

# Distributed Object Middleware to Support Dependable Information Sharing between Organisations

Nick Cook, Santosh Shrivastava  
University of Newcastle  
Department of Computing Science  
Newcastle upon Tyne NE1 7RU, UK  
{nick.cook, santosh.shrivastava}@ncl.ac.uk

Stuart Wheeler  
HP Arjuna Lab  
Newcastle upon Tyne NE1 3DY, UK  
stuart\_wheater@hp.com

## Abstract

*Organisations increasingly use the Internet to offer their own services and to utilise the services of others. This naturally leads to information sharing across organisational boundaries. However, despite the requirement to share information, the autonomy and privacy requirements of organisations must not be compromised. This demands the strict policing of inter-organisational interactions. Thus there is a requirement for dependable mechanisms for information sharing between organisations that do not necessarily trust each other. The paper describes the design of a novel distributed object middleware that guarantees both safety and liveness in this context. The safety property ensures that local policies are not compromised despite failures and/or misbehaviour by other parties. The liveness property ensures that, if no party misbehaves, agreed interactions will take place despite a bounded number of temporary network and computer related failures. The paper describes a prototype implementation with example applications.*

## 1. Introduction

Organisations increasingly use the Internet to offer their own services and to utilise the services of others. This naturally leads to multi-party information sharing across organisational boundaries. A trend that is reinforced by concentration on core business and the “out-sourcing” of non-core operations to external organisations. However, despite the requirement to share information, the autonomy and privacy requirements of organisations must not be compromised. This demands the strict policing of inter-organisational interactions. Thus the requirement is for dependable mechanisms for information sharing between organisations that do not necessarily trust each other.

This paper describes the design of a novel distributed object middleware that guarantees both safety and liveness in the above context. It is assumed that each organisation has a local set of policies for information sharing that is consistent with the overall information sharing agreement (business contract) between the organisations. The safety property ensures that local policies of an organisation are not compromised despite failures and/or misbehaviour by other parties. In essence, the middleware facilitates regulated information sharing through multi-party coordination protocols for non-repudiable access to and validation of shared state. The liveness property ensures that, if no party misbehaves, agreed interactions will take place despite a bounded number of temporary network and computer related failures.

Section 2 sketches three scenarios from which requirements are derived. Section 3 provides an overview of the distributed object middleware we call *B2BObjects*<sup>1</sup>. Coordination protocols are discussed in detail in Section 4. Section 5 presents the Application Programmer Interface (API), a prototype implementation and two proof-of-concept applications that use it. Related work is surveyed in Section 6. The paper concludes with a discussion of future work.

## 2. Application requirements

In this section we sketch three different application scenarios from which we derive requirements on middleware support for dependable information sharing between organisations.

**1. Order processing.** The process of ordering goods or services includes: requisition; agreement; delivery and payment. The process must be validated to ensure that organisational policy is adhered to (for example, that a customer is credit-worthy) and that agreements between the parties are observed (for example, that the supplier does not arbitrarily modify an order). There is also a requirement that payment

<sup>1</sup>subject of a patent application by Hewlett Packard

is made if and only if the items or services ordered are delivered. For simple orders, this last aspect of the process is the most significant. When the ordering process is more complex, requisition and agreement can acquire greater significance. Requisition may include a procurement process involving multiple parties; there may be a need to negotiate non-standard terms and conditions; order fulfillment may entail commitments from more than one supplier or from delivery agents; or the order may govern delivery of an ongoing service that should itself be regulated. In these cases it can be argued that business is better supported if the organisations involved are able to share the order and related agreements. This requires that all interested parties validate updates to the shared information.

## 2. Dispersal of operational support to the customer.

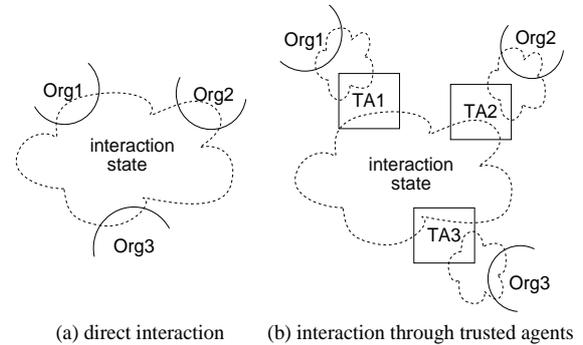
In the telecommunications industry, Operational Support Systems (OSS) manage service configuration and fault-handling on the customer's behalf [9]. For the most part, existing OSS are monolithic and centralised. Customers have little or no direct control over critical business processes that are carried out for them by the service provider. With the advent of more sophisticated services, the customer needs to be able to tailor their complete service. This requires the "dispersal of OSS" so that the customer controls the aspects that logically belong to them. The resultant devolution of processes and information allows business relationships to evolve to the benefit of all involved. To fulfill this promise, there is a requirement for regulated information sharing between the organisations.

**3. Distributed auction service.** In this scenario, autonomous, geographically dispersed auction houses wish to collaborate to deliver a trusted, distributed auction service to their clients (buyers and sellers). The clients act upon the state of an auction through servers that are controlled by the auction houses. These servers share and update auction state. The clients expect the service to guarantee the same chance of a successful outcome irrespective of which individual server is used. In effect, the auction houses are providing a distributed trusted third party (TTP) service to deliver a regulated market-place for buyers and sellers. The auction houses wish to maintain a long-lived, successful service and, therefore, continued interaction.

Each of the above examples entails multi-party interaction and information sharing. For each party the overarching requirements are: (i) that their own actions on shared information meet locally determined, evaluated and enforced policy; and that their legitimate actions are acknowledged and accepted by the other parties; and (ii) that the actions of the other parties comply with agreed rules and are irrefutably attributable to those parties. These requirements imply the collection, and verification, of non-repudiable evidence of the actions of parties who share and update information. If middleware is provided that presents the abstrac-

tion of shared (interaction) state, then the requirements can be met by regulating, and recording, access and update to that state.

Figure 1 shows two interaction styles. In the first (1a), or-



**Figure 1. Direct vs. indirect interaction styles**

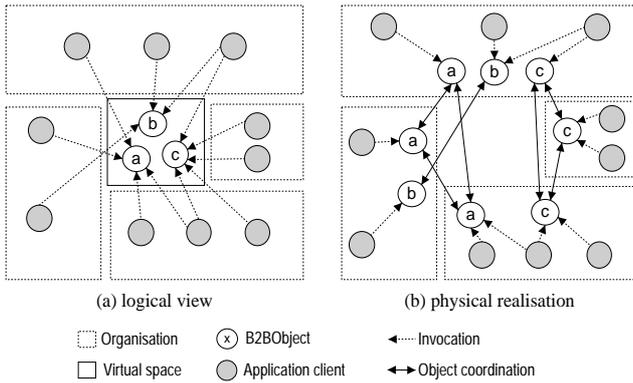
rganisations ( $Org_i$ ) disclose state and interact directly. In the second (1b), state disclosure is conditional and interaction is conducted via trusted agents ( $TA_i$ ). It is possible to envisage circumstances where both styles will be used: there may be an initial direct interaction to agree trusted agents before continuing the interaction through those agents; or relationships between organisations may change in such a way that indirect interaction evolves to direct interaction. The dotted clouds in Figure 1 represent the deployment of B2BObjects middleware to meet the application requirements outlined above. For simplicity, in the rest of the paper direct interaction (1a) is assumed unless stated otherwise.

