proposed by P_k
$currCS_i$	current control state as viewed by P_i
S_y	information state y

Table 3.7: Notation for coordination protocols

in the context of a given interaction. They may also serve to locate the participant. An email address may be used to identify an individual or organisational participant. A URL may be used to identify a participant service.

The group G_n is an ordered set of the participant identifiers for the members of sharing group n . Unless stated otherwise, G_n is the currently agreed sharing group and it has n members. As shown in Table 3.7, P_k identifies the proposer of a new control state. RG_k is the set of recipients for a proposed change. For example, the sharing group in Figure 3.10b on page 103 is $\{A, B, C\}$. If the proposer of a change, P_k , is A then the recipient group, RG_k , is $\{B, C\}$. Each member of a sharing group knows the membership of their group as identified by the currently agreed control state. That is, each member P_i of group G_n knows the identity of each of the other members of G_n .

S_y is an agreed representation of the information state (application data) that is shared. In the current implementation, information state is represented by an object for local application access. This object representation is then transformed to the agreed representation using a data-binding. For example, the data-binding may map an application object to some text representation such as an XML document (see Section 4.2.3). It is the agreed representation that is signed and propagated during protocol execution. Unless stated otherwise, S_y represents the currently agreed information state.

As described in Section 2.1.7, the protocol run identifier, $nrId$, can be generated by the protocol initiator from a hash of a random number that is contextualised by a URI that is unique to the initiator. For participant P_k , an $nrId$ will consist of $h(rn_k)$ pre-pended by a URI that is unique to P_k . This combination of elements should guarantee the uniqueness of an $nrId$

for the group sharing information — no correctly behaving party will produce an identifier that has been seen before by any member of the sharing group.

<i>nrId</i>	identifier of this control state
<i>prevAgreedNRId</i>	identifier of previously agreed control state
<i>proposerId</i>	identifier of the proposer of this control state
<i>groupId</i>	identifier of the sharing group
<i>stateId</i>	identifier of the information state

Table 3.8: Basic elements of control state

Table 3.8 defines the basic elements of information control state. The current protocol run identifier, *nrId*, uniquely identifies the given control state. *prevAgreedNRId* identifies the previously agreed control state and, therefore, provides the link between successive control states. These links allow the construction of a history of transitions between control states. The *proposerId* uniquely identifies the proposer of a given control state and, therefore, the proposer of the change to shared information. The *groupId* and *stateId* uniquely identify the associated sharing group and information state, respectively. These two identifiers need not be unique across all control states. Both group membership and information state may legitimately return to some prior state.

The *groupId* is generated by calculating a message digest over the identifiers of the members of the sharing group. For example, $h(P_1, P_2, \dots, P_n)$ is the *groupId* for group G_n . This identifier both uniquely identifies G_n and is cryptographically bound to G_n . The identifier can therefore be used to verify the integrity of group membership information.

The *stateId* is cryptographically bound to the information state it identifies. For state S_y , the *stateId* minimally includes $h(S_y)$. The *stateId* may also provide location information to access the identified information state. For example, shared information may be hosted by some third party data repository. In this case, a *stateId* could consist of the repository's URL and a Base64 representation of the digest to identify the specific information state, similar to the generation of protocol run identifiers in Section 2.1.7.

3.3.2.2 Protocol assumptions and invariants

The assumptions with respect to well-behaved parties in Section 2.1.4, and with respect to cryptographic primitives and protocol subversion model in Section 2.1.5, hold for the protocols described in this section. Note, as in earlier protocols, if a signing key is revocable without reference to the relying party, then trusted time-stamps should be associated with signed evidence. However, for brevity, these time-stamps are not shown in the coordination protocol descriptions that follow.

A proposed transition must be disclosed for validation by all members of group G_n . Therefore, it is assumed that disclosure of information about a proposed transition within the sharing group is unconditional. This is analogous to the precedence given to data integrity over disclosure in the Clark-Wilson model (see Section 2.3.1).

The aim of the proposer of a change, P_k , is to reach agreement on transition from their view of the current control state, $currCS_k$, to a new agreed control state, $newCS_k$. The proposer executes a non-repudiation protocol with the other members of group G_n that serves: (i) to disseminate both $newCS_k$ and the associated information required to update each member's local view of information state or group membership, and (ii) to obtain agreement on the transition from $currCS_k$ to $newCS_k$. To begin protocol execution, P_k first generates the elements of $newCS_k$ shown in Table 3.8. The other members of G_n , $P_j \in RG_k$, use the $prevAgreedNRId$ to verify that $newCS_k$ represents a transition from their view of the current control state: $currCS_j$. $prevAgreedNRId$ maintains the linked sequence of agreed transitions. For a valid transition to $newCS_k$, the following invariants must hold:

1. $newCS_k.nrId$ is unique for all proposals seen by the members of G_n .
2. $newCS_k.prevAgreedNRId = currCS_j.nrId$

That is, $\forall P_j \in RG_k$, P_j 's current control state is P_k 's previously agreed control state. If this invariant does not hold, then either: (i) one or more members of RG_k have submitted a concurrent proposal with P_k and, for these members, their current control state is not the previously agreed control state; or (ii) P_k is starting from a different view of agreed control state to that of one or more members of RG_k

3. If $newCS_k.groupId \neq currCS_j.groupId$
 then $newCS_k.stateId = currCS_j.stateId$
 else if $newCS_k.stateId \neq currCS_j.stateId$
 then $newCS_k.groupId = currCS_j.groupId$

That is, to maintain a definitive view of information state when membership changes, P_k cannot propose a membership change and a state change in the same protocol run (or control state transition).

Violation of the preceding invariants leads to invalidation of the proposed transition. A transition is also invalid if any of the non-repudiation evidence is found to be invalid or if any member of the sharing group determines that the proposed change is invalid when subjected to application-level validation.

3.3.2.3 Coordination of state changes

I use the first example state transition in Figure 3.11 on page 105 to describe the state coordination protocol. In the example, the proposer, P_k , wishes to reach agreement on transition from state S_y to state S_{y+1} . Table 3.9 defines the $newCS_k$ and $resp_j$ tokens used in the non-repudiable

Token	Definition
$newCS_k$	$nrId, prevAgreedNRId, P_k, h(G_n), h(S_{y+1})$
$resp_j$	$newCS_k.nrId, currCS_j.nrId, P_j, D_{k,j}$

Table 3.9: State coordination protocol control state and response tokens

two-phase commit protocol shown in Figure 3.13. $newCS_k$ is the new control state proposed by P_k . $resp_j$ is the form of the response that is expected from $P_j \in RG_k$. P_k is committed to the new state at initiation of a protocol run. Thus, a transition is only aborted if it is vetoed by one or more members of RG_k . The final protocol message represents the non-repudiable decision of the whole of G_n on the validity of the proposed transition.

In step 1, P_k propagates $newCS_k$, S_{y+1} and their signature on $newCS_k$. $newCS_k$ contains all the information necessary to identify and verify the transition proposed by P_k . The digest of S_{y+1} included in $newCS_k$ allows a recipient to verify the integrity of S_{y+1} . The signature on $newCS_k$ represents NRO of the proposal and binds the $nrId$ to P_k 's authenticator, rn_k .

-
- 1 $P_k \rightarrow RG_k$: $newCS_k, S_{y+1}, sig_k(newCS_k)$
 - 2 $\forall P_j \in RG_k$
 $P_j \rightarrow P_k$: $resp_j, sig_j(resp_j), sig_j(newCS_k)$
 - 3 $P_k \rightarrow RG_k$: $rn_k, \sum resp_j, \sum sig_j(resp_j), \sum sig_j(newCS_k)$
-

Figure 3.13: State coordination protocol

In step 2, each member of RG_k responds with a message that confirms their view of agreed control state through the identifier $currCS_j.nrId$ and provides their decision, $D_{k,j}$, on the validity of the proposed transition to $newCS_k$. The decision $D_{k,j}$ depends on verification of the integrity of the proposal and on application-level validation of the new state S_{y+1} . The inclusion of $newCS_k.nrId$ in $resp_j$, along with $currCS_j.nrId$, confirms the recipient's view of the proposed transition. The signature on $resp_j$, including P_j , binds the various elements together, provides NRO of the elements and preserves their integrity. The signature on $newCS_k$ provides NRR of the proposal from P_k . $newCS_k.nrId$ irrefutably binds $newCS_k$ to $resp_j$. Taken together, $resp_j$ and the signature on $newCS_k$ allow any party to verify whether the invariants identified in Section 3.3.2.2 hold and whether all members of RG_k have seen the same proposal from P_k .

In step 3, P_k disseminates the collected responses and NRR tokens along with the authenticator rn_k . Since only P_k knows the value of rn_k , no other party can generate the final message. Thus rn_k authenticates the group's decision and is sufficient receipt for the responses provided in step 2. Given $newCS_k$ and $\sum resp_j$, any party can compute the group's decision to commit or abort P_k 's proposal. The final message is irrefutably linked to the other messages in the same protocol run through the authenticator and the signed responses (which include $newCS_k.nrId$).

G_n 's authenticated decision on P_k 's proposal is:

$$rn_k, newCS_k, \sum sig_i(newCS_k), \sum resp_j, \sum sig_j(resp_j)$$

$$i \in 1 : n; \quad j \in \{1 : n \text{ and } j \neq k\}$$

This is non-repudiable evidence of acceptance or rejection of a proposed transition and of the consistency, or otherwise, of the information provided during a protocol run. A successful protocol run allows the consistent installation of a new view of the agreed state by all members of G_n . An unsuccessful run results in the consistent view that the proposed transition is invalid.

In this case, P_k 's *prevAgreedNRId* identifies the agreed view of shared information.

Modifications for partial update

To allow for update to, as opposed to overwrite of, state, the control state can be extended to include a digest of a proposed update, U_k . The extended control state is of the form:

$$newCS_k = \{nrId, prevAgreedNRId, P_k, h(G_n), h(S_{y+1}), h(U_k)\}$$

The inclusion of $h(U_k)$ allows a proposal recipient to distinguish a partial update from a complete overwrite to the information state. In step 1 of the protocol, the partial update to state, U_k , is provided in place of S_{y+1} . U_k may contain changes to certain attributes of S_y that are necessary to perform the update from S_y to S_{y+1} or U_k may define some function over S_y that achieves the update. Since both $h(U_k)$ and $h(S_{y+1})$ are provided in $newCS_k$, it is possible for a recipient to verify the integrity of U_k and to determine that, if the update is agreed and applied, a consistent new state will result.

State change protocol analysis

The protocol aims to reach agreement on a control state transition, to prevent the misrepresentation of that agreement, and, therefore, to prevent misrepresentation of the validity of the associated shared information. Evidence is generated both to ensure that the actions of well-behaved parties cannot be misrepresented by dishonest parties and to detect misbehaviour. In particular:

- a proposed transition is irrefutably bound to its proposer and to the decisions of the parties validating the proposal,
- validation decisions are irrefutably bound to their source — no party can sustain a claim that a vetoed transition is valid or vice versa, and
- there is irrefutable evidence of who participated in a protocol, through confirmation of the membership of the sharing group.

Given the well-behaved party assumption, if all parties behave correctly, liveness is guaranteed despite a bounded number of temporary failures. Incoming protocol messages are logged on receipt. Outgoing protocol messages are logged before sending. In the event of local failure, and upon recovery, a participant can determine the last known state of any active protocol run from the evidence in their local log. They can also use this evidence to request information on the status of a given protocol run from other participants. A well-behaved party may then be able to resume protocol execution following recovery from temporary local failure (see Section 4.4).

There is no guarantee of termination when even one party misbehaves. In part, this is a consequence of the local autonomy and the need for unanimous agreement to changes to shared information. The protocol concerns both the verification of the integrity of messages and the semantic validation of message content. This exacerbates the problem of guaranteeing termination because, for example, it is not possible to deduce anything about the validity of a transition from a failure to respond to a proposal. The protocols are designed to generate the evidence necessary for application-level resolution of any resultant blocking. In practice, timeouts will result in the local propagation of exceptions by the the protocol execution framework. The middleware can be configured to react to such events by, for example, proposing the eviction of members of the sharing group who cease to cooperate. Ultimately, non-cooperation can mean it is impossible for an application to progress. In this case, participants can use the evidence in their logs for application-level resolution of disputes. Guaranteeing termination for well-behaved parties is discussed further in Section 3.5.

I now present an informal analysis to support the assertion that no party can misrepresent the validity of a control state transition. This safety guarantee is that, irrespective of the behaviour of other parties, no well-behaved parties will view an invalid transition to be valid. To deliver the guarantee, the protocol must withstand subversion by members of G_n , whether through deliberate or accidental fault, as well as by intruders. Any attempt to subvert a protocol run by generating inconsistent message content can be detected. In which case, the proposed state transition is invalidated and evidence of misbehaviour is generated. It is possible to verify that the signed parts of protocol messages are consistent with the unsigned parts and, there-

fore, to detect internally inconsistent messages. It is possible to detect inconsistency between messages because $nrId$ and rn_k cryptographically link each message to its uniquely identified protocol run. I now show how the protocol allows detection of other attempts at subversion by members of G_n .

1. A member of G_n does not send a message. If P_k does not initiate the protocol then, by definition, P_k is unable to show that a new state is valid. If a member, P_j , of RG_k does not respond to a proposal, then P_j will have obtained the proposed new state without providing NRR. However, P_j cannot demonstrate the validity of the state (since they will not obtain authenticator rn_k). If P_k fails to send the final message, then all members of RG_k hold evidence that the protocol run is active and any subsequent coordination request will reveal inconsistencies between control states. If P_k has evidence that a new control state has been agreed, then any subsequent action on the information will reveal this agreement. That is, P_k must produce the non-repudiation evidence to support any assertion that the new control state is valid.
2. P_k performs selective sending. If different messages are sent to different members of RG_k , then the inconsistency will be detected in subsequent protocol steps or, in the case of the final message, during any subsequent coordination request. If the first message is not sent to a subset of RG_k , then it is not possible to reach a unanimous decision on the validity of the proposal and P_k cannot produce a valid final message for any member of RG_k . If the final message is not sent to a subset of RG_k , then this subset can show that the protocol run is still active. Further, any honest party who receives the final message can relay it to any other member of RG_k .
3. There is a temporary divergence of the view of agreed state. It is possible for a member, P_j , of RG_k to prepare two response messages: one representing an accept and the other a reject of P_k 's proposal. P_j sends one response to P_k , intercepts a subset of the final messages and, in those messages, substitutes the other response. In this case, if and only if all other members of RG_k accepted the proposal then some members of G_n will install a new view of shared information and others will remain with the previously agreed

view. However, the guarantee that no member of RG_k installs a view that has not been unanimously agreed still holds, as does the guarantee of eventual consistency of the view of agreed state. The next coordination request will reveal the divergent view, as will any attempt to take advantage of the divergence. Thus the divergence is temporary. All parties have the information necessary to install the new view and all parties eventually receive evidence of its validation, at next state or membership coordination. If P_k countersigns the responses, $\sum resp_j$, then divergence can only occur if P_k colludes with a member of RG_k to prepare two different sets of responses.

The non-repudiation evidence generated during a protocol run binds a party to their actions and those actions cannot be misrepresented. Given the subversion model in Section 2.1.5, it is not possible for an intruder to undetectably modify messages between members of G_n and no member of G_n can be forced to agree an invalid transition by the intruder. Thus the most that can be achieved is the detectable disruption of the protocol. In no case is it possible to force a correctly behaving party to install a new view of shared information unless the view has been unanimously agreed.

At the end of a protocol run, a correctly behaving party will either:

1. be able to install a new view of valid state, and hold evidence that it has been unanimously agreed, or
2. hold evidence that the proposed transition has been vetoed.

A misbehaving party cannot misrepresent invalid state as valid. Similarly, they cannot support a claim that valid, unanimously agreed state is invalid. Thus, the protocol delivers the stated safety guarantee.

3.3.2.4 Coordination of membership changes

In this section I describe the connection and disconnection protocols used to coordinate group membership. First, I clarify the roles of *subject* and *proposer* in membership changes. Then, I detail the data propagated with a new control state to effect a membership change. The section concludes with descriptions of the membership change protocols.

The protocols ensure the maintenance of a consistent, non-repudiable view both of the group membership and of the agreed information state at membership changes. The connection protocol is used to reach agreement on a member joining the group. An existing member uses a voluntary disconnection protocol to leave the group or temporarily suspend their membership. An eviction protocol is used to suspend or terminate the membership of uncooperative member(s). Information ceases to be shared after a succession of member disconnections. In this way, each party obtains a verified view of the agreed state of the information at the end of their involvement in sharing. The membership change protocols aim to reach agreement on a membership change that is represented by transition to a new control state. Since each party's view of agreed state is propagated during membership changes as part of the control state, the temporary divergence of view discussed in Section 3.3.2.3 can be resolved during a membership change.

Membership change subject(s) and proposer

The subject of the connection protocol is the party intending to join the sharing group. The subject of the voluntary disconnection protocol is the member leaving the group. The subject(s) of the eviction protocol are the member(s) to be evicted from the group.

The membership change proposer is the existing member of the group responsible for coordinating agreement to a membership change. In addition, if a connection request is agreed, the proposer provides the current agreed information state to the subject. The proposer is also responsible for blocking other coordination requests until a decision on a membership change is reached. Should a connection request be rejected, the proposer restricts the knowledge of the shared information that the subject can acquire. For a voluntary disconnection, the proposer limits the involvement of the subject in protocol execution. The subject(s) of an eviction have no involvement in protocol execution.

Proposer responsibility can be rotated in order to reduce reliance on a single member to propose membership changes. In this case, the proposer of a connection request is unambiguously identified as the most recently connected existing member of the group; that is, given $G_n = \{P_i | i \in 1 : n\}$, ordered by oldest member first, the proposer of a connection request is

P_n . If the connection is agreed, then the proposer of the next request will be P_{n+1} . Similarly, for group G_n , P_n is the proposer of disconnection requests unless P_n is a disconnection subject. If P_n is a disconnection subject then the next most recently connected member, P_{n-1} , is the proposer. If proposer rotation is not required, then the oldest member of the group can act as proposer of all membership changes unless they are the subject of a disconnection request (in which case the responsibility passes to the next oldest member). Since the current proposer can be unambiguously identified, any member of G_n can verify the legitimate proposer for a request and provide this information to a potential new member.

Membership change data and group identifier

To allow all parties to install a new view of group membership, a membership update of the following form is the payload that is propagated with the new control state:

$$MU = \{ADD|REMOVE, subjectList\}$$

where *subjectList* identifies the party joining or the member(s) leaving

The *groupId* included as part of the new control state uniquely identifies the proposed new sharing group. For a connection, *ADD* membership update, the *groupId* is generated by appending the single member of *subjectList* to the existing ordered set of group members and then computing a message digest over the new set. For a disconnection, *REMOVE* membership update, the *groupId* is generated by removing the member(s) of *subjectList* from the existing set of group members and then computing a message digest over the new set.

Membership change protocols

Each of the membership change protocols follows the same pattern:

1. The subject of a connection or voluntary disconnection, or the initiator of an eviction, sends a membership change request to the current membership proposer.
2. The proposer and the existing members of the sharing group, excluding any disconnection subject(s), execute non-repudiable two phase commit to confirm the membership

change. The two phase commit is essentially the same as the state change protocol except that the proposed control state transition represents a membership change and a membership update is the payload. The member eviction protocol terminates at this two phase commit stage.

3. For a connection request, the proposer then provides the connection subject with the group's decision. If the request is accepted, the proposer also provides both the information necessary to join the sharing group and the currently agreed application state of the shared information. So, upon agreement to connection by the existing sharing group, a new member knows the membership of the sharing group, which now includes themselves, and the state of the shared information. For a voluntary disconnection, the proposer provides the subject with the confirmed view of both group membership and information state at their disconnection.

If the outcome of the two phase commit stage is agreement to the membership change, then at protocol termination all members of the new sharing group will have an identical view of the membership of the new group and of the information state that they share. If the existing members of the sharing group that participate in the two phase commit do not agree to the change, then the agreed view of their shared information does not change.

Connection protocol

Figure 3.14 recalls the member change example from Section 3.3.1 in which transition from

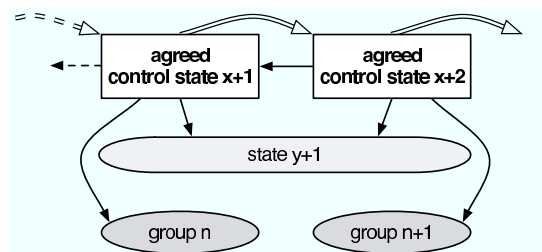


Figure 3.14: Member connection control state transition

control state $x + 1$ to control state $x + 1$ represents the formation of the new sharing group: G_{n+1} . To describe the connection protocol, we assume that connection subject P_{n+1} wishes to

join group G_n to form group G_{n+1} . Table 3.10 defines the MU_{n+1} , $newCS_k$ and $resp_j$ tokens

Token	Definition
MU_{n+1}	$\{ADD, P_{n+1}\}$
$newCS_n$	$nrId, prevAgreedNRId, P_n, h(G_{n+1}), h(S_y)$
$resp_j$	$newCS_n.nrId, currCS_j.nrId, P_j, D_{n,j}$

Table 3.10: Connection protocol tokens

used in the protocol description shown in Figure 3.15. MU_{n+1} is the proposed membership update. $newCS_n$ is the proposed new control state generated by membership change proposer, P_n . $resp_j$ is the form of the response that is expected from $P_j \in RG_n$.

-
- 1 $P_{n+1} \rightarrow P_n$: $MU_{n+1}, sig_{n+1}(MU_{n+1})$
 - 2 $P_n \rightarrow RG_n$: $newCS_n, MU_{n+1}, sig_n(newCS_n), sig_{n+1}(MU_{n+1})$
 - 3 $\forall P_j \in RG_n$
 $P_j \rightarrow P_n$: $resp_j, sig_j(resp_j), sig_j(newCS_n)$
 - 4 $P_n \rightarrow RG_n$: $rn_n, \sum resp_j, \sum sig_j(resp_j), \sum sig_j(newCS_n)$
 - 5 $P_n \rightarrow P_{n+1}$: $newCS_n, rn_n, G_n, S_y, \sum resp_j, \sum sig_j(resp_j), \sum sig_i(newCS_n)$
-

Figure 3.15: Connection protocol

To start the protocol, P_{n+1} sends a signed membership update request to the proposer, P_n . In steps 2 to 4, P_n coordinates agreement to the change within the existing sharing group: G_n , where $RG_n = G_n - P_n$. At step 5, assuming the sharing group agrees P_{n+1} 's membership, P_n provides P_{n+1} with the evidence of the agreement. As shown, P_n also provides G_n and S_y to enable P_{n+1} to join the group. It is assumed that a participant's identifier, P_i , provides access to the information necessary to communicate with P_i and to verify P_i 's signature. The authenticated decision to agree update MU_{n+1} and, therefore, P_{n+1} 's membership is given by:

$$rn_n, newCS_n, \sum sig_i(newCS_n), \sum resp_j, \sum sig_j(resp_j), sig_{n+1}(MU_{n+1})$$

$$i \in 1 : n; \quad j \in 1 : n - 1$$

Together with MU_{n+1} , this information allows all parties, including P_{n+1} , to install a consistent view of the new membership, G_{n+1} . So, on admission to the sharing group, P_{n+1} shares the group's knowledge of the membership of the new group G_{n+1} and of the agreed information state, S_y . P_{n+1} then also has the option to perform application-level validation of both G_{n+1} and

S_y . Should that validation fail, P_{n+1} may immediately initiate a voluntary disconnection.

A connection request from P_{n+1} may be rejected immediately by P_n or may be vetoed by one or more members of RG_n . In the case of immediate rejection, P_n simply responds with the following signed reject message.⁶

$$P_n \rightarrow P_{n+1} : REJECT, P_n, sig_n(REJECT, P_n, MU_{n+1})$$

In the case of veto by a member of RG_n , the protocol follows steps 1 to 4 shown in Figure 3.15. Then, in step 5, P_n sends P_{n+1} the signed reject message. Thus, P_{n+1} learns no more information than in the case of immediate rejection by P_n .

Disconnection protocols

The disconnection protocols ensure that the remaining members of the group have evidence of the decision to disconnect a subject and, in the case of voluntary disconnection, that the subject initiated the disconnection. Since any member of the group wishing to disconnect may in practice simply cease cooperation, voluntary disconnection cannot be vetoed. Thus, there is no group decision on a voluntary disconnection and the protocol is executed solely to maintain the consistent view of group membership and information state. Note that the response token in Table 3.11, $resp_j$, does not include any decision. The corresponding tokens for connection and eviction include the decision $D_{n,j}$.

The eviction protocol removes one or more subject(s) without reference to those subject(s). Thus the eviction protocol may result in the formation of distinct sharing groups. Any such groups can only claim that its members have agreed to the formation of their group. No claim can be made with respect to the agreement of the evictee(s) to the new group membership nor can any assumption be made about the evictee(s) agreement or otherwise to subsequent state changes in the group from which they have been evicted. The distinct groups act upon different views of the information. In effect, the shared information has been cloned and the state of the information may evolve along separate lines without disadvantage to the members

⁶The request may be rejected because P_n is not the current membership change proposer. In this case, the reject message will provide the information necessary to make a request to the current proposer.

of the distinct groups.

In the following descriptions, P_1 is the subject of a disconnection request, P_n is the current membership proposer, and P_k initiates the disconnection of P_1 . In the case of voluntary disconnection, $P_k = P_1$. In the case of eviction, $P_k \neq P_1$.⁷ The recipient group for a disconnection proposal from P_n is $RG'_n = RG_n - P_1$. The new group after successful completion of the disconnection protocol is $G'_n = G_n - P_1$.

