Dependable Information Sharing between Organisations Using B2BObjects*

Nick Cook[1], Santosh Shrivastava[1] and Stuart Wheater[2]

[1]School of Computing Science, University of Newcastle, UK

[2]Arjuna Technologies, Newcastle, UK

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 who 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.

Keywords: distributed object middleware, e-commerce, security, fault-tolerance

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 who 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

---

* This paper is a substantial revision and extension to an earlier version published in: Proc. IEEE DSN02 [Cook *et al.* 2002].

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 [Cook *et al.* 2002]. State coordination protocols are discussed in detail in Section 4 and Section 5. Section 6 presents the Application Programmer Interface (API), a prototype implementation and two proof-of-concept applications that use it. Related work is surveyed in Section 7. Section 8 discusses future work and Section 9 concludes the paper.

2 Application Requirements

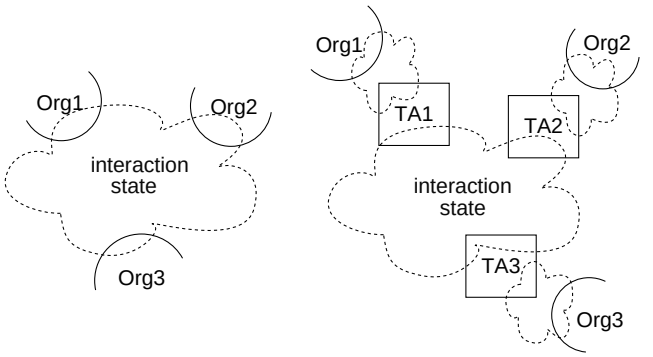
In this section we sketch three different application scenarios from which we derive requirements for middleware support for dependable information sharing between organisations**.**

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 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 on-going 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 [Mitchener *et al.* 1999]. 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 abstraction of shared (interaction) state, then the requirements can be met by regulating, and recording, access and update to that state.



(a) direct interaction (b) interaction through trusted agents

Figure 1: Direct vs. indirect interaction styles

Fig. 1 shows two interaction styles (a and b). In (a), organisations (Org_i) disclose state and interact directly. In (b), 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 Fig. 1 represent the deployment of B2BObjects middleware to meet the application requirements outlined above. For simplicity, in the rest of the paper direct interaction 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 Sections 4 and 5. The API and a prototype implementation are described in Section 6

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.

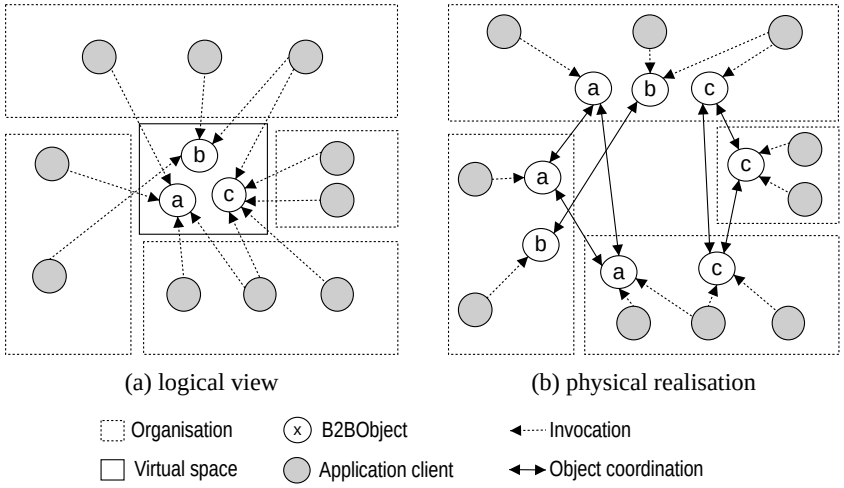


Figure 2: B2BObjects interaction

As shown in Fig. 2, the logical view of shared objects in a virtual space (a) is realised by the regulated coordination of actions on object replicas held at each organisation (b).