## 3. Overview of B2BObjects middleware

This section gives an overview of the B2BObjects middleware that is designed to address the requirement for information sharing between organisations. Detailed discussion of coordination protocols is deferred to Section 4. The API and a prototype implementation are described in Section 5.

B2BObjects provides non-repudiable coordination of the state of object replicas. State changes are subject to a locally evaluated validation process. State validation is application-specific and may be arbitrarily complex (and may involve back-end processes at each organisation). Coordination protocols provide multi-party agreement on access to and validation of state. As shown in Figure 2, the logical view of shared objects in a virtual space (2a) is realised by the regulated coordination of actions on object replicas held at each organisation (2b).

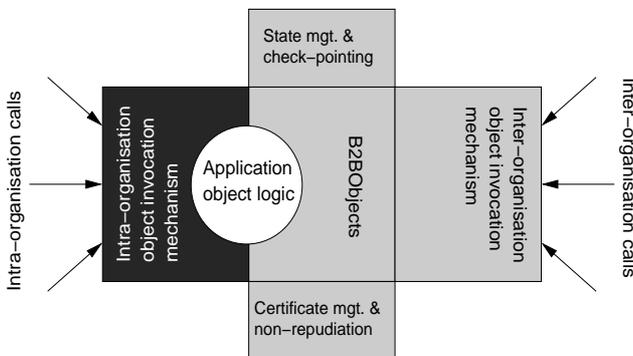
Multi-party validation of state changes supports the notion of the "joint ownership" of shared state. A state change



**Figure 2. B2BObjects interaction**

proposal comprises the new state and the proposer’s signature on that state. The proposal is dispatched to all other parties for local validation. Each recipient produces a response comprising a signed receipt and a signed decision on the (local) validity of the state change. All parties receive each response and a new state is valid if the collective decision is unanimous agreement to the change. The signing of evidence generated during state validation binds the evidence to the relevant key-holder. Evidence is stored systematically in local non-repudiation logs.

B2BObjects supports the evolution of enterprise applications to inter-organisation applications. Although an object’s implementation is augmented, the application-level use of the object may remain unchanged. Figure 3 depicts this augmentation of an application object. Calls to



**Figure 3. B2BObjects augmentation**

the object are mediated by the middleware. The intra-enterprise object invocation mechanism provides an internal interface to the object that guarantees that state changes are coordinated with other organisations through the inter-organisation object invocation mechanism. Systematic check-pointing of object state upon installation of a newly-validated state allows recovery in the event of general failures and rollback in the event of invalidation. The certificate

management and non-repudiation services provide: authentication of access to objects; verification of signatures to actions on objects; and logging of evidence of those actions. In summary, augmentation with B2BObjects provides: connection authentication and management; coordination and validation of state changes; persistence of both validated object state and of the information required to reach validation decisions; and the logging of non-repudiation evidence.

The careful separation of concerns means that the middleware can be configured to different application requirements; to suit a variety of interaction styles; and to use different underlying services (for example, to operate in synchronous, deferred synchronous or asynchronous communication modes).

## 4. B2BObjects coordination protocols

This section provides a detailed discussion of the state coordination protocol at the heart of B2BObjects. A discussion of the guarantees provided by the protocol is followed by the assumptions and notation that apply to its description. The protocol description includes modifications to protocol messages to validate update to, as opposed to overwrite of, object state. An informal analysis of protocol vulnerabilities follows the description. The discussion is in terms of a single object but applies just as well to the use of a composite object to coordinate the states of multiple objects.

The middleware also uses connection and disconnection protocols to manage membership of the group that has access to an object. These protocols are described in Section 4.5.

### 4.1. Protocol guarantees

The state coordination protocol regulates overwrites to the state of object replicas by validating state transitions. A proposed new state is valid if all parties who share the object have validated the transition to that state. A proposed state is invalid if any party has invalidated (vetoed) the transition. The notion of valid state is necessarily self-contained: a state is valid if it has been unanimously agreed; invalid otherwise.

The guarantees offered by the protocol relate to reaching agreement on a state transition; to the inability to misrepresent that agreement; and, therefore, to the inability to misrepresent the validity of object state. The safety guarantee is that invalid state cannot under any circumstances be imposed on a local object replica and that evidence is generated to ensure that the actions of honest parties cannot be misrepresented by dishonest parties. If all parties behave correctly, liveness is guaranteed despite a bounded number of temporary failures. The protocol generates evidence to

detect misbehaviour. It is assumed that, if necessary, this evidence can be used in *extra-protocol* arbitration to resolve disputes. Specific guarantees are detailed after clarifying what the protocol does not guarantee.

- Amongst the parties who share an object, there is no protection against the disclosure of a proposed state change to the object. State must be disclosed to be validated. As illustrated in Figure 1b, trusted agents can be used to provide conditional state disclosure. An example of this type of interaction is given in Section 5.1.
- There is no guarantee of termination when parties misbehave. The protocol is concerned both with verification of the integrity of messages and with the semantic validation of message content (a proposed state change). This exacerbates the problem of guaranteeing termination since, for example, we do not deduce anything about the validity of a state change from a failure to respond to a proposal. The protocol is designed to generate the evidence necessary for application-level resolution of the resultant blocking. The provision of stronger termination guarantees is discussed in Section 7.

The specific guarantees are:

- that a state transition proposal is irrefutably bound to its source and to the decisions of the parties validating the proposal; and that those decisions cannot be misrepresented and are irrefutably bound to their source;
- that irrefutable evidence of who participated in a protocol is generated;
- that no party can misrepresent the validity of object state, either by claiming that an invalid (vetoed) state is valid or that a valid (unanimously agreed) state is invalid; and
- that the protocol is *fail-safe*: faults or misbehaviour may result in the abort or blocking of a protocol run but cannot result in the installation of invalid object state at a correctly behaving party.

## 4.2. Assumptions and notation

It is assumed that the communications infrastructure provides eventual, once-only message delivery. If the underlying communications system does not support these semantics then the coordination middleware masks this and presents the assumed semantics. There is no requirement for the communications system to order messages. Network partitions are assumed to heal eventually. Nodes may crash but it is assumed that they will eventually recover and resume participation in a protocol run. For non-repudiation,

and recovery, protocol messages are held in local persistent storage at sender and recipient.

To generate non-repudiation evidence, each party has access to the following cryptographic primitives [11]: 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,  $h()$ ; and a secure pseudo-random sequence generator to generate statistically random and unpredictable sequences of bits. All parties are assumed to have the means to verify each other's signatures. Since a signature is only valid if it can be asserted that the signing key was not compromised at the time of use, all signed evidence must be time-stamped [15]. It is assumed that a trusted time-stamping service, or services, acceptable to all parties is available to each party to generate time-stamps. Given a message  $m = \{ev, sig_A(ev)\}$  by party  $A$ , a time-stamping service,  $TTS$ , will provide the following time-stamp as evidence of its generation at time  $T_g$ :  $\{T_g, sig_{TTS}(h(m), T_g)\}$ . For brevity, time-stamps are not shown in protocol descriptions.