Token	Definition
MU_1	$\{REMOVE, P_1\}$
$newCS_n$	$nrId, prevAgreedNRId, P_n, h(G'_n), h(S_y)$
$resp_j$	$newCS_n.nrId, currCS_j.nrId, P_j$

Table 3.11: Voluntary disconnection protocol tokens

Table 3.11 defines tokens used in the description of the voluntary disconnection protocol shown in Figure 3.16. This protocol is very similar to the connection protocol. The differences

-
- 1 $P_1 \rightarrow P_n$: $MU_1, sig_1(MU_1)$
 - 2 $P_n \rightarrow RG'_n$: $newCS_n, MU_1, sig_n(newCS_n), sig_1(MU_1)$
 - 3 $\forall P_j \in RG'_n$
 $P_j \rightarrow P_n$: $resp_j, sig_j(resp_j), sig_j(newCS_n)$
 - 4 $P_n \rightarrow RG'_n$: $rn_n, \sum resp_j, \sum sig_j(resp_j), \sum sig_j(newCS_n)$
 - 5 $P_n \rightarrow P_1$: $newCS_n, rn_n, \sum resp_j, \sum sig_j(resp_j), \sum sig_j(newCS_n), sig_n(newCS_n)$
-

Figure 3.16: Voluntary disconnection protocol

are in the form of the membership update (MU_1), the new group membership (G'_n), and the fact that there is no decision on voluntary disconnection in the response token ($resp_j$). The authenticated voluntary disconnection of P_1 is given by:

$$rn_n, newCS_n, \sum sig_i(newCS_n), \sum resp_j, \sum sig_j(resp_j), sig_1(MU_1)$$

$$i \in 2 : n; \quad j \in 2 : n - 1$$

This provides evidence that P_1 initiated voluntary disconnection and that all other members of G_n have seen the request. The other members of G_n can therefore install the new agreed view of group membership: G'_n .

⁷In the general case, for voluntary disconnection: $subjectList = \{P_k\}$, and for eviction: $P_k \notin subjectList$.

The protocol to evict a member, P_1 , differs from the voluntary disconnection protocol in the following ways.

1. Some member $P_k \neq P_1$ initiates the request to proposer P_n with the membership update $MU_k = \{REMOVE, P_1\}$.
2. $resp_j$ now includes $D_{n,j}$ — the decision of each member of RG'_n as to whether P_1 should be evicted.
3. The subject is not involved in protocol execution. Thus, it is possible to form a new sharing group without the agreement of the evictee. Since each respondent must make a decision on the validity of the eviction, this new group can only be formed by unanimous agreement of RG'_n and P_n .
4. If the current proposer is also the initiator of an eviction ($P_k = P_n$), the first step of the protocol is omitted.
5. Step 5 is omitted altogether since the initiator of the eviction, P_k , is a member of RG'_n .

The authenticated decision to evict member P_1 is given by:

$$rn_n, newCS_n, \sum sig_i(newCS_n), \sum resp_j, \sum sig_j(resp_j), sig_k(MU_k)$$

$$i \in 2 : n; \quad j \in 2 : n - 1; \quad k \neq 1$$

The eviction protocol can be generalised to remove one or more members of the group. In which case the membership update, MU_k , requested by P_k indicates a list of subjects. Following unanimous agreement to the eviction the new group is: $G'_n = G_n - subjectList$.

Membership change protocol analysis

The two phase commit at the core of the membership change protocols is essentially the same as the state change protocol. Therefore the analysis of the state change protocol in Section 3.3.2.3 applies to the core of each membership change protocol. The significant difference between state changes and membership changes is the use of a proposer to coordinate membership changes. The proposer has two important responsibilities:

1. to act as the identified contact that members joining and leaving the sharing group can submit membership change requests to and, thereby, initiate an appropriate membership change protocol, and
2. to prevent (or veto) other changes to membership or to information state during active execution of a member change protocol.

The latter responsibility means that the proposer can provide a member joining or voluntarily leaving the sharing group with a definitive view of the membership of the sharing group and of information state at the time of the membership state. Evidence is generated that this definitive view is shared by the new sharing group. Concurrent requests for state or membership changes cannot disrupt the establishment of the shared view. The proposer's additional responsibilities present the following additional opportunities for misbehaviour.

1. In the connection protocol, the proposer may unilaterally reject a request to join the sharing group. Proposer rejection on first receipt of a request is acceptable because each member of the sharing group, including the proposer, has the right to autonomous decision on the validity of a membership change request. The proposer may reject a join request in spite of the agreement to the request by the existing sharing group. In this case, the existing members will have evidence that the new member has been admitted to the sharing group. However, the new member will believe their request to join was rejected. Therefore, in the worst case there will be no further progress among the sharing group as the new member will not cooperate in coordination. The proposer's misbehaviour can be detected from the non-repudiation logs of the members of the sharing group and of the proposed new member.
2. In the voluntary disconnection protocol, the proposer may not initiate the disconnection protocol. However, the disconnecting member will cease cooperation in any case. So, this will simply lead to a lack of progress for the remaining members of the sharing group. The proposer may decline to provide the member leaving with the definitive state of shared information at the membership change. However, if necessary, the leaving member can legitimately request this information from any other member of the sharing

group.

3. In the eviction protocol, the proposer may unilaterally reject an eviction request. As for the connection protocol, this is acceptable because the proposer has the right to autonomous decision on the validity of the request.

It should be noted that the subject of a membership change, whether a new member of the sharing group or a member voluntarily disconnecting, is able to detect an attempt by the proposer to misbehave by providing inconsistent information on the current state of the shared information. The internal consistency of the joining or leaving information is guaranteed by the cryptographic binding between the evidence provided and by the fact that the information state at membership change must be the same as the information state identified through the previously agreed control state. Therefore, the subject can verify the integrity and consistency of the information and that it is genuinely the shared view of the membership of the sharing group.

In summary, the proposer does acquire some power from their additional responsibilities. However, the worst case is that the proposer can use this additional power to prevent forward progress for the sharing group, including themselves. This misbehaviour can be detected, is equivalent to blocking by the proposer and may be resolved at application-level. There is no way for the proposer to misrepresent the validity of decisions with respect to membership changes. The rotation of proposer responsibility reduces the burden on any one party and also reduces the opportunities for any individual member to exploit the additional power that a proposer acquires.

3.3.3 Concurrency control

Shared information, of its nature, is subject to concurrent access. Different parties may independently initiate a request to update information state or to change the membership of the sharing group. There is, therefore, a requirement for concurrency control to determine whether any of two or more competing requests should succeed. There are four different types of protocol-level concurrency control: (i) read locking of shared information, (ii) optimistic concurrency control with all fail semantics, (iii) optimistic concurrency control with at most one

succeeds semantics, and (iv) explicit write locking of shared information.

Read locking

Any member of the sharing group can veto proposed changes to shared information and, thereby, apply a local read lock on the information. The local acquisition of a read lock can be communicated to the proposer of the change by a recipient indicating in their decision, $D_{k,j}$, that the information is read-only.

Optimistic concurrency control with all fail semantics

The second invariant from Section 2.1.2 states that each recipient's view of the current agreed control state must be the same as the proposer's previously agreed control state. So:

$$\forall P_j \in RG_k : newCS_k.prevAgreedNRId = currCS_j.nrId$$

This invariant guarantees concurrency control with all fail semantics. Minimally, competing proposals are detected and rejected by their respective proposers. For example, if member P_k proposes transition to control state $newCS_k$ concurrently with a proposal from P_l for transition to control state $newCS_l$ then:

$$\begin{aligned} currCS_k = newCS_k \text{ and } currCS_l = newCS_l, \text{ therefore} \\ newCS_k.prevAgreedNRId \neq currCS_l.nrId \\ \text{and } newCS_l.prevAgreedNRId \neq currCS_k.nrId. \end{aligned}$$

P_k and P_l have inconsistent views of current control state. Therefore, they will veto each other's proposals and all other members will obtain evidence of the concurrent proposals.

Optimistic concurrency control with at most one succeeds semantics

It is possible to provide *at most one succeeds* semantics for competing proposals by adopting a convention for the precedence of one proposal over another. For example, the ordering of members to identify the membership proposer could be used to choose between concurrent proposals. In the case of competing proposals from P_k and P_l , P_k would not automatically veto

P_i 's proposal. P_j 's proposal would succeed providing it passed application-level validation by all members of the sharing group, including P_k .

Explicit write locking

Explicit write locking of shared information is supported by extending information control state to include a *lockId* that identifies the current holder of a write lock. Then a proposer may use the state coordination protocol to agree transition to a new, locked control state. The state coordination protocol can support optimistic locking where a *lockId* is associated with the new control state along with some new information state payload. Alternatively, the proposer can perform pessimistic locking by requesting transition to a new locked control state before proposing a change to the associated information state. Having acquired an explicit write lock on the information, the proposer may then make a sequence of state changes by appropriate executions of the state coordination protocol. Figure 3.17 shows a sequence of such control

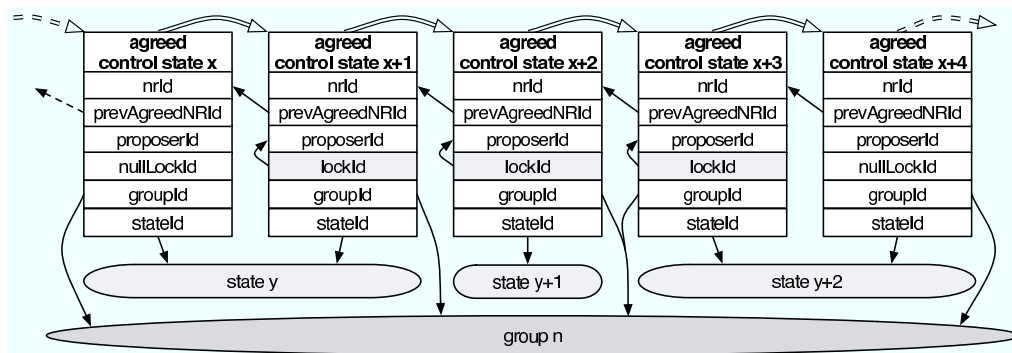


Figure 3.17: Control state transitions with locking

state transitions. Transition from control state x to control state $x + 1$ represents acquisition of the write lock by the proposer. While they hold the lock, the proposer obtains agreement to transition from control state $x + 1$ through to control state $x + 3$. So, the proposer holds a write lock for changes from state y , through state $y + 1$ to state $y + 2$. Finally, the proposer releases the lock in the transition from control state $x + 3$ to control state $x + 4$.

The following conditions hold with respect to explicit write-locking.

1. Any party may request that a lock holder release their lock by initiating transition to a new unlocked control state. The lock holder may veto such a request.

2. If the lock holder voluntarily disconnects or is evicted from the sharing group, their lock is relinquished as part of the protocol for transition to the new membership.
3. While a lock is held, the lock holder is the designated membership proposer. So, the identification of membership proposer discussed in Section 3.3.2.4 is overridden by the existence of a lock holder.

In Section 5.3.3 I show how transactional access to shared information relies similar extensions to control state. I now summarise the supporting infrastructure required for the non-repudiation services defined in Sections 3.2 and 3.3. I then conclude the chapter with an evaluation of the non-repudiation services.

3.4 Supporting infrastructure

The services described in Sections 3.2 and 3.3 rely on a supporting infrastructure. Here, I provide a brief overview of its main components. Section 4.3 presents service APIs and other implementation detail.

Credential (certificate) management: to support signature verification, and for storage and retrieval of digital certificates and associated public keys. Credential management also supports certificate revocation and the verification of certificate chains.

Cryptographic services: to sign information, to verify signatures, to encrypt information, to decrypt cyphers, to generate message digests, and to generate secure pseudo-random number sequences.

Information time-stamping: to provide trusted time-stamps on signed information as described in Section 2.1.3.

Message logging: to log the messages that protocol participants exchange. There are two types of message log: (i) a long-term non-repudiation log, and (ii) an active message log that provides temporary storage for messages that are still being processed in some way. Each participant's non-repudiation log provides their definitive view of an interaction

and can be used to recover protocol state. However, certain (invalid) incoming messages should not be stored in the non-repudiation log. For example, duplicate messages or messages with invalid cryptographic information should be discarded. Therefore, incoming messages that have been received for processing are stored in the active message log until a decision has been made to either discard them or to move them to the non-repudiation log. From each participant's point of view, a protocol run is active if they have sent a message for which they are entitled to expect one or more response(s). Thus, for each participant, the last message they sent determines whether a given protocol run is active. These messages are stored in, and can be recovered from, the non-repudiation log. However, protocol-specific knowledge is required to determine whether a given message relates to an active or terminated protocol run. Rather than perform potentially complex searches of the non-repudiation log, it is convenient to use the active message log to store a copy of the last outgoing message for any active protocol run. An outgoing message is removed from the active message log either: (i) when it is replaced by a subsequent outgoing message for the same protocol run, or (ii) when the message that terminates the protocol run is added to the non-repudiation log. For example, in the request phase of the non-repudiable invocation shown in Figure 3.5 on page 88, B keeps a copy of the message sent in step 1.3 in the active message log until it can be replaced by the message sent in step 1.6. B removes the latter message from the active message log when they receive the TTP's message in step 1.7 and, from B's point of view, the protocol has terminated. Though not essential, the temporary logging of these outgoing messages is useful for recovery.

An organisation may already have access to some of the identified infrastructure. For example, standard Java libraries can be used for the cryptographic services. In other cases, new implementations of support services are provided. Section 4.3 provides service implementation details.

3.5 Evaluation of non-repudiation services

In this chapter we have defined two services to address the requirements for accountability in B2B interactions identified in Section 1.2. The two services for non-repudiable service invocation and for non-repudiable information sharing address the two domains for action: on private and shared resources, respectively.

Non-repudiable service invocation is based on fundamental work on non-repudiation, using protocols for the direct exchange of evidence and for TTP-supported fair exchange. Section 3.2.1.2 highlighted extensions to Coffey-Saidha's in-line TTP protocol to integrate application-level validation without loss of fairness and to support timely termination. The straightforward extension for validation is possible because the TTP is involved in each message exchange and controls the release of information to other parties. For off-line TTP protocols, it was shown that application-level validation can be integrated by the correlation of two protocol runs. The first protocol run exchanges a message for its receipt. The second run exchanges validation evidence. For each approach to the exchange of non-repudiation evidence, correlated protocol runs are used to link the request and response phase of service invocation.

The choice of exchange mechanism will depend on application-specific requirements. For example, direct exchange may be appropriate when interacting with a trusted entity. The choice between in-line TTP and off-line TTP fair exchange may depend on the availability and capabilities of TTP services. An obvious consideration is the communication overheads of a protocol in terms of the number of messages needed to achieve the exchange of evidence. However, care must be taken when calculating these overheads. If signing keys are revocable and trusted time-stamps are required for signature verification and the long-term validity of evidence, then the costs of obtaining time-stamps from a TSA must be included in the calculation. Table 3.12 compares the costs for the request phase of a service invocation for five approaches to exchange: (i) direct exchange, (ii) direct exchange with a TTP that also acts as a TSA, (iii) in-line TTP fair exchange, (iv) in-line TTP fair exchange where the TTP is also the TSA, and (v) optimistic fair exchange. It should be noted that it is not possible to optimise optimistic fair exchange when a TTP also acts as TSA since there is no TTP involvement in the main exchange. Column 2 shows the number of protocol messages required for the exchange of receipting and

Approach (request phase only)	Protocol messages	Time-stamp messages	Total messages
(i) Direct exchange (Figure 3.4)	2	4	6
(ii) Direct exchange with TTP/TSA	2	0	2
(iii) In-line TTP fair exchange (Coffey-Saidha — Figure 3.5)	8	6	14
(iv) In-line TTP/TSA fair exchange (Coffey-Saidha — Figure 3.8)	8	0	8
(v) Optimistic, off-line TTP fair exchange (Wang — Figure 3.9)	6	8	14

Table 3.12: Communication costs for request phase non-repudiation with validation

validation evidence. Direct exchange is cheapest, then optimistic fair exchange and then in-line TTP exchange. Column 3 shows the number of messages required for trusted time-stamping. In each protocol, a time-stamp must be obtained the first time that signed evidence is generated. Two messages are required for each time-stamp, one to send the request to the TSA and the other for the TSA's response. As shown, the time-stamping overhead is highest for optimistic fair exchange, which requires time-stamps for 4 of the 6 protocol messages. Inline TTP exchange requires a time-stamp for 3 of 8 protocol messages. Direct exchange requires a time-stamp for each of its 2 protocol messages. As shown for exchanges (ii) and (iv), when a TTP can also act as a TSA there is no communications overhead for time-stamping. Not surprisingly then, as shown in column 4, direct exchange with a TSA has the lowest total message cost followed by direct exchange. Perhaps less obvious is that fair exchange with an in-line TTP/TSA is cheaper, in terms of total messages required, than optimistic fair exchange and that optimistic fair exchange has the same total message cost as in-line TTP fair exchange. So, the relative communication costs of the different approaches change when time-stamps from a TSA are required. Table 3.13 compares communication costs when application-level validation is not required. In this case, there is no exchange of validation evidence. Once again, the relative costs of the different approaches change when time-stamping is required. Interestingly, in-line TTP/TSA fair exchange is now cheaper than the simple direct exchange. Clearly, if communication costs are a factor in the choice of mechanism then important considerations include whether trusted time-stamping is required and whether guarantor TTP and TSA roles

Approach (request phase only)	Protocol messages	Time-stamp messages	Total messages
(i) Direct exchange (Figure 3.4)	2	4	6
(ii) Direct exchange with TTP/TSA	2	0	2
(iii) In-line TTP fair exchange (Coffey-Saidha — Figure 3.5)	5	4	9
(iv) In-line TTP/TSA fair exchange (Coffey-Saidha — Figure 3.8)	5	0	5
(v) Optimistic, off-line TTP fair exchange (Wang — Figure 3.9)	3	4	7

Table 3.13: Communication costs for request phase non-repudiation without validation

can be combined. Other factors, such as computational costs and relationships between business partners, will also inform the choice of mechanism. The preceding analysis underlines the desirability of being able to choose an appropriate mechanism for the given B2B context. The flexibility necessary to meet this challenge is based on the model of interceptor-mediated interaction described in Section 3.1.

The definition of non-repudiable information sharing addresses all the requirements for the service identified in Section 1.2 except for transactional access. The coordination protocols presented in this chapter guarantee the validity of decisions reached amongst the sharing group. They are also efficient in terms of the number of messages required ($O(n)$ for n parties) and are straightforward to implement. Protocol messages, and valid states, are persisted and, therefore, recovery is possible in many circumstances. These characteristics are achieved in the context of stated assumptions with respect to failures and, in particular, by not guaranteeing protocol termination when parties misbehave. The inability to terminate is detectable and may be resolved outside of a protocol run.

Application-level validation depends on the semantics of a proposed change to shared information as interpreted by each individual party. Thus all parties must be involved to validate a change. Protocol-level support for termination in the presence of misbehaviour therefore reduces to allowing timely abort of a proposed update without disadvantaging well-behaved parties. The arguments for the impossibility of deterministic fair exchange without a TTP will also apply in this case and, therefore, a safe abort protocol will require the involvement of a

TTP to guarantee that all well-behaved parties terminate with the same view of agreed state. In effect, a TTP would provide certified abort of a protocol run unless the required set of responses are available, in which case the TTP would provide a certified decision that reflects those responses. Another approach is to take advantage of the propagation of the agreed application state during membership changes. A membership change can be seen as the point at which any temporary divergence of application state can be resolved. If a TTP were used as coordinator for membership changes, then the TTP could guarantee termination to well-behaved parties. The development of sub-protocols to guarantee termination is left as an avenue for future research. Approaches to explore include the use of both on-line and off-line TTPs, and also the construction a *virtual* TTP from an honest majority of members of the sharing group. The protocols presented in this chapter are an easily understood base for the investigation of different approaches. Again, the resulting choice underlines the need for flexibility in order to the deploy the mechanism that is appropriate to the given context.

Taken together the definitions of non-repudiable invocation and non-repudiable information sharing are the foundation for systematic support to meet requirements for accountability and audit in the two domains of action identified in Chapter 1. Remaining challenges are: (i) the realisation of the non-repudiation services as middleware with the flexibility to adapt to different application requirements, and (ii) the support for transactional access to shared information. Chapter 4 presents the design and implementation of a generic middleware framework for protocol execution that supports the adaptable delivery of non-repudiation services and the flexible provision of the supporting infrastructure identified in Section 3.4. The framework supports interceptor-mediated non-repudiable interaction between application components. Chapter 5 describes the implementation of the non-repudiation services that use the protocol execution framework. The implementation of non-repudiable information sharing extends standards-compliant transactional access to resources to include shared information. Section 5.3.3 describes how information control state can be extended to support transactions, in a similar way to its extension for explicit write locking in Section 3.3.3. From the point of view of transaction middleware, the extensions allow shared information that is realised as application components to behave in the same way as any other transactional resource.

Chapter 4

Generic protocol execution framework

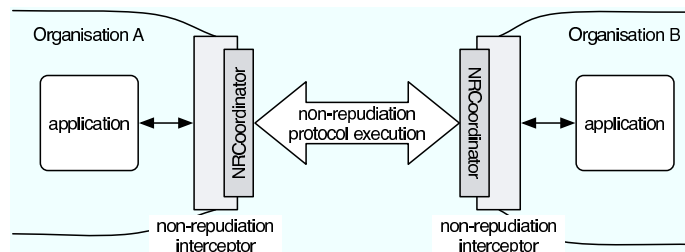
This chapter presents the design and implementation of the generic protocol execution framework that is basis for the non-repudiation service implementations described in Chapter 5. The generic framework has two layers: (i) a protocol-specific handler layer for message generation and processing, and (ii) a protocol-agnostic remote access and coordination layer for the exchange of messages between protocol participants. This separation of protocol-specifics from B2B communication is fundamental to the general applicability of the framework. The framework supports the execution of any protocol for which there is a handler that complies with the framework's API. The framework provides support for the run-time invocation of application-specific validation and for the application-level notification of protocol events. In addition, the framework provides support services that are not specific to a given non-repudiation service or protocol.

The framework is implemented in Java. Chapter 5 demonstrates its use to implement the non-repudiation services defined in Chapter 3 for interaction between and with components of a J2EE application server. As discussed in Chapter 6, preliminary work on a re-implementation for Web services demonstrates the flexibility of the approach [RCS05, CRS06]. The Web services version provides non-repudiation of SOAP message delivery as a mediation service or using the AXIS SOAP handler chain [Apa05]. Section 4.1 of this chapter provides an overview of the protocol execution framework. Then in Section 4.2 I provide details of B2B communication and protocol handling, the representation of protocol messages, the agreed representation of evidence, and of application-level validation and event notification. Section 4.3 provides service APIs and implementation details of the supporting infrastructure introduced in Sec-

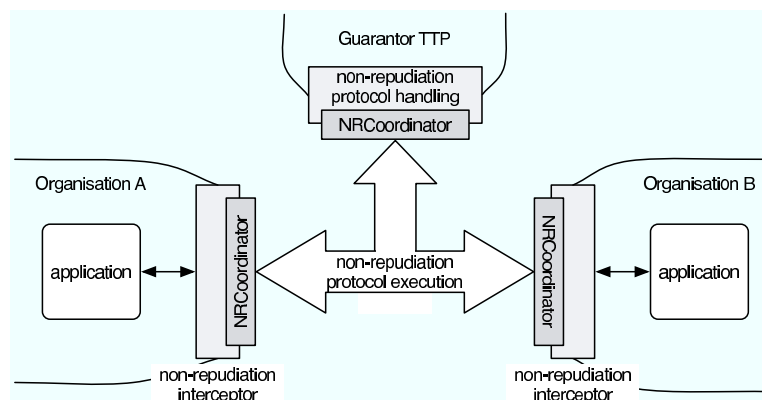
tion 3.4. As stated in Section 1.2, fault tolerance of exchange is beyond the scope of this dissertation. However, the necessity to log messages for non-repudiation and the design of the protocol execution framework means that it is possible to support local recovery, and re-engagement in exchange, at the middleware level. Section 4.4 discusses this middleware-level support for recovery in the event of temporary local failure. Such recovery is the basis for future, more comprehensive work on fault tolerance. Section 4.5 concludes the chapter with a summary of contributions, highlighting how the framework addresses the requirement for flexibility identified in Section 1.2.

4.1 Overview of framework

Each business partner in a non-repudiable B2B interaction is expected to expose a service for the exchange of protocol messages. This *NRCoordinator* service is the external entry point for remote access and coordination. As shown Figure 4.1a the coordinator services are deployed



(a) protocol execution between peer coordinator services



(b) guarantor TTP coordinator service

Figure 4.1: Interceptor-based protocol execution with NRCoordinator services

at non-repudiation interceptors that mediate the interaction between business partners. So, for B2B interactions that require regulation, the interception of communication between application components at organisations A and B results in the execution of non-repudiation protocols through the exchange of messages between the exposed coordinator services. This realisation of the interceptor-mediated interaction model introduced in Section 3.1 preserves the semantics of the interaction between application components at A and B. Figure 4.1b shows the extension of the model to include a guarantor TTP for protocol execution. The TTP also provides a coordinator service for interaction with A and B. The same protocol execution services are deployed at the TTP as at A and B. The services at A and B are invoked as a result of the interception of operations on application components. At the TTP, the services are standalone and are invoked as a result of calls made at the TTP's NRCoordinator interface by A's and B's interceptors.

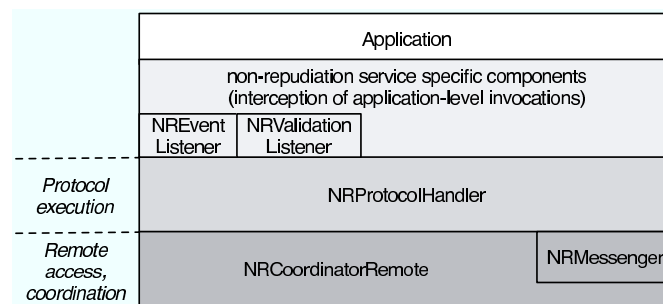


Figure 4.2: Non-repudiation protocol execution components

Figure 4.2 shows the main components of the protocol execution framework (deployed at non-repudiation interceptors). Operations at the application level are mediated by non-repudiation service-specific components. The service-specific layer is responsible for the local selection and initiation of protocols appropriate to the application context and to the given non-repudiation service, whether service invocation or information sharing. Thus, the service-specific layer mediates between the application and the underlying protocol execution framework. The protocol-specific layer of the framework manages the execution of specific protocols, including message generation and processing. The service-specific layer initiates a protocol through the NRProtocolHandler interface. The NRProtocolHandler also provides an interface for the remote access layer to forward incoming messages for processing. It is at

the protocol-specific layer that protocols appropriate to different application requirements are deployed. For example, different `NRProtocolHandler` implementations are provided for the different service invocation protocols described in Section 3.2. The framework is designed to execute any protocol for which there is a handler that complies with the `NRProtocolHandler` API.

