CORBA Transactions Through Firewalls

David Ingham*, Owen Rees, Andy Norman
Extended Enterprise Laboratory
HP Laboratories Bristol
HPL-1999-50
April, 1999

atomic transactions, electronic commerce, security, firewalls, CORBA

Electronic commerce on the Internet is evolving from simpler customer-to-customer (C2B) interactions, such as online shopping, to more complex business-to-business (B2B) applications, so called *extranet applications*. This class of application differs from C2B as back-office processing is typically required within each organisation. CORBA provides abstractions and transparencies that make it a good candidate technology for building such applications; organisations are required to agree on object interfaces but are free to implement objects in their preferred language using their chosen ORB vendor. Multi-party interactions introduce complex failure modes and, given the unpredictable quality of service of the Internet, occasional failures are likely to occur. Atomic transactions are a well known structuring technique for ensuring the overall consistency of system state in the presence of concurrent access and occasional failure. Transactions therefore appear an appropriate technology to support extranet applications. The use of CORBA transactions for supporting extranet applications is complicated by the use of organisational firewalls. Conventional firewall technology that operates by restricting access based on port numbers and protocols is not appropriate for CORBA, which abstracts away from these concepts. This paper describes the issues involved and shows how they can be addressed using an advanced CORBA object gateway.

Internal Accession Date Only
*Department of Computing Science, Newcastle University, Newcastle upon Tyne, UK

CORBA Transactions Through Firewalls

David Ingham
Dave.Ingham@ncl.ac.uk
Department of Computing Science,
Newcastle University,
Newcastle upon Tyne, NE1 7RU,
United Kingdom.

Owen Rees
Owen_Rees@hpl.hp.com
Hewlett-Packard Laboratories,
Filton Road, Stoke Gifford,
Bristol, BS34 8QZ,
United Kingdom.

Andy Norman
Ange@hplb.hpl.hp.com
Hewlett-Packard Laboratories,
Filton Road, Stoke Gifford,
Bristol, BS34 8QZ,
United Kingdom.

Abstract: Electronic commerce on the Internet is evolving from simple customer-to-business (C2B) interactions, such as online shopping, to more complex business-to-business (B2B) applications, so called *extranet applications*. This class of application differs from C2B as back-office processing is typically required within each organisation. CORBA provides abstractions and transparencies that make it a good candidate technology for building such applications; organisations are required to agree on object interfaces but are free to implement objects in their preferred language using their chosen ORB vendor. Multi-party interactions introduce complex failure modes and, given the unpredictable quality of service of the Internet, occasional failures are likely to occur. Atomic transactions are a well known structuring technique for ensuring the overall consistency of system state in the presence of concurrent access and occasional failure. Transactions therefore appear an appropriate technology to support extranet applications. The use of CORBA transactions for supporting extranet applications is complicated by the use of organisational firewalls. Conventional firewall technology that operates by restricting access based on port numbers and protocols is not appropriate for CORBA, which abstracts away from these concepts. This paper describes the issues involved and shows how they can be addressed using an advanced CORBA object gateway.

Keywords: atomic transactions; electronic commerce; security; firewalls; CORBA

Introduction

Much of the effort to support electronic commerce over public networks has focused on the provision of secure communication channels and electronic payment protocols, e.g., SET [Mastercard95]. While being necessary, secure payment is only one aspect of electronic business (e-business). Today, there are successful Internet-based shops selling both hard and soft goods over the Web. Such shops require a number of other computing systems in addition to payment to support their businesses, e.g., stock control, customer accounts, billing, ordering, etc. Co-ordinating the interactions between these sub-systems in a reliable manner is a non-trivial task. For example, it would be unsatisfactory if a computer failure caused a customer to be billed for an item that was not dispatched. Furthermore, this example is one of the more simplistic cases from the spectrum of possible e-business interactions since, although the purchase is initiated by the customer, once the ‘Go’ button is pressed then all of the processing takes place under the control of a single merchant.

More complex scenarios may require end-to-end guarantees that necessitate the participation of the client in the processing, e.g., the delivery of soft-goods [Little97]. Increasing in complexity further are a general class of business-to-business transactions that involve back-office processing within two or more organisations. In this scenario the problem of reliably ensuring the integrity of changes to overall system state is dramatically complicated due to the independent control and the widely distributed nature of the interaction.

Atomic actions (transactions) are a well known technique for addressing these application consistency requirements [Lomet77, Bernstein87]. An atomic action guarantees that, despite failures, either all of the items of work performed within its scope are completed successfully or all are undone. The Object Management Group specified Object Transaction Service (OTS) provides support for implementing distributed transactional applications based on CORBA technology [OMG95]. Additionally, the Microsoft Transaction Service (MTS) offers similar capabilities for applications implemented using distributed COM objects [Microsoft97]. It is the belief of the authors that distributed object-oriented transactional technology will become an indispensable component of the e-business infrastructure.

However, although e-business offers the potential for significant productivity and financial benefits, it must be conducted within a framework that does not compromise the security of an enterprise's data. Currently, many organisations operate behind firewalls to reduce the possibility of external attacks[1]. If e-business is to be carried out using objects and transactions then it will be necessary that these operate within the restrictions imposed by such security solutions.

Implementing secure inter-enterprise e-business applications using CORBA distributed object technology necessitates access to objects that are behind organisational firewalls. One of the primary issues in implementing such applications is the requirement that the access be *controlled*, that is, only the objects *published* by an organisation should be externally visible and in addition, access to these exposed objects should be authenticated. As far as the authors are aware, there are currently no commercial offerings that address these requirements. This paper presents our experience in building such applications using the OTSArjuna transaction system developed by Newcastle University and commercially supported by Arjuna Solutions Limited and CORBAgate, a research prototype object gateway from Hewlett-Packard Laboratories.