The different roles in  $n$ -party coordination of shared object state are distinguished as follows:

$Pset = \{P_i \mid i \in 1 : n\}$  is the set of participants

$P_k \in Pset$  is a proposer of new state

$Rset_k = \{P_j \mid j \in 1 : n \text{ and } j \neq k\}$  is the set of recipients of  $P_k$ 's proposal

$Gid_i$  is the group identifier of  $Pset$  as viewed by  $P_i$ . It is computed when the membership of  $Pset$  changes (see Section 4.5). Inconsistent group identifiers lead to invalidation of a proposal.

The state of an object is uniquely identified by a tuple:  $\langle seqno, h(rn), h(S) \rangle$ ; where  $seqno$  is a proposal sequence number,  $h(rn)$  is a hash of a random number, and  $h(S)$  is a hash of the state to which the tuple refers. All of these are generated locally by the proposer. The proposer creates a new sequence number by incrementing the sequence number of the last known coordination request. Thus, the sequence number of any proposed state is guaranteed to be greater than that of any agreed state and of any coordination request seen by the proposer. The combination of sequence number and hash of the random number disambiguates concurrent proposals and guarantees the uniqueness of the tuple. The hash of the state binds the tuple to the state identified to enable checks on the integrity of the tuple with respect to that state.

There are three tuples of interest:

$NSid_k = \langle seqno, h(rn_k), h(NS_k) \rangle$  is the tuple that identifies the **new state**,  $NS_k$ , proposed by  $P_k$  ( $rn_k$  is a random number generated by  $P_k$ )

$ASid_i$  is the tuple that identifies the **agreed state**,  
 $AS_i$ , as viewed by  $P_i$

$CSid_i$  is the tuple that identifies the **current state**,  $CS_i$ , as viewed by  $P_i$

To ensure ordered state transitions, the following invariants should hold during a protocol run:

1. for  $Rset_k$ :  $CSid_j = ASid_j = ASid_k$  (their current state is the agreed state as viewed by themselves and by  $P_k$ )
2. for  $P_k$ :  $CSid_k = NSid_k$  (its current state is the proposed state)
3. for  $Pset$ :  $NSid_k.seqno > ASid_i.seqno$  (which follows from generation of  $seqno$  and invariant 1)
4. for  $Pset$ :  $NSid_k$  is unique for all proposals seen

Breaches of these invariants are detected during a protocol run and lead to invalidation of a proposed state transition.

The following notation is used in addition to the above:

$sig_i(x)$  is  $P_i$ 's signature on value  $x$

$D_{k,i}$  is  $P_i$ 's decision on the validity of a state transition proposed by  $P_k$ . A decision is *accept* or *reject* plus optional diagnostic information.  $D_i$  is used as shorthand for  $D_{k,i}$  if the proposal to which it relates is unambiguous. ( $D_{k,k}$  is, by definition, *accept*.)

$P_k \rightarrow Rset_k : m$  means  $P_k$  sends message  $m$  to each member of  $Rset_k$

$Rset_k \rightarrow P_k : m_j$  means each member,  $P_j$ , of  $Rset_k$  sends a message of type  $m$  to  $P_k$

$\sum m_i$  is the concatenation of a set of messages, or parts of messages, of type  $m$

### 4.3. Protocol description

In essence, the state coordination protocol provides non-repudiable two-phase commit. However, the messages exchanged have a richer semantics than could be derived from simply signing and counter-signing two-phase commit messages. The proposer is committed to acceptance of the new state at initiation of a protocol run.  $P_k$  cannot later unilaterally reject the state transition. A state transition is only rejected if it is vetoed by one or more members of  $Rset_k$ . The final commit message represents more than  $P_k$ 's confirmation of whether to accept or reject the new state. It is the non-repudiable decision of  $Pset$  on the validity of the proposed state transition. The protocol has three steps:

$propose \quad P_k \rightarrow Rset_k \quad : \quad mp$   
 $respond \quad Rset_k \rightarrow P_k \quad : \quad mr_j$   
 $commit \quad P_k \rightarrow Rset_k \quad : \quad mc$

where :

$mp = \{prop, NS_k, sig_k(h(prop))\}$   
 $prop = \{P_k, Gid_k, ASid_k, NSid_k\}$   
 $mr_j = \{resp_j, sig_j(h(resp_j)), sig_j(h(prop))\}$   
 $resp_j = \{P_j, D_j, Gid_j, CSid_j, ASid_j, h(NS_k)\}$   
 $mc = \{rn_k, \sum mr_j\}$

Message  $mp$  comprises: a proposal, the proposed new state and  $P_k$ 's signature on the proposal. A proposal identifies  $P_k$  and  $Pset$  (to verify a consistent view of the group), and specifies the proposed state transition from  $AS_k$  to  $NS_k$ .  $h(rn_k)$ , sent as part of  $NSid_k$ , is  $P_k$ 's commitment to the random authenticator,  $rn_k$ , of the group's decision.

Message  $mr_j$  is a receipt from  $P_j$  for the proposal and a signed decision,  $D_j$ , on its validity. Inclusion of  $Gid_j$ ,  $CSid_j$  and  $ASid_j$  permits systematic consistency checks. The hash  $h(NS_k)$  represents  $P_j$ 's assertion of the integrity (or otherwise) of  $NS_k$  with respect to the hash sent as part of  $NSid_k$ .

Message  $mc$  is the aggregation of all decisions and of the non-repudiation evidence in the form of signed proposals and responses. Any party can compute the group's decision over  $\sum resp_j$  and  $prop$ .  $mc$  requires no signature since only  $P_k$  can produce the authenticator  $rn_k$ .  $mc$  is linked to the other messages in the same protocol run through the authenticator and the concatenated, signed responses.

$Pset$ 's authenticated decision on  $P_k$ 's proposal is:

$$rn_k, prop, \sum resp_j, \sum sig_i(h(prop)), \sum sig_j(h(resp_j)) \\ i \in \{1 : n\}; j \in \{1 : n \text{ and } j \neq k\}$$

This is non-repudiable evidence of acceptance or rejection of a proposed state transition and of the consistency, or otherwise, of the information provided during a protocol run. A successful protocol run allows the consistent installation of a new, validated object state at all replicas. An unsuccessful run results in the consistent view that a proposed state is invalid. In this case, the replicas remain in the state last agreed by all parties (and the proposer can rollback to that state).

#### 4.3.1 Modifications for state update

To allow for update to, as opposed to overwrite of, object state, the propose and respond messages are modified as follows:

$mp = \{prop, US_k, sig_k(h(prop))\}$   
 $prop = \{P_k, Gid_k, ASid_k, NSid_k, h(US_k)\}$   
 $resp_j = \{P_j, D_j, Gid_j, CSid_j, ASid_j, h(US_k)\}$

In message  $mp$ , the update,  $US_k$ , is provided along with the hash  $h(US_k)$ . A hash of the new state, after application of  $US_k$ , is still provided as part of  $NSid_k$ . It is therefore possible for members of  $Pset$  to determine that, if the update is agreed and applied, a consistent new state will result.