The remote access and coordination layer in Figure 4.2 provides protocol-independent communication between protocol participants. The `NRCoordinatorRemote` interface is provided for delivery of incoming messages and the `NRMessenger` interface is used by the protocol-specific layer to send outgoing messages. The `NRValidationListener` provides the means to plug-in application-specific validation and to trigger that validation during protocol execution. Similarly, `NREventListeners` can be used for higher-level reaction to events that are generated during protocol execution. I now provide further details of the different components of the framework and their relationship to each other.

4.2 Framework APIs and implementation

In Section 4.2.1 I describe the coordinator service and protocol handlers. Section 4.2.2 defines the self-describing protocol messages that protocol participants exchange. The agreed representation of application data and evidence is discussed in Section 4.2.3. Section 4.2.4 describes validation and event listeners.

4.2.1 Coordinator service and protocol handlers

Figure 4.3 is the API of the components that constitute the lower layers of Figure 4.2. Protocol-independent remote access and coordination are provided by `NRCoordinatorRemote` and `NRMessenger`. `NRCoordinatorRemote` is the entry point for the delivery of protocol messages from remote parties to be processed by local protocol handlers. Remote invocation of `deliverMessage` results in delivery of the given `NRProtocolMessage`, defined in Section 4.2.2, from the invoker to the invoked coordinator service. `deliverRequest` is a convenience method that allows a remote party to deliver a message and then to wait synchronously for a response. The concrete implementation of the `NRMessenger` interface is responsible for binding to the `NRCo-`

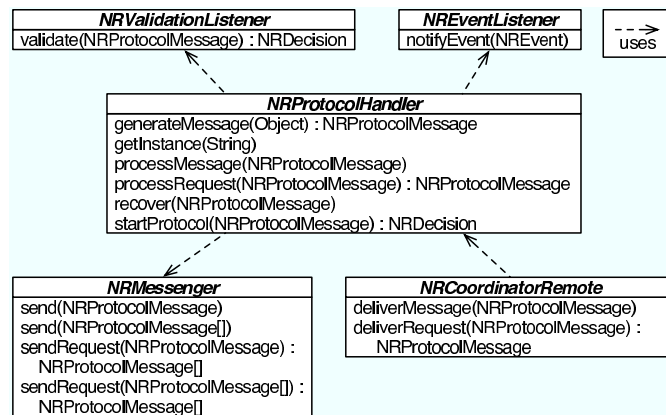


Figure 4.3: Non-repudiation protocol execution API

ordinatorRemote interface of remote coordinator services for the delivery of locally generated messages. As shown, the NRMessenger has two send methods: one to send a single message to one or more recipients, and a convenience method to send a collection of messages. The corresponding sendRequest methods send one or more messages and return a possibly empty array of replies from the intended recipients. Each participant's local coordinator service is also responsible for any necessary bootstrapping of the supporting infrastructure (see Section 4.3) and plays a role in initiating recovery in the event of temporary local failure.

At the heart of the framework are protocol-specific handlers, instantiated at the protocol execution layer, that generate and process protocol messages. Protocol handlers perform two main tasks:

1. to initiate protocol execution given a message generated from application data provided by the local non-repudiation middleware service, and
2. to manage participation in ensuing protocol execution by providing outgoing messages to the NRMessenger to send to remote parties and by processing incoming messages that have been delivered to the local coordinator service.

The protocol handlers use the support services described in Section 4.3 for tasks such as signature generation and verification, message logging and time-stamping.

Protocol handlers are instantiated by a NRProtocolHandlerFactory. Every protocol, and its associated sub-protocols, has a unique name that enables the handler factory to identify

the appropriate protocol handler. For example, Table 4.1 identifies six protocols for non-

Protocol	Name
Request phase main protocol	/nrinv/request/ext-coffey-saidha/main
Request phase abort sub-protocol	/nrinv/request/ext-coffey-saidha/abort
Request phase resolve sub-protocol	/nrinv/request/ext-coffey-saidha/resolve
Response phase main protocol	/nrinv/response/ext-coffey-saidha/main
Response phase abort sub-protocol	/nrinv/response/ext-coffey-saidha/abort
Response phase resolve sub-protocol	/nrinv/response/ext-coffey-saidha/resolve

Table 4.1: Extended Coffey-Saidha protocol names

repudiable invocation based on the extended Coffey-Saidha protocol suite described in Section 3.2.1.2. There are, therefore, six protocol handlers for this protocol suite. In addition, for all main protocols, there is a status update protocol handler, named <protocol-specific name>/main/getstatus, to support recovery of protocol status between participants. This allows parties to query the protocol status as seen by other participants. Information exchanged during a status update sub-protocol will determine whether the requesting party is entitled to the current protocol status. For local initiation of protocols, a protocol name is obtained from the middleware configuration. For incoming messages, the coordinator service obtains the protocol name from the protocol message (see Section 4.2.2) and uses the handler factory to instantiate an appropriate handler for the message.

All protocol handlers provide the NRProtocolHandler interface shown in Figure 4.3. The generateMessage method is used to generate a protocol-specific initial message from application-specific data. The form of the application data is dependent on the type of non-repudiation service involved. For service invocation, application data is either the request or the response intercepted by the middleware along with additional information such as service identity. For information sharing, it is the control state and related state change or membership change. The use of the generateMessage method allows the correlation of application-level information with protocol-specific information, such as as protocol run identifiers. The NRProtocolHandler processMessage methods are used by the coordinator service to pass protocol messages to an instantiated protocol handler for protocol-specific processing. The service-specific layer may also pass messages generated by generateMessage to one of these methods. The recover method can be used by the coordinator service to initiate recovery after temporary local failure.

During protocol execution, protocol handlers log messages to the non-repudiation log and to the active message log described in Section 3.4. To initiate recovery, for each message in the active message log, the coordinator instantiates an appropriate protocol handler and uses the recover method to pass the message to the handler for processing. The recovery process is protocol- and non-repudiation service-specific. Section 4.4 provides further details.

4.2.2 Representation of protocol messages

A protocol message is represented by the NRProtocolMessage class shown in Figure 4.4. Protocol messages are self-describing and consist of two parts: (i) an NRToken that includes

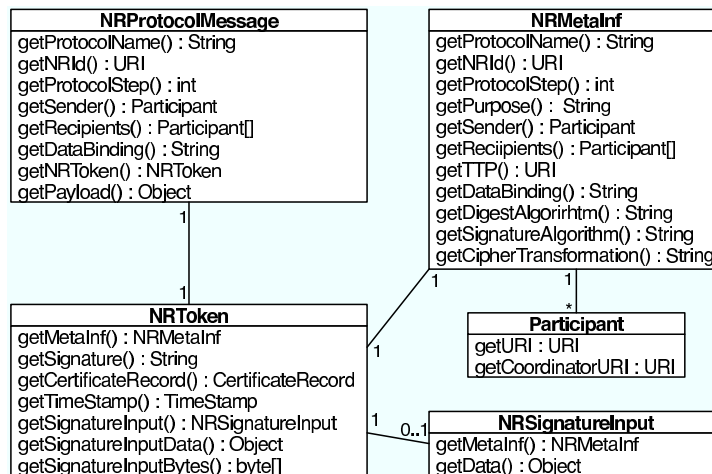


Figure 4.4: NRProtocolMessage API

protocol meta-information, input data for any signature, the signature itself and the information necessary to verify it; and (ii) unsigned payload that can be used for application-level information that is related to the signature input. As shown in Figure 4.4, a NRProtocolMessage provides accessor methods for its NRToken and the payload. It also provides some convenient wrapper methods to access NRToken meta-information that is used frequently during message processing.

All NRTokens have a NRMetaInf instance. The NRMetaInf provides the context information to associate a given message with its protocol and to enable the processing of any associated application-specific and/or protocol-specific data that is included in the message. The protocol name, non-repudiation identifier and protocol step unambiguously identify the

protocol run, and the step within the run, to which the message relates. The other `NRMetaInf` elements include: any purpose associated with the message, for example, “NRR”; the protocol participants — message sender, intended recipient(s), and any guarantor TTP; the data-binding in use; and cryptographic algorithms used for any message digests, signatures and/or ciphers. The sender and recipient(s) are represented by `Participant` objects that provide access to the participant’s URI and the URI of their coordinator service. The coordinator service URI allows `NRMessenger` implementations at each participant to bind to the service `NRCoordinatorRemote` interface for protocol execution. For example, a participating entity may be represented by an email address and their coordinator service may be referenced through an RMI URI. The data-binding is used to convert information to/from an agreed representation and the corresponding native Java representation (see Section 4.2.3).

An `NRToken` that includes signed content has an `NRSignatureInput` instance. If present, the signature input actually encloses the concrete representation of `NRMetaInf`. This ensures that the meta-information is signed along with the associated application-specific and/or protocol-specific data. The associated data, and the message payload, are represented by the Java Object type because their interpretation is dependent on the given application and the protocol to which a message relates. For example, for the first message of the state coordination protocol in Figure 3.13 on page 112, the `getData` method of `NRSignatureInput` will return a representation of the new information control state ($newCS_k$) — the data included for signing. The `NRProtocolMessage` `getPayload` method returns the proposed new application state (S_{y+1}). As shown in Figure 4.4, in addition to the `NRSignatureInput`, an `NRToken` includes the information necessary to verify the signature: the Base64 signature string itself, a `CertificateRecord` that conveys the signers certificate or a URI referring to it, and a trusted time-stamp over the signed information (see Section 4.3.3). Finally, a `NRToken` provides methods to access three representations of signature input: (i) a native Java `NRSignatureInput` object, (ii) the agreed representation as generated by the identified data-binding, and (iii) a conversion of the agreed representation to a byte array for signature generation.

4.2.3 Agreed representation of evidence

The primary requirement on non-repudiation services is that the evidence that relates to an action or event supports the subsequent undeniability of the action or event. This means that non-repudiation evidence exchanged during a B2B interaction must remain meaningful after the applications and services that constitute the interaction have ceased to execute. Further, the evidence must be rendered meaningful and verifiable by a third party who may have no knowledge of the implementation context in which the evidence was generated. This implies that there must be an agreed representation of the information, including application-specific data, that constitutes the non-repudiation evidence of an interaction. This agreed representation must be meaningful to the parties directly involved in the interaction and also to third parties who may later rely on the evidence or who may be called on to resolve disputes.

A service invocation can be represented, in part, by an array of Java objects that is comprised of the parameters to the service request. To provide NRO of these request parameters it is possible to perform standard Java object serialization of the array and then to sign the resulting byte array with the client's private key. Subsequently, because they share implementation details of the service, both client and server can deserialize the byte array and retrieve the request parameters as an object array. Thus, this representation is meaningful to both parties. However, the representation only remains meaningful as long as the Java implementation classes for the parameters are available because deserialization requires access to these classes. Further, any third party that subsequently needs to verify the request parameters will require access to the implementation classes. Thus, subsequent undeniability requires both the signed non-repudiation evidence and the availability of the implementation classes for deserialization. This is acceptable provided client, server and any third parties that may subsequently rely on the non-repudiation evidence agree to this representation. If the tight binding of evidence to its Java representation is not desirable, then there must be agreement on an implementation-independent representation of the information for the generation of non-repudiation evidence.

In the case of non-repudiable information sharing, shared state may be represented locally as an J2EE entity bean (see Section 5.3). The Java serialization of the bean could then be used as input to signing. As the previous discussion indicates, the implementation of the bean must

then be available to any party who subsequently relies on the evidence generated. In this case, the situation is further complicated because the shared information is represented in different applications at different organisations. These local applications may present different views of the shared information, even using different implementation classes. Thus there may be a natural distinction between the local presentation of shared information and its agreed representation. Given this, and the requirement for subsequent deniability, it may be convenient to use an implementation-independent agreed representation for non-repudiation evidence.

There is, then, a need to convert between a local Java representation of information and an agreed representation that may be implementation independent. To address this requirement, the framework provides data-bindings to perform the conversion. A `DataBindingFactory` is used to instantiate a `DataBinding` object for the agreed representation. As mentioned in Section 4.2.2, the agreed data-binding is identified in non-repudiation protocol messages. Figure 4.5 shows the `DataBinding` interface. The `bytesToData` and `dataToBytes` methods are

```
public interface DataBinding {
    Object bytesToData(byte[] bytes);
    byte[] dataToBytes(Object data);
    Object dataToObject(Object data);
    Object objectToData(Object obj);
}
```

Figure 4.5: `DataBinding` interface

used to convert between the agreed representation and a byte array for signing (since it is the agreed representation that must be signed). The `dataToObject` and `objectToData` methods convert between the agreed representation and the local Java representation. Table 4.2 shows

Data binding name	Agreed data representation	Conversion to byte array
binary	serialized Java object	serialized Java object
string	string serialization using programmer defined <code>toString</code> and <code>valueOf</code> methods	string to byte array conversion
xmlstring	serialization to XML document string	string to byte array conversion

Table 4.2: Pre-defined data bindings

three pre-defined bindings. The binary binding uses Java serialization as the agreed representation. The other two bindings are stringified forms. To illustrate these data-bindings, Figure 4.6

```
Java implementation:  
public class OfferImpl implements Offer {  
    // fields  
    private String description;  
    private String currency;  
    private double price;  
    // accessor methods etc.  
    ...  
    ...  
}  
  
String representation:  
description: widgetX, currency: EUR, price: 10.50  
  
XML representation:  
<Offer>  
    <description>widgetX</description>  
    <currency>EUR</currency>  
    <price>10.50</price>  
</Offer>
```

Figure 4.6: Example data-bindings for an Offer object

shows the fields of a simple business Offer object followed by a comma-separated string representation of an Offer instance and the corresponding fragment of an XML string representation. In practice, the string and XML representations may convey additional meta-information such as the element types. The examples are sufficient to illustrate the transformation from implementation-specific to implementation-neutral representation of the data. The use of a `DataBindingFactory` allows extension of the framework to support additional data-bindings and the customisation of the pre-defined bindings.

4.2.4 Application-level validation and protocol events

A key requirement identified in Section 1.2 is that the actions of interacting parties can be subject to application-level validation and that this validation is itself non-repudiable. For service invocation, the server should be able to subject a request to validation with respect to contract

to protect the server's interest. Similarly, the client should be able to subject any response to validation to protect their interests. For information sharing, a proposed state or membership change should be subject to validation by the recipients of the proposed change. As indicated in Chapter 3, the invocation of validation is, therefore, an integral part of non-repudiation protocol execution. However, the application-specific validation logic is not fixed for a given protocol and cannot be hard-coded in the protocol implementation. To address the problem of integrating validation with protocol execution, the framework supports the configuration of protocol handlers to instantiate and invoke validation listeners that are specific to a given business interaction. The application programmer, and not the protocol implementor, is responsible for implementing and identifying the validation listeners that a protocol handler should use for a given interaction.

```
public interface NRValidationListener {  
    NRDecision validate(NRProtocolMessage msg);  
}
```

Figure 4.7: NRValidationListener interface

All validation listeners implement the interface shown in Figure 4.7. The listener `validate` method takes a `NRProtocolMessage` and returns an `NRDecision`. A `NRProtocolMessage` carries all the context information necessary to support its validation. For example, an information sharing protocol message that requires application-level validation will include the information control state that allows the programmer to access previous control states. It is therefore possible to access previous information state(s) and to validate a proposed state change with respect to those state(s). The data-binding identified by the protocol message allows the programmer to convert application state from the agreed representation to the local implementation-specific representation. An `NRDecision` includes the protocol run identifier of a proposed change, a boolean indication of the validity of the change and an optional diagnostic message. The `NRDecision` is used by the protocol handler to construct the non-repudiation of validation that is returned during protocol execution.

It is possible to identify more than one validation listener for invocation by a protocol

handler. In this case, the listeners will be invoked as a chain, in the order registered, until the last `validate` method returns or a proposal is found to be invalid; whichever is the sooner. If validation succeeds for all listeners in the validation chain, then the message is considered valid. At the appropriate point in protocol execution, the local protocol handler passes a protocol message to the registered listeners for validation. By definition, validation of the protocol message constitutes validation of the related service request, service response or change to shared information.

In addition to validation events, protocol handlers can generate information events as a protocol executes. Example events are the arrival of acknowledgements during service invocation or a change to the membership of the group sharing information. Handling of protocol events is optional. To activate event handling, the application programmer identifies one or more event listeners for invocation as protocol execution progresses. Figure 4.8 shows the `NREventListener`

```
public interface NREventListener {  
    void notify(NREvent event);  
}
```

Figure 4.8: `NREventListener` interface

tenter interface. To notify an event, a protocol handler invokes the `notifyEvent` method to pass an `NREvent` to the event listener. Concrete implementations of `NREvent` convey the event type along with information such as the protocol run identifier and event diagnostics. Non-repudiation service specific events allow the application programmer to react to protocol and middleware-related events in order to manage a B2B interaction.

Chapter 5 describes how validation and event listeners have been used in demonstrator applications and, in particular, to perform the sort of contract monitoring described in [MJSSW04].

4.3 Supporting services

This section provides an overview of the services that have been defined to provide the supporting infrastructure introduced in Section 3.4. The approach taken is to define a set of high-level

interfaces for the non-repudiation middleware to use. Wrappers for underlying implementations that can provide these services have then been written to comply with the defined interfaces. Thus, there is a service abstraction layer between the underlying services available at a given organisation and the non-repudiation middleware that uses those services. For example, certificates may be stored and managed locally using a simple Java keystore. They may be accessible through an LDAP directory service. It is also conceivable that some third party may provide a certificate service. Whatever the implementation, for the most part, the non-repudiation middleware simply needs to be able to store and retrieve a certificate. Therefore, the implementation-specific details of the backing store are hidden by a higher-level interface. In the following, I define the minimal functional service interface for each component of the supporting infrastructure to meet the requirements of the non-repudiation protocol execution framework. Implementation details such as the configuration, administration and maintenance of the services are beyond the scope of this dissertation. From the point of view of the non-repudiation middleware, it suffices that there exists a service implementation, or a wrapper or adapter for the service, to satisfy each of the defined interfaces. In the simplest case, the underlying services may be provided by standard Java libraries.

4.3.1 CertificateService

A CertificateService provides for storage, retrieval and verification of certificates that provide access to a subject's public key that is in turn bound to the subject's private key. Each certificate

```
public interface CertificateService {
    String addCertificate(Certificate cert, String certId);
    String addCertificate(Certificate cert);
    Certificate getCertificate(String certId);
    boolean verifyCertificate(String certId);
    boolean verifyCertificate(String certId, long time);
}
```

Figure 4.9: CertificateService interface

stored with the service has an identifier that can be used to retrieve the certificate using the `getCertificate` method shown in Figure 4.9. There are two `addCertificate` methods to store

a certificate. The first takes a certificate and a suggested identifier for the service to use. The second just takes a certificate as a parameter. Both return the actual identifier for the certificate used by the CertificateService. If the certificate is already known to the service, then the service returns the existing identifier for the certificate. If the certificate is unknown and the first version of the method is invoked and the suggested identifier is also unknown, then the service uses the suggested identifier and returns it. Otherwise, for either version of the method, the service returns a newly generated certificate identifier. There are two forms of certificate verification. The first verifies that the identified certificate is currently valid. The second verifies that the identified certificate was valid at the time represented by the given number of milliseconds since the beginning of the epoch.¹ In both cases, certificate verification may involve verification of any supporting certificate chain, up to some trusted Certificate Authority.

4.3.2 Cryptographic services

Figure 4.10 shows the API for the following cryptographic services: (i) CryptoUtil to generate pseudo-random numbers (byte streams) and to generate and verify message digests, (ii) SigningService to generate and to verify digital signatures, and (iii) CipherService for data encryption and decryption.

For all services, cryptographic elements — message digests, signatures and secure pseudo-random numbers — are output as Base64 strings rather than as the native byte arrays generated by the standard Java cryptographic libraries. The Base64 encoding is used because: (i) it is a standard that is commonly used for cryptographic elements, (ii) it is suitable for any of the agreed representations of information discussed in Section 4.2.3, and (iii) it can be used in the generation of protocol run identifiers (see Section 2.1.7). Given the small size of cryptographic elements,² the overhead of conversion between native byte arrays and Base64 strings is small, as is the increase in data size. As shown in Figure 4.10, a Base64 utility class is used to perform the conversion.

Both the SigningService and CipherService require access to an entity's private key, as referenced by the Principal parameter to `sign` and `decrypt` methods. How the Principal is

¹The epoch is the standard base time: January 1, 1970, 00:00:00 GMT.

²For example, 20 bytes for a SHA-1 message digest.

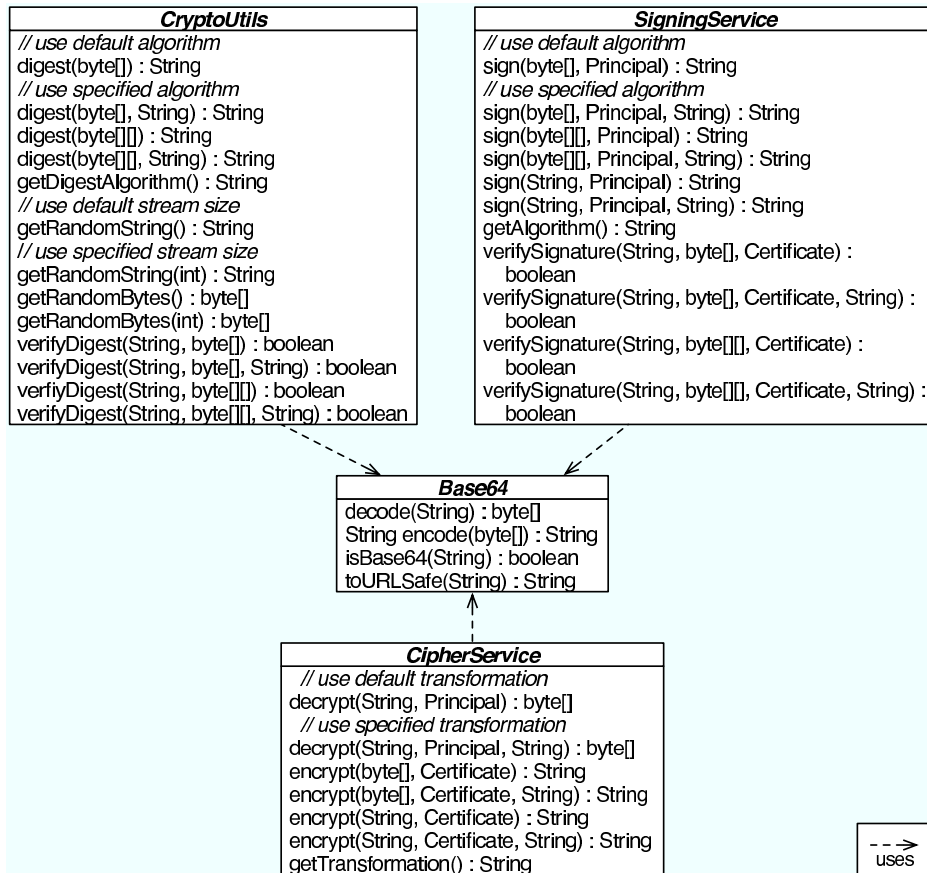


Figure 4.10: Cryptographic services API

interpreted by the services and how access to the keys is obtained is implementation-dependent. There are a number of options to authenticate a service for such access. For example, the service configuration may provide the necessary authentication. For simple services based on a Java keystore, this could be the approach taken; where the principal maps to a passphrase for a private key held in the keystore. Alternatively, the service may provide callback functionality that, for example, presents a principal with information to sign and prompts for authentication to access the user's private key. It should be noted that the CipherService is only used when message-level encryption is required; as opposed to network-level encryption. For example, for clarity, the description of the modified Coffey-Saidha protocol in Figure 3.5 shows message level encryption to emphasise when it is fundamental to the fairness guarantee to ensure that a message is kept private to sender and immediate recipient. However, for this and many other protocols, network-level encryption between protocol participants will suffice and may indeed

be a requirement of the business interaction. Technology such as Secure Sockets Layer can provide this level of privacy. The CipherService is used when, to guarantee fairness, parts of a message must be kept secret from the immediate recipient of the message.

CryptoUtil provides four digest methods. The first uses a default algorithm to generate a Base64-encoded message digest of binary input provided in the form of a byte array. The second digest method also generates a digest of a byte array but uses the algorithm specified by name in the second parameter to the method. The other two digest methods perform the corresponding functions for a sequence of byte arrays (`byte[][]`). The `getDigestAlgorithm` method returns the name of the default algorithm used by the CryptoUtil implementation. There are four methods for generation of secure pseudo random number streams. Two of them generate random Base64 strings of either a default length or a specified length. The other two are corresponding methods to generate random byte arrays. The remaining CryptoUtil methods verify that the Base64 string provided as first parameter is a valid digest of the byte array, or array of byte arrays, provided as second parameter. As with the corresponding digest methods, a default digest algorithm is used for verification unless another is specified.

SigningService provides methods for the generation of signatures and for the verification of signatures in their Base64 string representation. As with CryptoUtil, a default signature algorithm can be used, as identified by the `getAlgorithm` method; or the user can specify the algorithm to use. `sign` methods are provided to sign both byte arrays and Base64 strings. The `Principal` parameter to `sign` methods provides the service with the information it needs to gain access to the identified entity's private signing key. The exact mechanism used to access the key is service-specific. The `verify` methods use the Certificate provided to access the public key that corresponds to the signing key. To verify a signature, the relying party performs the following three steps: (i) verify any time-stamp associated with the signature (see Section 4.3.3); (ii) use a `CertificateService` to verify whether the signer's certificate was valid at the time of the time-stamp; and (iii) use a `verify` method of a `SigningService` to determine whether the signature is valid with respect to the input data — provided as a `byte[]` or `byte[][]` to the `verify` method.

CipherService provides decryption of Base64 strings using the private key associated with the specified Principal. Encryption is performed over byte arrays or Base64 strings using the public key accessed through the Certificate provided. A default transformation, identified by the `getTransformation` method, is used unless another is specified.