The remainder of this paper begins with an overview of atomic transactions, focusing on their use in a CORBA environment. Next, we introduce the CORBAgate architecture showing how it provides a secure object gateway for firewall machines. Throughout the paper, we illustrate the issues using a simple inter-enterprise e-business example. Finally, we compare our work with alternative technologies and draw conclusions.

Atomic transactions for e-business applications

This section provides an introduction into the concepts behind atomic transactions and shows how transactions can be used within CORBA applications using the Object Transaction Service. An example is provided to illustrate how CORBA e-business applications can be implemented using transactions.

Overview of transaction concepts

The concept of atomic transactions is an important programming paradigm to aid in the construction of reliable applications. Using transactions alleviates much of the programming burden of ensuring application integrity in the presence of failures. Atomic transactions possess the well-known ACID properties:

[1] The Gartner Group, in a recent report, predicted that the use of firewalls will expand from their current role in shielding organisations from the Internet to being used within organisations to protect Intranets.

atomicity: a transaction completes successfully (commits) or, if it fails (aborts), then all of its effects are undone (rolled back);

consistency: transactions produce consistent results and preserve invariant properties;

isolation: the intermediate states produced while a transaction is executing are not externally visible. Furthermore, transactions appear to execute serially, even if they are performed concurrently;

durability: the effects of a committed transaction are never lost (except by catastrophic failure).

Transactions were originally used in centralised database systems to allow concurrent access to be performed without interference. The Distributed Transaction Processing (DTP) Reference Model, developed by the X/Open Company, extended the concepts to include distributed applications so that a transaction could encompass more than a single application, process or machine [Xopen96]. In the DTP model, a Transaction Manager (TM), is responsible for managing transactions on behalf of multiple clients. The TM keeps track of the resources that are manipulated within the scope of each transaction and co-ordinates their completion when instructed to do so by the transaction originator. Transactions are committed by using a two-phase-commit (2PC) protocol; during phase 1, all involved parties are instructed to prepare to commit and return a result indicating whether they prepared successfully. Based on the responses, TM begins phase 2 of the protocol: if all of the parties prepared okay, then they are told to commit, otherwise, they are instructed to roll-back.

Although widely used in industry today, the DTP model is procedure-oriented and is therefore not well suited to the more modern concepts of object-oriented computing. For the CORBA platform, the OMG has specified the Object Transaction Service (OTS) which extends transaction semantics to distributed object-oriented applications. The next section provides an overview of the OTS.

CORBA Object Transaction Service

In the OTS model, there are three classes of application object, namely, transactional clients, transactional servers, and recoverable servers. A *transactional client* is simply an application program that creates a transaction and invokes operations on *transactional objects* within its scope. Transactional objects are those objects that are somehow affected by being invoked within the scope of a transaction. Any transactional object that directly manages persistent data is known as a *recoverable object* and has to participate in the transaction completion protocol. They do this by creating `Resource` objects and registering them with the transaction service. The transaction service drives the two phase commit protocol by issuing requests to the resources that are registered with the transaction. A *transactional server* is one that contains one of more transactional objects, none of which have recoverable state whereas a *recoverable server* is one that contains one or more recoverable objects.

The OTS is a flexible service that allows transaction applications to be implemented in a number of ways. Figure 1 shows the interfaces provided by the service and how they are typically used by transactional clients and recoverable servers.

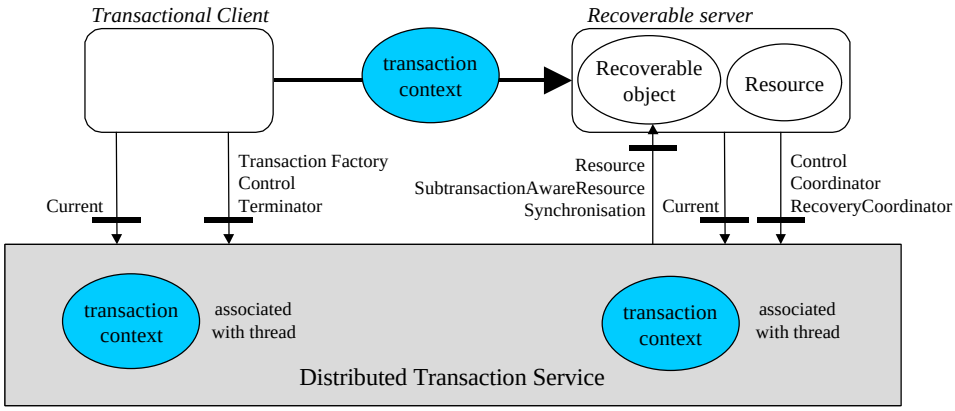

Figure 1: Overview of OTS interfaces

Fundamental to the OTS architecture is the notion of a *transaction context*. Each thread of control is associated with a context. This association may be null, indicating that the thread has no associated transaction, otherwise it refers to a specific transaction. Management of a transaction may be undertaken by an application in either a direct or indirect manner. In the direct approach, the transaction originator issues a `create` invocation on a `TransactionFactory` object to begin a new transaction. The factory returns a `Control` object for the transaction that enables two further interfaces to be obtained: the `Terminator`, which allows an application to end a transaction and the `Coordinator` which is used to register participants with the transaction. These objects are expected to be passed as explicit parameters in operations on transactional objects since transaction creation using these interfaces does not explicitly change a thread's current context. This is known as *explicit propagation*. Using the alternative indirect approach, transactions are created by invoking the `begin` method on the `Current` pseudo object, which automatically associates the newly created transaction with the current thread. With indirect context management, it is usual to use *implicit propagation*, to pass the transactional context to transactional objects. With this approach the signatures of the methods of the objects are not modified, rather the object inherits from the empty OTS interface `TransactionalObject`. This provides the necessary indication to the ORB that the transactional context should be propagated for all methods of the object[2]. The object can then use its local `Current` object to perform transactional operations.