#### 4.4. Protocol analysis

We now present an informal analysis to support the assertion of the safety guarantee in Section 4.1. To deliver the guarantee, the protocol must withstand subversion by members of  $Pset$ , whether through deliberate or accidental fault, as well as by intruders.

Any attempt to subvert a protocol run by generating inconsistent message content can be detected. In which case, the proposed state transition is invalidated and irrefutable evidence of misbehaviour is generated. It is possible to verify that the signed parts of protocol messages are consistent with the unsigned parts and, therefore, to detect internally inconsistent messages. It is possible to detect inconsistency between messages because all messages are linked to their predecessor(s) in a protocol run.  $NSid_k$  provides a unique label for each protocol run that is linked to each message in the run. It is therefore possible to detect any attempt to replay messages from a prior run. (Note: uniqueness refers to the tuple that identifies a state proposal ( $NSid_k$ ) and not to the proposed state ( $NS_k$ ) — it may be legitimate to propose the re-installation of an earlier state.) We now show how the protocol allows detection of other attempts at subversion by members of  $Pset$ :

- *A member of  $Pset$  omits to send a message:* If  $P_k$  does not send  $mp$  then, by definition,  $P_k$  is unable to show that the new state is valid. If a member,  $P_j$ , of  $Rset_k$  does not send  $mr_j$ , then  $P_j$  will have obtained the proposed new state without providing non-repudiable evidence of its receipt but cannot demonstrate the validity of the state. If  $P_k$  fails to send  $mc$ , then  $P_k$  will know, and can act upon, the group's decision but all members of  $Rset_k$  hold evidence that the protocol run is active. Any subsequent coordination request (whether connection, disconnection or state change) will reveal inconsistencies between state identifier tuples.
- *$P_k$  selectively sends to members of  $Rset_k$ :* If different messages are sent to different members of  $Rset_k$ , then the inconsistency will be detected in subsequent protocol steps. If  $mp$  is not sent to a subset of  $Rset_k$ , then it is not possible to reach a unanimous decision on the validity of the proposed state and  $P_k$  cannot produce a valid  $mc$  for any member of  $Rset_k$ . If  $mc$  is not sent to a subset of  $Rset_k$ , then this subset can show that the protocol run is still active. Further, any honest party who receives  $mc$  can relay it to any other member of

$Rset_k$ . Selective sending by  $P_k$  can be prevented if multicast semantics are guaranteed. In the absence of such a guarantee, members of  $Rset_k$  can detect selective sending.

- *$P_k$  proposes a null state transition:* On receipt of  $mp$  any member of  $Rset_k$  can detect that  $AS_j = NS_k$  and can reject a null state transition.

Assuming signatures are not compromised, the non-repudiation evidence generated during a protocol run binds a party to their actions (good or bad) and those actions cannot be misrepresented. An intruder in control of a member of  $Pset$  can act as a misbehaving party as outlined above. In no case can a correctly behaving party be forced to agree (and install) an invalid state.

With insecure channels between members of  $Pset$ , the well-known Dolev-Yao intruder [6] (who has full control over the network but cannot perform cryptanalysis) can obtain complete knowledge of proposed object state and of decisions with respect to proposals. In addition, they are able to modify the unsigned parts of any message. This results in inconsistent message content (dealt with above). Given secure channels between members of  $Pset$ , this intruder can only remove, delay or replay messages. With or without secure channels, it is not possible to undetectably modify messages between members of  $Pset$  and no member of  $Pset$  can be forced to agree invalid state. Thus the most that can be achieved is the detectable disruption of the protocol (including the blocking of a protocol run pending receipt of messages). In particular, inconsistency between signed and unsigned message content is detectable and will lead to exceptional abort of a protocol and invalidation of a proposed state change.

At the end of a protocol run a correctly behaving party will either: (i) be able to install a new, valid object state, and hold evidence that it has been unanimously agreed; or (ii) hold evidence that the proposed state transition has been vetoed. A misbehaving party may locally install invalid state but is not able to misrepresent it as valid. Similarly, they cannot support a claim that valid (unanimously agreed) state is invalid. A misbehaving party may prevent termination of a protocol run. This must be resolved at the application level by, for example, using the evidence generated to invoke a dispute resolution procedure.

#### 4.5. Connection and disconnection protocols

This section describes the connection and disconnection protocols used to manage membership of the participant set,  $Pset$ , for object coordination. The protocols ensure that at membership changes, all parties have a consistent, non-repudiable view both of the current membership of  $Pset$  and of the agreed object state.

#### 4.5.1 Subject and sponsor roles

There are two special roles in the connection/disconnection protocols:

1. the *subject* of a connection or disconnection request — the proposed new member or the member leaving, respectively.
2. the *sponsor* of a connection or disconnection request. The sponsor coordinates the decision of the current membership of  $Pset$  with respect to the request. If a connection request is agreed, the sponsor provides the current agreed object state to the subject of the request. The sponsor is also responsible for blocking new coordination requests pending decision on any active request.

The sponsor of a connection request is unambiguously identified as the most recently joined member of  $Pset$ . That is, given  $n$  members of  $Pset = \{P_i | i \in 1 : n\}$  ordered by most recently joined member, the sponsor of the current connection request is  $P_n$ . If the connection request is agreed, then the sponsor of the next request will be  $P_{n+1}$ . Any member of  $Pset$  can identify the legitimate sponsor for a connection request and provide this information to the subject of a request.

The sponsor of a disconnection request is  $P_n$  unless  $P_n$  is the subject of the request. If  $P_n$  is the subject, then  $P_{n-1}$  is the sponsor — the most recently connected member prior to  $P_n$ .

The use of a sponsor during connection reduces the information gained by the subject in the event of a request being rejected. In the case of disconnection, the use of a sponsor limits the participation of the subject. If a subject is to be evicted from  $Pset$ , then the sponsor may initiate the disconnection protocol without their involvement. Rotating the responsibility of sponsor reduces reliance on a single member of  $Pset$ <sup>2</sup>. Since the current sponsor can be unambiguously identified, any member of  $Pset$  can verify the legitimate sponsor for a request and the sponsor is able to enforce concurrency control during membership changes.

#### 4.5.2 Group identifiers

The membership of  $Pset$  is uniquely identified by a group identifier tuple:  $\langle seqno, h(rn), h(\sum P_i) \rangle$ .  $seqno$  and  $h(rn)$  are generated in the same way as for state change proposals.  $h(\sum P_i)$  is a hash over the members of  $Pset$ . There are two tuples of interest:

<sup>2</sup>If sponsor rotation is not required, then the initial member of  $Pset$  can sponsor all connection/disconnection requests unless they are the subject of a disconnection request (in this case the responsibility would pass to the next oldest member of  $Pset$ ).