4.3.3 TimeStampService

As shown Figure 4.11, a time-stamp service takes some Base64 input and returns a time-stamp

```
public interface TimeStampService {
    TimeStamp getTimeStamp(String base64data);
    TimeStamp getTimeStamp(String base64data, boolean useCertURI);
}
```

Figure 4.11: TimeStampService interface

over the input. The service binds a time, in milliseconds since the beginning of the epoch, to the Base64 input by generating a signature over their concatenation. Figure 4.12 defines the

```
public interface TimeStamp {
    long getTime();
    String getSignature();
    String getSignatureAlgorithm();
    CertificateRecord getCertificateRecord();
}
```

Figure 4.12: TimeStamp interface

interface to a `TimeStamp` that provides access to the time, the signature, the signature algorithm used by the time-stamping service and the service's certificate record. To verify a time-stamp, a relying party must verify that both the time-stamping service's certificate was valid at the time of the time-stamp, using the `CertificateService`; and that the signature, generated with the private key corresponding to the time-stamping service's certificate, is valid for the given time and the input data, using the `SigningService`. A `TimeStamp` provides access to the service's certificate through a `CertificateRecord`, returned by the `getCertificateRecord` method. A Cer-

certificateRecord either contains the appropriate public key certificate or provides a URI to access the certificate. As shown in Figure 4.11, a time-stamping service offers two `getTimeStamp` methods. The first will return a `TimeStamp` that includes the service certificate in the `CertificateRecord`. The second returns a `TimeStamp` with a certificate URI to access the certificate in the `CertificateRecord`.

4.3.4 NonRepudiationLog

Non-repudiation logs contain serialized `NRLogEntry` objects. An `NRLogEntry` object contains a non-repudiation protocol message (`NRProtocolMessage`) along with related meta-information such as time of entry in the log, references to certificates accessible through the `CertificateService` and any protocol-related annotations that may provide a more verbose description of the purpose of a message. The log stores entries in the serialized form specified by the enclosed message's data-binding. As shown in Figure 4.13, the Java interface used by the protocol exe-

```
public interface NonRepudiationLog {
    void add(NRLogEntry entry);
    boolean contains(NRLogEntry entry);
    boolean contains(Uri nrId);
    NRLogEntry[] getEntries(Uri nrId);
    NRLogEntry[] getEntries(long onDay);
    NRLogEntry[] getEntries(long fromTime, long toTime);
    NRLogEntry[] getEntriesAfter(long time);
    NRLogEntry[] getEntriesBefore(long time);
}
```

Figure 4.13: NonRepudiationLog interface

cution middleware provides methods to add a log entry; to determine whether the log already contains an entry, or entries, for a given `nrId`; and to retrieve entries by `nrId` or that were logged at specified time periods. Non-repudiation log implementations will provide richer functionality for subsequent access to evidence. For example, if the agreed data-binding is XML, then an XML database can be used as the backing store that allows Xpath queries over entries in the log.

4.3.5 MessageLog

Implementations of the active message log discussed in Section 3.4 provide the MessageLog

```
public interface MessageLog {
    void add(NRProtocolMessage msg);
    void remove(NRProtocolMessage msg);
    NRProtocolMessage[] getMessages(String protoName);
    NRProtocolMessage[] getMessages(Uri nrId);
    NRProtocolMessage[] getMessages();
}
```

Figure 4.14: MessageLog interface

interface shown in Figure 4.14. The interface provides methods to add and remove messages, to retrieve messages by protocol name or nrId, and to retrieve all messages from the log.

As stated in Section 1.2, fault-tolerant exchange is beyond the scope of this dissertation. However, the protocol execution framework described in this chapter and the preceding support services together provide a basis for recovery from temporary local failure that will facilitate future work on fault tolerance. I now provide a brief overview of the support for recovery, followed by a summary of the contributions of this chapter.

4.4 Recovery from temporary local failure

It is the responsibility of the coordinator service to initiate recovery from local failure. After restart, for each message in the active message log, the service instantiates an appropriate protocol handler and passes the message to the handler using its recover method (see Figure 4.3 on page 139). The protocol handler first determines whether a given message is an incoming or outgoing message. If it is an incoming message, then the handler knows that message processing was interrupted. Message processing is essentially stateless. Therefore, one option is to simply start processing the message again as if it had been received normally. In this case, the protocol handler passes the message to its own handleMessage method. The resulting processing leads to the removal of the message from the active message log. If it is a valid message for an active protocol run, then it will be entered in the non-repudiation log and pro-

tol participation will resume. If the given message is an outgoing message, then the handler knows that, according to the protocol, a response to the message is expected. In this case, an option is to assume that the response will eventually arrive and to simply leave the message in the active message log. In practice, the non-repudiation service and the protocol to which a message relates determine how such recovered messages are processed.

Recovery for non-repudiable service invocation

For a direct exchange protocol of the type described in Section 3.2.1.1, the only option is to attempt to resume the exchange to which a message relates. The protocol handler may be configured to issue a protocol status request to the other participant as part of the attempt to resume. For a fair exchange protocol, the handler also has the option to request either abort or resolution of the exchange. The option that is chosen will be determined by the configuration of the middleware. A handler could be configured to always request the resolution of the exchange. However configured, the outcome depends on the state of the exchange (see, for example, Section 3.2.1.2). If an exception handling sub-protocol is initiated, then any message that relates to the corresponding main protocol is cleared from the active message log.

Recovery for non-repudiable information sharing

Information sharing protocol handlers can determine the local view of agreed information control state from the messages passed to them and from messages in the non-repudiation log. They can also determine the state of any active protocol run. From this information they can attempt to resume execution of a coordination protocol. Optionally, they may also request protocol status from their remote peers and obtain their view of the currently agreed control state. If there is a divergence of view then the peers must have evicted the participant on whose behalf the handler is acting, because no new control state could have been agreed unless their eviction preceded its agreement. In this case, the handler may be configured to request re-connection to the sharing group and the resumption of participation in information sharing.

4.5 Summary

The protocol execution framework presented in this chapter is a generic base for the implementation of non-repudiation services. Central to the framework's design is a careful separation of concerns. Protocol-specific message processing and generation is separate from message exchange — coordinator services are protocol-independent. Application-specific issues such as the specific non-repudiation protocol to use, the validation of actions, the notification of events and the representation of evidence are all configurable. Thus the framework can adapt to both non-repudiation specific and application-specific requirements. There follow four specific aspects of the framework that contribute to its flexibility.

1. The definition of APIs for interaction between components of the framework and between coordinator services (in contrast, for example, to the lack of such APIs in the FIDES project discussed in Section 2.4).
2. The ability to customise behaviour through configurable, factory-based instantiation of protocol handlers, validation listeners, event listeners and data bindings for the agreed representation of evidence.
3. The use of self-describing protocol messages that underpin the protocol-independence of B2B communication and the ability to customise behaviour at message recipients.
4. The specification of a support service API that allows local administrative control over provision of the supporting infrastructure without modification to the protocol execution framework.

Thus the framework addresses the flexibility requirement identified in Section 1.2. Further, the framework provides support for recovery from temporary local failure that is the basis for future work on fault tolerance. I now demonstrate its use in the middleware implementation of the non-repudiation services defined in Chapter 3.

Chapter 5

Implementation of non-repudiation services

This chapter describes the implementation of the non-repudiation services defined in Chapter 3 using the protocol execution framework from Chapter 4. I concluded Chapter 3 by identifying the following remaining challenges: (i) to realise the defined non-repudiation services as flexible middleware, and (ii) to support access to shared information in the context of distributed transactions. Taken together, the protocol execution framework and the implementations presented in this chapter address these remaining challenges. Section 5.1 provides an overview of the J2EE application server that is the demonstrator platform for the service implementations. The middleware services provide non-repudiation and validation of interactions between and with application server components. Section 5.2 describes the implementation of non-repudiable service invocation that mediates interaction with J2EE session beans that are exposed as services for access by remote clients. Section 5.3 describes the implementation of non-repudiable information sharing that mediates access to standard Java objects and/or J2EE entity beans that represent the state of the shared information. Each member of the sharing group hosts an object replica and the non-repudiation service coordinates changes to the state of the replicas. Section 5.3.3 shows how extensions to the information control state introduced in Section 3.3.1 support the participation of shared information as a transactional resource in standards-compliant distributed transactions.

The use of both non-repudiation services is essentially declarative. The application programmer is responsible for identifying when non-repudiation is required and for configuration

of the relevant non-repudiation service. The only significant additional implementation effort is the provision of any necessary application-specific validation and event listeners (see Section 4.2.4). I detail application programmer responsibilities with respect to non-repudiable service invocation in Section 5.2.1, and with respect to non-repudiable information sharing in Section 5.3.2. The utility of each service is demonstrated with proof-of-concepts implementations of the applications introduced in Section 1.1. Section 5.2.2 demonstrates the use of non-repudiable service invocation for the ASP-hosted B2B auction application and Section 5.3.4 demonstrates non-repudiable information sharing for the negotiation of tenders to supply chemicals. Section 5.4 concludes this chapter with an overview of the novelty of the service implementations when compared to other approaches to middleware support for non-repudiation.

5.1 J2EE application server platform

This section provides a brief overview of relevant aspects of J2EE applications and of the JBoss J2EE-compliant application server [FR03] that has been chosen as the demonstrator platform for implementation of non-repudiation middleware services. Full details of the J2EE architecture are provided in Sun's specification [Sun03]. The JBoss application server was chosen because it provides straightforward mechanisms for realisation of the interceptor-mediated interaction introduced in Section 3.1. It is also a widely used platform with which to experiment. Nevertheless, as is apparent from the following discussion, the general approach to development of the middleware is neither JBoss nor J2EE specific.

J2EE applications are assembled from self-contained software units or components. The components include Enterprise JavaBeans (EJBs) that are deployed on an application server. There are three types of EJB:

1. session beans that typically provide the externally visible service interface to a J2EE application and, as the name suggests, represent client sessions for interaction with the application;
2. entity beans that provide persistent storage of application state; and

3. message driven beans that provide publish and subscribe notification to J2EE applications.

EJBs execute in an environment called an EJB container. Together, the application server and the container provide a bean's run-time environment. The container is responsible for invoking appropriate low-level services, such as persistence and transaction management, for each invocation on a hosted bean. The application programmer concentrates on the functional (business logic) aspects of the bean's behaviour. The container invokes services to ensure correct, non-functional behaviour. Figure 5.1 shows an EJB application client invoking an

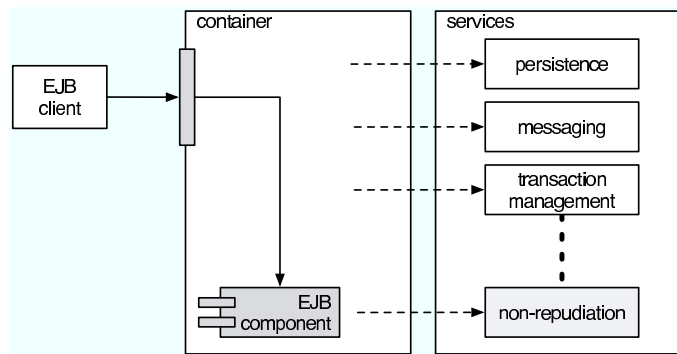


Figure 5.1: J2EE-based component architecture with non-repudiation

operation on an EJB component and the container interception of the invocation to provide various services. As shown, the intention is to add non-repudiation to the services available to the container in order to provide regulated access to EJBs.

The demonstrator implementations of the non-repudiation services extend version 3.2.5 of the JBoss J2EE application server. JBoss makes systematic use of reflection and invocation path interceptors to allow extension to existing services and to add new services. Interception on the EJB invocation path is a straightforward way to realise interceptor-mediated interaction. Although this exploits JBoss-specific mechanisms, similar support is found in other component-based systems. For example, the Jironde flexible transaction framework [Pro03] also makes use of interceptors. Furthermore, even if a J2EE implementation does not support the introduction of new interceptors, standard Java support for dynamic proxies can be used to introduce a layer between application clients and application server components. The use of dynamic proxies to support on-line upgrades to component systems [OTMMS02] is an ex-

ample of this approach. In fact, it is common for remote invocation middleware such as Java RMI, CORBA and .Net to support dynamic proxies or the interception of the service invocation path; as do many Web service frameworks. As far as possible, the implementations presented here isolate JBoss-specific details. The supporting infrastructure and much of the protocol execution framework is neither specific to JBoss nor to J2EE. For example, non-repudiation interceptors are instantiated from factories that could provide implementations for platforms other than JBoss. As stated previously, work on non-repudiation of Web service interactions illustrates the flexibility of the approach (see Chapter 6).

In JBoss, interceptors are used to invoke container-level services to meet requirements specified in a component's deployment descriptor. An application-level invocation passes through a chain of interceptors. Each interceptor performs some task before passing the invocation to the next interceptor in the chain. Existing services can be modified or new services added to a container by inserting additional interceptors in the chain. JBoss uses reflection to provide the interceptor with access to the application-level method that was called, the parameters to the method call, the target bean and its deployment descriptor. A dynamic proxy is used to instantiate both client-side and server-side interceptors. Thus the mechanism supports the execution of additional logic at the client-side on behalf of a container-level service.

In the non-repudiation middleware implementations, the application programmer identifies invocations on EJBs that require non-repudiation: whether as a result of remote service invocation or local access to shared information. Then, at run-time, JBoss interceptors invoke appropriate operations on the non-repudiation middleware. Typically, non-repudiable invocation applies to access to session beans. Non-repudiable information sharing applies to entity beans that represent the application state that is shared between organisations. As described in Chapter 4, the protocol execution framework provides a generic coordinator service for the exchange of protocol messages and supports the deployment of protocol-specific handlers to drive the execution of non-repudiation protocols. The remainder of this chapter details the implementation of non-repudiation middleware services on the JBoss/J2EE platform.

5.2 Implementation of non-repudiable service invocation

Service invocation in a J2EE application equates to the remote invocation of an operation on an enterprise bean. Figure 5.2 shows an application accessing a service object through its service

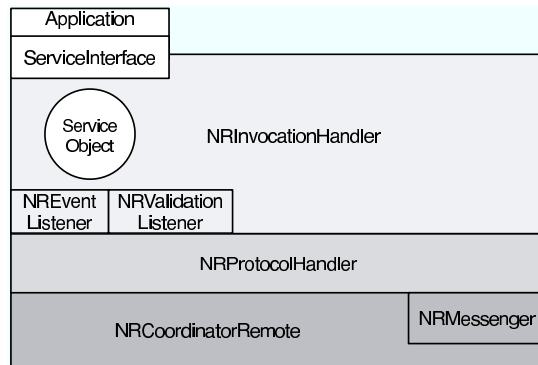


Figure 5.2: Non-repudiable service invocation components

interface. Typically, the service object is a session bean that implements a remote interface that is presented to client applications. To provide non-repudiable service invocation, an invocation at the service (remote) interface is intercepted before execution of the operation on the target service object (session bean). The `NRInvocationHandler` layer in Figure 5.2 performs this mediation and triggers execution of appropriate protocols using the framework described in Chapter 4. The `NRInvocationHandler` corresponds to the non-repudiation service specific layer shown in Figure 4.2 on page 137. I now describe how to use the JBoss facility for server- and client-side interceptors to implement this layer between the application and the generic protocol execution framework.

As shown in Figure 5.3, the JBoss client's reference to the service object is a dynamic proxy generated by the server. This proxy contains client-side interceptors that are typically used for context propagation. The implementation of non-repudiable service invocation adds an extra, non-repudiation interceptor to both client and server invocation paths. These non-repudiation interceptors are responsible for triggering execution of a non-repudiation protocol that achieves the exchange of evidence described in Section 3.2. The client-side non-repudiation interceptor accesses the client's non-repudiation middleware that, in turn, manages the client's participation in protocols and its access to supporting infrastructure to store evidence etc. Figure 5.3

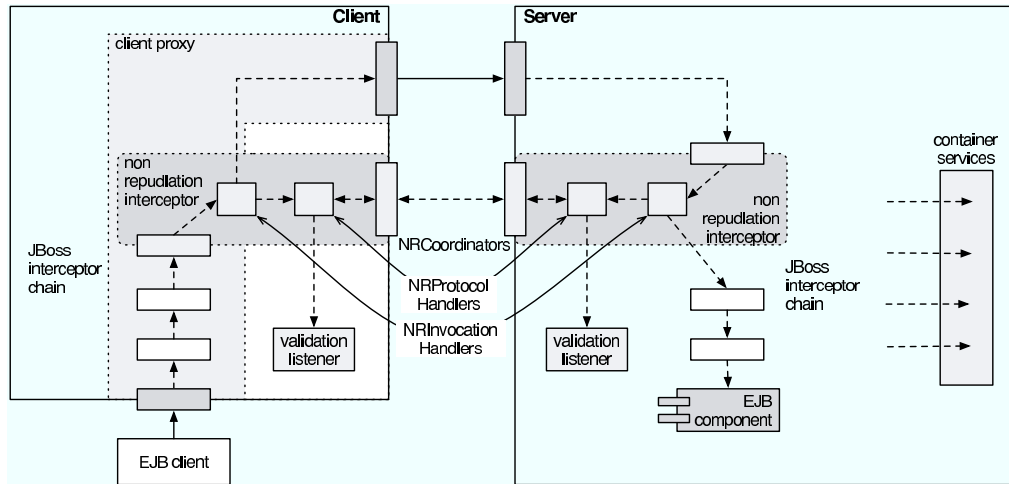


Figure 5.3: JBoss/J2EE-based implementation of non-repudiable invocation

shows direct execution of the request phase protocol between client and server coordinators. For protocols supported by a guarantor TTP, a coordinator service at the TTP would also be involved in the exchange (as depicted in Figure 4.1b on page 136).

Each interceptor in a chain may execute on both the outgoing and incoming invocation path. To achieve non-repudiation of the request as constructed by the client and to verify the integrity of the response presented to the client, the client-side non-repudiation interceptor is the last in the chain on the outgoing path (and first on the return path). On the server-side, to verify the integrity of the request as it enters the server and to provide non-repudiation of the response as it leaves the server, the interceptor is the first in the chain on the incoming path (and the last on the return path).

Each JBoss interceptor has an `invoke` operation that takes an `Invocation` object¹ as a parameter for the interceptor to process in some way. The interceptor then passes the `Invocation` to the next interceptor in the chain by calling that interceptor's `invoke` operation. The implementation of the `invoke` operation of the client-side JBoss non-repudiation interceptor is shown in Figure 5.4. First, the `NRInvocationHandler` `getInstance` factory method is used to obtain a handler for the given platform ("JBossJ2EE"). The concrete implementation of a `NRInvocationHandler` is under the control of the client. A `NRInvocation` object is a generic wrapper

¹An `Invocation` encapsulates the client's service invocation and includes the request parameters, context information and related payload.

```

public Object invoke(Invocation inv) {
    NRInvocationHandler nrInvHdlr =
        NRInvocationHandler.getInstance("JBossJ2EE");
    NRInvocation nrInv =
        new JBossNRInvocation(nextInterceptor(), inv);
    return nrInvHdlr.invoke(nrInv);
}

```

Figure 5.4: JBoss NRInterceptor invoke operation

for platform-specific representations of the service to invoke and the invocation parameters(s). In JBoss, the service to invoke is the next interceptor in the chain and the parameters are encapsulated by the Invocation object. As shown, a JBossNRInvocation object is instantiated with the next interceptor and the Invocation object as parameters. The NRInvocation object is then passed to the NRInvocationHandler for processing and to initiate request phase protocol execution. Eventually, after successful completion of both request and response phases of the invocation and of the protocols associated with each phase, this handler returns the result of the invocation. This result is then passed back through the client interceptor chain to the client application.

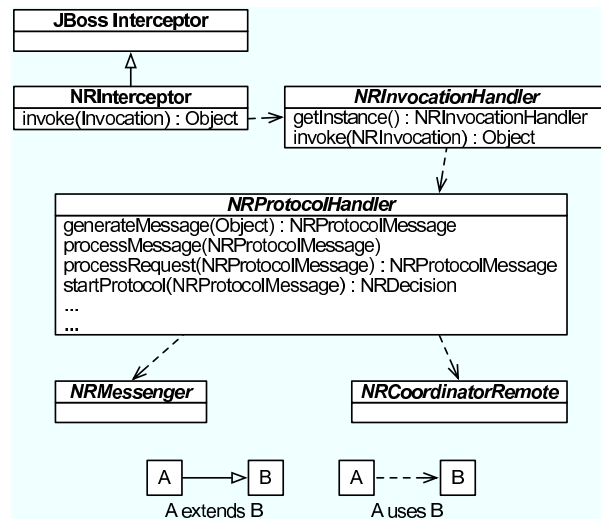


Figure 5.5: JBoss interceptor and NRInvocation API

Figure 5.5 shows the relationship of the NRInvocation interceptor API to protocol execution and coordination. The invocation handler manages any conversion of information be-

tween layers (including protocol outcomes), initiates protocol execution, and instantiates the protocol-specific handlers. When `invoke` is called, the general behaviour of the client-side `NRInvocationHandler` for the request phase of the invocation is to:

1. obtain the name of the protocol to execute from non-repudiation configuration information,
2. instantiate a protocol handler for the given protocol,
3. generate the first message of the protocol using the protocol handler's `generateMessage` method,
4. start the request phase protocol by passing the generated message to the protocol handler's `startProtocol` method,² and
5. wait for return of the `NRDecision` from the `startProtocol` method that indicates termination of the request phase protocol.

The `NRDecision` indicates whether the protocol completed successfully. It includes the `nrId` that identifies the request phase protocol run. If the protocol failed, then the invocation handler raises an appropriate exception that will be propagated to the client. If the protocol completed successfully, the client-side `NRInvocationHandler` replaces the parameters to the original JBoss Invocation object with the `NRDecision`. This modified Invocation is then passed up through the interceptor chain to the server by calling the `invoke` operation of the next interceptor that was provided in the `NRInvocation`.

On the server-side, the Invocation object is passed as a parameter to its non-repudiation interceptor's `invoke` method. The behaviour of the interceptor is the same as on the client side. A JBoss-specific `NRInvocationHandler` is instantiated and is passed a `JBossNRInvocation` that encapsulates a reference to server's next interceptor and the client-provided Invocation. The interceptor then calls the `NRInvocationHandler`'s `invoke` method with the `NRInvocation` as a parameter. The general behaviour of the server-side `NRInvocationHandler` is to:

²Depending on the level of involvement of any guarantor TTP, the first message may be sent directly to the server's coordinator or via the TTP.

1. obtain the original invocation request parameters that were exchanged during protocol execution and that were cached by the server-side protocol handler³;
2. replace the NRDecision in the Invocation object with the request parameters, in order to reconstruct the client's original Invocation, and pass the reconstructed Invocation through the server-side interceptor chain to the target EJB component for execution; and
3. use the result of service processing to initiate the response phase of the invocation.

The response phase is essentially the reverse of the request phase. The sever-side NRInvocationHandler initiates protocol execution, using the service response as application level data. On completion of protocol execution, an NRDecision is returned to the server-side NRInvocationHandler. This NRDecision is then passed back along the invocation path to the client as the result of the client-side NRInvocationHandler's call to `invoke` on its next interceptor. If the response phase protocol completed successfully, then the NRInvocationHandler will be able to retrieve the service response that was exchanged, and cached, during protocol execution. The handler uses the *nrId* from the server-provided NRDecision to identify the response. This response is then passed through the client-side interceptor chain to the client. If the response-phase protocol failed, then an appropriate exception is propagated to the client. If the target service raised an exception when processing the client's request, then this exception can be used as the subject of the response-phase protocol and be subsequently re-created for propagation at the client.

The NRInvocationHandlers, on both client- and server-side, cooperate with service invocation protocol handlers to maintain the correlation of application-level request and response with the protocol runs that effect a given exchange, identified by *nrIds*. In the case of a direct exchange (see Section 3.2.1.1), this correlation is achieved by inclusion of application-level data in the first message of a protocol run. In the case of a fair exchange (see Section 3.2.1.2), the protocol handlers persist application-level data passed to them by the invocation handlers when the first message of a protocol run is generated. This data is then released at the appropriate point in protocol execution. For example, in the protocol discussed in Section 3.2.1.2, the

³The request parameters are referenced through the protocol run *nrId* provided in the NRDecision. The request parameters can also be retrieved from the relevant message in the server-side non-repudiation log.

client request is subjected to validation at step 1.4. Similarly, in the response phase, validation occurs at step 2.4.

5.2.1 Application programmer responsibilities

The server-side application programmer identifies when non-repudiation is required and provides (server-side) non-repudiation configuration information. On the client-side, configuration information is provided to determine the client's reaction to a server's demand for non-repudiation. Thus, rendering a service invocation non-repudiable is declarative and, apart from validation or event listeners, requires no additional implementation effort on the part of the application programmer.

The application programmer on the server-side is responsible for identifying, in an EJB's deployment descriptor, when non-repudiation is required and for identifying the platform for client-side instantiation of the `NRInvocationHandler` by its non-repudiation interceptor. Thus the servers-side application programmer controls activation of non-repudiation. They can stipulate that the client participate in the generation of non-repudiation evidence at two levels: (i) for all components with a remote interface, or (ii) for one or more of the remote methods of selected components. The client-side application programmer determines whether the client engages in non-repudiable exchange through the configuration of their own non-repudiation middleware. Non-compliance will result in failure of a service request. Once the client is engaged in non-repudiable exchange, through instantiation of a non-repudiation interceptor, they control their participation. The client-side application programmer configures the locally owned and deployed implementations of non-repudiation middleware components, such as the `NRInvocationHandler`, `NRProtocolHandler` and the coordinator service. The client-side application programmer is also responsible for implementation of local validation and event listeners.

5.2.2 Regulated interaction with a B2B auction application

This section demonstrates how the B2B auction application introduced in Section 1.1.1 is augmented to regulate client interactions with the application. This augmentation does not involve

any changes to application code. The non-repudiation middleware renders application-level operations non-repudiable and validates them with respect to the terms and conditions of the auction. Section 5.2.2.1 describes application set-up and request processing. Section 5.2.2.2 demonstrates the non-repudiation of both a valid and an invalid auction service invocation. Section 5.2.2.3 evaluates the demonstration.