The OTS specification is also flexible as regards the degree of distribution inherent in the implementation of the service. Some OTS implementations, for example, OrbTP from Groupe Bull [OrbTP], adopt a centralised approach in which all transactions are created and managed by a single Transaction Manager. The libraries that are linked in to the transactional clients and servers map the OTS interfaces to remote invocations on objects that reside within the centralised transaction manager. An alternative approach (e.g., the Iona/Transarc OTS [OTM]) is to distribute transaction management so that each transaction is managed by the client that created it. This approach has the advantage of not having a single point of failure, i.e., the failure of a component of a transaction will cause only that transaction to block while others may proceed. The OTS implementation used for our experiments, OTSArjuna from Arjuna Solutions, can be configured to support both models [Arjuna].

[2] Implicit transaction propagation requires support from the underlying ORB. This feature is not available in all current ORB products.

A full description of the OTS is beyond the scope of this paper; for more details consult the OMG's specification [OMG95].

Example: an inter-organisation meeting arranger application

To illustrate how transactions can be used to support e-business, consider the simple example of a distributed meeting arranger application. The diagram in Figure 2 shows two diary objects, Alice's diary (`Diary`$_A$) and Bob's diary (`Diary`$_B$), located within their respective organisations (A and B). Alice's secretary in organisation A is using the application to schedule a meeting between Alice and Bob. In order to check availability and to create an appropriate appointment in the two diaries the application has to perform read/write operations on both of the diary objects. In order to ensure the integrity of the objects is preserved in the presence of possible concurrent access and occasional failure, the diary objects are implemented as OTS recoverable objects and the application performs its manipulation of them within the scope of transactions.

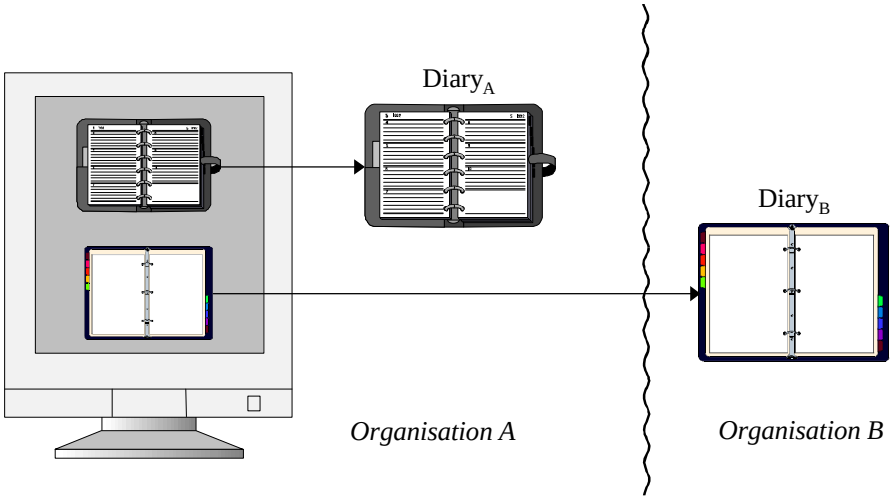

Figure 2: Inter-organisational meeting arranger application

For this example we shall assume indirect context management, explicit transaction propagation and the use of a centralised transaction manager executing within organisation A. The sequence of events that take place during the arrangement of the meeting are illustrated graphically in Figure 3 and are described below:

1-2 The client begins a transaction by invoking the `begin` method on the `Current` pseudo object within the client's address space. The implementation of `begin` in the OTS library causes a remote invocation of the `create` method on the `TransactionFactory` object residing in the transaction manager process. This causes the creation of `Control`, `Coordinator` and `Terminator` objects in the transaction manager process. In the client process, the newly created transaction context is associated with the client's current thread.

3 The client invokes the `get_control` method on `Current` which returns a reference to the `Control` object residing in the transaction manager.

4-6 The client then invokes the `make_appt` method of Alice's diary (`Diary`$_A$), passing the reference to the `Control` object as an explicit parameter. The implementation of `make_appt` creates a new `Resource` object that is responsible for managing the persistent state changes to the diary object. A reference to the `Coordinator` object

is obtained through the `get_coordinator` method of the `Control` object. The newly created `Resource` is then registered with the transaction by invoking the `register_resource` method of the `Coordinator`. Although not shown in Figure 1 for simplicity, `register_resource` returns a reference to a `RecoveryCoordinator` object (RC_A) that is created in the transaction manager. This object can be used to complete the transaction after recovery from a failure.

7-9 Repeat of operations 4 – 6 for $Diary_B$.

10 The client triggers the completion of the transaction by invoking the `commit` method of `Current`.

11-12 The implementation of `Current::commit` first obtains a reference to the `Terminator` object using the `get_terminator` method of `Control` and invokes the `commit` method on the `Terminator`.

13-16 This initiates the 2-phase commit protocol; in the first phase the `prepare` method is invoked on the two resource objects ($Resource_A$ and $Resource_B$). If both respond successfully then in the second phase the `commit` method is invoked on the two resources thereby completing the transaction.

