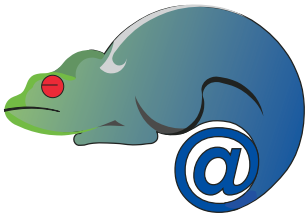
ADAPT
IST-2001-37126

Middleware Technologies for Adaptive and Composable Distributed Components

Security and Trust in Composite Services



Deliverable Identifier: D12
Delivery Date: **19** August 2004
Classification: Public Circulation
Authors: Santosh Shrivastava
Document version: Final, August 2004

Contract Start Date: 1 September 2002
Duration: 36 months
Project coordinator: Universidad Politécnica de Madrid (Spain)
Partners: Universitá di Bologna (Italy), ETH Zürich (Switzerland), McGill University (Canada), Universitá degli Studi di Trieste (Italy), University of Newcastle (UK), Arjuna Technologies Ltd. (UK)

Project funded by the European Commission under the Information Society Technologies Programme of the 5th Framework (1998-2002)

CONTENTS

Dependencies with other deliverables 2
1. Introduction 2
2. Requirements and Architectural Concepts 3
2.1. Conversations 3
2.2. Contract mediated conversations 4
3. Non-repudiated Interactions 6
4. Contract Monitoring and Enforcement 9
5. Concluding Remarks 12
References 12
Appendix: Component Middleware to Support Non-repudiable Service Interactions 14
1. Introduction 14
2. Motivating example 15
3. Building blocks for trusted interaction 16
4. Component-based implementation 24
5. Related work 29
6. Conclusions and future work 29

Dependencies with other deliverables

This report describes how trust and security issues can be addressed in composite services (CS). ADAPT deliverable report D9 (CS Middleware Architecture) gave the overall architecture of CSs within the context of virtual organisations [1]. Composition and enactment of CSs was described in ADAPT deliverable reports D6 (Service Specification Language) and D7 (Composition Language) [2,3]. The work described here has been carried out in collaboration with IST Project IST-2001-34069: "TAPAS (Trusted and QoS-Aware Provision of Application Services). In particular, this report describes how the ideas presented in TAPAS deliverable Reports D5 (TAPAS Architecture: concepts and protocols) and D9 (Component Middleware for Trusted Coordination) [4,5] can be used within ADAPT. A research paper based on the work reported here is attached in the appendix.

1. Introduction

Presence of a wide variety of services and resources over the Internet creates new opportunities for providing value added, inter-organisational services by composing multiple exiting services into new Composite Services (CSs). This naturally leads to resource sharing across organisational boundaries. An inter-organisational business relationship is commonly referred to as a virtual organisation (VO). Whether in the context of large-scale scientific experiments, eCommerce, eGovernment or any other collaborative effort, organisations need cost-effective ways of finding, purchasing and managing services performed by other organisations. It is therefore necessary for organisations to be able to set-up and manage business links with other organisations in a rapid, dynamic and flexible manner. A VO however, blurs the distinction between 'outsiders' and 'insiders' and yet, organisations forming a VO will want to preserve their individual autonomy and privacy. A central problem in VO management is therefore how organisations can regulate access to their resources by other organisations in a way that ensures that their individual policies for information sharing are honoured. Regulating access to resources by other organisations is made difficult as the organisations might not trust each other. Organisation will therefore require their interactions with other organisations to be strictly controlled and policed. How can this be achieved? First we need to understand trust management issues in open systems (see [6] for review of trust related issues).

Trust is a vital component of every business interaction. The Oxford dictionary defines trust as "Firm belief in the reliability or truth or strength of an entity". Following [7], we consider a trust relationship of the form: **A** *trusts* **B** *on matters of* **X** *at epoch* **T**

Here, *A* and *B* may be people, computers and their specific resources and services, or even small or large organisations that admit to *trust relationships*. In the proposition, *A* is placing a trust relationship (*dependence*) on *B* with respect to matters of *X*. Such matters constitute the set of *rights and obligations* of *A* with respect to *B*, such that *B* permits access to its specific resources (services) to *A* provided that *A* fulfils specific *obligations* (*conditions*) laid down by *B*. Epoch *T* represents the period during which both *A* and *B* observe the well being of the their trust relationship without incidence of failure.

In the paper-based world, businesses have been conducted using contracts. The concept and the use of contracts are not new to today's society. Legal contracts can be traced back to ancient times [8].

To form and manage VOs, we need to emulate electronic equivalents of the contract based business management practices; this means that interactions between organizations will need to be mediated by *electronic contracts*, and then enforced and monitored at run time. This aspect is explored in detail in this report.

2. Requirements and Architectural Concepts

2.1. Conversations

Workflow management systems for Web service enactment specify the composition of a CS as a business process. Such a specification should be sufficiently abstract, uncluttered with specific details of enactment. Specifying the enactment of the business process is a concrete operation which requires further information that is critically dependent on organisational issues. For example, the organisations involved in a CS may have a peer-to-peer business relationship, in which case, a decentralised enactment seems a natural choice, with each organisation responsible for its part of the process. Where as in a hierarchic relationship, a centralised enactment may well be deemed more appropriate. Fig.1 depicts centralised enactment of CS from organisation 3; here double arrowed lines indicated message exchanges between organisations. In order to realise the CS, organisation 3 will need to set up a contract each with organisations 1 and 2 detailing the terms and conditions of service usage (Web services WS1 and WS2 respectively). An alternative arrangement is also possible where there is a single contract between the three organisation for CS provision.

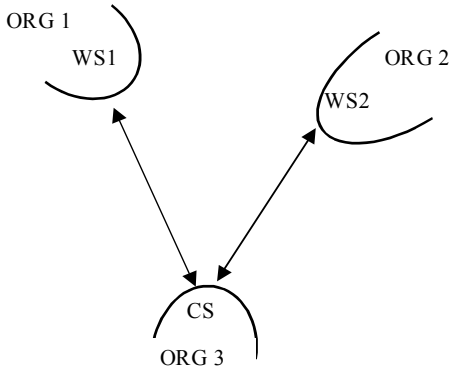


Fig. 1: Inter-organisational interactions

Our main requirement is to ensure that the enactment of a CS generates only those inter-organisation interactions that are consistent with the contracts in force between the organisations forming a VO.

We assume that trading partners of a VO explicitly or implicitly agree on the set of 'message interaction patterns' or 'conversations' associated with a given contract. There could be several ways of achieving this:

(i) Trading partners agree to use a common standard that has a well defined set of conversations for business activities. An example of such a standard is Rosettanet [9], that has defined a number of conversation specifications called partner interface processes (PIPs). In this case, partners agree on the set of PIPs that they will use within the VO. The diagram (see fig. 2) shows the business roles, messages, and their sequence of exchange in a PIP concerned with submission of a purchase order.

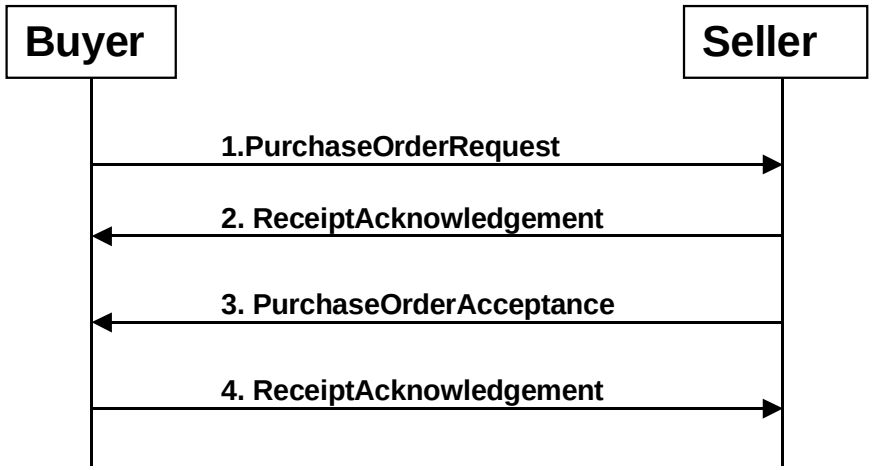


Fig. 2: A sample Rosettanet PIP Interaction

(ii) A variation on (i) above, trading partners jointly compose conversation specifications in terms of message ordering constraints and rules. The World Wide Web Consortium, W3C, is developing a standard on Web service choreography, WS-CDL [10] that defines a language for just such a task.
(iii) Conversation specifications are worked out by careful study of rights and obligations in contract clauses written in a natural language, and any sequencing constraints published by the individual services. We briefly describe in a subequent section (and in detail in [11]) how finite state machine descriptions can be obtained from contracts.

2.2. Contract mediated conversations

Next we examine how to ensure that enactment of a CS generates interactions that are consistent with the contracts in force between the organisations.
Each enterprise expects access to other's services. An operation on a service is allowed only if it is permitted by the rules of the contract and then only if it is invoked by a legitimate role player of a participating enterprise. Thus, a contract is a mechanism that is conceptually located in the middle of the interacting enterprises to intercept all the contractual operations that the parties try to perform. Intercepted operations are accepted or rejected in accordance with the contract clauses and role players' authentication.
We thus need to maintain an explicit representation of a contract. A *contract* can be defined as a paper document that stipulates that the signatories (two or more) agree to observe the clauses stipulated in the document. Each entry in the document is called a *term* or a *clause*. Moreover, it is common practice to specify what *role* (manager, accountant, supervisor, etc.) each of the signatories (Alice, Bob, Doug, etc.) play within their respective enterprises. An *e-contract* is an electronic version of a conventional contract. It is an electronic document that stipulates that the signing entities (two or more) agree to observe clauses stipulated in the document.

A *right* can be defined as an authorization to do something. Because it is only an authorization, a right *may or may not* be exercised. In the context of the execution of an e-contract, a right is an authorization to perform an operation that will affect the behaviour of the execution of the e-contract. For example, the e-contract can stipulate that Alice, as a manager of enterprise E1, has the right to send an offer to sell to Bob,

the manager of E2. Because this is a right, it is up to Alice to send or not to send the offer to Bob; Bob will not be disappointed if he does not receive the offer.