Multi-party validation of state changes supports the notion of "joint ownership" of shared state. A state change 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. Fig. 3 depicts this augmentation of an application object.

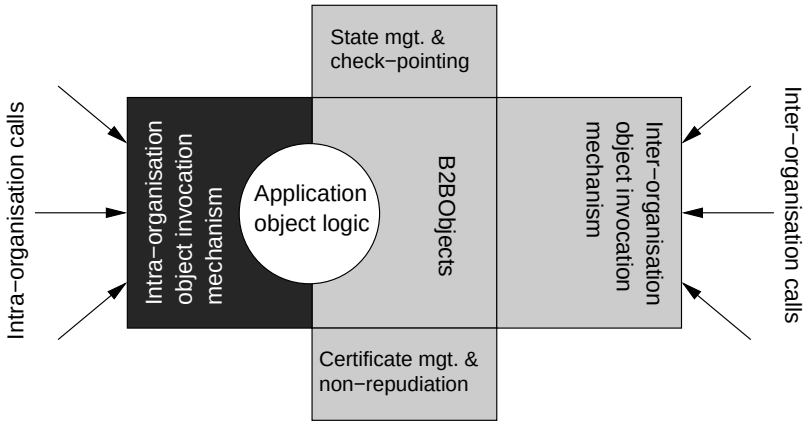


Figure 3: B2BObjects augmentation

Calls to 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 State Coordination

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 requirement for unanimous agreement to a proposed state change is fundamental to the following discussion. It is imposed because, in the context of mutually mistrusting parties, it cannot be assumed that majority agreement to a state change will suffice. For example, any majority decision on the validity of a change to an order that is shared by a customer, a supplier and a delivery agent is potentially disadvantageous to the minority party. The unanimity restriction ensures that the safety guarantee described below can be met.

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 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 Fig. 1b, trusted agents can be used to provide conditional state disclosure. An example of this type of interaction is given in Section 6.1.

- There is no guarantee of termination when parties misbehave. This is the price of local autonomy required for the safety guarantee. 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 any resultant blocking. The provision of stronger termination guarantees is discussed in Section 8

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 [Schneier 1996]: 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: $h(x)$; 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 [Zhoh & Gollman 1997]. 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 $m = \{ev,\ sig_A(ev)\}$ from 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

$Rset_k = \{P_j \mid j \in 1:n \ \ and \ \ j \neq k\}$ 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 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 (and 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. P_k is committed to acceptance of the new state at initiation of a protocol run (at *propose* step). They cannot unilaterally refute the state transition later. A state transition is only rejected if it is vetoed by one or more members of $Rset_k$. The final resolve message is the non-repudiable decision of the whole of *Pset* on the validity of the proposed state transition. The protocol has three steps:

$$
\begin{array}{lllll}
propose & P_k \rightarrow Rset_k & : & mp & = \{prop,\ NS_k,\ sig_k\,(h\,(prop))\} \\
respond & Rset_k \rightarrow P_k & : & mr_j & = \{resp_j,\ sig_j\,(h\,(resp_j))\} \\
resolve & P_k \rightarrow Rset_k & : & mrs & = \{rn_k,\ \sum mr_j\}
\end{array}
$$

where:

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

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 from P_j contains a receipt, $h(prop)$, for the proposal and a signed decision, D_j, on its validity. Inclusion of Gid_j, $ASid_j$ and $CSid_j$ permits systematic consistency checks.

The resolve message, *mrs*, 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 to commit or abort a proposal over $\sum resp_j$ and *prop*. *mrs* can only be generated by P_k since it contains the authenticator rn_k. *mrs* 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:

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

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).

4.4 Modifications for State Update

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

$$mp = \{prop,\ US_k,\ sig_k(h(prop))\};\qquad prop = \{P_k,\ Gid_k,\ ASid_k,\ NSid_k,\ h(US_k)\}$$

In message *mp*, the state 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.