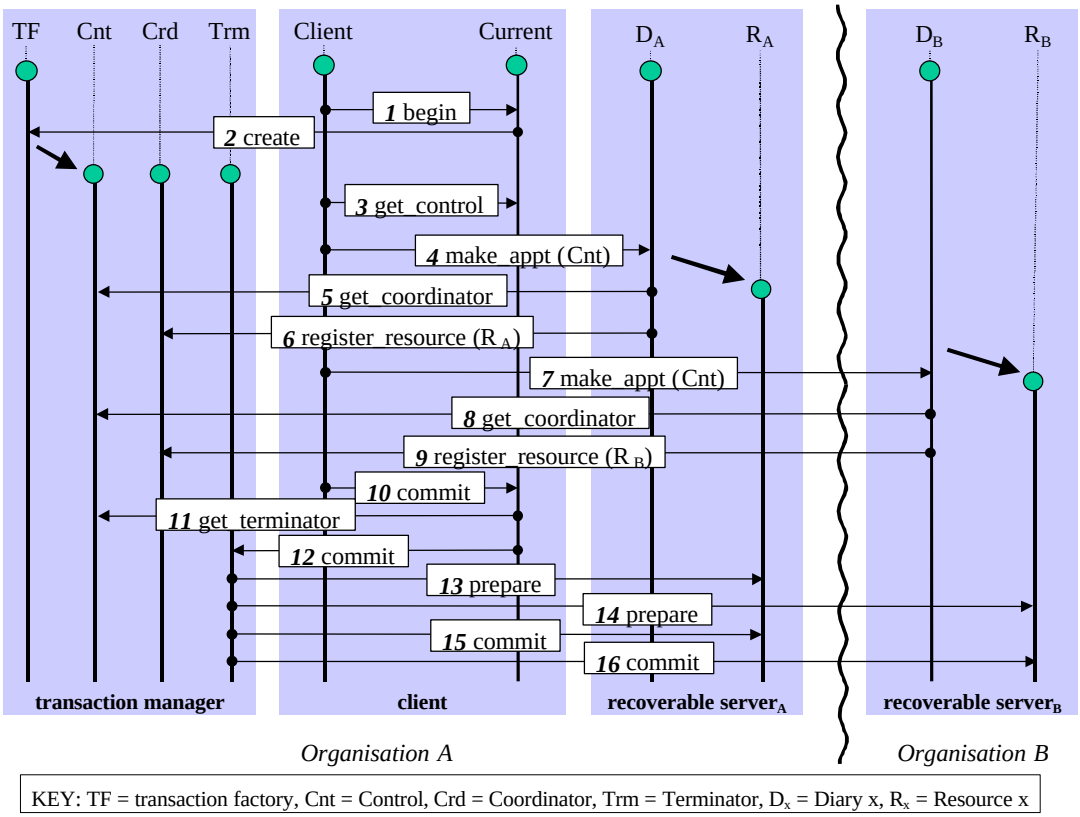

Figure 3: Meeting arranger application walk-through

Organisations protect their internal systems, and the business critical data they contain, by putting a firewall between an external network and their internal network The network topology is arranged so as to form enclaves where all traffic into or out of the enclave must go through the firewall. It is common for the whole of an organisation to form an enclave; the more security conscious organisations may have inner enclaves containing particularly sensitive information protected by internal firewalls.

The example in Figure 3 will now be extended to add a firewall belonging to each organisation.

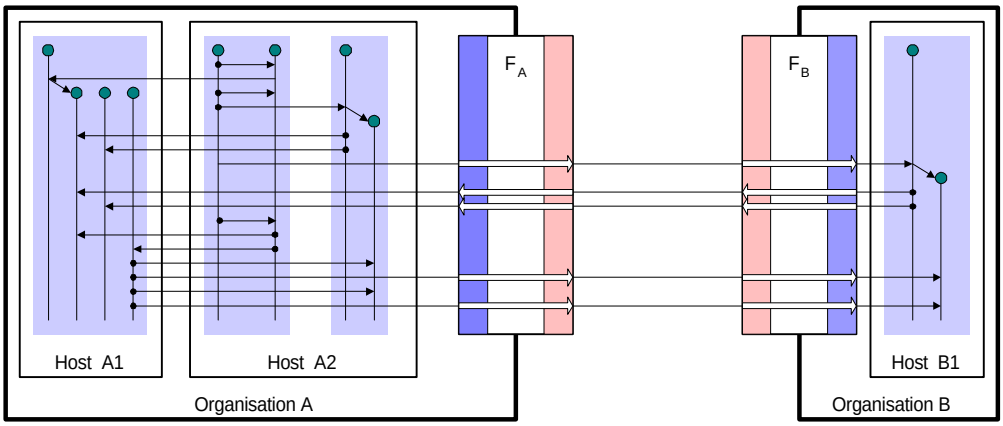

Figure 4: Meeting arranger interactions through firewalls

For the purpose of this example, we consider the case where each organisation forms a single enclave so that we have two firewalls, F_A and F_B, one belonging to each organisation.

Note that interactions go through both firewalls in both directions, and involve objects that already existed and new objects created for the transaction. This can present problems for conventional firewall mechanisms.