Similarly, an *obligation* can be defined as a duty that *must* be performed. In the context of the execution of an e-contract, an obligation is a duty to perform an operation that will affect the behaviour of the execution of the e-contract. A failure to perform such a duty means a breach of the e-contract. For example, the e-contract might stipulate that upon receiving an offer to sell from Alice, Bob has the obligation to reply to her with an *OfferAccepted* or *OfferRejected* message.

As an example, consider a simple request-response message exchange as related to a contract clause. Suppose the contract clause states that "Alice has the right to retrieve a copy of doc1 from Bob's enterprise, provided that her request is not submitted on Fri, Sat or Sun. Bob has the obligation to provide doc1 in less than 24 hrs." Fig. 3. illustrates the role of the contract monitor/enforcer that only permits a legitimate request-reply conversation interaction.

To the contract monitor, we add the additional requirement of supporting non-repudiated interactions. This is because to regulate the interactions involved, 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 disavow the request and/or consumption of a service. That is, to regulate an interaction we require attribution, validation and audit of the actions of the parties involved. 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 subsequent interactions.

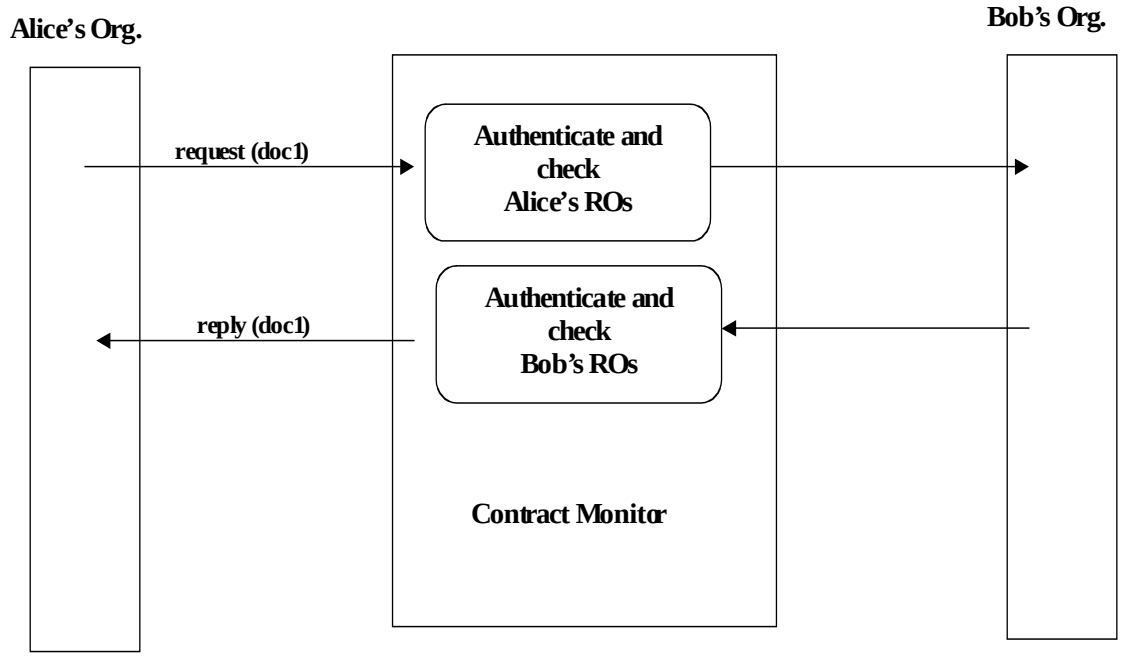


Fig. 3: Contract monitor

We propose a two level architecture for facilitating contract mediated interactions (see fig. 4). The lower level provides a mechanism for intercepting messages, generating non-repudiation logs and upcalling the higher level contract monitoring sub-system that maintains e-contracts and performs contract specific validation. The message is

forwarded to the intended destination only if the validation succeeds. The design of such an architecture that permits distributed implementation is covered in the following two sections.

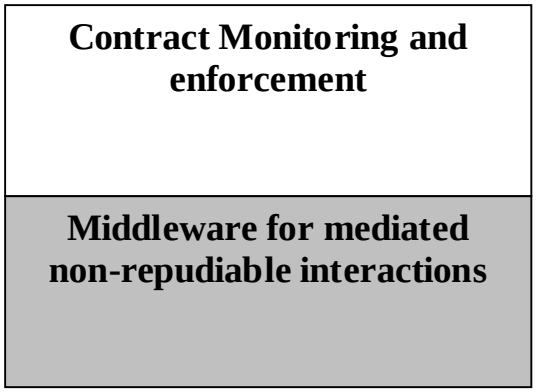


Fig. 4: Two level architecture for contract monitoring

3. Non-repudiated Interactions

We describe two building blocks for regulated (mediated) interaction between organisations: non-repudiable service invocation (NR-Invocation) and non-repudiable information sharing (NR-Sharing). Component middleware support for regulated service interactions ensures that actions of a member of a VO are non-repudiably bound to the member; the acceptance, or otherwise, of those actions is non-repudiably bound to the other members of the VO; and that service invocations, and the results of those invocations, are bound to the parties to the invocation. The ideas presented have been dicussed in detail in the TAPAS deliverable [5] and the research paper [12], attached as an appedix.

3.1 Trusted interceptor abstraction

In this section we introduce the abstraction of *trusted interceptors* that mediate inter-organisational interaction and then model non-repudiable service invocation and non-repudiable information sharing in terms of this abstraction. The trusted interceptor abstraction is sufficiently general to apply to a variety of interaction scenarios. For example, it is not bound to any particular non-repudiation protocols but can be seen as a flexible framework in which protocols can be deployed as appropriate to the regulatory regime governing an interaction or to the trust relationships between the parties to an interaction.

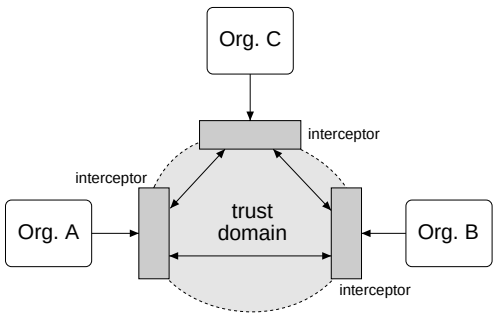


Figure 5. Trusted interceptors

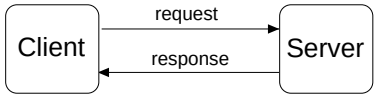
As shown in Figure 5, each organisation conceptually has a trusted interceptor that acts on its behalf. The introduction of the trusted interceptors transforms an unregulated domain into a trust domain that safeguards the interests of each party. The interaction

between interceptors is regulated, audited and fair [13]. That is, trusted interceptors provide a trust domain by policing access to the domain and regulating and auditing actions within the domain. The fairness guarantee is that honest parties will not be disadvantaged by the behaviour of dishonest parties. In the worst case, a break down in an interaction will lead to dispute. To support dispute resolution, the fact that trusted interceptors mediated the interaction will provide any honest party with irrefutable evidence of their own actions within the domain and of the observed actions of other parties. The trusted interceptor abstraction insulates the parties to the interaction from the detail of underlying mechanisms used to meet regulatory requirements. Interceptors can implement different mechanisms to meet different interaction requirements and can be reconfigured to meet changing requirements as inter-organisational relationships evolve.

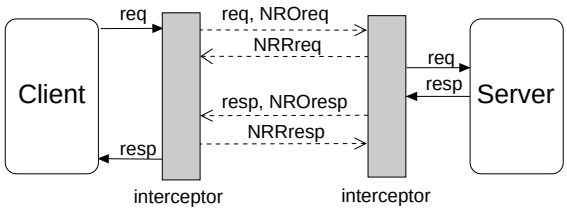
3.1.1 Non-repudiable service invocation (NR-Invocation)

Figure 6(a) shows a typical two-party, client-server interaction. The client invokes a service by sending a request to the server who issues a response. Non-repudiable service invocation provides the following assurances to the client:

1. that following an attempt to submit a request to a server, either: (a) the submission failed and the server did not receive the request; or (b) 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 produced the response.



(a) Service invocation



(b) Non-repudiable service invocation

Figure 6. NR-Invocation through trusted interceptors

For the server, the corresponding assurances are:

1. that if a request is received, there is proof identifying the client who submitted the request; and:
2. that following an attempt to deliver a response to the client, either: (a) the delivery failed and the client did not receive the response; or (b) delivery succeeded and there is proof that the response is available to the client.

To provide the above assurances, trusted interceptors execute a non-repudiation protocol that ensures the following:

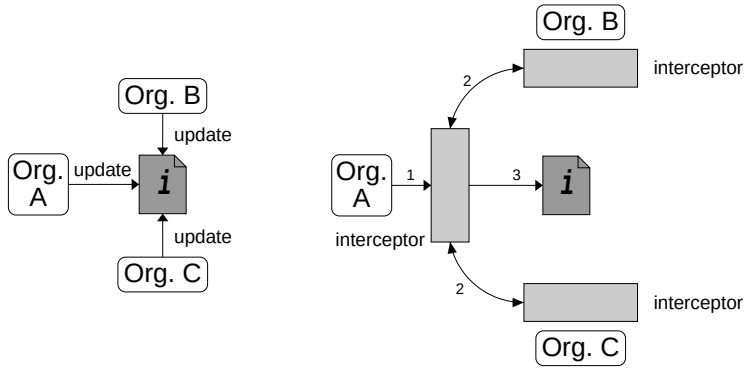
1. a request is passed to a server if, and only if, the client (or its interceptor) provides non-repudiation evidence of the origin of the request (NROreq) and the

server (or its interceptor) provides non-repudiation evidence of receipt of the request (NRRreq)

2. the response is passed to the client if, and only if, the server (or its interceptor) provides non-repudiation evidence of the origin of the result (NROresp) and the client (or its interceptor) provides non-repudiation evidence of receipt of the response (NRRresp).