5.2.2.1 Auction application set-up

As discussed in Section 1.1.1, a third party ASP has developed the auction application that

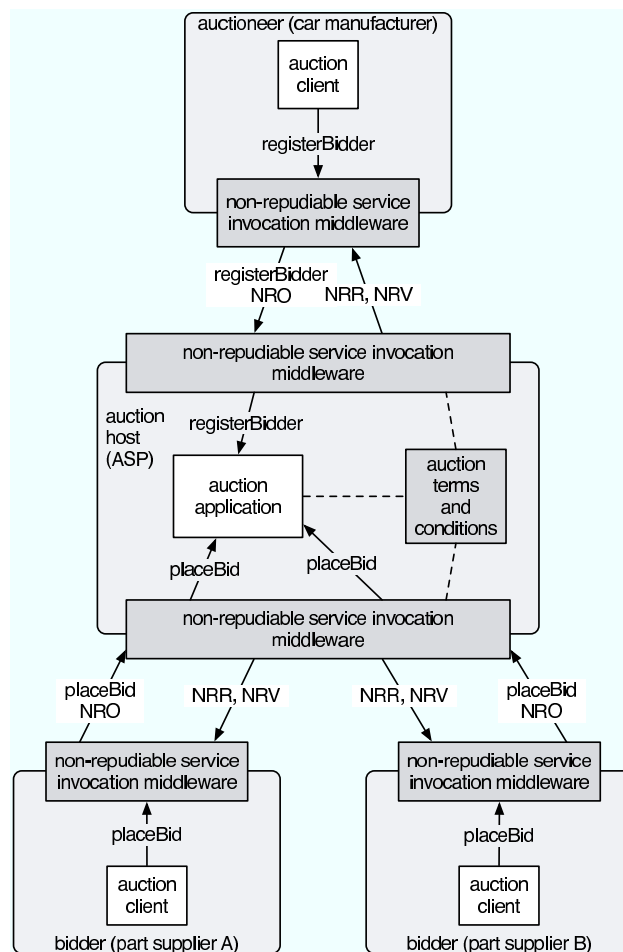


Figure 5.6: Regulated B2B auction application

they host on a cluster of JBoss/J2EE application servers. The following requirements on the non-repudiation middleware are recalled from Section 1.1.1.

1. Both the origin and receipt of requests must be non-repudiable; that is, evidence must be generated to irrefutably bind a request to an originating client and, given evidence of origin, the auction service must provide irrefutable evidence of receipt of a request. The middleware must also maintain non-repudiation logs of the evidence generated both at the clients and at the ASP. Since the ASP is trusted by the clients, the direct exchange of evidence described in Section 3.2.1.1 can be used to satisfy these requirements.
2. Requests must be validated with respect to business contract terms and conditions that govern the behaviour of clients participating in an auction.

Figure 5.6 shows the augmentation of the auction application with the non-repudiation middleware in order to meet the preceding requirements. Three auction clients, an auctioneer and two bidders, submit requests to the auction service. For example, Figure 5.6 shows these requests in the form of the auctioneer's registration of a bidder and the bidding clients' invocation of `placeBid` operations. The non-repudiation middleware at client- and server-side intercepts the requests to ensure the exchange of the request and evidence of origin of the request (NRO) from the client, in return for the receipt (NRR) and validation (NRV) of the request by the service. As described in Chapter 4, the middleware uses support services to, for example, log non-repudiation evidence, manage credentials and apply trusted time-stamps for signed evidence.

The ASP conducts validation against auction terms and conditions by registering validation listeners with the non-repudiation middleware. As depicted in Figure 5.6, the middleware invokes this application-level validation with respect to the auction terms and conditions when a request is intercepted. The validation listeners also access the state of the auction application to determine whether a request is valid given the current state of the auction. The validation ensures that the auction progresses along the lines specified in Section 1.1.1. If the middleware detects a contract violation, the violation is logged and the request that triggered the violation is rejected. This results in an application-level exception at the client that indicates that they submitted an invalid request. Such exceptions are essentially the same as any other exception raised by the application. Invalid requests are not passed to the auction service for processing and, therefore, its application state remains unchanged.

5.2.2.2 Auction demonstration

This section first demonstrates the auction application given a valid client request and then demonstrates the violation of one of the rules governing bidder behaviour. Before starting an auction, the auctioneer registers a new bidder from the *partsuppliers.com* company. The auc-



The screenshot shows a web browser interface with a left-hand navigation menu and a main content area. The navigation menu has four items: 'Auctions overview', 'Create auction', 'Register bidder', and 'Log out'. The 'Register bidder' item is highlighted in orange. The main content area is titled 'Register a new bidder' and contains a form with the following fields and values:

First Name:	Craig
Last Name:	Scanlon
Street:	Parts Dept
Postal Address:	Car Part Suppliers, Glasgow
Phone:	0123 456 789
Email:	j.scanlon@partsuppliers.com
Tax ID Number:	10

At the bottom of the form are two buttons: 'Confirm' and 'Reset'.

Figure 5.7: Auctioneer registering bidder

tioneer provides the bidder's details through the Web browser interface shown in Figure 5.7. This results in the interception of the registration request. Non-repudiation evidence is generated and logged, and the request validated. Figure 5.8 shows an extract of a non-repudiation log entry for the request and NRO evidence (note the use of the `xmlstring` data-binding). The invocation parameters of the request show the auctioneer, Mark Smith, creating a new person, Craig Scanlon, in the role of bidder. These parameters are signed along with the service URI and related information in the signature input. Figure 5.9 is an extract of a log entry for the NRR and NRV evidence. This entry indicates that the request is valid. The signed receipt includes a digest of the request, `receiptDigest`, that matches the signature input digest from the NRO evidence. The evidence shown is logged both at the ASP and the auctioneer. Since the request is valid, the indicated `createPerson` operation will be invoked on the auction application hosted by the ASP and the state of the application will be updated accordingly — bidder Craig Scanlon with id 24 is registered with the auction application.

Log entry date	Mon Feb 06 13:31:50 GMT 2006 [1139232710118]
Type	NRO-Request
NRId	Ehb7UQ8xIOBqQVOWsJjgSkoTpcE=
Signature input	<pre> <uk.ac.ncl.nrcore.NRSignatureInput> <metaInf class="uk.ac.ncl.nrcore.NRMetaInf"> <dataBinding>xmlstring</dataBinding> <digestAlgorithm>SHA1</digestAlgorithm> <nrid>Ehb7UQ8xIOBqQVOWsJjgSkoTpcE=</nrid> <protocolName>/nrinv/request/direct/main</protocolName> <protocolStep>1</protocolStep> <purpose>NRO</purpose> <recipient class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://tapasauction. adesso.de:55010/nrcoord <uri>jnp://tapasauction. adesso.de:1099/</uri> </recipient> <sender class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://nrcoord. vauxhallcars. com:55010/nrcoo <uri>mark. smith@vauxhallcars. com</uri> </sender> <signatureAlgorithm>SHA1withDSA</signatureAlgorithm> </metaInf> <data class="uk.ac.ncl.nrinvoication.NRInvocationData"> <invocationParams> <entry> <string>person</string> <de. adesso. tapas. vo. BidderVO> <address>Car Part Suppliers, Glasgow</address> <auctioneer> <address>Vauxhall Cars plc, Manchester</address> <bidders class="list"/> <email>mark. smith@vauxhallcars. com</email> <firstName>Mark</firstName> <id>22</id> <lastName>Smith</lastName> <loginName>Smith</loginName> <phone>0161 123456</phone> <role> <name>auctioneer</name> <orderId>1</orderId> </role> <street>Purchasing Dept</street> </auctioneer> <email>craig. scanlon@partsuppliers. com</email> <firstName>Craig</firstName> <id>24</id> <lastName>Scanlon</lastName> <phone>0123 456 789</phone> <role> <name>bidder</name> <orderId>0</orderId> </role> <street>Parts Dept</street> <taxId>10</taxId> </de. adesso. tapas. vo. BidderVO> </entry> </invocationParams> <serviceURI> jnp://tapasauction. adesso.de:1099/ de. adesso. tapas. business. PersonSession?void#createPerson() </serviceURI> </data> </uk.ac.ncl.nrcore.NRSignatureInput> </pre>
Input digest	nsEgPr18kx3P8EpIS1rbsKfxiB4=
Signature	SHA1withDSA MC0CFQCPiakKlwYa4z89vS4tRYECCGOoy6QIU2RN507LRDZQ7ag28Ck3tLMUxyRk=
Timestamp date	Mon Feb 06 13:31:49 GMT 2006 [1139232709924]
Timestamp signature	SHA1withDSA

Figure 5.8: Request to register a bidder with NRO

Log entry date	Mon Feb 06 13:31:50 GMT 2006 [1139232710181]
Type	NRR-NRV-Request
NRId	Ehb7UQ8xIOBqQVOWsJjgSkoTpcE=
Signature input	<pre> <uk.ac.ncl.nrcore.NRSignatureInput> <metaInf class="uk.ac.ncl.nrcore.NRMetaInf"> <dataBinding>xmlstring</dataBinding> <digestAlgorithm>SHA1</digestAlgorithm> <nrid>Ehb7UQ8xIOBqQVOWsJjgSkoTpcE=</nrid> <protocolName>/nrinv/request/direct/main</protocolName> <protocolStep>2</protocolStep> <purpose>NRR/NRV</purpose> <recipient class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://nrcoord.vauxhallcars.com:55010 /nrcoordinator</coordinatorUri> <uri>mark.smith@vauxhallcars.com</uri> </recipient> <sender class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://tapasauction.adesso.de:55010 /nrcoordinator</coordinatorUri> <uri>jnp://tapasauction.adesso.de:1099</uri> </sender> <signatureAlgorithm>SHA1withDSA</signatureAlgorithm> </metaInf> <data class="uk.ac.ncl.nrcore.NRRReceipt"> <decisionStatusCode>VALID</decisionStatusCode> <receiptDigest>nsEgPr18kx3P8EpISlrbsKfxiB4=</receiptDigest> <receiptTo>mark.smith@vauxhallcars.com</receiptTo> <serviceURI> jnp://tapasauction.adesso.de:1099/ de.adesso.tapas.business.PersonSession?void#createPerson() </serviceURI> </data> </uk.ac.ncl.nrcore.NRSignatureInput> </pre>
Input digest	P6YcR535znDw/n3CKuYwtfp4zBY=
Signature	SHA1withDSA MCwCFDDxE1wA1nPh0GNav7nBS0ku+RoCAhRC3aaSd3Bu3VaEroKU9KSRxk4few==
Timestamp date	Mon Feb 06 13:31:50 GMT 2006 [1139232710144]
Timestamp signature	SHA1withDSA MCwCFA60mcqgVuXCLRV5QUkpTbqWuxNQAhRdXKdQMtDq29+5yAy2o1I7Vhb0zg==

Figure 5.9: NRR/NRV of request to register a bidder

Auctions overview	
Log out	
Place Bid	
Last Bid	
Auction name:	Parts auction
Description of the auction good:	wheel trims
Actual round:	1
Number of bid rounds:	2
Last Bid:	
Best Bid:	
End of Round:	06.02.2006 14:10
Start of the Next Round:	06.02.2006 14:30
New Bid:	<input type="text" value="2.50"/>
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

Figure 5.10: Bidder placing bid

I now demonstrate violation of the following auction rule:

if a car part supplier has had a bid accepted in an bidding round, then that same car part supplier must not bid again in that round.

In this example, two people are registered as bidders for *partsuppliers.com*. This is legitimate. However, it is a violation of the preceding rule for both of them to place a bid in the same round. If this occurs, then the second bid in a round from a representative of *partsuppliers.com* should be detected as a violation of auction rules and recorded as such in the non-repudiation evidence. To demonstrate this, the same bidder as registered by the auctioneer in Figure 5.7 — Craig Scanlon with bidder id 24 — submits a second bid in the first round of an auction. Figure 5.10 shows the Web browser interface for placing a bid. The bid value is 2.50 in round one. The resulting `placeBid` operation is intercepted and rendered non-repudiable. Figure 5.11 shows an extract of the evidence of the request and its NRO. Since the bid is the second in the round from *partsuppliers.com*, the application-level exception shown in Figure 5.12 is raised at the client to indicate failure of the operation. The non-repudiation log entry shown in Figure 5.13 includes the corresponding evidence of the invalidity of the request. This entry is logged both at the ASP and at *partsuppliers.com* for access by the bidder, Craig Scanlon. The `decisionStatusCode` in the signature input is `INVALID` and the `decisionDiagnostic` states that the bid was rejected because two bidders from the same company attempted to place bids in round one. Since

Log entry date	Mon Feb 06 14:08:18 GMT 2006 [1139234898472]
Type	NRO-Request
NRId	N7zJ4VC0c200uHmgpush--uJelw=
Signature input	<pre><uk.ac.ncl.nrcore.NRSignatureInput> <metaInf class="uk.ac.ncl.nrcore.NRMetaInf"> <dataBinding>xmlstring</dataBinding> <digestAlgorithm>SHA1</digestAlgorithm> <nrid>N7zJ4VC0c200uHmgpush--uJelw=</nrid> <protocolName>/nrinv/request/direct/main</protocolName> <protocolStep>1</protocolStep> <purpose>NRO</purpose> <recipient class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://tapasauction.adesso.de:55010 /nrcoordinator</coordinatorUri> <uri>jnp://tapasauction.adesso.de:1099</uri> </recipient> <sender class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://nrcoord.partsuppliers.com:55010 /nrcoordinator</coordinatorUri> <uri>craig.scanlon@partsuppliers.com</uri> </sender> <signatureAlgorithm>SHA1withDSA</signatureAlgorithm> </metaInf> <data class="uk.ac.ncl.nrinvocation.NRInvocationData"> <invocationParams> <entry> <string>auctionId</string> <long>2</long> </entry> <entry> <string>bid</string> <double>2.5</double> </entry> <entry> <string>bidderId</string> <long>24</long> </entry> <entry> <string>round</string> <long>1</long> </entry> </invocationParams> <serviceURI>jnp://tapasauction.adesso.de:1099/ de.adesso.tapas.business.AuctionSession?void#placeBid() </serviceURI> </nrData> </uk.ac.ncl.nrcore.NRSignatureInput></pre>
Input digest	gzLXPXX5Ty5Jwq1SMsIepeBFu9c=
Signature	SHA1withDSA MCwCFQC0boxLuDObuogscA+KB1EspjyktgITFkp+IYyby1TQWAvCXGtSSURaRQ==
Timestamp date	Mon Feb 06 14:08:18 GMT 2006 [1139234898353]
Timestamp signature	SHA1withDSA MC0CFQCQR0oaiYAagcJwRnAQ6JGE9NsoagIURaNT28hv5D+Gdy2BP98u8IVaZPA=

Figure 5.11: placeBid request with NRO

the operation is invalid, there is no corresponding placeBid invocation on the target auction application. Thus, the state of the auction application is unchanged — as if the invalid second

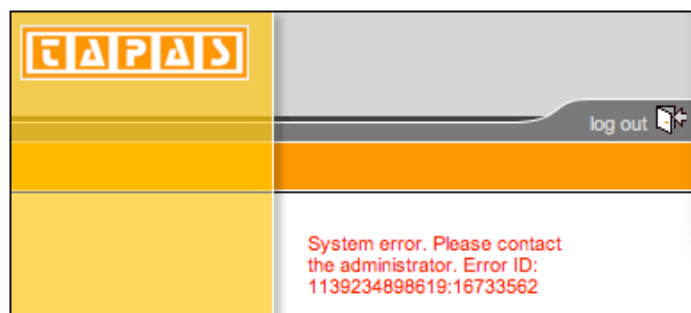


Figure 5.12: Client-side exception due to invalid bid

Log entry date	Mon Feb 06 14:08:18 GMT 2006 [1139234898522]
Type	NRR-NRV-Request
NRId	N7zJ4VC0c200uHmgpush--uJelw=
Signature input	<pre><uk.ac.ncl.nrcore.NRSignatureInput> <metaInf class="uk.ac.ncl.nrcore.NRMetaInf"> <dataBinding>xmlstring</dataBinding> <digestAlgorithm>SHA1</digestAlgorithm> <nrid>N7zJ4VC0c200uHmgpush--uJelw=</nrid> <protocolName>/nrinv/request/direct/main</protocolName> <protocolStep>2</protocolStep> <purpose>NRR/NRV</purpose> <recipient class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://nrcoord.partsuppliers.com:55010 /nrcoordinator</coordinatorUri> <uri>craig.scanlon@partsuppliers.com</uri> </recipient> <sender class="uk.ac.ncl.nrcore.Participant"> <coordinatorUri>rmi://tapasauction.adesso.de:55010 /nrcoordinator</coordinatorUri> <uri>jnp://tapasauction.adesso.de:1099</uri> </sender> <signatureAlgorithm>SHA1withDSA</signatureAlgorithm> </metaInf> <data class="uk.ac.ncl.nrcore.NRRReceipt"> <decisionDiagnostic>two or more bidders from same company: partsuppliers.com in round: 1</decisionDiagnostic> <decisionStatusCode>INVALID</decisionStatusCode> <receiptDigest>qzLXPXX5Ty5Jwq1SMsIepeBFu9c=</receiptDigest> <receiptTo>craig_scanlon@partsuppliers.com</receiptTo> <serviceURI>jnp://tapasauction.adesso.de:1099/ de.adesso.tapas.business.AuctionSession?void#placeBid() </serviceURI> </data> </uk.ac.ncl.nrcore.NRSignatureInput></pre>
Input digest	PLNlujiwtmuBjdx/oDEdLdbJOLA=
Signature	SHA1withDSA MCwCFCvE67ZwCI/JUr+qqcHCQKRX9+yTAhRgYJJ63P1velSpa2VCb1FgDQvLBA==
Timestamp date	Mon Feb 06 14:08:18 GMT 2006 [1139234898500]
Timestamp signature	SHA1withDSA MCwCFGf88gDqUy+DG9W5hYGEN9y9evyWAhRerU6kzXw11bxkpWB6QvAAcyaeAQ==

Figure 5.13: Evidence of invalidity of placeBid request

bid had not been made.

5.2.2.3 Evaluation of demonstration

The auction application successfully demonstrates the generation, exchange and persistence of non-repudiation evidence of service invocations. The fact that invocations are non-repudiable is transparent to the application-level processes at the auction clients (auctioneers and bidders) and at the ASP auction service. The rules governing the auction are enforced by application-level validation that is integrated with the non-repudiation protocol for the exchange of evidence. Client operations that violate the rules are detected and recorded but, critically, do not execute at the target auction service. Thus, neither the auction service nor its application state are compromised by contract violations. The demonstration shows that the middleware meets the requirements for non-repudiation and application-level validation identified in Section 1.2. Furthermore, the fact that no application code was modified to render the interactions

non-repudiable is an indication of the middleware's flexibility. The use of validation listeners supports context-specific customisation of the validation process. In the demonstration, the requirement for no unfair advantage for auctioneer or bidders is met because the ASP is a trusted host for the application and the invocations consist of one-way requests from the auction clients. As indicated by the protocol name — `/nrinv/request/direct/main` — in the signature input of each log entry (see Figures 5.8, 5.9, 5.11 and 5.13), this means that the request phase of the direct exchange protocol from Section 3.2.1.1 is used because the ASP guarantee a receipt to the client. The naming of the protocol in protocol messages indicates the potential for the protocol execution framework to adapt to the use of different protocols. This is reinforced by the use of the same execution framework to support non-repudiable information sharing, as described in the following.

5.3 Implementation of non-repudiable information sharing

This section presents the implementation of the non-repudiable information sharing service defined in Section 3.3. The implementation is a significant extension to the B2BObjects middleware that I first reported in [CSW02]. The middleware regulates information sharing by coordinating updates to object replicas hosted by the members of the sharing group. Shared information may be represented locally as a standard Java object⁴ or as an entity bean hosted by a J2EE application server. Section 5.3.1 provides a detailed description of the B2BObjects middleware, its API and its relationship to the protocol-execution framework described in Chapter 4. Section 5.3.2 describes the J2EE implementation of B2BObjects and application programmer responsibilities with respect to the J2EE implementation. The use of invocation path interceptors reduces this effort when compared to information sharing based on standard Java objects. Section 5.3.3 describes extensions to the middleware to address the remaining requirement to support transactional information sharing. Section 5.3.4 demonstrates the use of non-repudiable information sharing in the proof-of-concepts tender negotiation application introduced in Section 1.

⁴Commonly referred to as a “POJO” (Plain-Old Java Object).

5.3.1 B2BObjects middleware

Figure 5.14 shows the B2BObjects realisation of interceptor-mediated information sharing de-

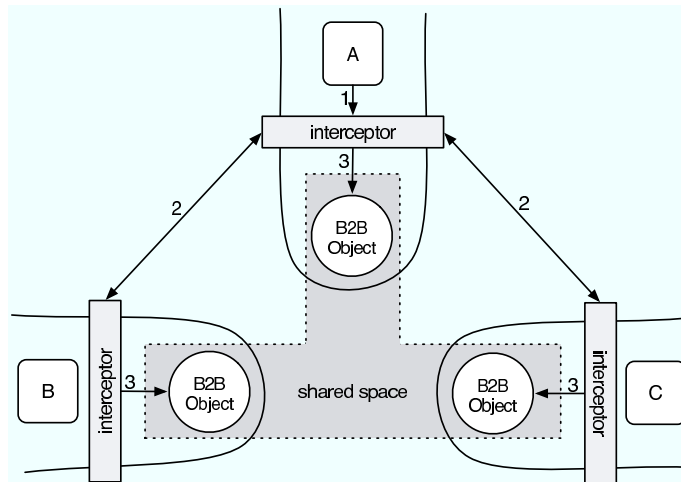


Figure 5.14: B2BObjects-based information sharing

finied in Section 3.3. The shared information is represented locally by object replicas. The middleware mediates both local and remote access to these objects. From the point of view of local application clients, access to an object replica is the same as access to a standard application object. The B2BObjects middleware ensures that application invocations on a local object replica are coordinated with other members of the sharing group using the protocols presented in Section 3.3.2. In step 1 in Figure 5.14, an application client at A makes an invocation on a local B2BObject replica. The middleware intercepts the invocation and, in step 2, executes an appropriate state coordination protocol between A, B and C. At step 3, if all parties agree, the state of each replica is updated to reflect the original invocation at A. If either B or C rejects the proposed update, there is no change to the state of any of the replicas.

Figure 5.15 shows the three layers of the B2BObjects middleware: (i) local access and state management that is mediated by the B2BObjectController, (ii) information sharing protocol execution based on NRProtocolHandler implementations, and (iii) remote access and coordination based on implementations of NRMessenger and NRCoordinatorRemote. The local access layer corresponds to the non-repudiation service specific layer of Figure 4.2 on page 137. The key component of the local access layer is the B2BObjectController. It performs two main

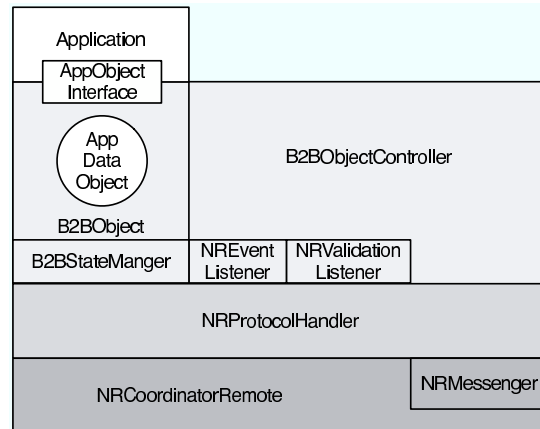


Figure 5.15: Non-repudiable information sharing components

functions: (i) to mediate access to and update of the underlying application object replica that is presented to the controller as a B2BObject, and (ii) to initiate non-repudiable information sharing protocols using the protocol execution framework.

As shown in Figure 5.15, an application client interacts with an underlying data object through an application object interface presented by a B2BObject wrapper. Thus, it is possible for the application need not be B2BObjects-aware. For example, an existing application object can be augmented to become a B2BObject without modification to any applications that use the object. Alternatively, it is possible for B2BObject-aware applications to use the B2BObject interface to interact programmatically with the B2BObjects middleware, and, in particular, an object's B2BObjectController. The API of the various components of the middleware and their relationships are shown in Figure 5.16. I now provide an example implementation of a B2BObject wrapper. This illustrates the use of the API to trigger state coordination via the B2BObjectController. In the J2EE implementation a non-repudiation interceptor provided by the middleware acts as the B2BObject wrapper (see Section 5.3.2).

The B2BObject wrapper and state coordination

A B2BObject can be defined as a new object that implements both the application object's interface and the middleware B2BObject interface, or by extension of an existing application object, or by generation of a proxy for an existing application object. Generation of a proxy object requires that the methods that write object state can be distinguished from read-only

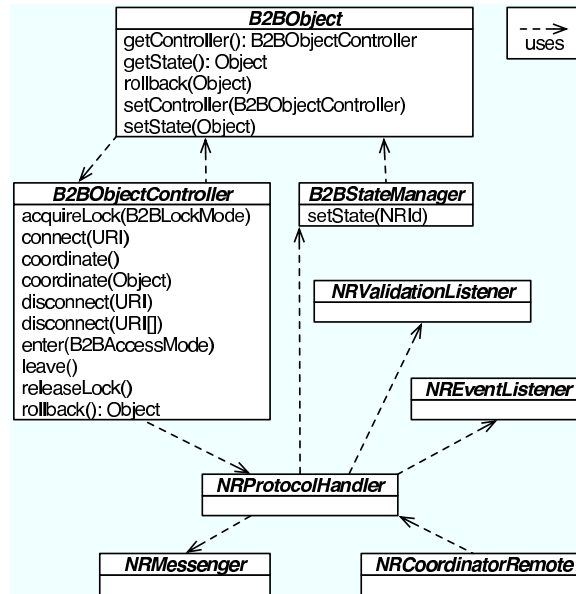


Figure 5.16: B2BObjects API

methods. For example, it is straightforward to generate a wrapper for an application object that uses set methods to write object state and get methods to read object state. However implemented, the method calls of the original application object are in effect wrapped to ensure that every access to object state is mediated by the B2BObjectController. The controller, in turn, accesses the application-level object through its implementation of the B2BObject interface.

```

public class OfferImpl implements Offer {
    // fields
    private String description;
    private String currency;
    private double price;
    // implementation of Offer interface
    public String getDescription() { return description; }
    public String getCurrency()    { return currency; }
    public double getPrice()       { return price; }
    public void setDescription(String description)
        { this.description = description; }
    public void setCurrency(String currency)
        { this.currency = currency; }
    public void setPrice(double price)
        { this.price = price; }
}
  
```

Figure 5.17: Application Offer implementation

Figure 5.17 shows an implementation of an application Offer that has get and set methods to read and write the description, currency and price of an offer. The corresponding B2BObject

```
public class B2BOffer implements Offer, B2BObject {
    // fields
    private B2BObjectController controller;
    private Offer offer;
    // implementation of Offer interface
    ...
    public void setDescription(String description) {
        // start of state access
        controller.enter(B2BAccessMode.OVERWRITE);
        try {
            // set offer description
            offer.setDescription(description);
            // coordinate change
            controller.coordinate();
        } finally {
            // end of state access
            controller.leave();
            // if coordinate failed, controller performs rollback
        }
    }
    ...
    // implementation of B2BObject interface follows
    ...
}
```

Figure 5.18: Extract of B2BOffer implementation

wrapper, B2BOffer, takes the form of the extract shown in Figure 5.18. The Offer methods are wrapped to ensure mediation by the middleware. The setDescription, setCurrency and setPrice methods involve state changes. Invocation of one of these B2BOffer methods results in coordination of the proposed change with members of the sharing group, including their validation of the change.