Firewalls and CORBA

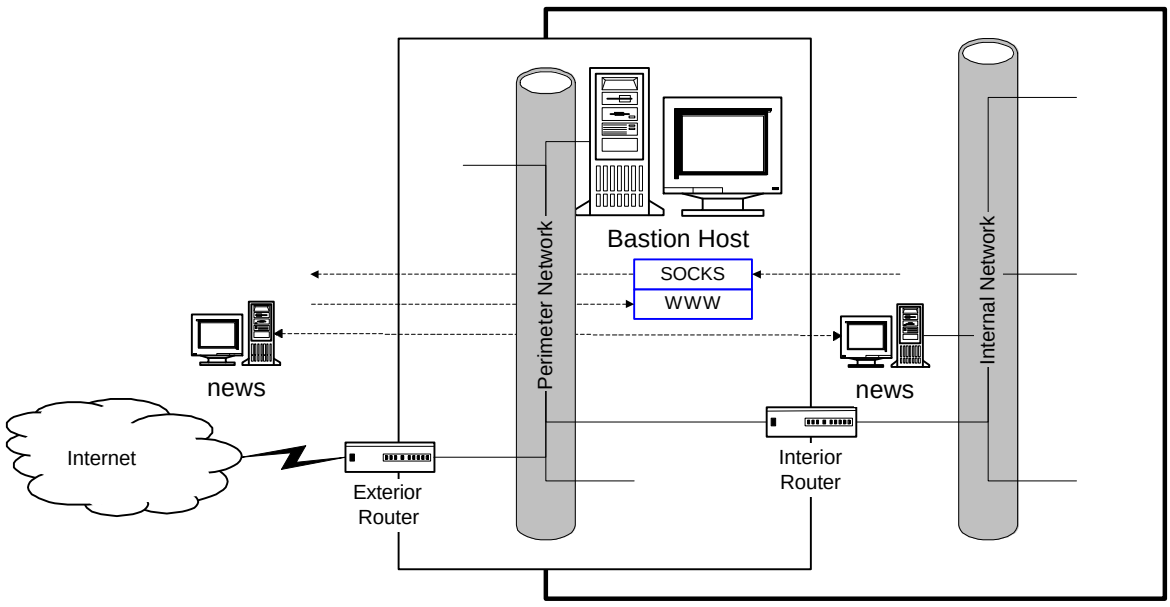

Figure 5: Typical screened subnet firewall

Figure 5 shows a typical screened subnet firewall configuration [Chapman95] with a bastion host. Outbound connections can be made via SOCKS on the bastion host, or via some other

proxy. Inbound connections, for services such as WWW are usually allowed in only as far as a bastion host. Some inbound connections may be permitted through to the internal network, but these will be restricted to connections from a specific external host to a specific internal host on a specific port. For example, a connection to deliver usenet news from the known news host of a service provider to the news host of the organisation on the standard NNTP (NetNews Transfer Protocol) port 119.

Exterior and interior routers configured in terms of IP addresses, ports and the ACK bit in the TCP headers. The more sophisticated routers can also inspect the traffic for some commonly used application protocols (e.g. HTTP, FTP) to grant some permissions dynamically, and to reject traffic where the application protocol is not followed correctly.

In our transactional application, the essence of the requirement it that the incoming invocations act on the live data, and data that must be held on the internal network. The invocations may originate from various external sources, and may be directed at various hosts on the internal network. The application requirements go beyond those normally satisfied by a firewall, and the challenge is to satisfy those requirements without compromising security.

A further difference from the conventional firewall is that CORBA does not fit particularly well into the model where control is based on hosts and port numbers known in advance. The strengths of CORBA as a platform for building distributed applications rest on hiding the location information from application objects. CORBA applications never need to deal with hosts by name (or IP address), nor do they need to be aware of port numbers. CORBA allows objects to be invoked without the invoker needing to know whether the object is local or remote, or where it is in relation to any other object.

With a typical ORB, there may be several objects in a process, several processes containing objects on a host, and several hosts within any enclave. Figure 4 shows that even for the simple example we have chosen this may be the case. This flexibility of CORBA makes it hard to know in advance which hosts will need to make or to receive invocations. CORBA also does not depend on any particular well-known port being used; it is common for servers to allow the system to choose an arbitrary port for incoming invocations. Figure 6 shows the connections that would be needed through the firewalls in order to support the meeting arranger application, if we were to permit direct connection.

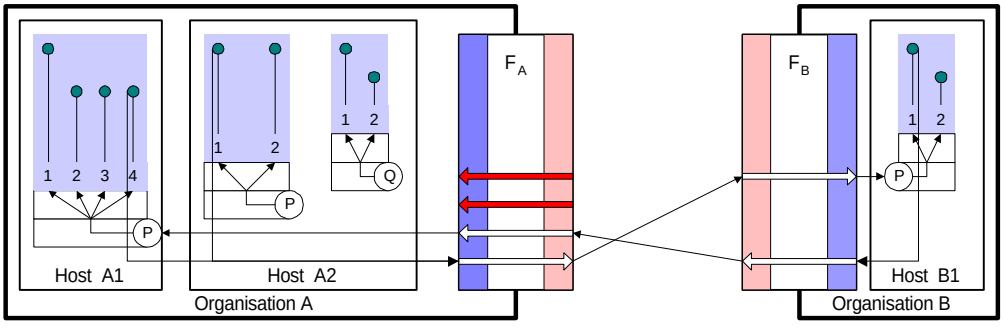

Figure 6: Meeting arranger connections through firewalls

Here we show a process on host A1 listening on port P for requests for its objects. A CORBA Interoperable Object Reference (IOR) for the Internet Inter-ORB Protocol (IIOP) contains the host and port, and an “object key” that identifies the object in the context of that host and port. In Figure 6 the objects have been labelled 1,2,3 and 4, and we will assume that these are the object keys. An IOR contains additional information, but it is the host, port and key that are significant in this discussion, and we will show IORs in the form IOR[h:p:k] where we

need to discuss their contents. For example, the IOR of the Transaction Factory in our example is IOR[A1:P:1]. Note that the same port may be used on different hosts, but need not be. The same object key may be used by different objects, or the keys may be different, and the keys may have internal structure that differs depending in which ORB implementations are used.

The key issues for configuring the firewall are:

- How do we know which connections need to be enabled?
- How do we limit connections so that only essential objects can be accessed?

In this scenario, the client object on host A2 invokes $Diary_B$ (at IOR[B1:P:1]) passing the reference to the Control object for the transaction (IOR[A1:P:2]). If there were no firewalls, the ORB hosting $Diary_B$ would simply connect to port P on host A1 to invoke the Control object. Picking the information A1:P out of the request requires knowledge not only of the IIOP protocol, but also the definition of the Diary interface.