Non-repudiation tokens include a unique request identifier, to distinguish between protocol runs and to bind protocol steps to a run, and a signature on a secure hash of the evidence generated. Figure 6(b) models the exchange of evidence achieved by the execution of an appropriate non-repudiation protocol between interceptors acting on behalf of client and server. The client initiates a request for some service. The client's interceptor generates an NROreq token and then sends both the request and the token to the server's interceptor. The server's interceptor generates an NRRreq token and returns it to the client's interceptor. The server's interceptor then passes the request to the server to generate a response. On receipt of the response, the server's interceptor generates an NROresp token and sends both the response and the token to the client's interceptor. The interceptors ensure that irrefutable evidence of the exchange is both generated and stored.

3.1.2 Non-repudiable information sharing (NR-Sharing)



(a) Information sharing (b) Non-repudiable information sharing

Figure 7. NR-Sharing through trusted interceptors

Figure 7(a) shows three organisations (A, B and C) accessing and updating shared information. If, for example, A wishes to update the information, then they must reach agreement with B and C on the validity of the proposed update. For the agreement to be non-repudiable: (i) B and C require evidence that the update originated at A; and (ii) A, B and C require evidence that, after reaching a decision on the update, all parties have a consistent view of the agreed state of the shared information. The latter condition implies that there must be evidence that all parties received the update and all parties know whether there was unanimous agreement to it being applied to the information. Figure 7(b) shows A proposing an update to the information shared by A, B and C. Interceptors are used to mediate each organisation's access to the information. In step 1, A attempts an update to the information. A's interceptor intercepts the update and, in step 2, executes a non-repudiable state coordination protocol with B and C to achieve the following:

1. That A's update is irrefutably attributable to A and proposed to B and C.

2. That B and C independently validate A's proposed update, using a locally determined and application-specific process, and their respective decisions are made available to A and are irrefutably attributable to B and C.

3. That the collective decision on the validity of the update (in this case, responses from B and C to A) are made available to all parties (A, B and C).

If the resolution of the protocol executed at step 2 represents agreement to the update then the shared information is updated in step 3. Otherwise, the information remains in the state prior to A's proposed update. Non-repudiable connect and disconnect protocols govern changes to the membership of the group of organisations sharing the information The use of interceptor's allows us to abstract away the details of state coordination and insulate the application from protocol specifics. From the application viewpoint, the update to shared information is an atomic action that succeeds or fails dependent on the agreement of the parties sharing the information. Thus the interceptors may execute any protocol that achieves non-repudiable agreement on: the origin and state of a proposed update; the state of the shared information after application of an update; and the membership of the group that agreed to, or vetoed, the update.

4. Contract Monitoring and Enforcement

We use finite state machines (FSMs) for representing conversations. We describe how to map the rights and obligations extracted from the clauses of the contract into the states, transition and output functions, and input and output symbols of a FSM.

At the level of rights and obligations a contract is often more easily understood as a set of FSMs, one for each contracting party. So, from our example in Fig.3, we will have one FSM for the purchaser (Alice) and one FSM for the supplier (Bob). The physical location of each FSM is irrelevant to the functionality of the contract and is decided at the time of implementation, influenced by the way interceptor have been deployed. We will now discuss how the rights and obligations stipulated in an e-contract can be mapped into the FSMs.

It is easy to reason about the operations of an e-contract, with the following general syntax in mind:

if $event_1$ *&* $condition_q$
perform $operation_1$ *and switch to* $state_1$
else if $event_2$ *&* $condition_q$
perform $operation_2$ *and switch to* $state_2$
… … …
else if $event_m$ *&* $condition_q$
perform $operation_m$ *and switch to* $state_m$

This syntax expresses the idea that, at some point an e-contract can be at any of *n* possible conditions ($condition_1$, $condition_2$,…,$condition_n$). If the e-contract is in a given $condition_q$ (for example, *WatingForOffer*), there is a finite and well defined set of events ($event_1$, $event_2$, …,$event_m$) that can affect the future behaviour of the contract. The occurrence of $event_i$ determines what objects (variables, files, database, etc.) within the system change their values, that is, the event determines to which new condition the systems switches. Similarly, there is a finite and well defined set of operations ($operation_1$, $operation_2$,…, $operation_m$) that can be executed when the system is in $condition_q$. The $event_i$ determines the operation to be executed.

Thus, in terms of FSMs, we can express the above syntax as shown in Fig.8, where *e* and *o* stand for event and operation, respectively.

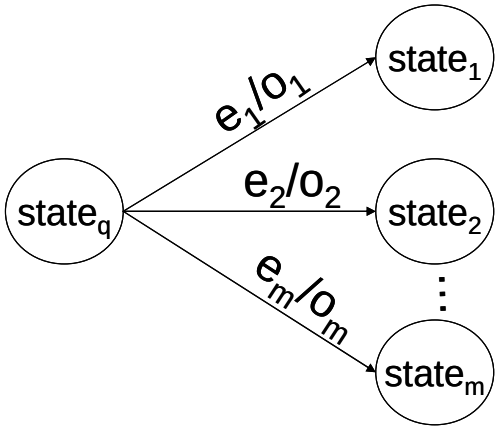


Fig. 8. FSM representation

To show what rights and obligations look like, we will discuss a very simple contract for offering and purchasing goods. As an attentive reader will notice, the contract has some ambiguities that could lead to deadlocks (The contract text does not specify the time for sending the offer. Neither does it specify the time for sending the notification about rejecting or accepting the offer). However, here we will ignore these. In practice, such ambiguities will have to be removed, for example, using the process of model checking, as discussed in [14].

1. Offering
1.1 The supplier may use his discretion to send offers to the purchaser.
1.2 The purchaser is entitled to accept or reject the offer, but he shall notify his decision to the supplier.
2. Commencement and completion
2.1 The contract shall start immediately upon signature.
2.2 The purchaser and the supplier shall terminate the e-contract immediately after reaching a deal for buying an item.

From this English text contract clause we can extract the sets of rights and obligations for the purchaser and the supplier and express them in terms of operations for FSMs.

Purchaser's rights:

R_1^P : SendAccepted -- right to accept offers.

R_2^P : SendRejected -- right to reject offers.

Purchaser's obligations:

O_1^P : StartEcontract -- obligation to start the e-contract.

O_2^P : SendAccepted or SendRejected -- obligation to reply to offers.

O_3^P : EndEcontract -- obligation to terminate the e-contract.

Supplier's rights:

R_1^s : SendOffer -- right to send offers.

Supplier's obligations:

O_1^S : StartEcontract -- obligation to start the e-contract.

O_2^S : EndEcontract -- obligation to terminate the e-contract.

As shown in Fig.9, we have used two FSMs to describe the conversation implied by the English text contract of our example.

With appropriate support from the underlying middleware (see below), each FSM can be used to monitor and enforce the rights and obligations of its owner. Thus the supplier's FSM will allow the supplier to execute only the operations he has the right to execute and nothing else. Likewise, the FSM enforces the supplier to execute the operations he has the obligation to execute. The purchaser's FSM works in a similar way. The FSM based contract representation can be further enriched with access control mechanisms, e.g, role based access control, as dicussed in [4].

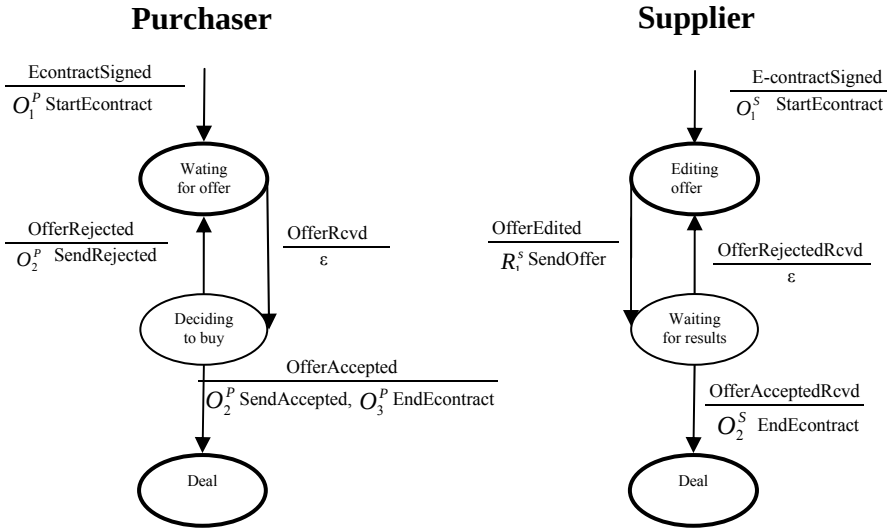


Fig. 9. FSM Representation.

With this background, we can hint at the overall implementation of a distributed contract monitoring system. The implementation of that involves a purchaser and a supplier is shown in Fig.8. Each party maintains a copy of the contract object, encoded as one or more shared objects that support non-repudiable information sharing; in the diagram they are referred to as B2Bobj; operations on these objects are controlled by the contract FSMs. The dashed line that goes from the supplier to the purchaser shows what happens when the supplier sends an offer. When the offer is ready, the supplier invokes a send operation, and the supplier's FSM switches to its *Waiting for response* state and makes a *SendOffer* call to the local copy of a shared B2Bobj (that implements the operation). The local B2Bobj collects, and signs, evidence of the operation and requests coordination of the proposed update to its state with the purchaser's B2Bobj.

The purchaser's B2Bobj verifies the evidence provided and makes an up-call to the purchaser's FSM to validate the B2Bobj operation. Upon receiving the up-call, the purchaser's FSM switches to the *Deciding to buy state*.

The dashed line from the purchaser's FSM to the supplier's FSM shows how the purchaser's response is transmitted to the supplier. The middleware ensures that all operations performed by the purchaser and the supplier are recorded and are non-repudiable.