The controller enter and leave methods, shown in the B2BOffer setDescription implementation, demarcate the scope of access to object state. These calls may be nested as long as there is a matching leave invocation for each invocation of enter. Nesting allows a series of state changes to be “rolled-up” into a single coordination event. As shown, enter is parameterised with an access mode. B2BAccessMode.OVERWRITE indicates that the whole of the object state is to be overwritten and coordinated with remote parties. B2BAccessMode.UPDATE indicates

a partial update to object state. `B2BAccessMode.READ` indicates that access is read-only. The controller coordinate method is called after writing object state to indicate that the state coordination protocol should be executed with remote parties. State coordination only actually takes place when coordinate is called in the context of the outer-most enter/leave block of any nested blocks. To initiate protocol execution, the controller first obtains a copy of object state using the `B2BObject.getState` method. The controller then constructs a new `B2BControlState` — the Java representation of information control state defined in Section 3.3.1. This is passed, with the object state, to the `generateMessage` method of the local protocol handler. The generated message, with object state as payload, is subsequently passed to the `startProtocol` method of the protocol handler to initiate coordination. The result of coordination is recorded by the controller. If, for example, coordination fails because a remote party vetoed a proposed object state change, the proposer's controller will rollback object state when the final leave is called.

Protocol handlers at remote parties validate a proposed change via calls to local validation listeners. If a proposed change is accepted by all parties, the protocol handlers subsequently update both the local object state and the currently agreed control state by invocation of `setState` through the `B2BStateManager` interface. This in turn leads to invocation of `setState` on each `B2BObject` replica to install the new validated state.

A single `B2BObjectController` is statically instantiated for a given uniquely identified `B2BObject`. This guarantees a one-to-one mapping between a `B2BObject` and its controller. Thus, the controller can ensure that local access to its `B2BObject` is thread-safe both with respect to multiple local threads and with respect to remote access during protocol execution, and for the acquisition of a write lock by a remote party. The one-to-one mapping is also essential for transactional access to a `B2BObject` (see Section 5.3.3). Both the `B2BObjectController` and protocol handlers can determine the existence of a lock from the currently agreed control state, which is persisted and accessible to both components of the middleware.

The membership change process is similar to the state change process. The `B2BObjectController` `connect` and `disconnect` methods initiate the different kinds of membership change discussed in Section 3.3.2.4, using the specified membership change protocols.

Object identifiers

Each B2BObject replica is uniquely identified by a URI of the following form:

```
rmi://<hostname>:<port>/nrcoordinator?localObjectName
```

Together, the bold components — URI scheme, authority and path — identify the coordinator service that, from the point of view of remote parties, hosts a partner's B2BObject. An implementation of NRMessenger can use this base part of the URI to bind to a remote coordinator service. The query component of the URI, *localObjectName*, is a local name that identifies a specific B2BObject. The local name is also used to access middleware configuration information, such as: credentials/certificate identifier, non-repudiation log to use, data-binding to use, and any validation and/or event listeners to call during protocol execution. The complete URI is used as the participant identifier and, in particular, as the control state *proposerId* in coordination protocols. It is bound to the entity responsible for a given replica by the credentials used in non-repudiation evidence. The URI is also used by the local protocol handler to identify a controller for the named B2BObject.

The other control state identifiers related to a B2BObject are the *groupId* and *stateId*. The *groupId* is generated by taking a message digest of the current set of participant URIs. The URI of each object replica is added to the message digest in the order of addition to the sharing group. The *stateId* is a message digest of an agreed serialized form of the application object. The serialized form is determined by the agreed representation of application state (see Section 4.2.3).

B2BObjects and reentrancy

The preceding discussion of B2BObjects middleware assumes that it is safe for a B2BObject-Controller to make reentrant invocations on a B2BObject. For example, in the implementation of the *setDescription* method in Figure 5.18, the B2BObject invocation of the controller's *coordinate* method results in the reentrant invocation by the controller of the B2BObject's *getState* method. However, in some contexts it can be unsafe and/or explicitly forbidden to make reentrant calls on objects. For example, the J2EE specification discourages reentrant

calls on EJBs. To support non-reentrant B2BObjects, the B2BObjectController provides alternative versions of its coordinate and rollback methods. Figure 5.19 shows a non-reentrant

```

// non-reentrant B2BOffer setDescription method
public void setDescription(String description) {
    // start of state access
    controller.enter(B2BAccessMode.OVERWRITE);
    try {
        // set offer description
        offer.setDescription(description);
        // coordinate change
        controller.coordinate(getState());
    } catch (Exception e) {
        // change rejected, rollback to
        // controller-provided state
        rollback(controller.rollback());
        ...
    } finally {
        // end of state access
        controller.leave();
    }
}

```

Figure 5.19: Non-reentrant B2BOffer

version of the B2BOffer setDescription method. The offer state is explicitly passed to the controller when coordinate is invoked. If coordination fails, the exception handler code explicitly invokes rollback on the B2BOffer instance with the appropriate rollback state obtained from the controller.

5.3.2 J2EE implementation and application programmer responsibilities

Figure 5.20 illustrates the component-based implementation of B2BObjects when two organisations, A and B, share a B2BObject and A is updating the object state. As in a standard J2EE application, an EJB client makes invocations through an application interface (a session bean) that may result in access and update to an associated entity bean. In this case, the entity bean has been identified as a B2BObject that should be coordinated with remote replicas. An interceptor traps invocations on the entity bean to ensure that a B2BObjectController controls access and update to the bean. The update is only applied to the replicas if B agrees to the proposal. The process is the same for an update proposed by B.

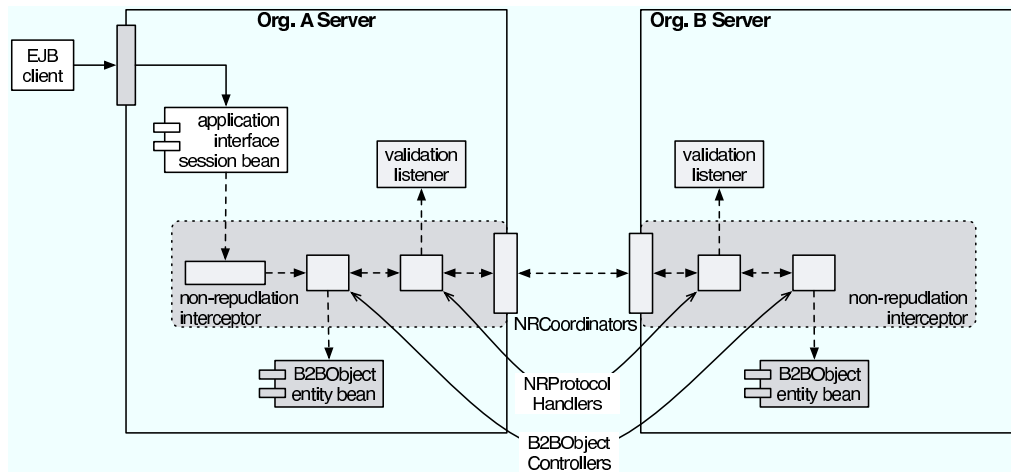


Figure 5.20: JBoss/J2EE-based implementation of non-repudiable information sharing

The middleware-provided non-repudiation interceptor is responsible for interaction with the B2BObjectController, and, through the controller, with the B2BObjects middleware. The application programmer is responsible for: (i) identifying an entity bean as a B2BObject, (ii) providing configuration information in the bean's deployment descriptor, for example, to identify validation listeners, and (iii) providing the validation listener implementations. Optionally, the application programmer may specify that a method in the application interface should result in the "rolling-up" of series of operations on the underlying B2BObject bean into a single coordination event. The enhancement of an entity bean to become a B2BObject is effectively transparent to the local EJB client and its application interface.

The application programmer defines the local entity bean representation of shared state. Get and set methods provide access to the state by co-located business logic session beans. To ensure that local invocations on these accessor methods are mediated by the B2BObjects middleware, the application programmer must do the following.

1. Declare that the entity bean's local interface extends B2BEJBLocalObject, which in turn extends the standard EJBLocalObject interface and the B2BObject interface. Extension of B2BEJBLocalObject provides the middleware with an interface to the entity bean to control access and update to the bean's state. The middleware-provided JBoss interceptor reflects on the local object interface to determine which controller operations should be invoked as a result of invocations on the bean.

2. Declare that the entity bean implementation extends `B2BEntityBean`. The `B2BEntityBean` is a middleware provided implementation of the standard `EntityBean` interface and of the `B2BObject` interface. Middleware invocations on the `B2BEJBLocalObject` interface result in execution of code defined in the `B2BEntityBean` implementation. The middleware automatically generates the `B2BEntityBean` implementation by reflecting on the application programmer defined entity bean.
3. Specify, in the bean's deployment descriptor, an object identifier that is used by the middleware to key configuration information and to construct the URI for remote parties to identify the object as a peer replica for coordination. The implementation classes of any validation listeners and event listeners are also identified in the deployment descriptor.

As shown in Figure 5.16 on page 178, the `B2BObject` interface defines methods that are invoked by the middleware to manage a bean's state during coordination with remote parties. The implementation of these methods is automatically generated for the J2EE version of the middleware. The application programmer simply ensures that an entity bean extends this interface. The interceptor uses the `B2BObjectController` interface of the middleware to wrap application-level access to the information state in appropriate `enter/leave` blocks and thereby effect state coordination, concurrency control etc. This wrapper code is generated automatically and is transparent to the application programmer. For example, given a J2EE version of the Offer object described in Section 5.3.1, the interceptor injects controller calls such as those shown in the non-reentrant implementation of the `B2BOffer setDescription` method in Figure 5.19 on page 182.

Figure 5.21 illustrates the relationship between the various components for an entity bean version of the Offer object. `OfferBeanLocal` defines the business methods (`get/set` accessors) that can be invoked by co-located beans. This interface extends `B2BEJBLocalObject` that in turn extends both the standard J2EE `EJBLocalObject` interface and the middleware-provided `B2BObject` interface. `OfferBean` is the abstract implementation of the offer provided by the application programmer. This extends the middleware-provided `B2BEntityBean` that in turn implements both the standard J2EE `EntityBean` interface and the middleware-provided `B2BObject` interface. The declaration of the relationships shown is sufficient for an entity

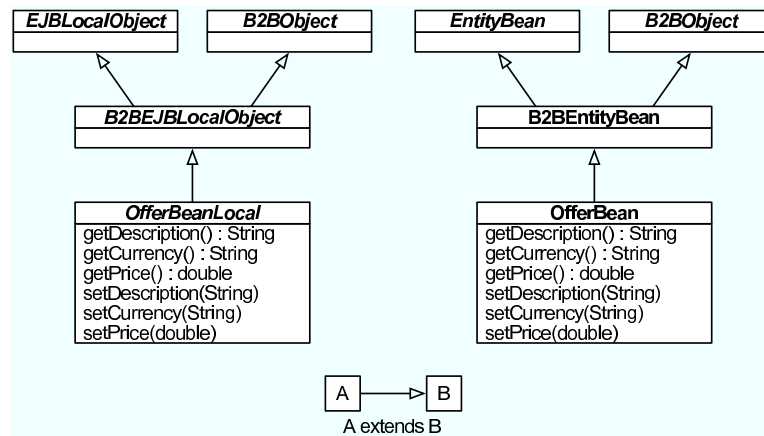


Figure 5.21: Shared Offer entity bean

bean to become a B2BObject that can be coordinated with remote parties. Application programmer responsibility could be reduced further by using a code generation tool such as XDoclet [XDo05]. In this case, the OfferBean class would be annotated to identify it as a B2BEntityBean and all other code and relationships could be generated automatically.

The B2BObjects middleware described thus far realises non-repudiable information sharing either as standard Java objects or as entity beans hosted by J2EE application servers. In fact, the use of an agreed representation of information state allows different business partners to choose their local representation of shared information and still use the B2BObjects middleware to agree updates to the shared information (in its agreed representation). I now address the remaining requirement to enable shared information to participate as a transactional resource in standards-compliant distributed transactions. The description of the support for transactional access is followed by a proof-of-concepts application of non-repudiable information sharing.

5.3.3 Transactional information sharing

This section describes support for transactional information sharing. First, I provide an overview of typical middleware support for distributed transactions. Then I describe an extension to information control state that facilitates transactional access to shared information. This allows one or more B2BObjects to participate as transactional resources in a distributed transaction with other local transactional resources.

Middleware support for transactions

Transactions have long been used to ensure the consistency of local resources despite concurrent accesses and system failures — delivering the well-known ACID properties of Atomicity, Consistency, Isolation and Durability. Figure 5.22 illustrates how transactional update to a set

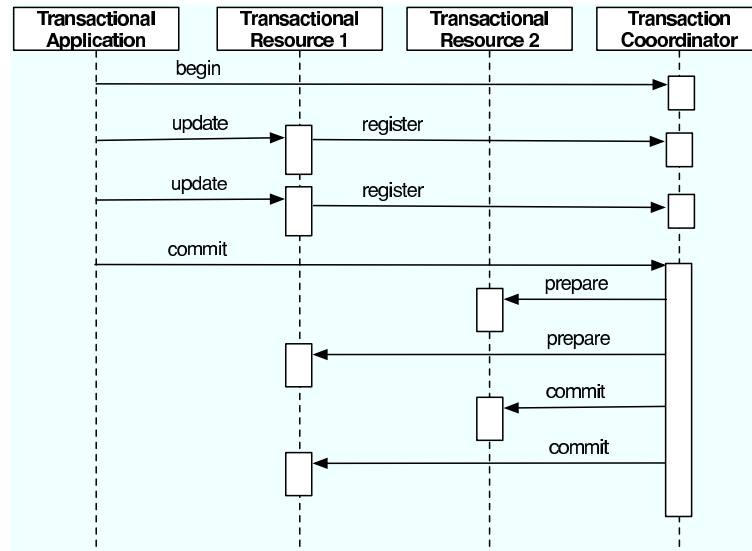


Figure 5.22: Middleware supported distributed transaction

of local resources is supported by commonly used middleware, such as CORBA and J2EE.

There are three basic roles in a distributed transaction:

1. the transactional application (or client) that is responsible for setting the transaction boundaries,
2. the transactional resources or services, such as enterprise databases, that can be updated consistently in the context of a transaction, and
3. the transaction coordinator that coordinates delivery of the ACID properties.

As shown, the client first requests the begin of a new transaction from the coordinator. Then the client updates the resources. The transaction-aware resources register their participation in the transaction with the coordinator. The application indicates the end of the transaction by requesting that the coordinator commit. As a result, the coordinator executes a two-phase commit protocol with the resources. In the prepare phase, each resource votes either to commit the

transaction (if their updates can be made durable) or to abort (if not). If, as shown, all resources vote to commit, the coordinator then invokes `commit` on each resource. On completion of the commit phase, control is returned to the application. If any resource had voted to abort the transaction, the coordinator would have invoked `rollback` on each resource.

Different transactional resources may use different mechanisms to meet transactional requirements and may have different interfaces to those mechanisms. To manage the resulting heterogeneity, the XA standard [Ope91] defines the contract, or interface, between transactional resources and the transaction coordinator in a distributed transaction. The Java Transaction API (JTA) [CM02] is a standard interface to Java-based transaction management that includes a mapping of the XA interface (`XAResource`). Access to enterprise resources is mediated by resource managers that export the `XAResource` interface to a JTA Transaction Manager. In the following I describe how the `B2BObjectController` also exports an `XAResource` interface to present `B2BObjects` as transactional resources to a Transaction Manager. In this way, distributed transactions can be combined with non-repudiable information sharing. First, I describe the necessary extension to shared information control state.

Extending control state to facilitate transaction

To support transactions, we consider shared information to be in either a prepared state or a committed state. Both prepared and committed states are subject to the agreement of the sharing group. The distinction is that a prepared state is revocable. If a prepared state is revoked, then the information returns to the most recently agreed committed state — the *rollback state*. A prepared state may be revoked because a transaction coordinator requests rollback of transactional resources or because of the invalidation of a subsequent state transition. To support the concept of prepared and committed states, I extend information control state to include a *rollbackNRId* that identifies the rollback state. A control state describes committed shared information if the control state *rollbackNRId* is the same as the control state *nrId*. Otherwise the control state describes shared information that is in a prepared state. Table 5.1 shows the five permitted transitions in shared information in terms of a tuple of rollback state and proposed new state. Transition 1 is equivalent to the state coordination described in Section 3.3.2.3

Transition type	Example transition in $\langle \text{rollback state}, \text{new state} \rangle$
1 committed to committed	$\langle S_y, S_y \rangle \rightarrow \langle S_{y+1}, S_{y+1} \rangle$
2 committed to prepared	$\langle S_y, S_y \rangle \rightarrow \langle S_y, S_{y+1} \rangle$
3 prepared to prepared	$\langle S_y, S_{y+1} \rangle \rightarrow \langle S_y, S_{y+2} \rangle$
4 prepared to committed (abort)	$\langle S_y, S_{y+2} \rangle \rightarrow \langle S_y, S_y \rangle$
5 prepared to committed (commit)	$\langle S_y, S_{y+2} \rangle \rightarrow \langle S_{y+2}, S_{y+2} \rangle$

Table 5.1: Permitted state transitions

— transition from the current committed state to a new committed state with no intermediate prepared state. Transitions 2 and 3 to prepared states can be mapped to the prepare phase of a distributed transaction. In these cases, the rollback state must be the same before and after the transition. Transition 4 maps to transaction abort and rollback to the previously committed state: $\langle S_y, S_y \rangle$. Transition 5 maps to transaction commit of the new committed state: $\langle S_{y+2}, S_{y+2} \rangle$.

```

1 // start transaction txId context
2 enter(b2bObjA, txId, B2BAccessMode.READ)
3 enter(b2bObjB, txId, B2BAccessMode.READ)
4 // perform state changes
5 enter(b2bObjA, B2BAccessMode.OVERWRITE)
6 // locally update b2bObjA state
7 // coordinate b2bObjA to prepared state
8 // and acquire lock
9 coordinate(b2bObjA)
10 leave(b2bObjA)
11 enter(b2bObjB, B2BAccessMode.OVERWRITE)
12 // locally update b2bObjB state
13 // coordinate b2bObjB to prepared state
14 // and acquire lock
15 coordinate(b2bObjB)
16 leave(b2bObjB)
17 // Perform further state changes. For each enter/leave
18 // block, object state is coordinated
19 // commit transaction txId
20 // coordinate objects to committed states
21 leave(b2bObjA, txId, TX_SUCCESS)
22 leave(b2bObjB, txId, TX_SUCCESS)

```

Figure 5.23: Pseudo-code for a B2BObjects transaction

Figure 5.23 is a pseudo-code example of a transaction that spans two B2BObjects: b2bObjA and b2bObjB. The enter operations in lines 2 and 3 indicate the start of the transaction. Lines 4 to 18 show state changes to b2bObjA and b2bObjB in nested enter/leave blocks. Each enclosed

coordinate operation, including those implied in lines 17 and 18, results in transition to a prepared state (of type 2 or 3 in Table 5.1). Finally, the leave operations in lines 21 and 22 trigger transition to committed states — transition 5 in Table 5.1.

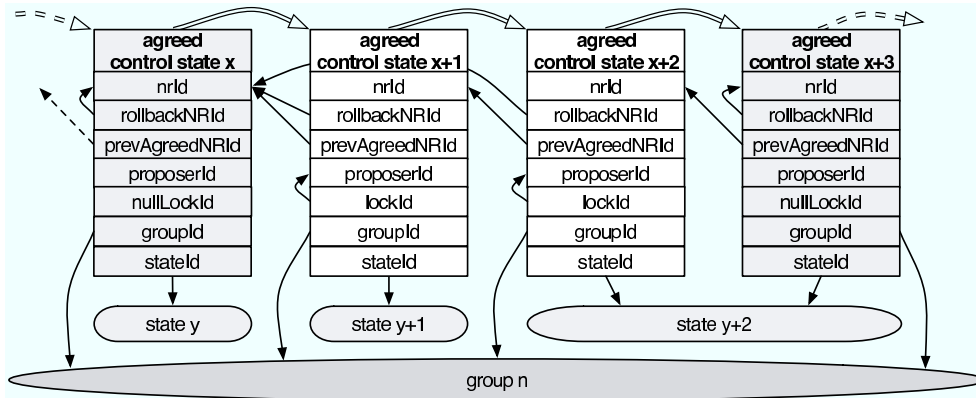


Figure 5.24: Transaction commit control state transitions

Figures 5.24 and 5.25 show how the transitions in Table 5.1 are represented by control state transitions to effect transaction commit and abort. Until the final transition of Figure 5.24, the *rollbackNRId* identifies control state x . Therefore, up to that point, state y is the agreed rollback state. After transition from control state $x+2$ to control state $x+3$, the *rollbackNRId* is the same as the *nrId* and, therefore, the information is in a new committed state with agreed rollback state $y+2$. Control states x and $x+3$, in grey, describe committed states. Control states $x+1$ and $x+2$ describe the prepared states that may be revoked. In Figure 5.25 transition

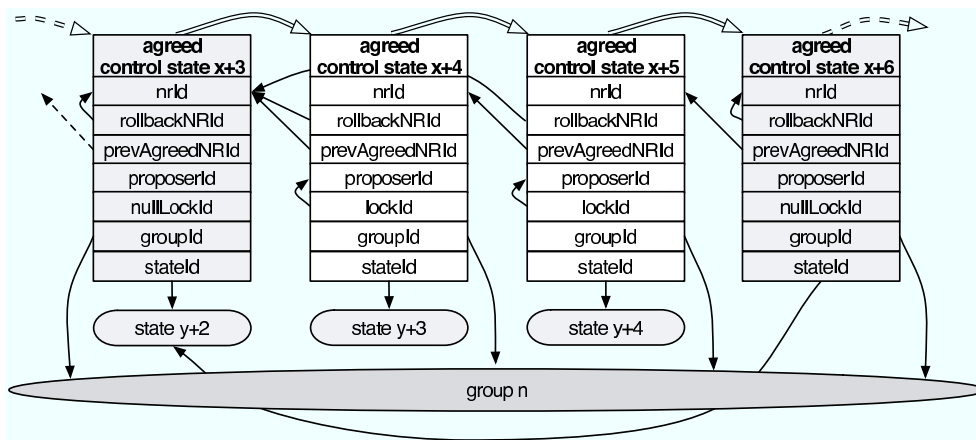


Figure 5.25: Transaction abort control state transitions

from control state $x + 5$ to control state $x + 6$ effects the abort of states $y + 3$ and $y + 4$. The information is then returned to the state identified through the *rollbackNRId* of control state $x + 5$. Control state $x + 6$ is confirmed as the new committed state because its *rollbackNRId* is the same as its *nrId*. Assignment of the *lockId* in control states $x + 1$, $x + 2$, $x + 4$ and $x + 5$ represents proposer lock acquisition in transaction context. The write lock is acquired in the same transition as a proposed state change, by way of contrast to the example in Figure 3.17 on page 127.

The sharing group can reach agreement on any of the transitions shown in Table 5.1 by using the extended control state when executing the state coordination protocol described in Section 3.3.2.3. Agreement to a new prepared state, for example $\langle S_y, S_{y+1} \rangle$, represents: (i) validation of proposed state S_{y+1} ; and (ii) a commitment to be able to subsequently install either of the related committed states: $\langle S_y, S_y \rangle$ or $\langle S_{y+1}, S_{y+1} \rangle$. A transition to one of these committed states does not require application-level validation of the known and agreed information state. Thus, the group can reach agreement on transitions 4 and 5, transaction abort and commit, by executing the state coordination protocol without the information state payload. Non-repudiable coordination from a prepared to a committed state is still required to ensure that all parties maintain the same view of shared information and, therefore, of the new committed state.

B2BObjects as transactional resources

I now show how to modify the B2BObjects middleware to enable the participation of shared information in JTA distributed transactions. The essential requirements are that a JTA transaction manager can control the participation of B2BObjects in transactions through an XAResource interface, and that the B2BObjects middleware can be instrumented to support JTA-compliant transactions. The approach is to combine extended control state transitions to deliver expected transactional behaviour. A transaction-aware controller, `TXB2BObjectController`, both exports an XAResource interface and provides additional methods to manage transactional context. The `TXB2BObjectController` is responsible for mapping requests at the XAResource interface to appropriate control state transitions. The controller guarantees both the

persistence of B2BObject state to facilitate recovery and rollback, and the persistence of transaction state information. For example, it maintains a persistent link between a transaction identifier, provided through the XAResource interface, and the state coordination events associated with the transaction. The one-to-one relationship between a controller and the object it controls ensures that transaction context is correctly maintained for all operations on a given B2BObject. In effect, the TXB2BObjectController is the transactional resource. Figure 5.26

```
public interface TXB2BObjectController
    extends B2BObjectController {
    void enter(Object txId, B2BAccessMode);
    void leave(Object txId, int flag);
    XAResource getXAResource();
}
```

Figure 5.26: TXB2BObjectController interface

shows the TXB2BObjectController interface that extends the B2BObjectController in Figure 5.16. The interface adds new enter and leave methods to demarcate transaction context for the controller; see lines 2, 3, 21 and 22 of Figure 5.23 for the use of these new methods. A getXAResource method provides access to the controller's XAResource interface to allow participation in a JTA-compliant transaction. The XAResource interface defined in [CM02] includes start and end methods to demarcate work on behalf of a given transaction, and prepare, commit and rollback methods for participation in the transaction two-phase commit protocol.