4.5 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 they cannot demonstrate the validity of the state. If P_k fails to send *mrs*, then P_k will know, and can act upon, the group's decision. In this case, all members of $Rset_k$ hold evidence that the protocol run is active and any subsequent coordination request (whether for a state change or for a connection or a disconnection — see Section 5) 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 or, in the case of *mrs*, during any subsequent coordination request. 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 *mrs* for any member of $Rset_k$. If *mrs* 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 *mrs* 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.

- *Temporary divergence of view of agreed state:* it is possible for a member, P_j, of $Rset_k$ to prepare two response messages: one representing an accept and the other a reject of P_k's proposal. P_j sends one response to P_k, intercepts a subset of the resolve messages

and, in those messages, substitutes the other response. In this case, if, and only if, all other members of $Rset_k$ accepted the proposal then some members of *Pset* will commit the state change and others will not (remaining in the currently agreed state). However, the guarantee that no member of *Pset* installs a state that they have not validated still holds. Furthermore, the next coordination request will reveal the divergent view, as will any attempt to take advantage of the divergence. Thus the divergence is temporary: all parties have the information necessary to install the new state and all parties eventually (at the next coordination request) receive evidence of its validation. If P_k signs the responses, $resp_j$, sent as part of the resolve message, *mrs*, then this attack can only be mounted if a member of $Rset_k$ colludes with P_k to prepare two different sets of resolve message.

Assuming signatures are not compromised, the non-repudiation evidence generated during a protocol run binds a party to their actions 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 [Dolev & Yao 1983] (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.

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. Section 5.1 describes the roles of *subject* and *sponsor* in membership changes. Section 5.2 specifies the group identifier for the current membership of *Pset* and the proposed new membership. Sections 5.3 and 5.4 provide details of the connection and disconnection protocols respectively.

The protocols ensure the maintenance of a consistent, non-repudiable view both of the membership of *Pset* and of agreed object state at membership changes. The connection protocol is used to reach agreement on members joining, or re-joining, *Pset*. Members use the voluntary disconnection protocol to leave, or temporarily suspend, membership of *Pset*. The eviction protocol is used to suspend, or end, the participation of uncooperative members of *Pset*. Since each party's view of agreed state is propagated during membership changes, any temporary divergence of view (see Section 4.5) can be resolved during a membership change protocol run. An object is withdrawn from coordination by a sequence of voluntary disconnections by the members of *Pset*. In this way, each party obtains a verified view of the agreed state of the object at the end of their involvement in its coordination. In essence, the protocols propagate non-repudiable evidence of the subject(s) and sponsor of a membership change and the agreed state of the object. For connection and eviction, decisions on the validity of the proposed change are also propagated.

It should be noted that, either as part of an interaction agreement or locally-determined policy, state changes may require validation by some fixed set of parties. That is, application requirements may determine that state changes will be vetoed until all members of the fixed group have connected and will be suspended if any member subsequently leaves or is evicted. Conditions such as this can be imposed during state coordination as part of the validation process.

5.1 Subject and Sponsor Roles

In connection and disconnection protocols, the *subject* of a request is the member joining or leaving, respectively; and 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 other coordination requests until a decision on a membership change is reached.

To reduce reliance on a single member of *Pset*, sponsorship is rotated* . The sponsor of a connection request is unambiguously identified as the most recently joined member of *Pset*. That is, given n members of $Pset = \{P_i \mid i \in 1:n\}$ 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}. Thus, 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. During 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. Since the current sponsor can be

---

* 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*).

unambiguously identified, any member of *Pset* can verify the legitimate sponsor for a request and the sponsor is able to block other requests during a membership change.

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:

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

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.

5.3 The Connection Protocol

In the protocol description, 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. As for state changes, $D_{n,i}$ represents P_i's decision on the validity of a connection request sponsored by P_n and, in the description, D_i is shorthand for $D_{n,i}$.