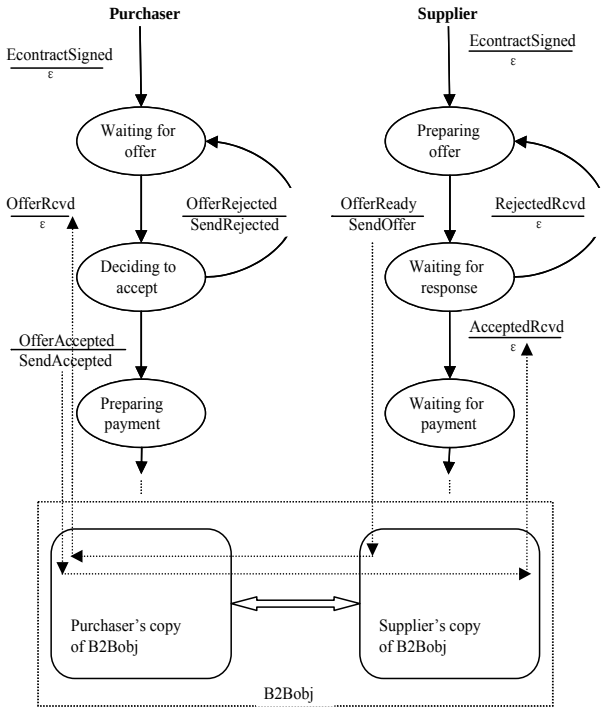


Fig. 14. Implementing contract monitoring

5. Concluding Remarks

A central problem in VO management is therefore how organisations can regulate access to their resources by other organisations in a way that ensures that their individual policies for information sharing are honoured. We have desribed how contracts can be used to regulate inter-organisation interactions. A contract is conceptually located in the middle of the interacting enterprises to intercept all the contractual operations that the parties try to perform. Intercepted operations are accepted or rejected in accordance with the contract clauses and role players' authentication. We have described a two level architecture that permits distributed implementation. Design details of the architecture were presented.

References

[1] ADAPT deliverable report D9: CS Middleware Architecture, September 2003, http://adapt.ls.fi.upm.es/adapt.htm
[2] ADAPT deliverable report D6: Service Specification Language, September 2003, http://adapt.ls.fi.upm.es/adapt.htm
[3] ADAPT deliverable report D7: Composition Language, September 2003, http://adapt.ls.fi.upm.es/adapt.htm
[4] TAPAS deliverable report D5: TAPAS Architecture: concepts and protocols, March 2003, http://www.newcastle.research.ec.org/tapas/
[5] TAPAS deliverable report D9: Component Middleware for Trusted Coordination, March 2004, http://www.newcastle.research.ec.org/tapas/
[6] T. Grandison and M. Sloman, "A survey of trust in Internet applications", IEEE Communications Surveys, Fourth Quarter 2000, www.comsoc.org/pubs/surveys.

[7] E. Gerck, *Towards Real-World Models of Trust: Reliance on Received Information*, published on 23rd June 1998 in the mcg-talk list server.
[8] Paul Halsall, "Ancient History Sourcebook: A Collection of Contracts from Mesopotamia, c. 2300--428 BCE", http://www.fordham.edu/halsall/ancient/mesopotamia-contracts.html, March, 1999.
[9] Rosettanet implementation framework: core specification, V2, Jan 2000. http://rosettanet.org
[10] Web service choreography Description Language, Version 1.0, http://www.w3.org/TR/ws-cdl-10/
[11] C. Molina-Jimenez, S.K. Shrivastava, E. Solaiman and J. Warne, "Run-time Monitoring and Enforcement of Electronic Contracts", Electronic Commerce Research and Applications (ECRA), Elsevier, Vol.3, No.2, 2004, pp. 108-125.
[12] Nick Cook, Paul Robinson and Santosh Shrivastava, "Component Middleware to Support Non-repudiable Service Interactions", IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2004), Florence, June 2004, pp. 605-614.
[13] Kremer, S., Markowitch, O. and Zhou, J., "An Intensive Survey of Fair Non-repudiation Protocols." Computer Communications, 25:1601-1621, Elsevier, 2002.
[14] Ellis Solaiman, Carlos Molina-Jimenez, and Santosh Shrivastava, "Model Checking Correctness Properties of Electronic Contracts", International Conference on Service Oriented Computing 2003, M.E. Orlowska et al. (Eds.), ICSOC 2003, LNCS 2910, pp. 303-318, 2003. ISBN 3 540 20681 7.

Appendix: Component Middleware to Support Non-repudiable Service Interactions[1]

Nick Cook

Paul Robinson

Santosh Shrivastava

School of Computing Science

University of Newcastle upon Tyne

Abstract. The wide variety of services and resources available over the Internet presents new opportunities to create value added, inter-organisational Composite Services (CSs) from multiple existing services. The resulting CS may involve close interaction between the constituent services of participating organisations. In order to preserve their autonomy and privacy, each organisation needs to regulate access both to their services and to shared information within the CS. Key mechanisms to facilitate such regulated interactions are the collection and verification of non-repudiable evidence of the actions of the parties to the CS. The paper describes how component based middleware can be enhanced to support non-repudiable service invocation and information sharing. These mechanisms can be incorporated in the service delivery platforms at each organisation or at one or more trusted third parties who offer non-repudiation services, or some combination of these options. A generic implementation, based on a J2EE application server, is presented.

Keywords: System Security; FT Architecture/Middleware Software Engineering; Non-repudiation; Service Composition

1. Introduction

The wide variety of services and resources available over the Internet presents new opportunities to create value-added, inter-organisational Composite Services (CSs) from multiple existing services. The resulting CS may involve close interaction between the constituent services of participating organisations. However, while cooperating to form a CS, each organisation needs to maintain their autonomy and privacy. This implies the regulation of access both to the services offered within a CS and to information that is shared in a CS. Regulation of access to shared information includes validation by all interested parties of any proposed changes to that information. Since the intention is to compose a CS from existing services, regulatory requirements should be met by the extension, as opposed to replacement, of existing services. The main contribution of this paper is to address this requirement by extending component based middleware to provide a flexible framework to support regulated interaction between organisations.

[1] Extended version of the paper presented at the IEEE/IFIP Int. Conf. on Dependable Syst. and Networks, Florence, Italy, 2004.

It is assumed that each organisation has a local set of policies for an interaction that is consistent with an overall agreement (or set of agreements) between organisations (the business contract). The formation and operation of the CS must not compromise local policies and must comply with the business contract. There are two aspects to regulation in this context:

1 high level mechanisms to specify and enforce contractual rights and obligations (examples include work on Law Governed Interaction [15] and on contract representation and monitoring [16]); and

2 lower level mechanisms to generate a non-repudiable audit trail that can be used to record and to verify that observed interaction behaviour adheres to agreements.

An interaction is non-repudiable if it is impossible for any party to the interaction to subsequently deny their participation. This paper presents two mechanisms that together form the basic building blocks for trusted interaction: non-repudiable service invocation and non-repudiable information sharing. These provide abstractions that are familiar from the intra-organisational context and result in regulated interaction in the inter-organisational 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.

The contributions of this paper are that it: (i) introduces the abstraction of *trusted interceptors* that mediate the interaction between organisations to achieve the exchange of non-repudiation evidence and to validate changes to shared information; (ii) shows that this abstraction is sufficiently general to apply to a variety of interaction scenarios; and (iii) demonstrates the practicality of the abstraction through a prototype implementation in component based middleware (such as J2EE [21]). Section 2 provides a motivating example. Section 3 discusses the trusted interceptor abstraction and our model of non-repudiable interaction. Section 4 describes the prototype component-based implementation of non-repudiation services. Related work is discussed in Section 5. Section 6 concludes the paper with an overview of future work.

2. Motivating example

This section describes the scenario of a specialist car manufacturer that combines components from various part suppliers to satisfy the requirements of a specialist car dealer (acting on behalf of the ultimate customer).

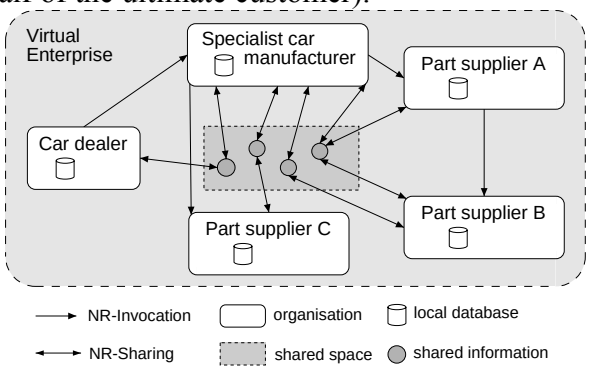


Figure 1. Specialist car manufacturer application

Figure 1 presents the overall structure of the interaction between the specialist car dealer, the car manufacturer and, in this example, three car part suppliers. In effect these

enterprises collaborate to form a virtual enterprise (VE) to deliver a specialist car to the car dealer's customer. That is, the VE creates a Composite Service (CS) for the specification and delivery of a specialist car. The CS interactions must be regulated to ensure that each member of the VE obtains the value they expect from the collaboration and are bound to the corresponding commitments they make.

CS interactions involve invocation of services between members of the VE and the sharing of information that is held in common by the VE. For example, Figure 1 depicts the car manufacturer and suppliers A and B negotiating the delivery of some component. The component is required to meet an overall specification negotiated between the dealer and the manufacturer. The manufacturer is then required to reach agreement with the suppliers on details such as: interfaces between parts, cost of customisation and delivery schedules. It is natural to share this information so that each party can update it (subject to the agreement of the other parties). Other artifacts that are shared, and may be subject to renegotiation, are the agreements governing the interaction. In addition to update to shared information, the process of reaching agreement on the specification of a car component, and the car as a whole, will involve requests between parties that some action is performed. Actions may range from the resolution of queries on the range of parts available to requests to act on shared information (initiating a transfer of control). These requests are naturally expressed as service invocations.