$NGid = \langle seqno, h(rn_n), h(\sum P_i) \rangle$  is the tuple that identifies the **new group** that would result from the proposed membership change. The sponsor of a connection/disconnection request generates  $NGid$ .

$Gid_i$  is the tuple that identifies the current group membership as viewed by  $P_i$ . Inconsistent group identifiers lead to invalidation of a proposal.

#### 4.5.3 The connection protocol

In the protocol description below, the participant identifier,  $P_i$ , is assumed to provide access to the information necessary both to establish a connection with  $P_i$  and to verify  $P_i$ 's signature.  $D_i$  represents  $P_i$ 's decision on the validity of a connection request sponsored by  $P_n$ .

The proposed new member,  $P_{n+1}$ , initiates the connection protocol by sending a request to  $P_n$ .  $P_n$  then relays the request to  $Rset_n (= Pset - P_n)$  to obtain the group's decision. A random number,  $rn_{n+1}$ , generated by  $P_{n+1}$  uniquely labels the initial request.  $P_{n+1}$  is assumed to have access to the information necessary to communicate with  $P_n$ . Assuming the connection request is unanimously agreed, the protocol proceeds as follows:

$$\begin{array}{lll}
 request & P_{n+1} \rightarrow P_n & : \{req, sig_{n+1}(h(req))\} \\
 propose & P_n \rightarrow Rset_n & : mp \\
 respond & Rset_n \rightarrow P_n & : mr_j \\
 commit & P_n \rightarrow Rset_n & : mc_1 \\
 & P_n \rightarrow P_{n+1} & : mc_2
 \end{array}$$

where :

$$\begin{array}{l}
 req = \{P_{n+1}, rn_{n+1}\} \\
 mp = \{prop, sig_n(h(prop)), sig_{n+1}(h(req))\} \\
 prop = \{P_n, req, Gid_n, NGid, ASid_n\} \\
 mr_j = \{resp_j, sig_j(h(resp_j)), sig_j(h(prop))\} \\
 resp_j = \{P_j, D_j, Gid_j, ASid_j\} \\
 mc_1 = \left\{ rn_n, \sum resp_j, \sum sig_j(h(resp_j)), \right. \\
 \left. \sum sig_j(h(prop)) \right\} \\
 mc_2 = \{mc_1, AS_n, prop, sig_n(h(prop))\}
 \end{array}$$

An authenticated decision to agree to the connection of  $P_{n+1}$  is given by:

$$\begin{array}{l}
 rn_n, prop, \sum resp_j, \\
 sig_{n+1}(h(req)), \sum sig_i(h(prop)), \sum sig_j(h(resp_j)) \\
 i \in \{1 : n\}; j \in \{1 : n-1\}
 \end{array}$$

At successful completion of the protocol, the membership of the object coordination group is:  $Pset + P_{n+1}$ . All members of this group have evidence of unanimous agreement to admit  $P_{n+1}$ .  $P_{n+1}$  has also acquired the agreed state,  $AS_n$ , and this agreed state can be verified against each

of the signed agreed state tuple,  $ASid_i$ , supplied by members of  $Pset$ . Thus a consistent view of the membership, identified by  $NGid$ , is installed by all parties.

A connection request from  $P_{n+1}$  may be rejected immediately by the sponsor,  $P_n$ , or may be vetoed by a member of  $Rset_n$ . In the case of immediate rejection,  $P_n$  simply responds to a request with a signed reject message:

$$\begin{aligned} request & P_{n+1} \rightarrow P_n : \{req, sig_{n+1}(h(req))\} \\ commit & P_n \rightarrow P_{n+1} : mc_3 \end{aligned}$$

where :

$$mc_3 = \{P_n, REJ, sig_n(h(P_n, REJ)), sig_n(h(req))\}$$

and  $REJ$  indicates the request,  $req$ , was rejected

In the case of veto by a member of  $Rset_n$ , the protocol follows the same steps as for a successful run except the final message,  $mc_2$ , is replaced by  $mc_3$ . That is,  $P_{n+1}$  learns no more information than in the case of immediate rejection by  $P_n$ . Message  $mc_1$  is still sent to all members of  $Rset_n$ . As can be seen, the sponsor has an advantage since, from the perspective of the proposed new member, there is no distinction between immediate rejection by  $P_n$  or rejection by any number of members of  $Pset$ . However, it is assumed that some advantage accrues to all members of  $Pset$  from the legitimate involvement of  $P_{n+1}$  and, therefore, that there is an incentive to cooperate and include  $P_{n+1}$  in the interaction [2]. The unwillingness of any member of  $Pset$  to admit a new member must ultimately be the subject of extra-protocol dispute resolution. The protocol as presented meets the requirement of providing a consistent, non-repudiable view of membership changes.

#### 4.5.4 The disconnection protocols

Disconnection protocols are required both for voluntary disconnection and for eviction of a member of  $Pset$ .

We assume  $P_1$  is the subject of a disconnection request,  $P_n$  is the current request sponsor and  $P_k$  is the proposer. For voluntary disconnection:  $P_k = P_1$ ; and for eviction:  $P_k \neq P_1$  (and may be  $P_n$ ). The disconnection protocols aim to ensure that the remaining members of the object coordination group have evidence of the decision to disconnect  $P_1$  and that, in the case of voluntary disconnection,  $P_1$  initiated the disconnection.  $Rset'_n = Rset_n - P_1$  is the recipient set for a disconnection proposal sponsored by  $P_n$ . For eviction,  $D_i$  represents  $P_i$ 's decision on the validity of the eviction request. Since any member of  $Pset$  wishing to disconnect may in practice simply cease cooperation, voluntary disconnection cannot be vetoed. Thus,  $D_i$  is omitted from the voluntary disconnection protocol.

The eviction protocol is:

$$\begin{aligned} request & P_k \rightarrow P_n : \{req, sig_k(h(req))\} \\ propose & P_n \rightarrow Rset'_n : mp \\ respond & Rset'_n \rightarrow P_n : mr_j \\ commit & P_n \rightarrow Rset'_n : mc \end{aligned}$$

where :

$$\begin{aligned} req &= \{P_k, P_1, rn_k\} \\ mp &= \{prop, sig_n(h(prop)), sig_k(h(req))\} \\ prop &= \{P_n, req, D_n, Gid_n, NGid, ASid_n\} \\ mr_j &= \{resp_j, sig_j(h(resp_j)), sig_j(h(prop))\} \\ resp_j &= \{P_j, D_j, Gid_j, ASid_j\} \\ mc &= \left\{ rn_n, \sum resp_j, \sum sig_j(h(resp_j)), \right. \\ &\quad \left. \sum sig_j(h(prop)) \right\} \end{aligned}$$

An authenticated eviction decision is given by:

$$\begin{aligned} &rn_n, prop, \sum resp_j, \\ &sig_k(h(req)), \sum sig_i(h(prop)), \sum sig_j(h(resp_j)) \\ &i \in \{2 : n\}; j \in \{2 : n-1\}; k \neq 1 \end{aligned}$$

If the current sponsor is also the proposer of an eviction ( $P_k = P_n$ ), the request step is omitted from the above protocol and message  $mp$  is modified as follows:

$$\begin{aligned} mp &= \{prop, sig_n(h(prop))\} \\ prop &= \{P_n, P_1, Gid_n, NGid, ASid_n\} \end{aligned}$$

It is possible to modify the eviction protocol to allow for eviction of subsets of  $Pset$  and arrive at cooperating subsets that may make forward progress. In this case, an evictee subset,  $Eset$ , is identified at the request stage instead of a single member of  $Pset$ . If the eviction is agreed, a new coordination group is formed:  $Pset - Eset$ . Clearly, distinct subgroups can be formed in this way. A group formed following eviction(s) can only claim that its members have agreed to its formation. No claim can be made with respect to the agreement of the evictee(s) to the new group membership nor to the validity of subsequent state updates agreed by the new group.