Even if we could construct a router with enough CORBA capability to discover the host and port, simply enabling incoming connections on that host and port combination would expose all of the objects hosted by that ORB instance.

The problem faced by the router also applies to a transport level relay approach such as SOCKS. A SOCKS proxy would have to be extended to understand CORBA well enough to find the host and port data.

In order to achieve the degree of control that we want, we add a CORBA service to the firewall that can mediate the interactions to ensure that only the authorised interactions take place. This service must permit all of the interactions that are necessary to complete the transaction, and we shall adopt the policy that by authorising the invocation of the transactional object – the diary in our example – we implicitly authorise invocations of the objects created to support the transaction.

This mediation service is provided by the Object Gateway.

The CORBAgate Object Gateway

Design Goals

The object gateway should not require changes to the application. For the meeting arranger example, this means that the existing meeting arranger client and the diary servers should not need to be changed when the object gateways are introduced.

The object gateway should enforce access control at a level that matches the application. The enforcement mechanism must work at the level of individual objects. Management of access should be in terms of collections of objects that participate in a series of related interactions.

The administration overhead should be low. Adding new applications and adjusting the access control policy should be the only administration required.

The gateway should be extensible to allow new kinds of application and new kinds of access control to be introduced.

Deployment of new applications should be rapid. The time from developing or acquiring a new application to having it deployed through the gateway should be short.

Object Gateway Structure

The design goals are achieved by creating proxy objects in the gateway. Outbound proxies accept invocations from clients in their own enclave, and then become the client making a corresponding invocation on an object outside the enclave. Inbound proxies accept invocations from clients outside the enclave, and become the client making a corresponding invocation on an object inside the enclave.

As far as the client is concerned, the proxy is the server; as far as the server is concerned, the proxy is the client. This means that one proxy may invoke another proxy just as if the first were the original client, and the second the final server. This makes it possible for each organisation to deploy an object gateway and the interactions to go through both with no additional functionality or configuration requirements.

Figure 7 shows the object interfaces that need to be exposed for the meeting arranger application. Note that since our example is using a centralised transaction manager then the client's invocations on the `Current` pseudo object are mapped *under the covers* onto remote invocations on the `TransactionFactory` object.

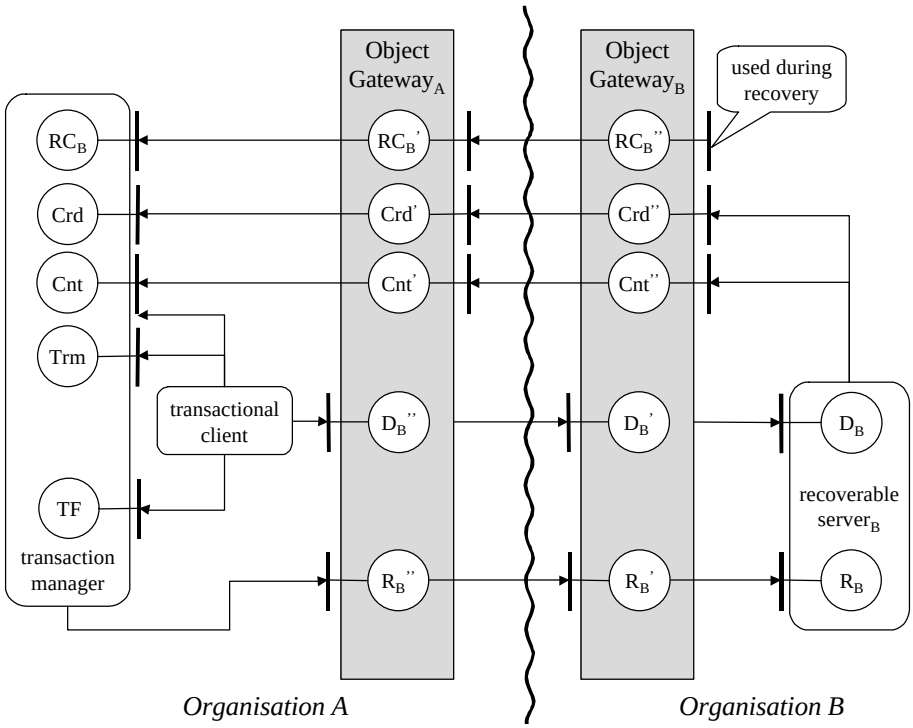

Figure 7: Meeting arranger application through object gateways

Some of the object references have to be known in advance, e.g., the transaction manager and the Diary object, D_B. Either, these IORs must be *well-known* to all of the parties through some external mechanism, or, more likely, the IORs will be found through naming or trading services. Other objects are created dynamically as the application executes, such as the resource object, R_B, and the recovery coordinator, RC_B. In the case of the well-known references, it would be possible to create the object gateway interceptors a priori. However, this cannot be done for the objects that are created dynamically by the application. In general, if the application passes an object reference through the firewall then there is a requirement to create a proxy for that object reference, inbound or outbound as appropriate.

Proxies and reference mapping

Using an individual proxy for each target object provides access control at the level of objects. Access can be denied by choosing not to create a proxy for a target object. The gateway also supports revocation of access. Proxies can be shut down to deny access that had previously been granted without having to shut down the target service. The access control and revocation mechanisms that have been implemented are described below.