To regulate interactions of the above type, 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 disavow the request and consumption of a service. Similarly, it should not be possible for the service provider to subsequently deny having delivered a service. If information is shared then the parties sharing the information should be able to validate a proposed update, the update should be attributable to its proposer and the validation decisions with respect to the update attributable to the other parties. That is, to regulate an interaction we require attribution, validation and audit. 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. This paper addresses these requirements by providing two building blocks for regulated interaction between organisations: non-repudiable service invocation (NR-Invocation) and non-repudiable information sharing (NR-Sharing). Component middleware support for regulated service interactions ensures that actions of a member of a VE are non-repudiably bound to the member; the acceptance, or otherwise, of those actions is non-repudiably bound to the other members of the VE; and that service invocations, and the results of those invocations, are bound to the parties to the invocation.

3. Building blocks for trusted interaction

This section discusses the abstraction of trusted interceptors that mediate inter-organisational interaction and describes our model of non-repudiable interaction in terms of this abstraction. We argue that the trusted interceptor abstraction is sufficiently general to apply to a variety of interaction scenarios. For example, it is not bound to particular non-repudiation protocols but can be seen as a flexible framework in which protocols can be deployed as appropriate to the regulatory regime governing an interaction or to the trust relationships between the parties to an interaction.

3.1 Trusted interceptors and trust domains

Inter-organisational interaction requires regulatory mechanisms to ensure: (i) that misbehaviour by dishonest parties does not disadvantage honest parties and (ii) that honest parties share a verifiable, consistent view of the nature of the interaction. However, different types of interaction will demand different mechanisms. The choice of mechanisms to deploy will be determined by application-specific factors such as: the relationship between the parties to the interaction, the legal framework and agreements that govern the interaction, and the application domain within which the organisations operate. The common feature of all regulatory mechanisms is that they somehow mediate the interaction between parties. The trusted interceptor abstraction generalises this notion of mediation. As shown in Figure 2, conceptually, each party has a trusted interceptor that acts on its behalf. The introduction of trusted interceptors transforms an unregulated domain into a trust domain for the conduct of regulated, audited and fair interaction. Informally, a fair interaction is one in which honest parties cannot be disadvantaged by the behaviour of dishonest parties (for details, see Markowitch et al [14] who discuss the evolution of the notion of fairness in exchange protocols). The trusted interceptor abstraction insulates the parties to the interaction from the detail of underlying mechanisms used to meet regulatory requirements. Interceptors can implement different mechanisms to meet different interaction requirements and can be reconfigured to meet changing requirements as relationships evolve.

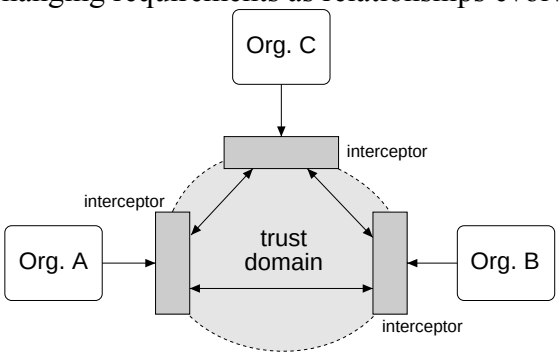
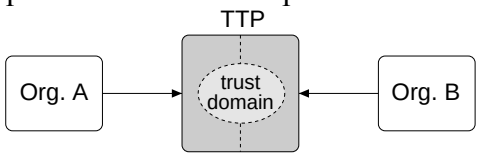


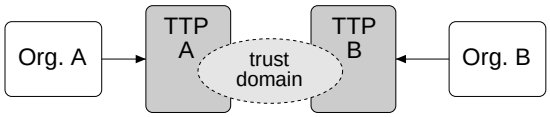
Figure 2. Trusted interceptors

Trusted interceptors provide a trust domain by policing access to the domain and regulating and auditing actions within the domain. To support dispute resolution, the fact that trusted interceptors mediated the interaction will provide any honest party with irrefutable evidence of their own actions within the domain and of the observed actions of other parties. The regulatory mechanisms used to support a trust domain will vary according to the degree of trust between parties. For example, a more lightweight mechanism can be used when parties, who otherwise trust each other, need a verifiable audit trail of their interaction compared to the situation where parties are mutually mistrusting (and require strong fairness guarantees). Also, certain types of interaction may be inherently more trustworthy than others. For example, there may be stronger incentives to good behaviour in a long-running interaction involving update to shared information between members of a VE compared with a one-off service invocation. This observation is supported by work on the Iterative Prisoner's Dilemma [1] where the prospect of and payoff from future interaction can even induce antagonists to cooperate. Ultimately, trusted interceptors construct a trust domain that, under assumptions agreed between the parties to an interaction, delivers safety and liveness guarantees. Safety guarantees ensure that the interaction complies with agreements between organisations — for example, that changes to shared information are unanimously agreed. Liveness

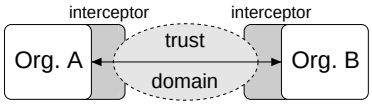
guarantees address forward progress — for example, that honest parties can resolve an exchange despite non-cooperation of dishonest parties.



(a) inline TTP trust domain



(b) distributed inline TTP trust domain



(c) direct trust domain

Figure 3. Trust domains using trusted interceptors

Figure 3 shows three approaches to the use of trusted interceptors to provide a trust domain (for simplicity, between two organisations). In both Figure 3(a) and 3(b), communication between organisations A and B is routed via Trusted Third Parties (TTP(s)). Figure 3(a) shows a single TTP acting on behalf of both organisations. Figure 3(b) is the construction of an inline TTP from TTPs acting on behalf of A and B. However constructed, the inline TTP is an interceptor between the organisations and is responsible for ensuring that agreed safety and liveness guarantees are delivered to honest parties.

The alternative to interaction through inline TTPs is the formation of a *direct* trust domain by the organisations themselves. As shown in Figure 3(c), in this case, each party to the interaction hosts its trusted interceptor. The interceptors execute protocols that deliver the guarantees required to form a trust domain appropriate to the given interaction. Depending on the relationship between organisations and the specific interaction requirements, this direct trust domain may demand the availability of one or more TTPs. These TTP(s) are not directly involved in all communication between the parties but may be called upon to resolve or abort a protocol run to deliver fairness and/or liveness guarantees to honest parties. The organisations forming a trust domain can agree on the deployment of different interceptors to deliver different fairness or reliability guarantees or to satisfy different evidentiary requirements. An advantage of the formation of a direct trust domain is that it is easier to make trade-offs between different requirements. For example, the implementation of non-repudiable information sharing described in Section 4.3 involves direct interaction between organisations without the support of a TTP. Nevertheless, as shown in [5], it has the safety property that an honest party can irrefutably assert the validity of the (agreed) state of shared information despite failure and/or misbehaviour by other parties. It has the liveness property that if no party misbehaves, agreed interactions take place despite a bounded number of temporary network and computer related failures. In effect, the risk of a loss of liveness and the resultant breakdown of an interaction leading to dispute is traded against the advantage of direct interaction between parties without the involvement of a TTP. An alternative implementation, using different interceptors, could involve a TTP to deliver a stronger liveness guarantee.

The above models for implementation of a trust domain are not mutually exclusive. One part of an interaction may deploy interceptors at trusted third parties while another uses interceptors hosted within each organisation. As an interaction evolves it may be appropriate to change the deployment of interceptors.
In the remainder of this section we describe how trusted interceptors are used to achieve regulated service invocation and information sharing. First, we enumerate the trusted interceptor assumptions (some of which are trivially met when a single TTP acts as interceptor for all parties):

1. Trusted interceptors use perfect cryptography. For example, signatures cannot be forged and encrypted data cannot be decrypted except with the appropriate decryption key.

2. The communication channel between trusted interceptors provides eventual message delivery (there is a bounded number of temporary network and computer related failures).

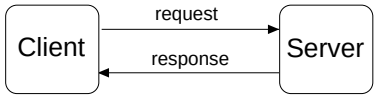
3. Trusted interceptors have persistent storage for messages (or, more precisely, evidence extracted from messages). The minimum requirement is that interceptors ensure evidence is available for as long as is necessary to meet their obligations to the other interceptors mediating an interaction. Longer term storage to protect the interests of the party on whose behalf an interceptor acts will be determined by agreement between the party and its interceptor.

4. Trusted interceptors only exchange messages that are well constructed with respect to the interaction they are mediating. For example: interceptors do not relay information provided by the organisation they represent that is invalid with respect to a given protocol execution; and messages exchanged are either tamper-resistant (encrypted), or tampering is detectable and interceptors will cooperate to ensure a well-constructed message is eventually delivered.

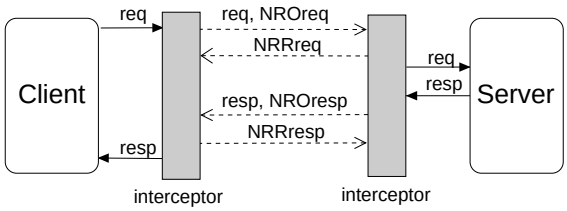
5. Trusted interceptors execute on reliable nodes or the interaction between them is made fault tolerant by employing mechanisms such as those described by Ezhilchelvan and Shrivastava [7].

Given these assumptions, trusted interceptors can cooperate to ensure fairness and liveness for honest parties to an interaction. Ultimately, since cooperation of dishonest parties cannot be enforced, the guarantee is that trusted interceptors will support the conclusion of dispute resolution in favour of honest parties. The infrastructure requirements implied by the above assumptions are discussed in Section 3.5.
The following descriptions of non-repudiation services apply to all three approaches to constructing a trust domain. In the case of a single inline TTP, trusted interceptors acting on behalf of each party are co-located and communication between them is internal to the TTP. In practice, this may mean that the interceptors are constructed from components hosted by the same application server and interfaces to interact through the interceptors are presented to participating organisations.