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:

$$
\begin{array}{lllll}
request & P_{n+1} \rightarrow P_n & : & mrq & = \{req,\ sig_{n+1}(h(req))\} \\
propose & P_n \rightarrow Rset_n & : & mp & = \{prop,\ sig_n(h(prop)),\ mrq\} \\
respond & Rset_n \rightarrow P_n & : & mr_j & = \{resp_j,\ sig_j(h(resp_j))\} \\
resolve & P_n \rightarrow Rset_n & : & mrs_1 & = \{rn_n,\ \sum mr_j\} \\
 & P_n \rightarrow P_{n+1} & : & mrs_2 & = \{mrs_1,\ AS_n,\ prop,\ sig_n(h(prop))\}
\end{array}
$$

where :

$$
\begin{array}{l}
req = \{P_{n+1},\ rn_{n+1}\} \\
prop = \{P_n,\ Gid_n,\ NGid_n,\ ASid_n,\ h(req)\} \\
resp_j = \{P_j,\ D_j,\ Gid_j,\ ASid_j,\ h(prop)\}
\end{array}
$$

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

$$rn_n,\ req,\ prop,\ \sum resp_j,\ sig_{n+1}\left(h(req)\right),\ \sum sig_n\left(h(prop)\right),\ \sum sig_j\left(h\left(resp_j\right)\right)$$
$$j \in \{1 : n-1\}$$

At successful completion of the protocol, the membership of the 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 tuples, $ASid_i$, supplied by members of *Pset*. Thus a consistent view of the membership, identified by $NGid_n$, 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:

$$\begin{array}{lllll} request & P_{n+1} \rightarrow P_n & : & mrq & \\ resolve & P_n \rightarrow P_{n+1} & : & mrs_3 & = \{rej,\ sig_n\left(h\left(rej\right)\right)\} \end{array}$$

where:

$rej = \{P_n,\ D_n,\ h\left(req\right)\}$

and D_n is P_n's decision to reject the request

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, mrs_2, is replaced by mrs_3 above. That is, P_{n+1} learns no more information than in the case of immediate rejection by P_n. Message mrs_1 is still sent to all members of $Rset_n$.

In practice, 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 to include P_{n+1} in the interaction [Axelrod 1990]. The unwillingness of a member of *Pset* to admit a new member can ultimately only be resolved through extra-protocol dispute resolution. The protocol presented meets the requirements of maintaining a consistent, non-repudiable view both of a membership change and of object state at a membership change.

5.4 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 of the request. For voluntary disconnection: $P_k = P_1$; and for eviction: $P_k \neq P_1$. The disconnection protocols aim to ensure that the remaining members of *Pset* 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 not relevant in the

voluntary disconnection protocol, which simply ensures that a consistent view of membership and object state is maintained.

The voluntary disconnection protocol is:

$$\begin{array}{lllll} request & P_1 \rightarrow P_n & : & mrq & = \{req,\, sig_1\,(h\,(req))\} \\ propose & P_n \rightarrow Rset'_n & : & mp & = \{prop,\, sig_n\,(h\,(prop))\,,\, mrq\} \\ respond & Rset'_n \rightarrow P_n & : & mr_j & = \{resp_j,\, sig_j\,(h\,(resp_j))\} \\ resolve & P_n \rightarrow Rset'_n & : & mrs_1 & = \{rn_n,\, \sum mr_j\} \\ & P_n \rightarrow P_1 & : & mrs_2 & = \{mrs_1,\, prop,\, sig_n\,(h\,(prop))\} \end{array}$$

where :

$$req = \{P_1,\, rn_1\}$$
$$prop = \{P_n,\, Gid_n,\, NGid_n,\, ASid_n,\, h\,(req)\}$$
$$resp_j = \{P_j,\, Gid_j,\, ASid_j,\, h\,(prop)\}$$