Using proxies also means that neither the application code nor the ORB it runs on need be aware of the gateway or the firewall. Provided that the application is given references to the proxies rather than the target objects, the underlying CORBA mechanisms hide the differences. The proxies in the gateway must be able to detect and replace object references as required in the invocations that pass through. Figure 8 illustrates this process. A client object Ca has a reference to a server object Sa, and passes this in an invocation request. The target of this invocation is server object Sb, but Ca has a reference to Sb' a proxy for Sb. The proxy Sb' replaces the reference to Sa with a reference to Sa', a proxy for Sa, when it forwards the invocation to Sb. When Sb receives this reference, it cannot and need not know that the reference is to a proxy and not the ultimate target object. If Sb uses the reference in an invocation, the invocation request goes to Sa' which forwards it to Sa.

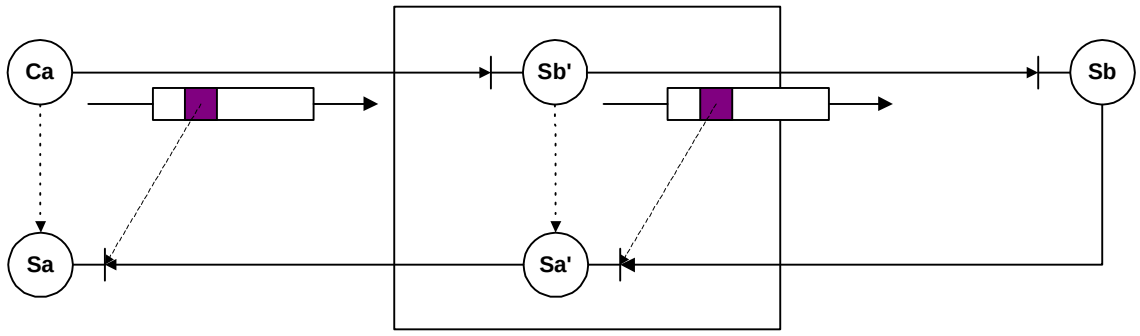

Figure 8: Proxy creation and reference mapping

The proxy Sa' could be created when Sb' creates the reference for it, but it need not be created until it is invoked. The gateway implementation uses the lazy incarnation policy and does not create proxies until they are invoked.

The gateway is designed to be easy to install, and to require no routine maintenance. It manages its own resources so that it can run indefinitely. Access control is enforced automatically according to an extensible set of policies. An example of such a policy is described below.

The gateway is designed to be extensible. Application specific proxies that provide custom functionality can be added easily, and in particular, additional access control policies can be added.

Application specific proxies

Custom functionality is provided by application specific proxies. These are specific to the type of the target object as described by its IDL definition.

An application specific proxy is written as a server for the required type that is also a client for that type. A body must be implemented for each operation in the interface. This body must

translate any object references in the parameters that it receives. Support classes with the required translation operations are provided.

The operation bodies can also include application specific logic. This has been used to modify the object reference translation so as to implement particular access control policies.

Since the IDL is known, so are the places where object references are passed as parameters. This means that application specific proxies can give the best performance that is possible for a proxy implemented as a CORBA object.

The disadvantage of an application specific proxy is that the type specific code needs to be written, even if it has no additional application specific logic. Development time must be allocated, and this could delay the introduction of a new service.

Generic proxies

In addition to any application specific proxies, the gateway has generic proxies. These proxies retrieve interface definitions from the Interface Repository - a standard CORBA service - and use that information to discover where there are object references in the parameters.

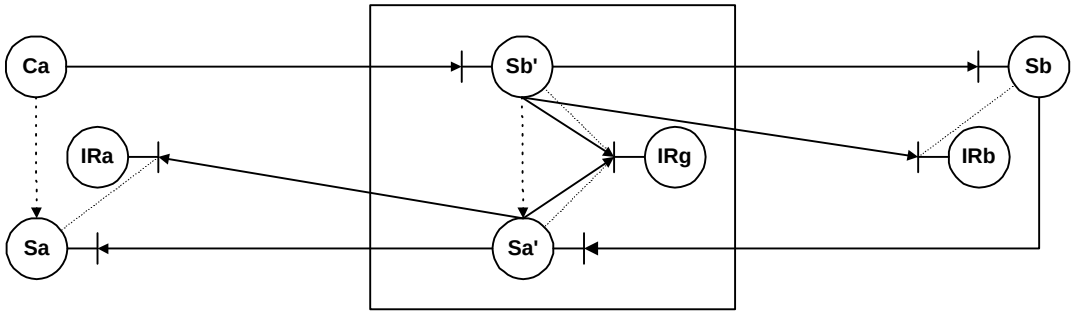

Figure 9: Generic Proxies and Interface Repositories

Figure 9 shows the relationships between the application objects, generic proxies, and Interface Repositories. In principle, CORBA objects should always be able to respond to a ‘get_interface’ request with a reference to an object that describes the interface to the object. These interface description objects are usually implemented by an Interface Repository. Some ORBs do not have an Interface Repository, and even if they do, the necessary definitions might not have been installed. For example, the object Sb may be able to respond with a reference to an object in IRb, its local Interface Repository. If it cannot then the generic proxy Sb’ can look up the definition in its local interface Repository IRg. The necessary interface definition must have been installed in one or other of these interface repositories for a generic proxy to be able to process the invocation. Similarly, the generic proxy Sa’ may be able to obtain a reference to the definition object in IRa from Sa, or it may look up the definition in IRg.

A generic proxy can act for an object of any type. This means that the gateway can handle any service that has its IDL installed in the Interface repository. To deploy a new application, all that is required is to add the interface definitions to the Interface Repository. New definitions can be added while the gateway is running, so deploying a new type of application is a very simple and rapid process.

The main disadvantage of the generic proxy is that it has more work to do to discover where object references occur in the parameters. The other disadvantage compared to an application specific proxy is that there is no way to add custom functionality.

Revocation

It is useful to be able to revoke access to an object without shutting down the object. Since the target objects can be accessed only through proxies, access can be revoked by shutting down proxies. The revocation mechanism must match the automatic reference mapping and proxy creation mechanisms; it must undo whatever has been done by those automatic mechanisms.