The voluntary disconnection protocol is:

$$\begin{aligned} request & P_1 \rightarrow P_n : \{req, sig_1(h(req))\} \\ propose & P_n \rightarrow Rset'_n : mp \\ respond & Rset'_n \rightarrow P_n : mr_j \\ commit & P_n \rightarrow Rset'_n : mc \\ & P_n \rightarrow P_1 : \{mc, prop, sig_n(h(prop))\} \end{aligned}$$

where :

$$\begin{aligned} req &= \{P_1, rn_1\} \\ mp &= \{prop, sig_n(h(prop)), sig_1(h(req))\} \\ prop &= \{P_n, req, Gid_n, NGid, ASid_n\} \\ mr_j &= \{resp_j, sig_j(h(resp_j)), sig_j(h(prop))\} \\ resp_j &= \{P_j, Gid_j, ASid_j\} \\ mc &= \left\{ rn_n, \sum resp_j, \sum sig_j(h(resp_j)), \right. \\ &\quad \left. \sum sig_j(h(prop)) \right\} \end{aligned}$$

An authenticated voluntary disconnection is given by:

$$rn_n, prop, \sum resp_j, sig_{g_1}(h(req)), \sum sig_i(h(prop)), \sum sig_j(h(resp_j))$$

$i \in \{2 : n\}; j \in \{2 : n - 1\}$

This provides evidence that  $P_1$  initiated voluntary disconnection and that all other members of  $Pset$  have seen the request.

The evidence generated during eviction or voluntary disconnection ensures that the remaining members of  $Pset$  maintain a consistent view of group membership and of agreed object state. After voluntary disconnection, the disconnected member has evidence of the group membership and agreed object state when they disconnected.

### 5. B2BObjects API and implementation

This section describes the B2BObjects API and a prototype implementation of the middleware. The prototype is written in Java using Java RMI for remote invocation. Two proof-of-concept applications have been developed using the prototype. Both applications illustrate two-party, synchronous coordination. However, neither the API, nor coordination protocols, are specific to this mode of operation.

The primary B2BObjects API classes are *B2BObject* — the application-specific augmentation of a local object, and *B2BObjectController* — the local interface to configuration, initiation and control of information sharing. The in-

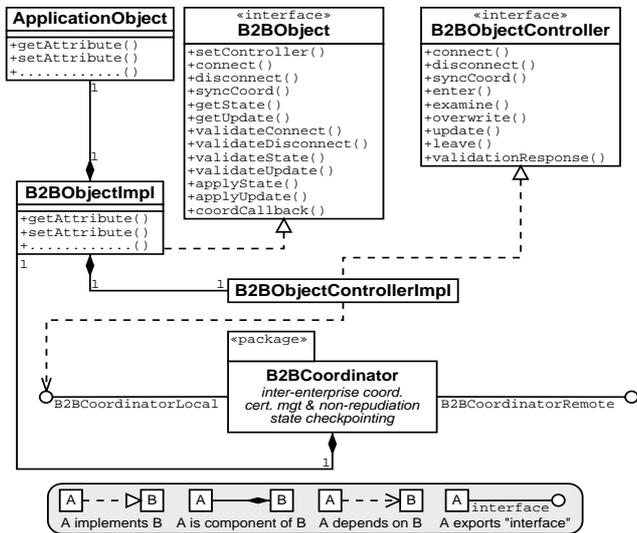


Figure 4. B2BObjects API

terfaces to these classes and the relationship between them and the *B2BCoordinator* package are shown in Figure 4.

The coordinator package manages inter-organisational connection to and communication between objects, and implements coordination protocols. It also provides state checkpointing, certificate management and non-repudiation services.

The *B2BObject* interface is implemented by the application programmer. The programmer decides whether to produce a new application object that implements both the *B2BObject* interface and the application logic, or to extend an existing application object, or to wrap the object with an implementation of the *B2BObject* interface. For example, the *ApplicationObject* operation:

```
setAttribute(AType a);
```

shown in Figure 4 has a corresponding *B2BObjectImpl* wrapper operation that could be implemented as follows:

```
setAttribute(AType a) {
    // start of state access
    controller.enter();
    // indicate overwriting object state
    controller.override();
    // set the attribute
    appObject.setAttribute(a);
    // end of state access
    controller.leave();
}
```

Similarly, the *B2BObjectImpl* *getAttribute* wrapper is:

```
AType getAttribute() {
    // start of state access
    controller.enter();
    // indicate reading object state
    controller.examine();
    // get the attribute
    AType attr = appObject.getAttribute();
    // end of state access
    controller.leave();
    return attr;
}
```

Given knowledge of an application object’s state access operations, the wrapper methods of a *B2BObjectImpl* class could be generated automatically. As indicated, the *B2BObjectController* *enter* and *leave* operations are used to demarcate the scope of access to object state. These calls may be nested provided that a *leave* is invoked for each *enter*. Nesting allows the application programmer to “roll-up” a series of state changes into a single coordination event. If *overwrite* has been called within the current state change scope (as in the *setAttribute* example), then state coordination is initiated at invocation of the final *leave*, as we describe now.

The controller obtains a copy of the object’s state (using the *B2BObject* *getState* operation) and passes that state to the coordinator for propagation to remote parties for state validation. *B2BCoordinatorLocal* provides the following propagation interface :

```

public interface B2BCoordinatorLocal {
    public void propagateConnect(String
        coordAlias);
    public void propagateDisconnect(String
        subjectAlias);
    public void propagateNewState(
        NewStateRequest stateRequest);
    ...
    ...
}

```

A call to `propagateNewState` results in state validation at the remote parties via invocation of `validateState` on their copy of the shared object. The *B2BObjectController* `validationResponse` communicates the result of this application-specific validation. It can be invoked synchronously or asynchronously as a callback on the local controller. If a proposed change is accepted by all parties, an `applyState` call on each replica installs the newly validated state. Thus the `leave` operation implicitly invokes the state coordination protocol, via the local coordinator, and the validation, or otherwise, of a state change proposal. If a proposed change is invalidated, the proposer's coordinator will rollback their local object state using a call to `applyState` with the previously agreed state. A similar process to that outlined applies to update, as opposed to overwrite, of object state. In this case, the *B2BObjectController* `update` operation is used to indicate the type of state coordination required. The `examine` operation indicates that object state will be read but not written in the current scope. Together with `enter` and `leave`, the three access type indication operations (`examine`, `overwrite` and `update`) can be used as hooks for concurrency control mechanisms and transactional access to objects.