An authenticated voluntary disconnection is given by:

$$rn_n,\, req,\, prop,\, \sum resp_j,\, sig_1\,(h(req))\,,\, \sum sig_n\,(h(prop))\,,\, \sum sig_j\,(h\,(resp_j)) \quad j \in \{2 : n-1\}$$

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

The eviction protocol is:

$$\begin{array}{lllll} request & P_k \rightarrow P_n & : & mrq & = \{req,\, sig_k\,(h\,(req))\} \\ propose & P_n \rightarrow Rset'_n & : & mp & = \{prop,\, sig_n\,(h\,(prop))\,,\, mrq\} \\ respond & Rset'_n \rightarrow P_n & : & mr_j & = \{resp_j,\, sig_j\,(h\,(resp_j))\} \\ resolve & P_n \rightarrow Rset'_n & : & mrs & = \{rn_n,\, \sum mr_j\} \end{array}$$

where:

$$req = \{P_k,\, P_1,\, rn_k\}$$
$$prop = \{P_n,\, D_n,\, Gid_n,\, NGid_n,\, ASid_n,\, h\,(req)\}$$
$$resp_j = \{P_j,\, D_j,\, Gid_j,\, ASid_j,\, h\,(prop)\}$$

An authenticated eviction decision is given by:

$$rn_n,\, req,\, prop,\, \sum resp_j,\, sig_k\,(h(req))\,,\, \sum sig_n\,(h(prop))\,,\, \sum sig_j\,(h\,(resp_j)) \quad j \in \{2 : n-1\}\,;\, k \neq 1$$

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:

$$mp = \{prop,\, sig_n\,(h\,(prop))\}\,; \quad prop = \{P_n,\, P_1,\, Gid_n,\, NGid_n,\, ASid_n\}$$

and the authenticated decision does not include: $\{req,\ sig_k(h(req))\}$.

It is possible to modify the eviction protocol to allow for eviction of subsets of *Pset*. 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 new 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 can any assumption be made about the evictee(s) agreement or otherwise to subsequent state changes.

The evidence generated during eviction or voluntary disconnection ensures the new group, $Pset'$, has 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 were disconnected.

6 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.

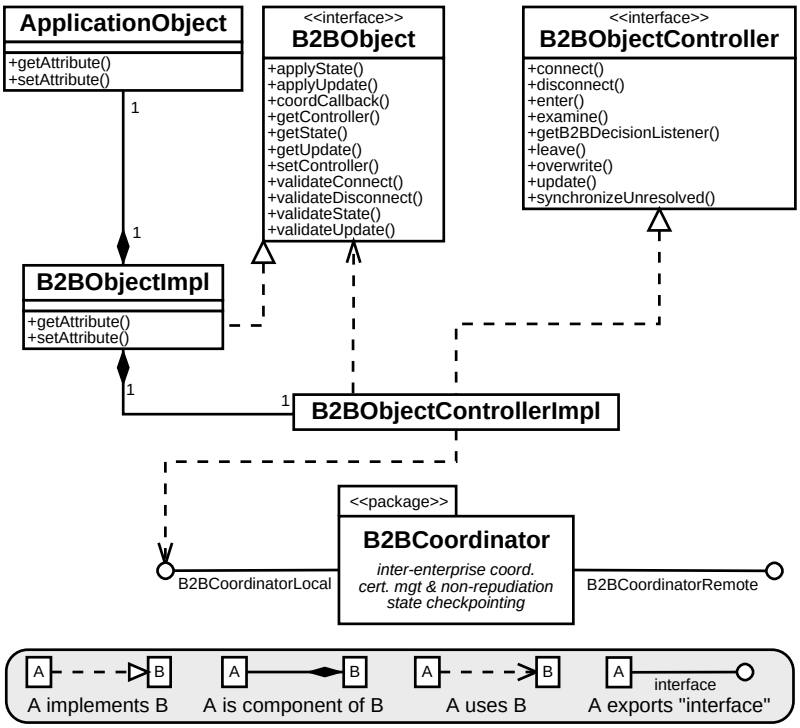


Figure 4: B2BObjects API

The interfaces to these classes and the relationship between them and the *B2BCoordinator* package are shown in Fig. 4. The coordinator package manages inter-organisational

