Component Middleware to Support Non-repudiable Service Interactions

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

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. 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. 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 can involve close interaction among the constituent services of participating organisations. Nevertheless, 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.

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: (i) high level mechanisms to specify and enforce contractual rights and obligations (examples include work on Law Governed Interaction [13] and on contract representation and monitoring [14]); and (ii) 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 [16]). 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.

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 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.

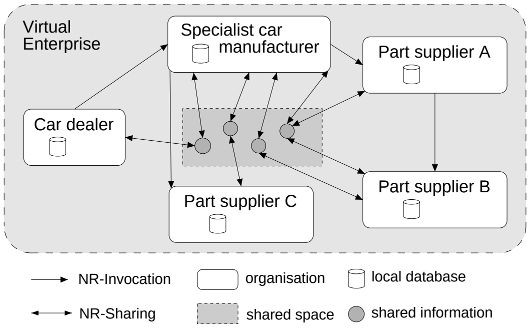


Figure 1. Specialist car manufacturer application

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 to perform actions on each other's behalf. 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 [12] 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.

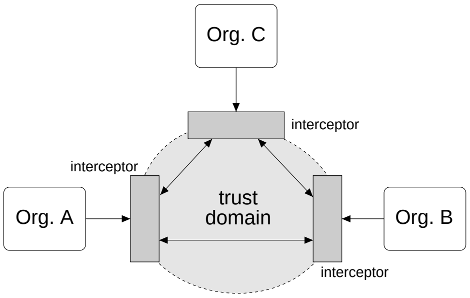
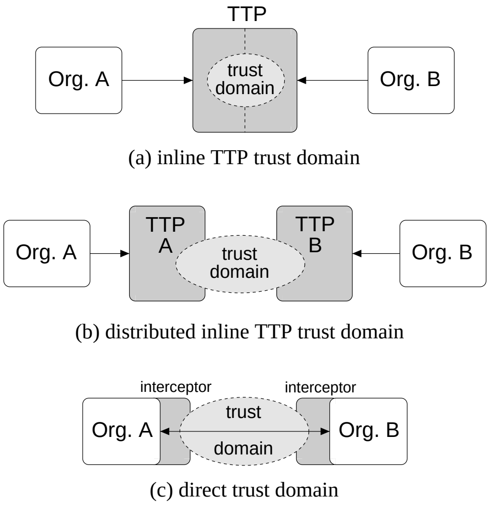


Figure 2. Trusted interceptors

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 provides 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 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.

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.

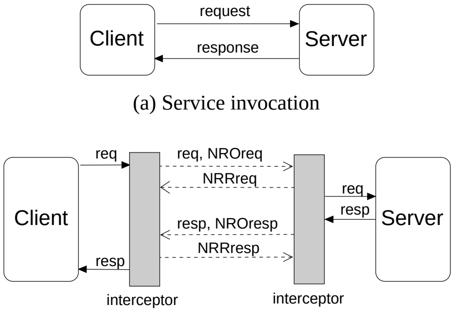
4. Trusted interceptors only exchange messages that are well constructed with respect to the interaction they are mediating. For example: interceptors do not relay information provided by the organisation they represent that is invalid with respect to a given protocol execution; and messages exchanged 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 the extended version of this paper [4].

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 response (*NROresp*) **and** the client (or its interceptor) provides non-repudiation evidence of receipt of the response (*NRResp).*

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.

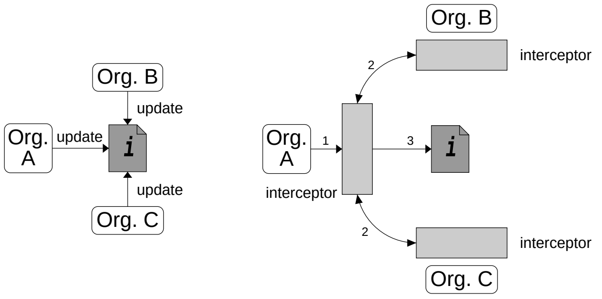
Assuming the server-side response *(resp*) includes evidence as to whether the request was made available to the server, the above model of the interaction between client interceptor (CI) and server interceptor (SI) can be simplified to:

$$\begin{array}{lcll} \mathrm{CI} & \rightarrow & \mathrm{SI} & : \ req,\, NROreq \\ \mathrm{SI} & \rightarrow & \mathrm{CI} & : \ resp,\, NRRreq,\, NROresp \\ \mathrm{CI} & \rightarrow & \mathrm{SI} & : \ NRRresp \end{array}$$

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 response was generated that the client did not consume.

3.3. 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 that they all agreed to it being applied to the information.