The implementation of the *B2BObjectController* is provided as part of the middleware. Together, *B2BObject* and *B2BObjectController* provide connection management; state change scoping and access type indication; and upcalls for application-level validation. `connect` and `disconnect` operations initiate connection to and disconnection from the set of objects being coordinated (leading to initiation of connection and disconnection protocols via the *B2BCoordinatorLocal* propagation interface). `validateConnect` and `validateDisconnect` allow application-specific validation of connection and disconnection requests.

The semantics of `connect`, `disconnect` and `leave` vary with the communication mode. In synchronous mode, they block until the relevant coordination process completes (an exception is raised if validation fails). In asynchronous mode, they return immediately and completion is signalled by the coordinator through invocation of `coordCallback`. In deferred synchronous mode they return immediately and a blocking call to `coordCommit` can be used to wait for completion. `coordCallback`

is also used by the coordinator to communicate protocol progress information to the application.

The *B2BCoordinatorLocal* interface is independent of both the communication mode and the coordination protocols executed between coordinators through the *B2BCoordinatorRemote* interface. The *B2BCoordinatorRemote* interface is protocol-specific and cooperating coordinators must export compatible interfaces to execute a given protocol. The *B2BCoordinatorLocal* propagation interface insulates the application from protocol specific detail. Thus it is possible to configure the middleware to use different coordination protocols without altering the application's interface to coordination. Implementations of coordinator interfaces are provided as part of the *B2BCoordinator* package provided by the middleware.

We now describe two simple applications that are each illustrative of a wider class of problem and demonstrate the adaptability of the middleware to application requirements. *Tic-Tac-Toe* is a two-party game in which the players take turns to modify its shared state according to well-defined, symmetrically applied rules. Turn-taking access to shared state is characteristic of other applications such as shared white boards. The order processing example demonstrates sharing between two parties according to asymmetric rules. As described in Section 5.2, it is relatively straightforward to extend this to involve multiple parties.

## 5.1. Tic-Tac-Toe application

The aim of a game of Tic-Tac-Toe is to claim a horizontal, vertical or diagonal line of squares before your opponent. Players take turns to play. The rules of the game are symmetric. For Nought, a vacant square is claimed by marking it with a zero; Nought cannot mark any square with a cross; and Nought cannot overwrite an already claimed square.

An object that implements the *B2BObject* interface represents the state of the game and encapsulates the rules. Servers representing each player share the object and coordinate the object state. A player communicates a move to their server using their local application client's "Save" operation. The servers validate each proposed move (state change) via the `validateState` upcall. A validated move is retrieved by the application client using its "Load" operation. Apart from encoding the rules of the game, the application programmer's task mainly concerns the instantiation of the *B2BObjects* infrastructure and provision of the user interface (the "Load" and "Save" operations are part of this interface and are not mandated by *B2BObjects*).

Figure 5 shows an example of the Tic-Tac-Toe game in progress after the following sequence of moves: Cross claims middle row, centre square; Nought claims top row,

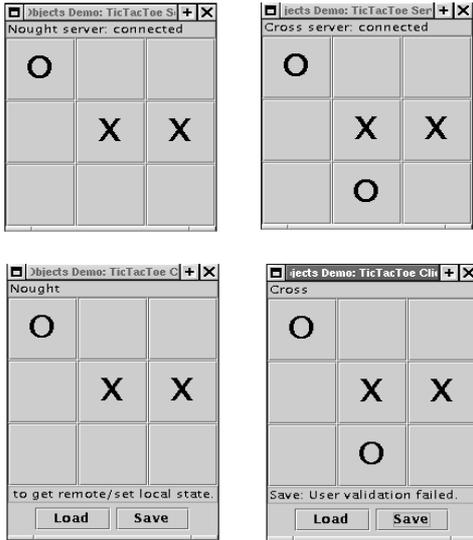


Figure 5. Tic-Tac-Toe game

left square; Cross claims middle row, right square; then Cross attempts to mark bottom row, centre square with a zero. The final move is an attempt by Cross to gain advantage by pre-empting Nought's next move. The state change is invalid and, as can be seen, is not reflected at Nought's server. The agreed state of the game has not been updated and Nought will have evidence of the attempt to cheat. Cross forfeits the game.

As an alternative to playing the game directly between two players, it may be desirable to validate moves at a TTP in order to guarantee that they are encoded and observed correctly. Figure 6 represents an instance of the game be-

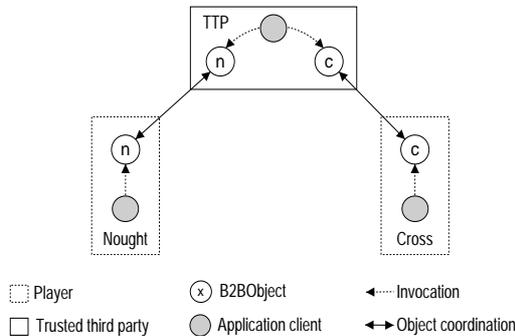


Figure 6. Tic-Tac-Toe through a TTP

ing played through a TTP that validates each player's move before it is disclosed to their opponent.

## 5.2. Order processing application

In this application a customer and supplier share the state of an order. Asymmetric validation rules apply to state

changes. The customer is allowed to add items and the quantity required to an order but is not allowed to price the items. The supplier can price items but cannot amend the order in any other way.



Figure 7. B2BObjects order processing

Figure 7 shows an example of an order being updated. The customer and supplier each have a replica of an order object that implements the *B2BObject* interface. The state of each replica is coordinated with that of its peer. In the example, the customer orders 2 *widget1s*. This is a valid entry. The supplier then prices *widget1* at 10 per unit. The supplier's action is validated and reflected in the customer's copy of the order. The customer then amends the order for the supply of 10 *widget2s*. This entry is validated and reflected in the supplier's copy. Then the supplier attempts to both price *widget2* (a valid action) and change the quantity required (an invalid action). As can be seen, this update to the order is rejected and is not reflected in the customer's copy.

An alternative instantiation of order processing could involve an approver to sanction the items ordered by the customer and a dispatcher to commit to delivery terms. The order object would then be shared between four parties and the validation rules modified to reflect their roles.

## 6. Related work

The problem of fair exchange of information, or items, of value has received considerable attention recently. A system is considered fair if it does not discriminate against a correctly behaving party. Fair exchange protocols [1, 7, 13] aim to guarantee fairness during a protocol run or, in the case of *optimistic fair exchange*, through an exchange protocol and associated resolve and abort sub-protocols. All known fair exchange protocols require either that a TTP is actively involved or is used to guarantee termination. Pagnia et al [10] provide a formal proof of the impossibility of (strong) fair exchange without a TTP. A distinction can be made between one-off exchange and information sharing that is ongoing. It has been shown that relationships that are characterised by an indefinite series of interactions have

quantifiable, and often strong, incentives to co-operative behaviour [2]. These incentives even hold between antagonists. This insight is relevant to the configuration of middleware support for evolving interaction styles (hinted at in Section 2 with respect to Figure 1).