Figure 5.27 is a sequence diagram for update of a B2BObject in the context of a JTA distributed transaction that shows the process from application demarcation of the transaction to successful commit. In a typical distributed transaction other resources, such as enterprise databases, would be enlisted with a transaction in the same way and participate in the two-phase commit protocol via their own XAResource interface. For simplicity, these other resources are not shown. First the application indicates the beginning of a transaction to the TransactionManager and obtains a Transaction object from the manager. Then the application obtains the TXB2BObjectController's XAResource interface and enlists it with the Transaction. This results in the propagation of transaction context to the controller via the parameterised enter

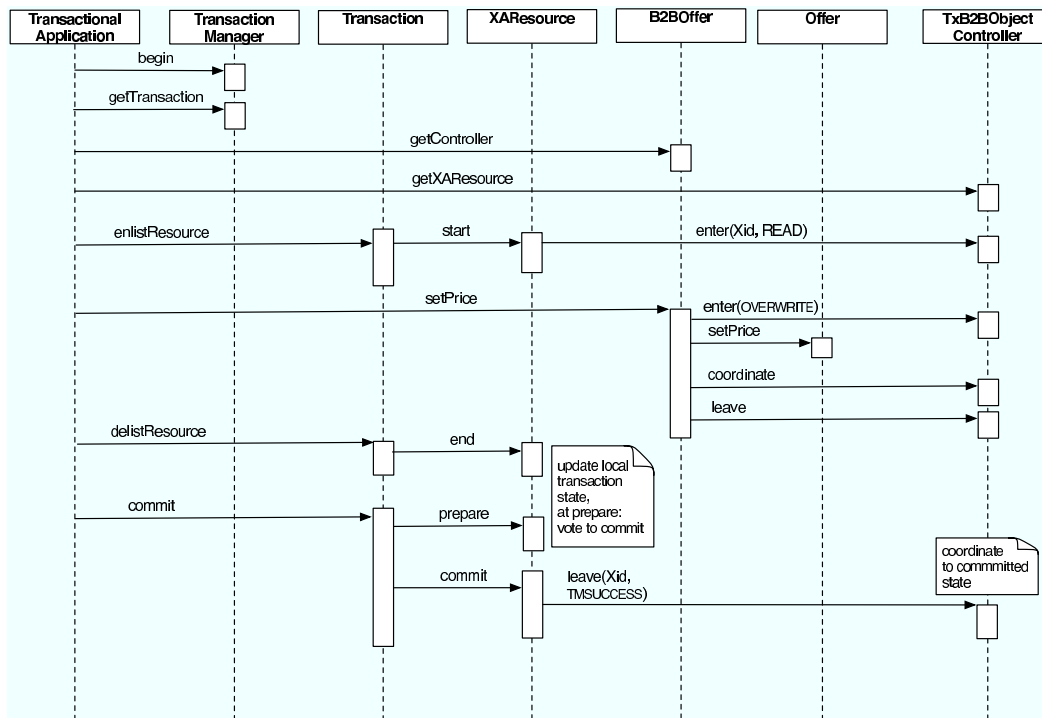


Figure 5.27: B2BOffer transaction sequence diagram

call. The application-level `setPrice` is invoked on the `Offer` interface of the `B2BOffer`. This leads to invocation of `setPrice` on the underlying `Offer` implementation in the context of an `enter/leave` block. Since the controller is aware that this `enter/leave` block is within the context of a transaction, execution of `coordinate` leads to coordination of the `B2BOffer` to a prepared, and locked, state. The controller records the success or failure of coordination. In the example, the call succeeds and the `prepare` call to the `XAResource` results in a vote to commit the changes. The `commit` call on the `XAResource` leads to a parameterised `leave` call on the controller that in turn leads to coordination of the `B2BOffer` to a committed, and unlocked, state. As shown, `XAResource` calls are mapped to operations on the `TXB2BObjectController`. It should be noted that for J2EE applications, the preceding application-level operations for transaction demarcation etc. are performed at the container-level and not directly by application clients.

Read and write locks are acquired and released in the way described in Section 3.3.3. In transaction context, a read lock is granted locally by the controller. For example, in Figure 5.27,

B2BAccessMode READ is used in the first enter call and acquisition of a write lock is deferred until the first coordination to a prepared state in transactional context. If B2BAccessMode OVERWRITE or UPDATE were used instead, a write lock would be acquired at the first enter call. The middleware can be configured locally to use either sequence of lock acquisition. Locks are released during coordination to a committed state, at transaction completion.

Deferred coordination

It is possible to configure the middleware to execute either immediate or deferred coordination. Immediate coordination is as described previously — each update to an object in the context of a transaction results in coordination with remote replicas. In this case, invalidity with respect to remote parties is detected early; with consequent rollback of the transaction. Deferred coordination is an optimisation in which updates to an object are performed locally and coordination with remote replicas is deferred to a single coordination event during the prepare phase of the transaction.

The controller's support for nested enter/leave blocks is used to implement deferred coordination. There is an additional enter invocation on the controller as a result of the start invocation on the XAResource. On completion of local state updates, corresponding coordinate and leave calls are then made as a result of the invocation of prepare on the XAResource. This triggers coordination to a prepared state that encompasses all of the intervening updates. Figure 5.28 shows how deferred coordination modifies the pseudo-code presented in Figure 5.23. For *objS*, the single transition $\langle s_i, s_i \rangle \rightarrow \langle s_i, s_{i+m} \rangle$ replaces coordination through a series of intermediate prepared states described.

Deferred coordination can also optimise object locking since, until the prepare phase, it is sufficient to veto remotely-initiated update of object state by acquiring a read lock with respect to remote replicas. Then a write lock is acquired as part of the coordination to prepared state. Deferred coordination results in less interaction with remote parties at the expense of delayed validation of state changes. An advantage is the confinement of local failure during a series of updates to a B2BObject or related resources. Deferring coordination means that local failures will preclude coordination with remote parties.

```

// start transaction txId context
enter(objS, txId, B2BAccessMode.READ)
enter(objT, txId, B2BAccessMode.READ)
  // defer coordination
  enter(objS, B2BAccessMode.READ)
  enter(objT, B2BAccessMode.READ)
    // perform state changes
    enter(objS, B2BAccessMode.OVERWRITE)
    // locally update objS state
    // but do not coordinate
    leave(objS)
    enter(objT, B2BAccessMode.OVERWRITE)
    // locally update objT state
    // but do not coordinate
    leave(objT)
    // Perform further state changes without
    // coordination
    // locally objS is in state:  $\langle s_i, s_{i+m} \rangle$ 
    // and objT is in state:  $\langle t_j, t_{j+n} \rangle$ 
    // prepare (and coordinate to prepared states)
    coordinate(objS)
    leave(objS)
    coordinate(objT)
    leave(objT)
  // commit transaction txId
  leave(objS, txId, TX_SUCCESS) // coordinate objS to
                                // committed state:  $\langle s_{i+m}, s_{i+m} \rangle$ 
  leave(objT, txId, TX_SUCCESS) // coordinate objT to
                                // committed state:  $\langle t_{j+n}, t_{j+n} \rangle$ 

```

Figure 5.28: Pseudo-code for a transaction with deferred coordination

Majority voting during commit phase

The requirement at the transaction commit phase is to ensure that the proposer does not attempt to issue a commit to some parties and abort to others. As noted previously, transition from a prepared to a committed state does not require further application-level validation because the relevant application state has already been subject to validation by all parties. During transaction commit, or abort, it is therefore possible for the proposer to short-circuit the response phase of the state coordination protocol after receipt of a majority of responses. The proposer need only show that a majority of parties have seen the signed transition to the new committed state. The proposer then completes the protocol by propagating to all parties the evidence that a majority of parties have seen and signed the new control state.

5.3.4 Validated negotiation of a tender to supply chemicals

This section demonstrates the use of non-repudiable information sharing in the simple tender negotiation application described in Section 1.1.2. The application involves the negotiation of a tender for the supply of chemicals. The negotiation process and the tender contract that governs the process are as described in Section 1.1.2. A supplier and purchaser share offers that represent the terms and conditions for the supply of chemicals. Negotiation proceeds through the non-repudiable validation of each other's updates to the state of an offer. I now describe the use of the B2BObjects middleware to support the negotiation process. Section 5.3.4.1 describes application set-up and the representation of an offer as a B2BObject replica at supplier and purchaser. Section 5.3.4.2 demonstrates the use of the information sharing middleware to validate proposed updates to an offer and to generate non-repudiation evidence of the negotiation. Section 5.3.4.3 evaluates the demonstration.

5.3.4.1 Tender application set-up

Figure 5.29 shows the augmentation of the application from Section 1.1.2 with non-repudiable

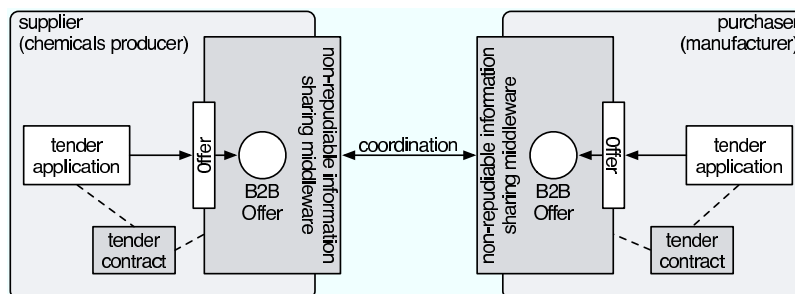


Figure 5.29: B2BOffer tender application

information sharing middleware. Each party has a local tender application and a tender contract implemented as a validation listener that encodes the rules specified in Section 1.1.2. Each tender application accesses a B2BOffer replica through its Offer interface. The non-repudiable information sharing middleware coordinates updates to the B2BOffer replicas. Users at each party interact with their local tender application through a Web browser interface that allows the user to create, edit and save offers. When a user saves an offer, the state of the B2BOffer is coordinated with, and validated by, the remote peer. If the proposed update is valid with

respect to the tender contract, then the replicas will be updated to reflect the new offer state. If the proposed update is invalid, then the replicas will remain in the previously agreed offer state. Figure 5.30 shows the read and write methods of the application Offer interface. Any local

<i>Offer read methods</i>	<i>Offer write methods</i>
getContractStart : Date	setContractStart(Date)
getContractEnd : Date	setContractEnd(Date)
getId : int	setId(int)
getSupplier : URI	setSupplier(URI)
getPurchaser() : URI	setPurchaser(URI)
getDescription : String	setDescription(String)
getUnit : String	setUnit(String)
getCurrency : String	setCurrency(String)
getPrice : double	setPrice(double)
getQuantity : double	setQuantity(double)
getExpiryDate : Date	setExpiryDate(Date)
getAcceptanceStatus : Acceptance Status	setAcceptanceStatus(AcceptanceStatus)
	setOffer(Offer)

Figure 5.30: Application Offer interface

application invocation of the write (set) methods at the Offer interface triggers coordination of the B2BOffer with the remote party. The `setOffer` method allows more than one attribute of an offer to be updated at a time. If a supplier complies with the tender contract, then they should set all attributes except the `AcceptanceStatus`. If a purchaser complies with the tender contract, then they should only set the `AcceptanceStatus`. The `AcceptanceStatus` is an enumeration with one of the following values: `UNKNOWN` (the initial status), `ACCEPT` or `REJECT`.

5.3.4.2 Demonstration of tender negotiation

I now show both a valid update to a shared offer and an attempt at an invalid change. In the first phase of negotiation, the supplier (ChemicalsRUs) edits a new offer — see Figure 5.31a. Note that the offer price for a kilogram of methane is set to 0.75 Euro. When the “Save” button is pressed, the ChemicalsRUs user is given the opportunity to confirm or cancel the update to the offer — see Figure 5.31b. The application presents the agreed string representation of the offer for confirmation. It is this representation of the state of an offer that is signed and coordinated with the purchaser (ChemProc Ltd). When the ChemicalsRUs user presses the “Confirm” button their local tender application invokes the `setOffer` method on the local B2BOffer replica, through the Offer interface. This results in state coordination with the B2BOffer replica at ChemProc Ltd. During state coordination, the update is validated with respect to the tender

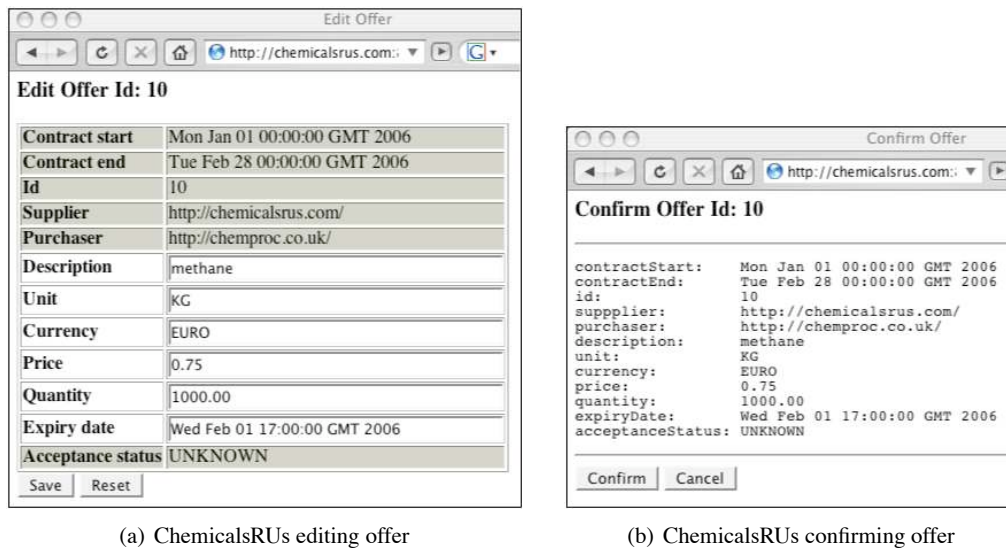


Figure 5.31: ChemicalsRUs editing and saving offer

contract at ChemProc Ltd. Since the proposed change is valid, the B2BOffer replicas at both ChemicalsRUs and ChemProc Ltd are updated to the state shown in Figure 5.31b.

In the second phase of negotiation, the ChemProc Ltd user views the current state of the offer. As can be seen in in Figure 5.32a, their local view of offer state is the same as for the

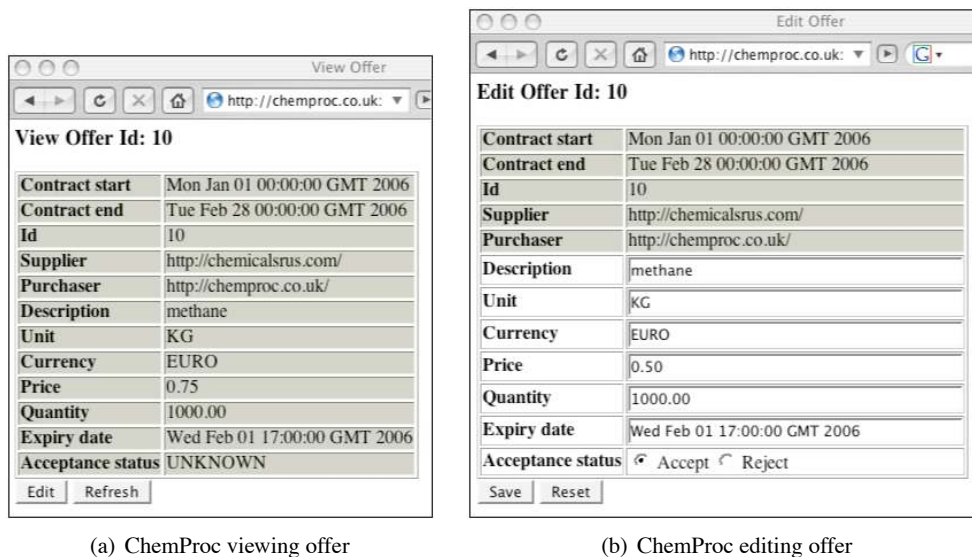


Figure 5.32: ChemProc Ltd viewing and editing offer

ChemicalsRUs user in Figure 5.31b. The ChemProc Ltd user attempts to cheat by both editing

the price of the offer and setting its `AcceptanceStatus` to `ACCEPT` — see Figure 5.32b. Note the change of the price of the offer to 0.50 Euro for a kilogram of methane. When the ChemProc Ltd user decides to commit their proposed changes, the middleware triggers state coordination between the B2BOffer replicas. However, this time the proposed change is found to be invalid with respect to the tender contract at ChemicalsRUs and the change is rejected. Figure 5.33



Figure 5.33: Invalid proposed update from ChemProc Ltd

shows notification of the alleged contract violation to the ChemProc Ltd application. The exception provides a reference to the protocol run to allow retrieval of related messages from the local non-repudiation log. The ChemicalsRUs application can also register an event listener to be notified of the violation and, therefore, to react to it. The violation is “alleged” because ChemProc Ltd may wish to contest the allegation of misbehaviour.

At the end of the example, ChemicalsRUs and ChemProc Ltd each have non-repudiable evidence of the creation of a valid offer by ChemicalsRUs. Similarly, they have non-repudiable evidence of the invalidation of ChemProc Ltd’s proposed update, and of the reason for the contract violation. Therefore, offer 10 in the given contract period remains in the agreed state shown in Figures 5.31b and 5.32a. Since the contract states that each party can only edit a uniquely identified offer once, there can be no further valid changes to the offer and it will expire on the date set by ChemicalsRUs.

5.3.4.3 Evaluation of demonstration

The tender negotiation application successfully demonstrates non-repudiable, validated update to shared information. From the point of view of a local tender application, access the shared offer is indistinguishable from access to local application state. The distinction is that the infor-

mation sharing middleware coordinates actions on the local instance of the offer with remote peers. This coordination is achieved by an initialising execution of the connection protocol defined in Section 3.3.2.4 that establishes sharing. This is followed by an execution of the state coordination protocol defined in Section 3.3.2.3 for the supplier's successful update to the offer and then another execution of the state coordination protocol that detects, and prevents, the purchaser's attempt at an invalid update to the offer. It is straightforward to add new parties to the interaction, such as the fulfillment agent suggested in the application description in Section 1.1.2. Similarly, it is straightforward to support an extended interaction for negotiation of more complex tender terms and conditions than those specified in the example offer. Though the resulting application would be more complex, from the middleware perspective such changes are simply a matter of further executions of the information sharing coordination protocols. As noted at the end of Section 5.2, the use of the same protocol execution framework to implement services for both information sharing and service invocation demonstrates the flexibility of the non-repudiation middleware.

5.4 Summary

I have now demonstrated a unified approach to the design and implementation of middleware for both non-repudiable service invocation and non-repudiable information sharing. The middleware implementation addresses all of the requirements identified in Section 1.2, namely: non-repudiation, no unfair advantage, application-specific validation, flexibility and transactional access to shared information. Key characteristics of the middleware are its basis in fundamental work on non-repudiation, the fact that it addresses both of the domains for action identified in Chapter 1 (the private and the shared), and its application-independence. No other non-repudiation middleware shares this collection of characteristics. The FIDES project is the only other system found that is based on fundamental work on non-repudiation and fairness. However, their work is specific to an application for document exchange and cannot be generalised to support other applications. None of the other middleware systems identified in Section 2.4, such as Wichert et al's CORBA non-repudiation service or BEA's Trading Partner Integration Engine, take account of the fundamental work nor do they exhibit the flexibility

of the middleware presented in this chapter. No other work addresses the complete set of requirements specified in Section 1.2 and in particular the need for non-repudiable information sharing. I now conclude this dissertation with a review of the contributions made and suggestions for future work.

Chapter 6

Summary of contributions and future work

In this chapter I summarise the contributions of my work and then conclude the dissertation with suggestions for future work. Chapter 1 identified two domains for action: one over private resources that are exposed as services to business partners and the other over shared resources that are jointly owned by business partners engaged in some collaborative venture. There is then a need for systematic support to regulate (monitor and control) B2B interactions in each domain. To this end I proposed two middleware services for non-repudiation and validation of B2B interactions — non-repudiable service invocation and non-repudiable information sharing that address requirements in the private and shared domains, respectively. The core of the work in this dissertation comprises:

1. the definition in Chapter 3 of the non-repudiation services based on fundamental work on protocols for the exchange of non-repudiation evidence,
2. the design and implementation of a generic framework for protocol execution in Chapter 4, and
3. the demonstration in Chapter 5 of the practical implementation of the non-repudiation services using the protocol execution framework.

Taken together, this work addresses the requirements for middleware support specified in Section 1.2. In the following, I summarise the support provided with respect to each of the requirements.

Non-repudiation Both services provide systematic generation, exchange and recording of evidence of B2B interactions at run-time. The evidence cryptographically binds actors to their actions. Thus, business partners can be held to account for their actions, which they cannot subsequently deny.

No unfair advantage For service invocation, fairness is addressed by the guaranteed receipt of authenticated service requests, responses and validation information. So, the middleware ensures that well-behaved parties obtain the information to which they are entitled. Fairness guarantees are based on fundamental work on fair exchange discussed in Section 2.2. Section 3.2 shows how to extend protocols to correlate request and response phases of service invocation, and how to integrate the application-specific validation of each phase, and at the same time preserve fairness guarantees. For information sharing, the guarantee of no unfair advantage relates to the integrity of the shared information. That is, it must not be possible to deny the validity of a well-behaved party's view of the agreed state of shared information and of the membership of group that shares the information. The coordination protocols defined in Section 3.3, and demonstrated in Section 5.3, guarantee this consistent, non-repudiable view of shared information.

Application-specific validation The requirement was to integrate application-specific validation of actions with the generation of non-repudiation evidence at run-time. The service definitions in Chapter 3 addresses the integration of application-specific validation at the non-repudiation protocol level. The middleware framework in Chapter 4 provides the validation listener mechanism to invoke potentially arbitrary application-specific validation of an interaction. Chapter 5 demonstrates the use of this mechanism during protocol execution for both service invocation and information sharing.

Flexibility The middleware demonstrated in Chapter 5 is application-independent. It does not require any change to application business logic and it preserves application-level semantics. There are a number of aspects to the flexibility of the middleware. It can adapt to different requirements for application-specific validation. The underlying mechanisms to achieve non-repudiation can be tailored to a given application context and to different

relationships between participants. It is possible to use different local representations of application data and, yet, specify agreed representations for evidence generation and information sharing. The specification of a support service API allows organisational autonomy with respect to the implementation, deployment and use of supporting infrastructure.

Transactional information sharing Section 5.3.3 shows how to combine the work on information sharing protocols in Section 3.3 with standards-compliant middleware support for distributed transactions in order to provide transactional access to shared information.

A notable feature of my approach is a careful separation of concerns. Figure 6.1 recalls the

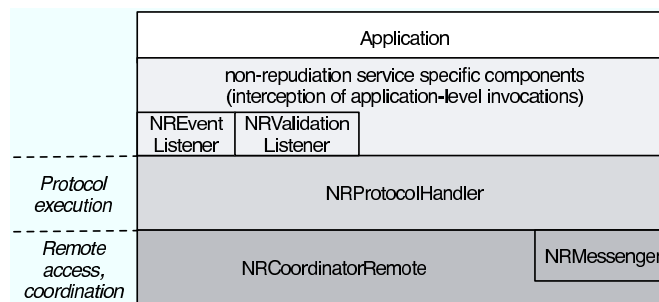


Figure 6.1: Generic protocol execution framework

protocol execution framework from Chapter 4. Application-level abstractions are maintained by the realisation of the non-repudiation services as middleware. The application-independent middleware itself is separated into three layers: a protocol-independent layer to address non-repudiation service specific issues, a layer for deployment of protocol implementations and a protocol-independent layer for B2B communications. This separation is fundamental to the flexibility of the middleware. Mechanisms to provide non-repudiation and validation of interactions are independent of the applications to which they apply and yet can be adapted to a specific application context. Protocol-specifics do not leak into the abstractions provided by the non-repudiation services nor into the communications interface between organisations. Thus, different protocols can be deployed to suit different requirements. The separation of concerns is expressed in the middleware through the specification of APIs for interaction with and between the different middleware components. So, there is a well-defined interface between

each layer of Figure 6.1, including the specification of self-describing protocol messages for processing by the middleware. A support service API isolates implementation detail of the necessary supporting infrastructure. A data-binding API supports the distinction between the local representation of data and its agreed representation in non-repudiation evidence and, in particular, as shared information. The definition of an API for validation and event listeners allows the application programmer to adapt the middleware to application-specific requirements and to customise the rules that are applied in different interaction contexts.

I have demonstrated the novelty of the design and implementation of the non-repudiation middleware with respect to the related work discussed in Chapter 2. Novel contributions include: (i) the development of flexible middleware that is based on fundamental work on non-repudiation and fairness, (ii) the provision of systematic support for regulation that is adaptable to different application contexts, and (iii) the specification of the two domains for action and the definition and implementation of a service for non-repudiable information sharing in order to regulate actions in the shared domain. In summary, as proposed in Chapter 1, through a unified approach to the development of middleware support for non-repudiation and validation of B2B interactions, I have established that:

1. a set of middleware services can provide a flexible framework for protocol-based interaction,
2. that the framework can deliver non-repudiation services that are appropriate to the two domains for action — the private and the shared, and
3. that the services can be used to regulate B2B interactions and at the same time preserve application-level semantics.

I now conclude this dissertation with a discussion for future work that continues the theme of exploring fundamental concepts in the context of realistic systems implementation.

Future work

It is increasingly common to view B2B interactions in terms of the exchange of business messages between loosely-coupled services (or business processes). This has led to the devel-

opment of open standards for business conversations. For example, RosettaNet PIPs [Ros05] define the externally observable aspects of a B2B interaction. They standardise the XML-based business messages that should be exchanged between partners to execute some function, such as order processing. The work I propose in this section concerns support to regulate interactions of this kind.

In [RCS05, CRS06] we present some preliminary work on a Web services based re-implementation of the framework for non-repudiation protocol execution. This will be further developed and will provide an experimental platform for investigations in other areas. Work that I envisage includes: (i) the composition of message delivery primitives into message exchange patterns, (ii) the application of non-repudiable information sharing in loosely-coupled interactions, (iii) the automatic derivation of protocol implementations, (iv) the performance analysis and characterisation of non-repudiation protocols to inform the choice of mechanism to use, and (v) the use of non-repudiation services to support the recording of provenance information.