connection to and communication between objects, and implements coordination protocols. It also provides state check-pointing, 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(SomeType attr)`, shown in Fig. 4, has a corresponding *B2BObjectImpl* wrapper operation that could be implemented as follows:

```
setAttribute(SomeType attr) {
    controller.enter(); // start of state access
    controller.overwrite(); // will overwrite object state
    appObject.setAttribute(attr); // set the attribute
    controller.leave(); // end of state access, trigger coordination
}
```

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

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

The *B2BObjectImpl* is then used in an application in the same way as the original application object, for example:

```
try { b2bObj.setAttribute(a); }
    catch (..) { // handle exceptions }
```

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 B2BResult propagateConnect(String sponsorName);
    public B2BResult propagateDisconnect(String subjectName);
    public B2BResult propagateDisconnect(String[] subjectNames);
    public B2BResult propagateNewState(NewStateRequest stateRequest);
}
```

The call to `propagateNewState` results in state validation at the remote parties via invocation of `validateState` on their copy of the shared object. 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. It is envisaged that, 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 *B2BObject* `getController` method provides application-level access to *B2BObjectController* operations such as: `connect`, `disconnect`, `getB2BDecisionListener` and `synchronizeUnresolved`.

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 call to `synchronizeUnresolved` can be used to wait for completion. `coordCallback` is also used by the coordinator to communicate protocol progress information to the application. `getB2BDecisionListener` supports asynchronous validation of coordination requests. The application programmer uses this method to pass a validation decision listener to some arbitrary validation process. In this case, the *B2BObject* validate methods return immediately, with no decision, and the decision listener is subsequently invoked to communicate the result of application-level validation.

The *B2BCoordinatorLocal* interface is independent of both the communication mode and the coordination protocols executed between coordinators through the *B2BCoordinatorRemote* interface. In asynchronous or deferred synchronous mode, the *B2BResult* returned by the propagation methods may simply indicate that a decision is not yet available. The

B2BCoordinator Remote interface is protocol-specific and cooperating coordinators must export compatible interfaces to execute a given protocol. The *B2BCoordinatorLocal* propagation interface insulates the application from this 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 included in the middleware as part of the *B2BCoordinator* package.

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. This 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.

6.1 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.

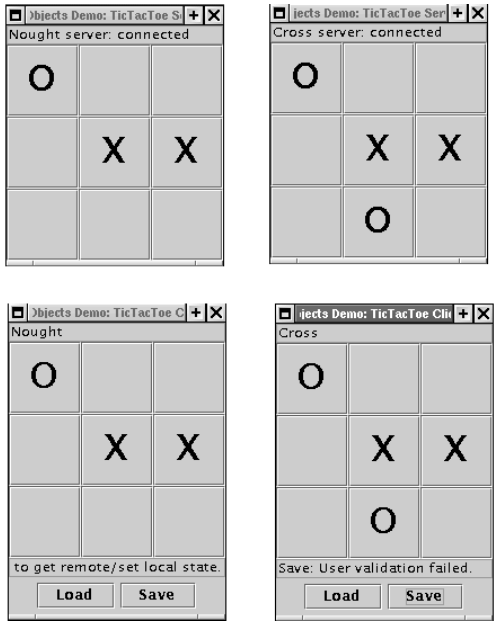


Figure 5: Tic-Tac-Toe game

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).

Fig. 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, 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.

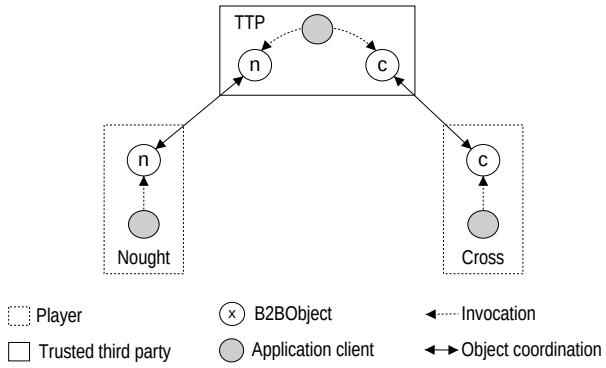


Figure 6: Tic-Tac-Toe through a TTP

Fig. 6 represents an instance of the game being played through a TTP that validates each player's move before it is disclosed to their opponent.

6.2 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.

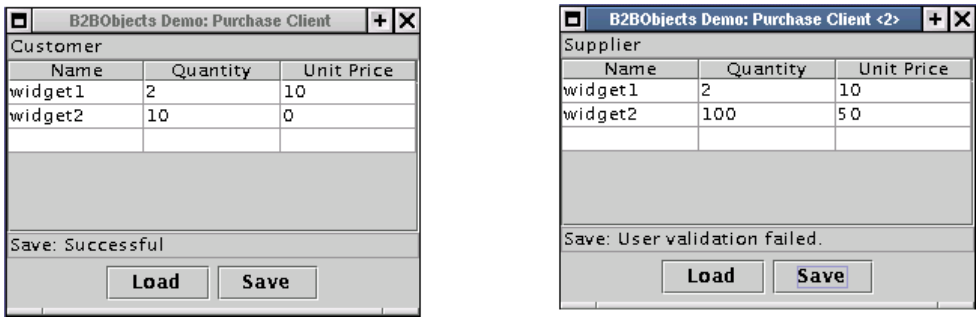


Figure 7: B2BObjects order processing

Fig. 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 *widget1*s. 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 *widget2*s. 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.