The work of Wichert et al [14] is close to our approach to systematic generation of non-repudiation evidence. They propose the generation of evidence at invocation of “tagged” methods. They provide non-repudiable RPC but do not address validation of state changes for information sharing.

Work in the MAFTIA project on distributed trusted computing services [12] is relevant to our plans to investigate the deployment of the functions and services provided by the *B2BCoordinator* package (see Section 5) at a trusted computing base. MAFTIA’s work on tolerating the corruption of a proportion of participants in agreement protocols [3] is relevant to protocol termination through majority voting.

In the area of policy-controlled interaction, Ponder [4] is of interest because of its unified approach to the specification of both security and management policy for distributed object systems. It also allows the import of policy across administrative domains. The work of Minsky et al on Law Governed Interaction (LGI) [8] provides an infrastructure for interaction between parties governed by global policy. Communication between parties is mediated by agents. An agent enforces agreed policy as it relates to the party on whose behalf the agent acts. (The agent role is similar, in effect, to that of trusted agents in the indirect interaction style of Figure 1b.) Another approach to the automated control of interactions through agreements between organisations is IBM’s tpaML language for business-to-business integration [5]. Their model of long-running conversations, the state of which is maintained at each party, is similar to the notion of shared interaction state. Policy-based approaches can be seen as complementary to B2BObjects. For example, policy controlling an interaction could be expressed using Ponder, LGI or tpaML constructs, and the underlying infrastructure for regulated information sharing could be instantiated using B2BObjects.

## 7. Conclusions and future work

We have presented middleware that addresses the requirement for dependable information sharing between organisations. The middleware presents the abstraction of shared state and regulates updates to that state. Safety is guaranteed even in the presence of misbehaving parties. If all parties behave correctly, liveness is guaranteed despite a bounded number of temporary failures. The middleware presents a familiar programming abstraction to the application programmer and frees them to concentrate on the business logic of applications.

Our state coordination protocol provides strong guarantees with respect to the validity of decisions reached. It is also efficient in terms of the number of messages required ( $O(n)$  for  $n$  parties) and is straightforward to implement. These characteristics are achieved in the context of stated assumptions with respect to failures and, in particular, by not guaranteeing protocol termination when parties misbehave. However, the middleware provides persistence both of valid state and of protocol messages and, therefore, recovery is possible in many circumstances. The inability to terminate is detectable and may be resolved outside of a protocol run. This *extra-protocol* resolution will necessarily involve appeal to a third party or parties (as is the case for all known fair exchange protocols). If validation depends on the semantics of a state change as interpreted by any individual party, then it can be argued that no protocol can guarantee termination (since a decision cannot be made on that party’s behalf). In which case, a state change cannot be validated without the participation of all concerned and ultimately application-level resolution will be required to compensate for a failure to participate. Thus our protocol simply results in earlier invocation of dispute resolution. Nevertheless, we intend to investigate the impact of relaxing failure assumptions (for example: a crashed node not recovering) and of providing stronger termination guarantees. Approaches to guaranteeing termination include: automatic resolution or abort by resorting to majority decision on state changes; and the imposition of deadlines on decision-making. The imposition of deadlines requires the involvement of a TTP to guarantee that all honest parties terminate with the same view of agreed state. In effect, a TTP would provide certified abort of a protocol run unless a complete set of responses were available (in which case the TTP would provide a certified decision derived from those responses). We intend to develop protocols that use both on-line and off-line TTPs and where the TTP is constructed from trusted agents acting on behalf of the participants to an interaction. The flexibility inherent in the B2BObjects API allows us to experiment with different instantiations of the middleware that use different coordination protocols and to investigate configuration of coordination protocols to suit application requirements. A merit of the current protocol is that it is an easily understood base for such investigation.

Future work will also include investigation of support for concurrency control and transactional access to B2BObjects; and the use of a wider range of communications mechanisms. For example, to support loosely-coupled inter-organisational interaction, we intend to provide implementations of the middleware that are based on Message Oriented Middleware and on the use of SMTP and HTTP/SOAP for message delivery.

## 8. Acknowledgements

This work is part-funded by the UK EPSRC under grant GR/N35953/01: “Information Co-ordination and Sharing in Virtual Environments”; by the European Union under Project IST-2001-34069: “TAPAS (Trusted and QoS-Aware Provision of Application Services)”; and by Hewlett-Packard. We thank our colleague Paul Ezhilchelvan for useful discussion of this work.

## References

- [1] N. Asokan. Fairness in Electronic Commerce. Research Report RZ3027, IBM Zurich Research Lab, 1998.
- [2] R. Axelrod. *The Evolution of Co-operation*. Penguin Books, 1990.
- [3] C. Cachin. Distributing Trust on the Internet. In *Proc. IEEE Int. Conf. on Dependable Syst. and Networks (DSN)*, June 2001.
- [4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proc. Int. Workshop on Policies for Distributed Syst. and Networks (POLICY)*, Springer-Verlag LNCS 1995, Bristol, UK, Jan. 2001.
- [5] A. Dan, D. Dias, R. Kearney, T. Lau, T. Nguyen, M. Sachs, and H. Shaikh. Business-to-business integration with tpaML and a business-to-business protocol framework. *IBM Syst. J.*, 30(1):68–90, 2001.
- [6] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Trans. Inf. Theory*, 29(2):198–208, 1983.
- [7] S. Ketchpel and H. Garcia-Molina. A sound and complete algorithm for distributed commerce transactions. *J. Distributed Computing*, 12:13–29, Jan. 1999.
- [8] 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, July 2000.
- [9] J. Mitchener, A. Pengelly, D. Freestone, and A. Childerhouse. Distributing operational support to transform business operations. *BT Technology J.*, 17(2):74–81, Apr. 1999.
- [10] H. Pagnia and F. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical Report TUD-BS-1999-02, TU Darmstadt, 1999.
- [11] B. Schneier. *Applied Cryptography*. John Wiley and Sons, 2nd edition, 1996.
- [12] P. Veríssimo, N. Neves, C. Cachin, M. Correia, T. McCutcheon, B. Pfitzmann, B. Randell, M. Schunter, W. Simmonds, R. Stroud, M. Waidner, and I. Welch. Service and Protocol Architecture for the MAFTIA Middleware. Deliverable D23, EU MAFTIA Project IST-1999-11583, Jan. 2001.
- [13] H. Vogt, H. Pagnia, and F. Gärtner. Modular Fair Exchange Protocols for Electronic Commerce. In *Proc. IEEE Annual Comput. Security Applications Conf.*, Phoenix, Arizona, Dec. 1999.
- [14] M. Wichert, D. Ingham, and S. Caughey. Non-repudiation Evidence Generation for CORBA using XML. In *Proc. IEEE Annual Comput. Security Applications Conf.*, Phoenix, Arizona, Dec. 1999.
- [15] J. Zhou and D. Gollmann. Evidence and non-repudiation. *J. Network and Comput. Applications*, 20(3):267–281, July 1997.