3.2 Non-repudiable service invocation



(a) Service invocation



(b) Non-repudiable service invocation

Figure 4. Non-repudiable service invocation

Figure 4(a) shows a typical two-party, client-server interaction. The client invokes a service by sending a request to the server who issues a response. We assume at-most-once service invocation semantics (supported by most middleware): if the client receives the response then this means that the invoked operation has been executed once; if no response is received then the operation may or may not have been executed. Non-repudiable service invocation provides the following additional assurances to the client: (1) that following an attempt to submit a request to a server, either: (a) the submission failed and the server did not receive the request; or (b) 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 produced the response. For the server, the corresponding assurances are: (1) that if a request is received, there is proof identifying the client who submitted the request; and: (2) that following an attempt to deliver a response to the client, either: (a) the delivery failed and the client did not receive the response; or (b) delivery succeeded and there is proof that the response is available to the client.

To provide the above assurances, trusted interceptors execute a non-repudiation protocol that ensures the following:

1. a request is passed to a server if, and only if, the client (or its interceptor) provides non-repudiation evidence of the origin of the request (*NROreq*) **and** the server (or its interceptor) provides non-repudiation evidence of receipt of the request (*NRRreq*)

2. the response is passed to the client if, and only if, the server (or its interceptor) provides non-repudiation evidence of the origin of the result (*NROresp*) **and** the client (or its interceptor) provides non-repudiation evidence of receipt of the response (*NRRresp*).

Non-repudiation tokens include a unique request identifier, to distinguish between protocol runs and to bind protocol steps to a run, and a signature on a secure hash of the evidence generated. Figure 4(b) models the exchange of evidence achieved by the execution of an appropriate non-repudiation protocol between interceptors acting on behalf of client and server. The client initiates a request for some service. The client's interceptor generates an *NROreq* token and then sends both the request and the token to the server's interceptor. The server's interceptor generates an *NRRreq* token and returns it to the client's interceptor. The server's interceptor then passes the request to the server to generate a response. On receipt of the response, the server's interceptor generates an

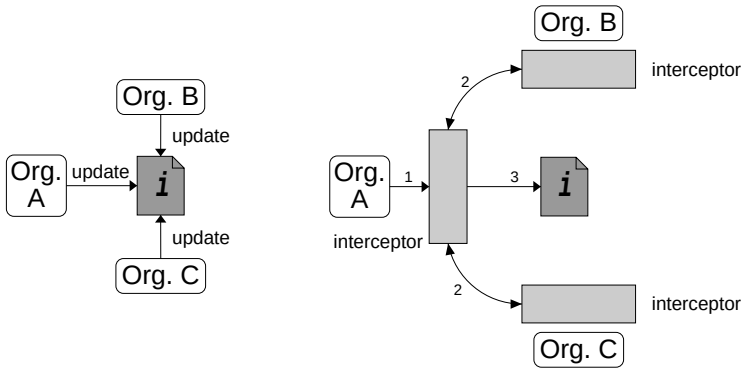
NROresp token and sends both the response and the token to the client's interceptor. As noted in Section 3.1, the interceptors are responsible for verification and persistence of evidence generated during the exchange. The exact meaning of generation of non-repudiation evidence will be dependent on the actual protocol used to execute the exchange. Client and server may sign evidence, or their interceptors may sign on their behalf, or, as with some fair exchange protocols, a combination of client/server signing in the normal case and TTP signing in case of recovery will be used. Minimally, the interceptors ensure that irrefutable evidence of the exchange is generated.

Assuming the server-side response (*resp*) includes evidence as to whether the request was made available to the server, the above model of the interaction between client interceptor and server interceptor can be simplified to:

client interceptor → *server interceptor* : *req, NROreq*
server interceptor → *client interceptor* : *resp, NRRreq, NROreq*
client interceptor → *server interceptor* : *NRRresp*

If the request was made available to the server, then resp is either the result of normal execution of the request at the server or interceptor-generated evidence that the request failed or that the server did not respond within some agreed timeout or that the client initiated an abort of the request before a result was available. If the request was not made available to the server, then resp indicates that the request was received but not executed. Similarly, the client-side receipt for the server-side response, *NRRresp*, may include evidence as to the client's consumption of the response. For example, if the interceptor can prevent access to the result of the server's execution of the client's request, then the *NRRresp* can indicate that the response was received but not consumed by the client. This equates to at-most-once semantics where a server may do work on behalf of a client that is not consumed. Given these semantics, the client may fail or timeout and the server will receive evidence that a result was generated that the client did not consume.

3.3 Non-repudiable information sharing



(a) Information sharing (b) Non-repudiable information sharing

Figure 5. Non-repudiable information sharing

Figure 5(a) shows three organisations (A, B and C) accessing and updating shared information. If, for example, A wishes to update the information, then they must reach agreement with B and C on the validity of the proposed update. For the agreement to be non-repudiable: (i) B and C require evidence that the update originated at A; and (ii) A, B and C require evidence that, after reaching a decision on the update, all parties have a consistent view of the agreed state of the shared information. The latter condition

implies that there must be evidence that all parties received the update **and** all parties know whether there was unanimous agreement to it being applied to the information.
Figure 5(b) shows A proposing an update to the information shared by A, B and C. Interceptors are used to mediate each organisation's access to the information. In step 1, A attempts an update to the information. A's interceptor intercepts the update and, in step 2, executes a non-repudiable state coordination protocol with B and C to achieve the following:

1. That A's update is irrefutably attributable to A and proposed to B and C.

2. That B and C independently validate A's proposed update, using a locally determined and application-specific process, and their respective decisions are made available to A and are irrefutably attributable to B and C.

3. That the collective decision on the validity of the update (in this case, responses from B and C to A) are made available to all parties (A, B and C).

If the resolution of the protocol executed at step 2 represents agreement to the update then the shared information is updated in step 3. Otherwise, the information remains in the state prior to A's proposed update. Non-repudiable connect and disconnect protocols govern changes to the membership of the group of organisations sharing the information.
Our previous work on B2BObjects [5] presents a realisation of the above abstraction of regulated information sharing. The paper gives a detailed description of a non-repudiable state coordination protocol used to reach agreement on update to shared information that offers the liveness and safety guarantees discussed in Section 3.1 A Java RMI-based implementation of B2BObjects is also described. This implementation is the starting point for the component middleware support for regulated information sharing described in Section 4.3.
As with non-repudiable service invocation, the use of interceptor's allows us to abstract away the details of state coordination and insulate the application from protocol specifics. From the application viewpoint, the update to shared information is an atomic action that succeeds or fails dependent on the agreement of the parties sharing the information. Thus the interceptors may execute any protocol that achieves non-repudiable agreement on: the origin and state of a proposed update; the state of the shared information after application of an update; and the membership of the group that agreed to, or vetoed, the update.

3.4 Evidence generation requirements

To meet non-repudiation requirements the evidence generated, and signed, during service invocation or update to shared information must be in a form that cannot be subsequently disputed. For non-repudiable service invocation, the requirement is that a meaningful snapshot of the invocation is signed and stored. An invocation has two parts: (i) the request comprising the service invoked, identified by a globally resolvable name such as a Uniform Resource Identifier (URI), and any parameters to the request, and (ii) the result of the invocation. For both the parameters to the invocation and the result, there are three different types to consider.

1. **value types**, or references to local objects, must be resolved to an agreed representation of their state at invocation (or at response for the result).

2. **service references** must be resolved to a meaningful, agreed representation of the service such as a URI.

3. **shared information** must be resolved both to a representation of the state of the information and a reference to the mechanism for sharing the information that is resolvable by the remote party. The combination of this evidence allows the remote party to determine the state of the shared information at invocation time and also to access the shared information locally after the invocation has completed.

For non-repudiable information sharing, the main requirements are: (i) that an agreed representation of information state is stored; and (ii) that there can be no dispute that a subsequent reconstruction of information state is a state previously agreed by the organisations who share the information.

3.5 Infrastructure requirements

Trusted interceptors require the following underlying services:

- Cryptographic primitives [20]: a signature scheme such that signature $sig_A(x)$ by A on data x is both verifiable and unforgeable; a secure (one-way and collision-resistant) hash function; and a secure pseudo-random sequence generator to generate statistically random and unpredictable sequences of bits. Random numbers are used to generate unique identifiers and random authenticators during non-repudiation protocols.

- Credential (certificate) management: a service to support signature verification that stores certificates and certificate revocation information, and can be used to verify certificate chains.

- Time-stamping: non-repudiation evidence should be time-stamped for logging and to support the assertion that the signature used to sign evidence was not compromised at time of use [26]. Recently, forward-secure signature schemes have been proposed that obviate the need for a third party signature on time-stamps [25].

- Persistence: persistence services are required both to log non-repudiation evidence and to store the state of invocation parameters/results and of shared information. Non-repudiation evidence will include a signed secure digest of state that is held in a state store. Persistence services should support the mapping of the state digest to the representation of state in the state store.

- Access control: to map credentials to roles between organisations. The exchange of credentials at first connection to shared information or on service invocation can be used as hooks to trigger the mapping of credentials to roles in a virtual enterprise. In this area, there is considerable existing work on credential exchange [11, 24]. An approach that seems fruitful is Cambridge's event-based access control system [2] where roles are activated, based on credentials presented, and de-activated in response to events in the system or changes in the environment.

- Membership service: for information sharing, the membership of the group that shares information must be identified. It must also be possible to map member identifiers (for example, URIs) to credentials in the credential management service.

4. Component-based implementation

This section presents a component middleware implementation of the services described in Section 3. The implementation is based on a J2EE application server. J2EE applications are assembled from components (self-contained software units). The components include Enterprise JavaBeans (EJBs) that are deployed on an application server. EJBs run in an environment called an EJB container. Together, the server and container provide a bean's runtime environment. The container intercepts remote invocations on the bean and is responsible for invoking appropriate low-level services, such as persistence and transaction management, for each operation on the bean. The application programmer concentrates on the functional (business logic) aspects of a bean's behaviour while the container provides services to ensure correct, non-functional behaviour.