7 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 [Asokan 1998, Ketchpel & Garcia-Molina 1999, Vogt *et al.* 1999] 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 & Gärtner 1999] provides 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 cooperative behaviour [Axelrod 1990]. 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 Fig. 1).

The work of [Wichert *et al.* 1999] 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 [Veríssimo *et al.* 2001] is relevant to our plans to investigate the deployment of the functions and services provided by the *B2BCoordinator* package (see Section 6) at a trusted computing base. MAFTIA's work on tolerating the corruption of a proportion of participants in agreement protocols [Cachin 2001] is relevant to protocol termination through majority voting.

In the area of policy-controlled interaction, Ponder [Damianou *et al.* 2001] 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. Law Governed Interaction (LGI) [Minsky & Ungureanu 2000] 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 Fig. 1b.) Another approach to the automated control of interactions through agreements between organisations is IBM's tpaML language for business-to-business integration [Dan *et al.* 2001]. 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.

8 Future Work

This section describes plans for future work on: coordination protocols; concurrency control and transactions; and support for loosely-coupled interaction.

8.1 Protocol Development

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. The middleware provides persistence both of valid state and of protocol messages and, therefore, recovery is possible in many circumstances. These characteristics are achieved in the context of stated assumptions with respect to failures and, in particular, by not guaranteeing protocol termination when parties misbehave. The inability to terminate is detectable and may be resolved outside of a protocol run. 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 each individual party, then it can be argued that no protocol can guarantee termination — all parties must be involved to validate a state change. 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. 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.

Resort to majority decision-making is an application/interaction-level issue. For example, some interactions will demand unanimity, others may permit simple majority decisions on state changes or decisions that are made by an identified, minimum subset of participants. We intend to investigate configuration of the middleware, and underlying protocols, to meet such requirements.

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 the required set of responses were available (in which case the TTP would provide a certified decision derived from those responses). We intend to investigate protocols that use both on-line and off-line TTPs in these circumstances. We will also investigate the construction of a *virtual* TTP both from trusted agents acting on behalf of the participants to an interaction (where some parties may themselves be trusted agents) and from a majority of participants.

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.

8.2 Concurrency Control and Transactions