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

Figure 5. Non-repudiable information sharing

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.

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 generated (including details of the request, the service invoked and the response). For non-repudiable information sharing, the main requirement is that an agreed representation of information state is used for evidence generation. Additional details of the components and form of evidence are provided in the extended version of the paper [4].

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.

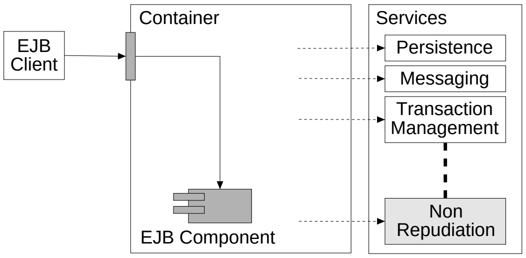


Figure 6. J2EE-based non-repudiation

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.

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 from the remote party (as a parameter to the call). `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 content that is 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

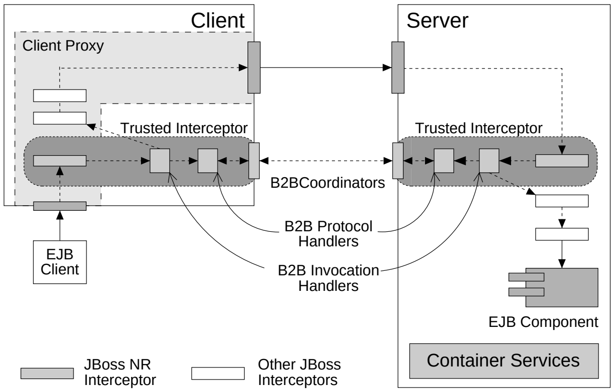


Figure 7. JBoss/J2EE-based NR-Invocation

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 3.2. 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[1] 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 =
       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: (i) obtain a reference to or instantiate the local B2BCoordinator service; (ii) obtain a reference to or instantiate a protocol handler for the given protocol and register the handler with the coordinator service; (iii) request that the protocol handler execute its non-repudiation protocol using the given service and invocation parameters; and (iv) 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: (i) obtain a reference to or instantiate the local B2BCoordinator service; (ii) 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 (iii) 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

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 as-

[1] an encapsulation of the client's service invocation, including contextual information and related payload

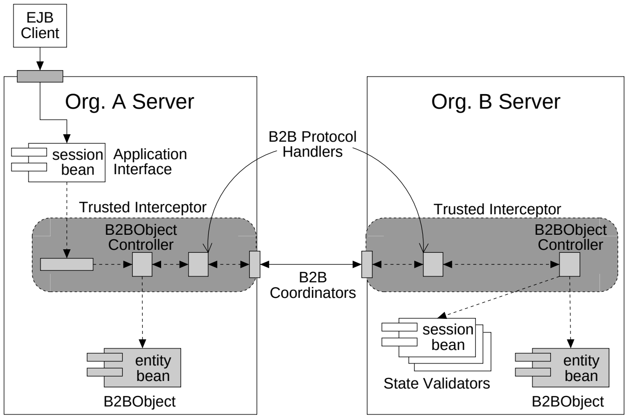


Figure 8. JBoss/J2EE-based NR-Sharing

sociated 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, with 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 can also 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 (atomic) 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 [9] 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 [15] proposes XML request/response protocols for signing, verifying and time-stamping data. The Universal Postal Union has proposed the Global Electronic Postmark [17] (EPM) standard. This is a TTP service for generation, verification, time-stamping and storage of non-repudiation evidence. 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 [3] 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 [10] summarise the state of the art and provide a useful classification of protocols according to types of fairness and the role of TTPs in protocols. There have also been contributions on the transformation of fair exchange to meet fault tolerance requirements [11, 7]. 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) [13] 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 [18] 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 is irrefutable.

6. Conclusions

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 [2]) 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 include the use of this framework to provide a suite of protocols and other mechanisms that can be deployed to meet different application requirements.

Acknowledgements

This work is part-funded by the EU under projects IST-2001-34069: "TAPAS (Trusted and QoS-Aware Provision of Application Services)" and IST-2001-37126: “ADAPT (Middleware Technologies for Adaptive and Composable Distributed Components)”; and by the UK EPSRC under grant GR/S63199: “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] 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.

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

[4] N. Cook, P. Robinson, and S. Shrivastava. Component Middleware to Support Non-repudiable Service Interactions. Technical Report CS-TR 834, School of Computing Science, Univ. Newcastle, 2004.

[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. 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-non-repudiation-05.pdf, Apr 2003.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.oasis-open.org/committees/dss, Dec 2003.

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

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

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