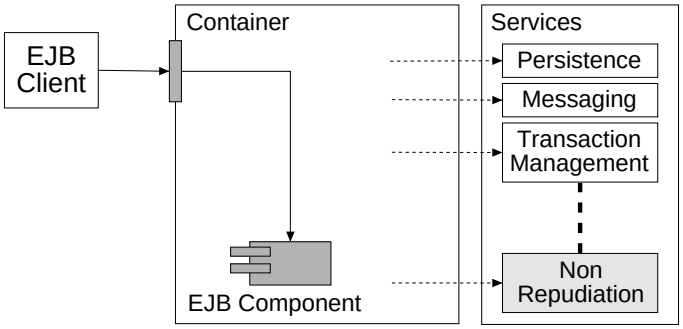


Figure 6. J2EE-based component architecture with non-repudiation

Figure 6 shows an EJB client invoking an operation on an EJB component and the container interception of the invocation to provide various services. As shown, the intention is to add a non-repudiation service to regulate access to EJBs.

Our prototype extends the JBoss J2EE application server [8]. JBoss makes systematic use of reflection and invocation path interceptors to support extension to its existing services and the addition of new services. This provides a straightforward mechanism for the implementation of the trusted interceptors introduced in Section 3. Although this exploits JBoss-specific mechanisms, similar support is found in other component-based systems (for example, the use of interceptors in the Jironde flexible transaction framework [19]). Furthermore, even when the introduction of new interceptors is not directly supported by a component system, the well-known smart proxy design pattern [9] can be followed to introduce a layer between application clients and application server components. An example of this approach is the use of smart proxies to support on-line upgrades to component systems [17].

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 completing 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 called, the method parameters, the target bean and its deployment descriptor. JBoss provides interceptors both at the server and the client (using a

dynamic proxy). Thus the mechanism supports the execution of additional logic at the client-side on behalf of a container-level service.
The prototype implementation uses JBoss interceptors to access our non-repudiation middleware that uses a generic B2BCoordinator service for the exchange of protocol messages. Custom protocol handlers are registered with the coordinator to execute non-repudiation protocols. The coordinator service also provides access to generic services that support execution of protocols (such as credential management and state storage). The combination of generic coordinator service and custom protocol handlers provides a middleware that is adaptable to different application requirements, for example to execute different protocols and to support the different interaction styles described in Section 3.1
The implementations are based on the direct trusted interceptor interaction shown in Figure 3(c). Furthermore, no TTP is used to support protocol execution. Thus, the implementation of service invocation guarantees safety and liveness if client and server satisfy the trusted interceptor assumptions. The implementation of information sharing guarantees: (i) no invalid changes to shared information whatever the behaviour of participants, and (ii) liveness if all parties satisfy the trusted interceptor assumptions. The flexibility inherent in our approach means that we can transform these implementations by introducing a TTP to support execution of fault-tolerant fair exchange protocols of the kind described in [7]. This transformation would then allow us to relax the strong assumptions about the parties to the interaction.

4.1 B2BCoordinator service and protocol handlers

Each trusted interceptor provides a B2BCoordinator service for the exchange of messages with other trusted interceptors. In the J2EE implementation, this service is exported as a remote object that remote trusted interceptors make invocations on to deliver messages. This service is the external entry point for execution of non-repudiation protocols. The interface is:

```
B2BCoordinatorRemote {
    void deliver(B2BProtocolMessage msg);
    B2BProtocolMessage deliverRequest(B2BProtocolMessage msg);
}
```

Remote invocation of `deliver` results in delivery of the given message (as a parameter to the call) from the remote party. `deliver` can be used for synchronous or asynchronous protocol execution. `deliverRequest` is a convenience method that allows a remote party to deliver a message and then to wait synchronously for a response (the result of the call). A B2BProtocolMessage is an interface to information common to non-repudiation protocol messages — request (protocol run) identifier, sender, protocol step, signed content, payload etc. Concrete implementations of B2BProtocolMessage meet protocol-specific requirements.
To execute specific protocols, and meet different application or platform requirements, custom protocol handlers are registered with the coordinator service. The coordinator is responsible for mapping an incoming protocol message to an appropriate handler. The coordinator also provides access to local services that are not protocol or platform specific. All protocol handlers provide the following interface to the local coordinator service to process incoming messages:

```
B2BProtocolHandler {
    void process(B2BProtocolMessage msg);
```

```
    B2BProtocolMessage processRequest(B2BProtocolMessage msg);
}
```

Protocol handlers use the coordinator service provided by remote parties to deliver outgoing protocol messages. As discussed below, for non-repudiable service invocation, a B2BInvocationHandler initiates protocol execution by an appropriate protocol handler. For non-repudiable information sharing, a B2BObjectController initiates protocol execution.

4.2 Implementation of non-repudiable service invocation

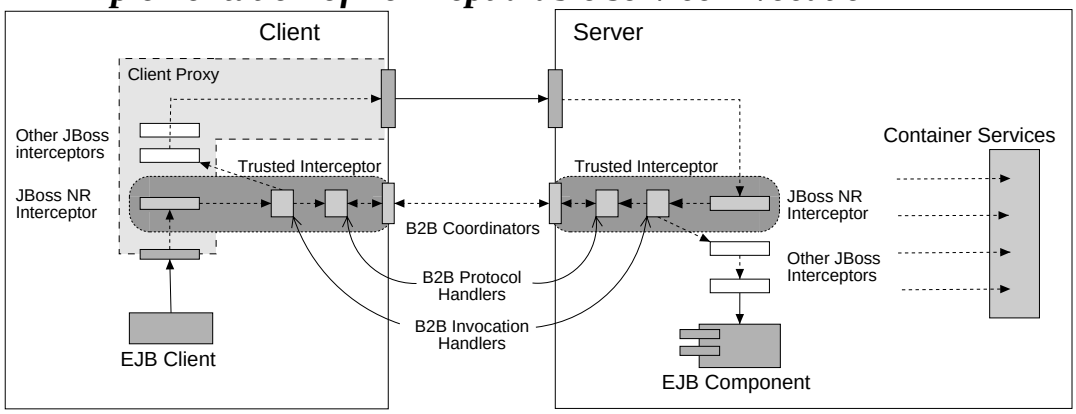


Figure 7. JBoss/J2EE-based implementation of non-repudiable invocation

In J2EE, service invocation equates to the remote invocation of an operation on an enterprise bean. As shown in Figure 7, the JBoss facility for server- and client-side interceptors is used to render the operation non-repudiable. The client's reference to the remote bean is a dynamic proxy generated by the server. This proxy contains client-side interceptors that are typically used for context propagation. We add an extra interceptor — the JBoss NR interceptor — to both client and server invocation paths. These NR interceptors are responsible for triggering execution of a non-repudiation protocol that achieves the exchange described in Section 32. The client-side NR 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.

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 NR interceptor is the first in the chain on the outgoing path (and last on the return path). On the server-side, to verify the integrity of the request as it entered the server and to provide non-repudiation of the response as it leaves the server, the NR interceptor is the first in the chain on the incoming path (the last on the return path).

Each JBoss interceptor has an `invoke` operation that takes an Invocation object* 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 `invoke` operation of the client-side JBoss NR interceptor is:

```
Public Object invoke(Invocation inv) {
    B2BInvocationHandler b2bInvHdlr =
```

* an encapsulation of the client's service invocation, include contextual information and related payload

```
            B2BInvocationHandler.getInstance("JBossJ2EE", "direct");
      B2BInvocation b2bInv =
            new JBossB2BInvocation(nextInterceptor(), inv);
      Return b2bInvHdlr.invoke(b2bInv); }
```

`getInstance` is a factory method that returns a reference to a B2BInvocationHandler for the given platform ("JBossJ2EE") to execute the given protocol ("direct"). The concrete implementation of a B2BInvocationHandler is under control of the client. A B2BInvocation object is a generic wrapper for platform-specific representations of the service to invoke and the invocation parameter(s). For a JBossB2BInvocation, the service to invoke is the next interceptor in the chain and a JBoss Invocation object encapsulates the invocation parameters. When `invoke` is called, the general behaviour of the client-side B2BInvocationHandler is:

1. obtain a reference to or instantiate the local B2BCoordinator service;
2. obtain a reference to or instantiate a protocol handler for the given protocol and register the handler with the coordinator service;
3. request that the protocol handler execute its non-repudiation protocol using the given service and invocation parameters; and
4. return the outcome of protocol execution (normally the server's response) to the client.

To start execution of the protocol, the client-side B2BInvocationHandler replaces the arguments to the service invocation with the first message of the protocol and a reference to its local coordinator service. These are then passed up through the interceptor chain to the server. When the server-side NR interceptor receives the Invocation object, it instantiates a JBoss-specific B2BInvocationHandler object and calls the B2BInvocationHandler's `invoke` method with the Invocation object as a parameter. The general behaviour of the server-side B2BinvocationHandler is:

1. obtain a reference to or instantiate the local B2BCoordinator service;
2. obtain a reference to or instantiate a protocol handler for the type of B2BProtocolMessage encapsulated in the Invocation object and register the handler with the coordinator service; and
3. request that the protocol handler execute its non-repudiation protocol using the protocol message and remote coordinator reference (obtained from the Invocation object).

At the appropriate point during execution of the non-repudiation protocol, the client's request is actually passed through the interceptor chain to the EJB component for execution. The result of this execution is then used to complete the non-repudiation protocol.

The application programmer on the server side is responsible for identifying, in a bean's deployment descriptor, when non-repudiation is required and for identifying the platform and protocol for instantiation of the B2BInvocationHandler by the NR interceptor. Thus the server controls activation of non-repudiation. However, the client controls its own participation, through its own implementations of B2BInvocationHandler, B2BProtocolHandler and B2BCoordinator. Thus, for example, the client may change the behaviour of its B2BInvocationHandler to attempt to re-

negotiate the non-repudiation protocol to execute. As shown, the NR interceptor, B2BInvocationHandler, B2BProtocolHandler and B2BCoordinator comprise each party's trusted interceptor.