A second area of future work is concurrency control for B2BObject coordination and the participation of B2BObjects in distributed transactions. Currently, we guarantee *all fail* semantics for concurrent state coordination proposals — the inconsistency in state identifiers of two or more concurrent proposals leads to the invalidation of them all. It is possible to provide *at most one success* by adopting some convention for deciding the precedence of competing proposals. For example, there could be a consistent logical ordering of participants that enables each party to independently derive the same priority order of competing proposals. Another form of concurrency control to investigate is agreed lock acquisition for B2BObjects. Such object locking will be required for work on the participation of B2BObjects as transactional resources in distributed transactions. With respect to transactions, we envisage scenarios such as a change to a B2BObject requiring modification to information stored in a backend database and to the state of other B2BObjects. To deliver ACID (all-or-nothing update) semantics in this situation, we must provide transactional B2BObjects.

8.3 Loosely-coupled Interaction and Asynchrony

B2BObjects provides callback and synchronization mechanisms to support asynchronous and deferred synchronous operation. These mechanisms are a basis for support for loosely-coupled inter-organisational interaction. In addition, we intend to provide implementations of the underlying coordination infrastructure that use a wider range of communication mechanisms such as Message Oriented Middleware and SMTP for message delivery. In this way, a variety of interaction styles can be supported. Issues to address include the impact of asynchronous interaction on the application programming model and the participation of B2BObject in extended (non-ACID) transactions such as the Business Transaction Protocol [Potts *et al.* 2002].

9 Conclusions

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. Finally, we have identified areas for further investigation and for development of this middleware.

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 the UK DTI and Hewlett-Packard under project "GridMist". We thank our colleague Paul Ezhilchelvan for useful discussion of this work.

References

[Asokan 1998] N. Asokan, "Fairness in Electronic Commerce," IBM Zurich Research Lab, Research Report RZ3027, 1998.

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

[Cachin 2001] C. Cachin, "Distributing Trust on the Internet," in *Proc. IEEE Int. Conf. on Dependable Syst. and Networks (DSN)*, 2001.

[Cook *et al. 2002*] 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, 2002.

[Damianou *et al.* 2001] 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, 2001.

[Dan *et al.* 2001] 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.*, vol. 30, no. 1, pp. 68–90, 2001.

[Dolev & Yao 1983] D. Dolev and A. Yao, "On the Security of Public Key Protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[Ketchpel & Garcia-Molina 1999] S. Ketchpel and H. Garcia-Molina, "A sound and complete algorithm for distributed commerce transactions," *J. Distributed Computing*, vol. 12, pp. 13–29, 1999.

[Minsky & Ungureanu 2000] N. Minsky and V. Ungureanu, "Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems," *ACM Trans. Softw. Eng. and Methodology*, vol. 9, no. 3, pp. 273–305, 2000.

[Mitchener *et al.* 1999] J. Mitchener, A. Pengelly, D. Freestone, and A. Childerhouse, "Distributing operational support to transform business operations," *BT Technology J.*, vol. 17, no. 2, pp. 74–81, 1999.

[Pagnia & Gärtner 1999] H. Pagnia and F. Gärtner, "On the impossibility of fair exchange without a trusted third party," TU Darmstadt, Tech. Rep. TUD-BS-1999-02, 1999.

[Potts *et al.* 2002] M. Potts, B. Cox, and B. Pope, "Business Transaction Protocol Primer," OASIS Committee Supporting Document, 2002.

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

[Veríssimo *et al.* 2001] 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," EU MAFTIA Project IST-1999-11583, Deliverable D23, 2001.

[Vogt *et al.* 1999] H. Vogt, H. Pagnia, and F. Gärtner, "Modular Fair Exchange Protocols for Electronic Commerce," in *Proc. IEEE Annual Comput. Security Applications Conf.*, Phoenix, Arizona, 1999.

[Wichert *et al.* 1999] 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, 1999.

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