The Web services protocol execution framework (WS-NRExchange) is based on: (i) a well-defined, generic Web services interface for the exchange of protocol messages, (ii) an extensible XML schema that defines the content of self-describing protocol messages, and (iii) a well-defined API for message processing by the middleware that includes the registration of protocol-specific message handlers. Figure 6.2 shows the interactions between the main components and services that constitute the implementation. The use of an in-line TTP is illustrated. However, as with the framework described in Chapter 4, WS-NRExchange is not restricted to interaction through an in-line TTP. Each protocol participant has an NRExchange Web service that manages their participation in non-repudiation protocols. At end-users — A and B in Figure 6.2 — the NRExchange service is deployed as an interceptor to mediate Web service interactions that require non-repudiation. This interceptor may be co-located with the local application that uses it, or, for example, may be deployed as part of a corporate firewall service. A and B may 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 applies signatures to the parts of messages that have not already been signed. This service may be an implementation of the Digital Signature Service (DSS)

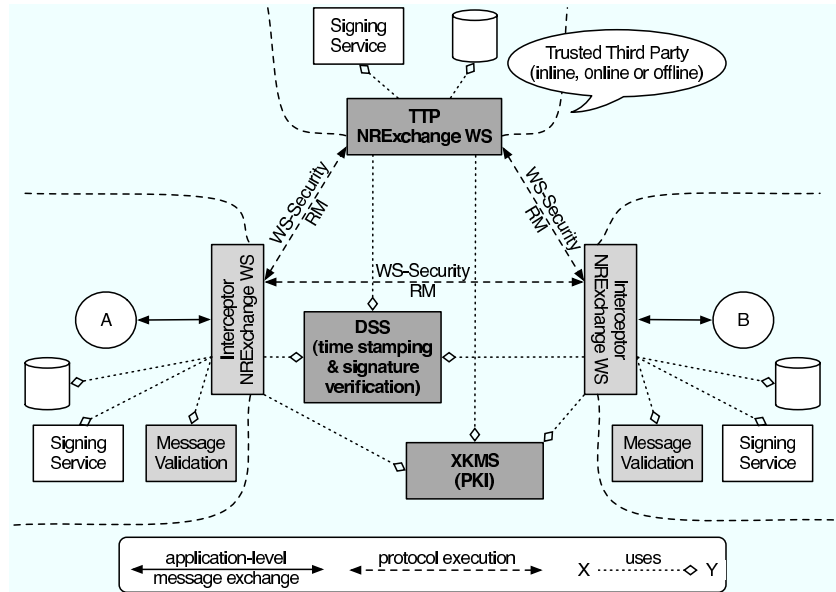


Figure 6.2: WS-NRExchange architecture

[PAC⁺04] or some other mechanism for obtaining private keys to apply signatures to SOAP messages to comply with the WS-Security standard [NKHBM04]. The NRExchange services also access trusted time-stamping services and public key management services, such as DSS and XML Key Management (XKMS) [HBM05] services provided by third parties. For protocols that use an in-line TTP, trusted time-stamps may optionally be applied by the TTP Web service. There is also a local interface to register application-specific listeners for message validation and for protocol-related events. Messages that are found to be invalid with respect to contract are logged but are not passed to the target application for processing.

The WS-NRExchange middleware currently provides a primitive for fair, non-repudiable and validated one-way message delivery. To support more complex interactions, and different application-level semantics, the middleware must be developed to allow the composition of message delivery primitives into message exchange patterns (MEPs). Figure 6.3 illustrates this composition of MEPs. At the lowest level, message delivery is achieved through protocol execution. Three message delivery primitives will be built on the protocol execution layer: (i) send without non-repudiation, (ii) send with NRO, and (iii) fair send with guaranteed exchange of NRO for NRR. Generic MEPs can then be composed from these primitives. An example is correlated sends to achieve request/response application semantics. Generic MEPs can also

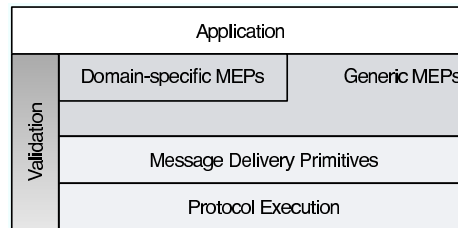


Figure 6.3: Message exchange pattern composition

be composed into domain-specific MEPs. The provision of domain-specific MEPs that, for example, execute PIPs and compositions of PIPs will allow us to regulate such interactions. As shown in Figure 6.3, message validation may be invoked at each level — providing a chain of responsibility for validation that is appropriate to the given level.

The exchange of business messages in conversations such as PIPs leads to each party to an interaction having a view of the state of the interaction that is derived from both the messages they have sent and the messages they have received. A problem, common to any distributed system, is that this view of the state of the interaction can become inconsistent; because, for example, messages may be delayed or lost. Therefore, in addition to regulating the message exchanges, it would be useful to provide mechanisms to synchronise the state of a business interaction as viewed by its various participants. This leads to another avenue for investigation: the application of non-repudiable information sharing to the synchronisation of interaction state. For this, we need to identify synchronisation points in a business interaction and then apply mechanisms similar to those described in Section 3.3 to provide an agreed view of the interaction state at the given synchronisation point.

We have implemented a version of the modified Coffey-Saidha protocol to provide the single WS-NRExchange message delivery primitive. As for the middleware presented in this dissertation, we intend to implement additional protocols with different characteristics. There are two significant problems with protocol implementation. The first is how to have confidence that a protocol design is correct with respect to stated assumptions. The second is how to have confidence that the corresponding implementation executes the protocol that the designer intended. The considerable body of work on the formal verification of security protocols addresses the former problem (see [Low97, SM00], for example). To address the latter problem,

as with any software, we require tool support to generate an implementation from its formally verified specification. We have begun work that, in principle, shows how to automate the generation of protocol implementations that are suitable for deployment in the WS-NRExchange framework. We are confident that this work applies to TTP-based deterministic fair exchange protocols. The approach needs to be demonstrated in practice. It also needs to be extended to support other protocol families. For example, it is not clear how to apply this work to the small set of deterministic protocols that set explicit deadlines for message delivery, as opposed to participants setting local timeouts and then pro-actively seeking termination of a protocol run. Work is also required on the implementation of probabilistic protocols that do not require a TTP or that construct a virtual TTP from tamper-resistant security modules hosted by the protocol participants (see Section 2.2.2.3). In principle, implementations could be deployed in the middleware framework. However, it is not clear whether we can apply our proposed techniques for the auto-derivation of deterministic protocol implementations to these protocols. Work is also needed on the implementation of fault-tolerant fair exchange as described in Ezhilchelvan and Shrivastava [ES05].

An important question when deciding how to regulate B2B interactions is: "are the proposed mechanisms proportionate to the value of the transaction(s) being secured?". There has been work on the relative computational costs of cryptographic algorithms. There has also been some work on cryptography-related message processing costs in the context of Web services (notably [LPF05]). But this work can only inform the choice of algorithms to use for signing and encryption. To make informed choices, for example, about non-repudiation protocols to adopt (if any), we need to be able to answer questions such as "what are the characteristics and associated costs of each candidate protocol?" and "where should the costs fall? Predominantly at guarantor TTPs or at the end users?". Protocols are typically characterised by the level of involvement of guarantor TTPs and the number of message rounds. As with cryptographic algorithms, and as illustrated by the discussion in Section 3.5, this can only provide a partial answer. We require a systematic classification of protocols by a range of characteristics that contribute to the execution overhead as well to trade-offs between the costs and the benefits that accrue. Characteristics that contribute to performance include message size, message

complexity, number of message rounds, applicable cryptographic schemes, and persistence requirements. The contribution of these characteristics to the overall performance overhead needs to be quantified. We also need to compare protocols by their security guarantees, the responsibilities of participants — where the costs fall, and by TTP involvement (including CA and TSA involvement). An ambitious goal is that a comprehensive study of the performance of protocols could lead to a generally applicable approach to their classification. This would allow the straightforward incorporation of new protocols as they are developed. The method may then apply to other areas of security.

Recalling the discussion in Section 2.4.1.3, the PASOA project analysed use cases in bioinformatics, particle physics, chemistry and software engineering to capture provenance service requirements [MGBM06]. They concentrated on generic, re-usable aspects with a view to deriving a domain-independent software architecture for recording, querying and processing provenance data. Their approach is of particular interest because, as far as possible and in contrast to many others, they explicitly aim for application-independence. This corresponds to the systematic recording of non-repudiation evidence, and of the validation of actions to which the evidence relates. They argue for a separation of concerns between the recording of and access to provenance data, the querying of provenance data, and the processing of provenance data (for example, to present to the user or to re-enact experiments). They suggest that recording should be based on the documentation of the processes, or interactions, that produce the data. The non-repudiation services I provide can be seen as performing this documentation of interactions. When both non-repudiation and provenance recording are requirements, it seems reasonable to suggest that non-repudiation logs could be the source of the provenance information. A question to explore is whether there are additional provenance requirements that could be addressed through this use of non-repudiation services as the recording mechanism. For example, non-repudiation protocol messages contain information about the identity of participants, their credentials, message time-stamps, and message validity. Higher-level functions for querying and processing may be able to exploit this information. In existing provenance services, information of this type is often explicitly added to messages as meta-information. An issue to address here is that information in non-repudiation logs is to some extent protocol-

specific. The information to answer a given provenance query may be scattered across different protocol messages because a single application-level message is represented by two or more protocol messages in a non-repudiation log. Another area for investigation, of particular relevance to e-Science, is how to perform efficient non-repudiation with large data-sets or streams of data, or with references to such information. In any case, the use of non-repudiation services to support provenance services has the potential to avoid duplication of effort and provide useful additional interaction information.

References

- [ABCS05] Ross Anderson, Mike Bond, Jolyon Clulow, and Sergei Skorobogatov. Cryptographic Processors - a Survey. *Proceedings of the IEEE: Special Issue on Cryptography and Security (to appear, also available as: University of Cambridge Tech. Rep. CL-TR-641)*, 2005.
- [AGGV05] Gildas Avoine, Felix Gärtner, Rachid Guerraoui, and Marko Vukolic. Gracefully Degrading Fair Exchange with Security Modules. In *Proc. 5th European Dependable Computing Conf. (EDCC)*, Springer LNCS 3463, Budapest, Hungary, 2005.
- [ALP⁺00] Ameneh Alireza, Ulrich Lang, Marios Padelis, Rudolf Schreiner, and Markus Schumacher. The Challenges of CORBA Security. In *Proc. Workshop Sicherheit in Mediendaten, Gesellschaft für Informatik (GI)*, Berlin, Germany, 2000.
- [AM03] Xuhui Ao and Naftaly Minsky. Flexible Regulation of Distributed Coalitions. In *Proc. 8th European Symp. on Research in Computer Security (ESORICS)*, Gjøvik, Norway, 2003.
- [Apa05] Apache Software Foundation. *AXIS Web Services*. <http://ws.apache.org/axis/>, 2005.
- [Aso98] N. Asokan. Fairness in Electronic Commerce. Research Report RZ3027, IBM Zurich Research Lab, 1998.
- [ASW98] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Proc. IEEE Symp. on Research in Security and Privacy*, pages 86–99, Los Alamitos, CA, USA, 1998.
- [Axe90] Robert Axelrod. *The Evolution of Co-operation*. Penguin Books, 1990.
- [BCDv88] Ernest F. Brickell, David Chaum, Ivan B. Damgard, and Jeroen van de Graaf. Gradual and Verifiable Release of a Secret. In *Proc. Advances in Cryptology - Crypto'87*, Springer LNCS 293, pages 156–166, Santa Barbara, CA, USA, 1988.
- [BEA05] BEA. *WebLogic 8.1 Trading Partner Integration*. BEA, <http://e-docs.bea.com/wli/docs81/pdf/tpintro.pdf>, 2005.
- [BS04] Werner Beckmann and Santosh K. Shrivastava. TAPAS QoS-aware Platform: technology and demonstrations. Project Deliverable D15, TAPAS EU Project IST-2001-34069, School of Computing Science, University of Newcastle, 2004.

- [CB02] Richard Clayton and Mike Bond. Experience Using a Low-Cost FPGA Design to Crack DES Keys. In *Proc. Workshop on Cryptographic Hardware and Embedded Sys.*, San Francisco Bay (Redwood City), CA, USA, 2002. Springer.
- [CD04] Jan Cederquist and Muhammad Torabi Dashti. Formal Analysis of a Fair Payment Protocol. In *Proc. IFIP World Comp. Congress Workshop on Formal Aspects in Security and Trust (FAST)*, Toulouse, France, 2004.
- [CM02] Susan Cheung and Vlada Matena. *Java Transaction API (JTA version 1.0.1B)*. Sun Microsystems Inc., <http://java.sun.com/products/jta/index.html>, 2002.
- [Com05] European Commission. REACH: Registration, Evaluation, Authorisation and Restriction of Chemicals. Technical report, European Commission, http://ec.europa.eu/enterprise/reach/index_en.htm, 2005.
- [CRS04a] Nick Cook, Paul Robinson, and Santosh K. Shrivastava. Component Middleware for Trusted Coordination. Project Deliverable D9, TAPAS EU Project IST-2001-34069, School of Computing Science, University of Newcastle, 2004.
- [CRS04b] Nick Cook, Paul Robinson, and Santosh K. Shrivastava. Component Middleware to Support Non-repudiable Service Interactions. In *Proc. IEEE Int. Conf. on Dependable Syst. and Networks (DSN)*, Florence, Italy, 2004.
- [CRS06] Nick Cook, Paul Robinson, and Santosh K. Shrivastava. Design and Implementation of Web Services Middleware to Support Fair Non-repudiable Interactions. *Int. J. of Cooperative Information Systems (IJCIS) Special Issue on Enterprise Distributed Computing (Invited Paper)*, 15(4):565–597, Dec 2006.
- [CS96] Tom Coffey and Puneet Saidha. Non-repudiation with mandatory proof of receipt. *ACM SIGCOMM Comp. Commun. Review*, 26(1):6–17, 1996.
- [CSW02] Nick Cook, Santosh K. Shrivastava, and Stuart Wheeler. 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.
- [CSW03] Nick Cook, Santosh K. Shrivastava, and Stuart Wheeler. Middleware Support for Non-repudiable Transactional Information Sharing between Enterprises. In *Proc. IFIP Int. Conf. on Distributed Applications and Interoperable Syst. (DAIS)*, Springer LNCS 2893, Paris, France, Nov 2003.
- [CW87] David R. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proc. IEEE Symp. on Security and Privacy*, pages 184–194, Oakland, CA, USA, 1987.
- [Dat04] DataPower. *XML Security Gateway*. <http://www.datapower.com/products/xs40.html>, 2004.
- [DY83] Danny Dolev and Andy C. Yao. On the Security of Public Key Protocols. *IEEE Trans. Inf. Theory*, 29(2):198–208, 1983.
- [EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A Randomized Protocol for Signing Contracts. *Commun. ACM*, 28(6):637–647, 1985.

- [ES05] Paul D. Ezhilchelvan and Santosh K. Shrivastava. A Family of Trusted Third Party based Fair-Exchange Protocols. *IEEE Trans. on Dependable and Secure Computing*, 2(4):273–286, 2005.
- [EY80] Shimon Even and Yacov Yacobi. Relations Among Public Key Signature Systems. Technical Report CS175, Technion Israel Institute of Technology, Haifa, Israel, 1980.
- [FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, April 1985.
- [FR03] Marc Fleury and Francisco Reverbel. The JBoss Extensible Server. In *Proc. ACM/IFIP/USENIX Int. Middleware Conf.*, Springer LNCS 2672, Rio de Janeiro, Brazil, Jun 2003.
- [GDWJ05] Tom Goovaerts, Bart De Win, and Wouter Joosen. Security Services in Mainstream Enterprise-Oriented Middleware Platforms. Technical Report CW 406, Dept. of Computer Science, Katholieke Universiteit Leuven, 2005.
- [GEN⁺01] Anders Grangard, Brian Eisenberg, Duane Nickull, et al. ebXML Technical Architecture Specification v1.0.4. OASIS Final Draft, <http://www.ebxml.org/>, 2001.
- [GHJV93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In *Proc. European Conference on Object-Oriented Programming (ECOOP)*, Springer LNCS 707, pages 406–431, Kaiserslautern, Germany, 1993.
- [GLM04] Paul Groth, Michael Luck, and Luc Moreau. A protocol for recording provenance in service-oriented grids. In *Proc. 8th Int. Conf. on Principles of Distributed Systems*, Grenoble, France, 2004.
- [Gol99] Dieter Gollmann. *Computer Security*. John Wiley and Sons, 1999.
- [HBM05] Phillip Hallam-Baker and Shivaram Mysore. XML Key Management Specification (XKMS 2.0). W3C Recommendation, <http://www.w3.org/TR/xkms2/>, 2005.
- [HFPS99] Russell Housley, Warwick Ford, Tim Polk, and David Solo. Internet X.509 Public Key Infrastructure: Certificate and CRL Profile. IETF RFC 2459, Internet Engineering Task Force, 1999.
- [Hol91] Gerald J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.
- [Hol04] Gerald J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.
- [ISO89] ISO. *Information Processing Systems - Open Systems Interconnection - Security Frameworks for Open Systems - Basic Reference Model - Part 2: Security Architecture*. ISO 7498-2, 1989.

- [ISO96] ISO/IEC. *Information Technology - Open Systems Interconnection - Security Frameworks for Open Systems - Part 1: Overview*. ISO/IEC 10181-1, 1996.
- [Jos03] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. IETF RFC 3548, Internet Engineering Task Force, 2003.
- [KMZ02] Steve Kremer, Olivier Markowitch, and Jianying Zhou. An Intensive Survey of Fair Non-repudiation Protocols. *Computer Communications*, 25(17):1601–1621, 2002.
- [LNJ00] Peng Liu, Peng Ning, and Sushil 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, NY, USA, 2000.
- [Low97] Gavin Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *Proc. 10th IEEE Comp. Security Foundations Workshop*, Rockport, MA, USA, 1997.
- [LPF05] Hongbin Liu, Shrideep Pallickara, and Geoffrey Fox. Performance of Web Services Security. In *Proc. 13th Annual Mardi Gras Conf. on Frontiers of Grid Applications and Technologies*, Baton Rouge, Louisiana, 2005.
- [MD02] Zoran Milosevic and Geoff Dromey. On Expressing and Monitoring Behaviour in Contracts. In *Proc. IEEE Int. Conf. on Enterprise Distributed Object Computing (EDOC)*, Lausanne, Switzerland, 2002.
- [MGBM06] Simon Miles, Paul Groth, Miguel Branco, and Luc Moreau. The requirements of using provenance in e-Science experiments. Technical Report 12566, Electronics and Computer Science, University of Southampton, <http://eprints.ecs.soton.ac.uk/12566/>, 2006.
- [MGK02] Olivier Markowitch, Dieter Gollmann, and Steve Kremer. On Fairness in Exchange Protocols. In *Proc. 5th Int. Conf. on Information Security and Cryptology (ISISC 2002)*, Springer LNCS 2587, Seoul, Korea, 2002.
- [MGL⁺04] Zoran Milosevic, Simon Gibson, Peter F. Linnington, James Cole, and Sachin Kulkarni. On design and implementation of a contract monitoring facility. In *Proc. 1st IEEE Workshop on Electronic Contracting*, San Diego, CA, USA, 2004.
- [MJSSW04] Carlos Molina-Jimenez, Santosh K. Shrivastava, Ellis Solaiman, and John Warne. Run-time monitoring and enforcement of electronic contracts. *J. Electronic Commerce Research and Applications*, 3(2):108–125, 2004.
- [MJSW05] Carlos Molina-Jimenez, Santosh K. Shrivastava, and John Warne. A Method for Specifying Contract Mediated Interactions. In *Proc. 9th IEEE Int. EDOC Enterprise Computing Conf.*, Enschede, Netherlands, 2005.
- [MK01] Olivier Markowitch and Steve Kremer. An Optimistic Non-Repudiation Protocol with Transparent Trusted Third Party. In *Proc. 4th Int. Conf. on Information Security (ISC2001)*, Springer LNCS 2200, pages 363–378, Malaga, Spain, 2001.

- [MR99] Olivier Markowitch and Yves Roggeman. Probabilistic Non-repudiation without Trusted Third Party. In *Proc. 2nd Workshop on Security in Communication Networks*, Amalfi, Italy, 1999.
- [MU00] Naftaly Minsky and Victoria Ungureanu. Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. *ACM Trans. Software Eng. and Methodology*, 9(3):273–305, 2000.
- [NKHBM04] Anthony Nadalin, Chris Kaler, Phillip Hallam-Baker, and Ronald Monzillo. Web Services Security: SOAP Message Security 1.0. OASIS Standard 200401, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>, 2004.
- [NZB04] Aleksandra Nenadic, Ning Zhang, and Stephen Barton. FIDES - a Middleware E-Commerce Security Solution. In *Proc. 3rd European Conf. on Inf. Warfare and Security (ECIW)*, London, UK, 2004.
- [OMG98] OMG. CORBA Security Service. OMG Specification 98-12-17, <http://www.omg.org/>, 1998.
- [Ope91] Open Group. Distributed Transaction Processing: The XA Specification. X/Open CAE Specification XO/CAE/91/300, X/Open Company Ltd., 1991.
- [OTMMS02] S.A. Oberg, L.A. Tewksbury, Louise E. Moser, and Peter M. Melliar-Smith. Online Upgrades for CORBA and EJB/J2EE. In *Proc. IEEE Workshop on Dependable Middleware-Based Systems (WDMS 2002)*, Washington, DC, USA, 2002.
- [PAC⁺04] Trevor Perrin, Dimitri Andivahis, Juan Carlos Cruellas, Frederick Hirsch, et al. Digital Signature Service Core Protocols, Elements and Bindings. OASIS Committee Working Draft 26, <http://www.oasis-open.org/committees/dss>, 2004.
- [PCC⁺06] Panos Periorellis, Nick Cook, Adrian Conlin, Hugo G. Hiden, et al. Service Oriented Middleware for the Formation and Operation of Virtual Organisations: The GOLD Project. Technical Report CS-TR 940, School of Computing Science, Univ. Newcastle, 2006.
- [PG99] Henning Pagnia and Felix Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, Dept. of Computer Science, TU Darmstadt, 1999.
- [Pro03] Marek Prochazka. Jironde: A Flexible Framework for Making Components Transactional. In *Proc. IFIP Int. Conf. on Distributed Applications and Interoperable Syst. (DAIS)*, Springer LNCS 2893, Paris, France, Nov 2003.
- [PVG03] Henning Pagnia, Holger Vogt, and Felix Gärtner. Fair Exchange. *The Computer Journal*, 46(1):55–75, 2003.
- [RCS05] Paul Robinson, Nick Cook, and Santosh K. Shrivastava. Implementing Fair Non-repudiable Interactions with Web Services. In *Proc. 9th IEEE Int. EDOC Enterprise Computing Conf.*, Enschede, Netherlands, 2005.

- [Ros05] RosettaNet. *eBusiness Standards for the Global Supply Chain*. RosettaNet, <http://www.rosettanel.org/RosettaNet/>, 2005.
- [Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, 2nd edition, 1996.
- [SM00] Vitaly Shmatikov and John C. Mitchell. Analysis of a Fair Exchange Protocol. In *Proc. Internet Soc. Symp. on Network and Distributed Syst. Security (NDSS)*, San Diego, CA, USA, 2000.
- [SMJ05] Santosh K. Shrivastava and Carlos Molina-Jimenez. TAPAS Architecture. Project Deliverable D6, TAPAS EU Project IST-2001-34069, School of Computing Science, University of Newcastle, 2005.
- [SPG05] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Survey of Data Provenance in e-Science. *SIGMOD Record*, 34(3), September 2005.
- [Sun03] Sun Microsystems. Java 2 Platform Enterprise Edition (J2EE). Specification 1.4, Sun Microsystems Inc., <http://java.sun.com/j2ee/>, 2003.
- [Ted85] Tom Tedrick. Fair Exchange of Secrets. In *Proc. Advances in Cryptology - Crypto'84*, Springer LNCS 196, pages 434–438, Santa Barbara, CA, USA, 1985.
- [UPU02] UPU. Global EPM Non-repudiation Service Definition and the Electronic Postmark 1.1. Universal Postal Union 1.1, <http://www.globalepost.com/prodinfo.htm>, Oct 2002.
- [Ver04] Verisign. Verisign Trust Gateway: Simplifying Application and Web Services Security. Verisign White Paper, <http://www.verisign.com/products-services/security-services/intelligence-and-control-services/application-security/index.html>, 2004.
- [Wan05] Guilin Wang. Generic Fair Non-repudiation Protocols with Transparent Offline TTP. In *Proc. 4th Int. Workshop for Applied PKI*, pages 51–65, Singapore, 2005.
- [WIC99] Michael Wichert, David Ingham, and Steve Caughey. Non-repudiation Evidence Generation for CORBA using XML. In *Proc. IEEE Annual Comp. Security Applications Conf.*, Phoenix, AZ, USA, 1999.
- [XDo05] XDoclet. *XDoclet: Attribute-Oriented Programming*. <http://xdoclet.sourceforge.net/>, 2005.
- [ZBD03] Jianying Zhou, Feng Bao, and Robert Deng. Validating Digital signatures without TTP's Time-stamping and Certificate Revocation. In *Proc. 2003 Inf. Security Conf.*, Springer LNCS 2851, Bristol, UK, 2003.
- [ZG97] Jianying Zhou and Dieter Gollmann. Evidence and non-repudiation. *J. Network and Comp. Applications*, 20(3):267–281, 1997.
- [Zho01] Jianying Zhou. *Non-repudiation in Electronic Commerce*. Artech House Computer Security Series, 2001.

-
- [Zho02] Jianying Zhou. Maintaining the Validity of Digital signatures in B2B Applications. In *Proc. Australasian Conf. on Inf. Security and Privacy*, Springer LNCS 2384, pages 303–315, Melbourne, Australia, 2002.
- [ZL99] Jianying Zhou and Kwok-Yan Lam. Securing Digital Signatures for Non-repudiation. *Computer Communications*, 22(8):710–716, 1999.

Authored publications

I am co-author of the following six publications cited in this dissertation: [CRS04a, CRS04b, CRS06, CSW02, CSW03, RCS05]