4.3 Implementation of non-repudiable information sharing

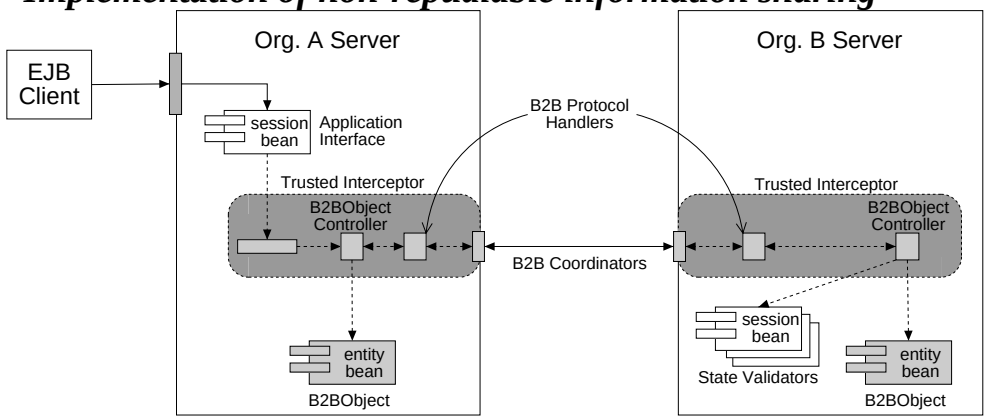


Figure 8. JBoss/J2EE-based implementation of non-repudiable information sharing

The implementation of non-repudiable information sharing is based on our previous work on B2BObjects. This provides the abstraction of shared information depicted in Figure 5(b) by coordinating the state of local (object) replicas that encapsulate the information. Figure 8 illustrates the component-based implementation 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 controller is the local interface to configuration, initiation and control of information sharing. It uses protocol handlers and a coordinator service to execute non-repudiable state and membership coordination protocols with remote parties. Implementations of the interceptor, controller, protocol handlers and coordinator are all provided by the middleware, as is the supporting infrastructure to store evidence etc. The controller uses application-specific validation listeners to validate state and membership changes proposed by remote parties. Figure 8 shows B's controller validating A's proposed update by appealing to one or more state validators (implemented as session beans). 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. Furthermore, the implementation supports sharing by more than two parties.

The middleware-provided JBoss interceptor is responsible for interaction with the B2BobjectController, and, through the controller, the B2Bobjects middleware. The application programmer is responsible for: identifying an entity bean as a B2BObject; providing configuration information in the bean's deployment descriptor (for example, to identify validator beans); and providing implementations of one or more session beans to perform validation. Optionally, the application programmer may specify that a method in the application interface should result in a series of operations on an underlying B2BObject bean being “rolled-up” 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.

5. Related work

We are not aware of other work that provides systematic integration of services for trusted interaction with component middleware. There is a Web Services non-repudiation proposal [10] that specifies a mechanism to request and send a signed receipt for a SOAP (XML-encoded) message in order to support so-called "voluntary" non-repudiation. The OASIS Digital Signature Service [18] proposes XML request/response protocols for signing, verifying and time-stamping data. The Universal Postal Union has proposed the Global Electronic Postmark [22] (EPM) standard. This is a TTP service for generation, verification, time-stamping and storage of non-repudiation evidence. The service would also support linking of evidence under a unique transaction identifier to allow business transaction events to be bound together. None of these proposals provide for the exchange of non-repudiation evidence or the governance of complex interactions. These would have to be delivered at the application level with the proposed services used as back-end infrastructure (which in the case of EPM would be provided by a TTP).

Early work by Clark and Wilson [4] on security policy stressed the importance of data integrity in the commerce domain (as opposed to the military domain's focus on disclosure). In the Clark-Wilson model constrained data items are only manipulated through verified transformation procedures as part of well-formed transactions. This ensures that transformations respect an organisation's integrity rules, for example respecting good accounting practice, and are logged for audit. The model was concerned with enforcement of policy within organisations. The use of verified transformation procedures that mediate the actions within an organisation is similar to the use of trusted interceptors as mediators between organisations.

There has been much recent work on fair exchange and fair non-repudiation, and on the formal verification of protocols. Kremer et al [12] summarise the state of the art and provide a useful classification of protocols according to types of fairness and the role of TTPs in protocols. There have also been contributions on the transformation of fair exchange [13, 7] to meet fault tolerance requirements. This body of work can be brought to bear on the choice of protocols that trusted interceptors execute to meet interaction requirements.

The work of Minsky et al on Law Governed Interaction (LGI) [15] represents one of the earliest attempts to provide coordination between autonomous organisations. Trusted agents act as mediators that comply with a global policy. This is similar to the trusted interceptor abstraction in that the interaction between agents is assumed to be legal. LGI does not address systematic non-repudiation.

Wichert et al [23] used filters in CORBA to provide non-repudiable invocation on a remote object. However, there approach is asymmetric — the client provides the server with non-repudiation of origin of a request but there is no exchange to provide corresponding evidence to the client. Their work did provide useful insights into representation of evidence in XML documents. In our system the exact representation of evidence is a matter for agreement between parties concerned, the important requirement is that the representation can be subsequently rendered meaningful and irrefutable.

6. Conclusions and future work

This paper presented a unified approach to regulated interaction based on the abstraction of trusted interceptors that mediate interactions. The component-based middleware implementation provides the basic building blocks for the construction of a composite service by organisations collaborating to form a virtual enterprise. This can be extended

to support transactional interaction. Our preliminary work in this area [6] shows how B2BObjects can participate in distributed (JTA [3]) transactions. We intend to build on this work to provide component-based transactional and non-repudiable interaction.

In effect, the trusted interceptor abstraction, and its realisation in middleware, provides a flexible framework for implementation of different approaches to non-repudiable service invocation (fair exchange) and regulated information sharing. Future work will use this framework to provide a suite of protocols and other mechanisms that can be deployed to meet different application requirements.

We intend to integrate the underlying mechanisms presented here with work on run-time monitoring of contracts [16]. Contracts are represented as executable finite state machines that can be verified using model-checking tools. We will, for example, use implementations of the verified state machines to validate changes to shared information for contract compliance.

There is a considerable body of work on Byzantine agreement and consensus in distributed systems. We will explore the relationship between this work and the problem of reaching unanimous, non-repudiable agreement on changes to shared information.

We also intend to investigate the use of Aspect Oriented Programming to allow the declaration of non-repudiation as a non-functional aspect of a service that results in support to exchange non-repudiation evidence etc.

Another area of work is the deployment of the middleware presented to render Web Service interactions non-repudiable.

Finally, we are not aware of systematic work on the performance costs of non-repudiation services (as opposed to the relative performance of cryptographic algorithms). There are a number of aspects to non-repudiation that impact on performance, including the computational overhead of cryptographic algorithms; the space overhead of evidence generated and the communication overhead of additional messages to execute protocols. Our interceptor-based framework will allow us to compare different implementations and their impact on performance.

Acknowledgements

This work is part-funded by the EU under projects IST-2001-34069: “TAPAS (Trusted and QoS-Aware Provision of Application Services)”, IST-2001-37126: ADAPT (Middleware Technologies for Adaptive and Composable Distributed Components) and by the UK EPSRC under e-Science project GR/S63199/01: “Trusted Coordination in Dynamic Virtual Organisations”. We thank our colleague Paul Ezhilchelvan for useful discussion of this work.

References

[1] R. Axelrod. *The Evolution of Co-operation*. Penguin Books, 1990.

[2] J. Bacon, K. Moody and W. Yao. Access Control and Trust in the use of Widely Distributed Services. In *Proc. IFIP/ACM Int. Middleware Conf.*, Springer LNCS 2218, Heidelberg, Germany, 2001.

[3] S. Cheung and V. Matena. *Java Transaction API (JTA version 1.0.1B).* Sun Microsystems Inc., http://java.sun.com/products/jta/index.html, 2002.

[4] D. R. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proc. IEEE Symp. on Security and Privacy*, pp. 184–194, 1987.

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

[6] N. Cook, S. Shrivastava, and S. Wheater. 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.

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

[8] M. Fleury and F. Reverbel. The JBoss Extensible Server. In *Proc. ACM/IFIP/USENIX Int. Middleware Conf.*, Springer LNCS 2672, Rio de Janeiro, Brazil, Jun 2003.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] E. Gravengaard, G. Goodale, M. Hanson, B. Roddy, and D. Walkowski. *Web Services Security: Non Repudiation Proposal Draft 05*. Reactivity,http://schemas.reactivity.com/2003/04/web-services-nonrepudiation-05.pdf, Apr 2003.

[11] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In *Proc. IEEE Symp. on Security and Privacy*, Oakland, USA, 2000.

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

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

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

[15] N. Minsky and V. Ungureanu. Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. *ACM Trans. Softw. Eng. and Methodology*, 9(3):273–305, 2000.

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

[17] S. Oberg, L. Tewksbury, L. Moser, and P. Melliar-Smith. Online Upgrades for CORBA and EJB/J2EE. In *Proc. IEEE Workshop on Dependable Middleware-Based Systems (WDMS 2002)*, Washington DC, USA, 2002.

[18] T. Perrin, D. Andivahis, J. C. Cruellas, F. Hirsch, P. Kasselman, A. Kuehne, J. Messing, T. Moses, N. Pope, R. Salz, and E. Shallow. *Digital Signature Service Core Protocols and Elements*. OASIS Committee Working Draft, http://www.oasisopen.org/committees/dss, Dec 2003.

[19] M. 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.

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

[21] Sun. *Java 2 Platform Enterprise Edition (J2EE) Specification*. Sun Microsystems Inc., http://java.sun.com/j2ee/, 1.4 edition, 2003.

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

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

[24] W. Winsborough, K. Seamons, and V. Jones. Automated Trust Negotiation. In *Proc. DARPA Inf. Survivability Conf. and Exposition*, Hilton Head, USA, 2000.

[25] B. F. Zhou, J. and R. Deng. Validating Digital signatures without TTP's Time-stamping and Certificate Revocation. In *Proc. 2003 Inf. Security Conf.*, Springer LNCS 2851, Bristol, UK, 2003.

[26] J. Zhou and D. Gollmann. Evidence and non-repudiation. *J. Network and Comp. Applications*, 20(3):267–281, 1997.