Each proxy has a tag, implemented as a string, which it propagates into the references that it creates in the mapping process, and the proxies incarnated to support those references. The revocation mechanism operates by shutting down all the active proxies with a given tag. With the lazy incarnation policy mentioned above, it is also necessary to have a check at incarnation time that will fail when the tag has been revoked.

Figure 10 illustrates tags in both generic and application specific proxies. The revoke operation will shut down both kinds of proxy; it shuts down all proxies with the specified tag.

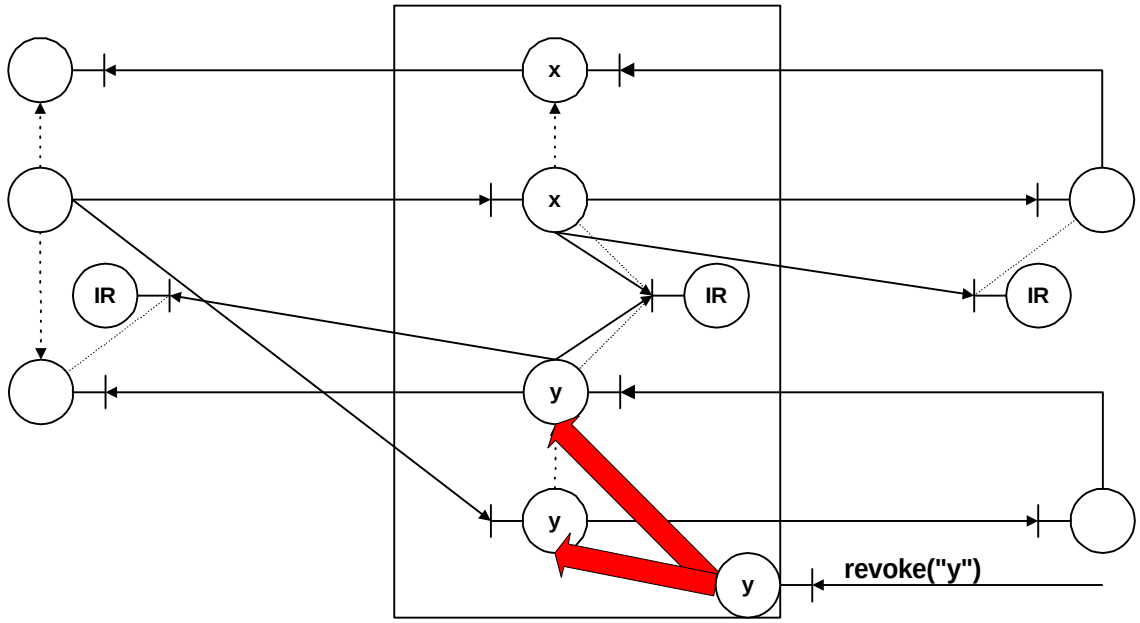

Figure 10: Tagging and revocation

Since tags propagate to references and proxies as references are passed as parameters, it is the means by which a reference is obtained that determines which access is revoked. A target object may have several proxies with different tags. Revoking the proxies with one of these tags will not affect the other proxies. This means that access granted for one purpose is not affected if access granted for a different purpose is revoked.

The access control policy is determined by how new tags are generated, and by the validity test applied to a tag before a proxy is incarnated. These are controlled by application specific proxies. When mapping a reference, an application specific proxy can generate a new tag rather than just propagating the existing tag. An application specific proxy can also register itself as the validator for certain tags. An example of this is given below where the naming service based access control strategy is described.

Access control strategies

One of the access control strategies that has been implemented uses the standard CORBA naming service.

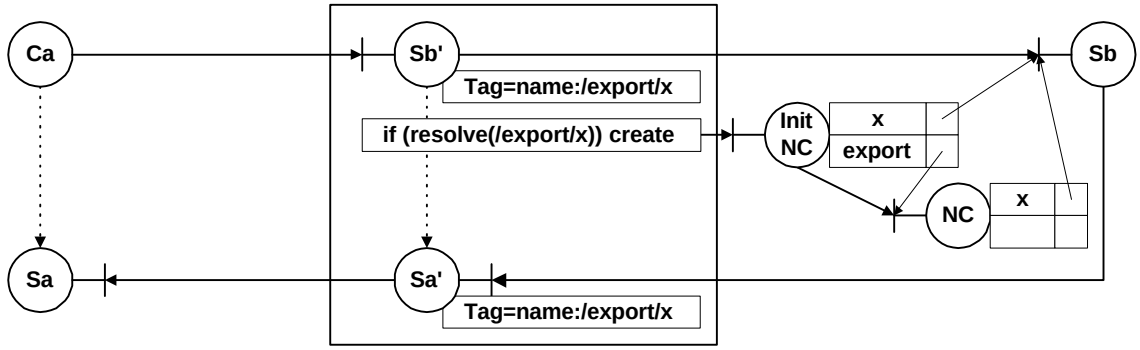

Figure 11: Access Control using the Naming Service

Figure 11 shows a server Sb that we shall assume has bound its reference under the name “x” in its initial naming context which is implemented by the object “Init NC”. A second naming context, implemented by the object “NC”, has been bound under the name “export”, and another binding of Sb has been made in that context, also under the name “x”. The proxy Sb’ has the tag “name:/export/x” which it propagates to the reference for the proxy Sa’. When the time comes to incarnate Sa’, this tag indicates that the access control policy is based on using the naming service, and that the validation test is to check that the name “/export/x” can be resolved. Incarnation of Sa’ is controlled by the presence of the binding with name “x” in the context bound under the name “export” in the initial naming context. This binding can be deleted or recreated without affecting the server itself, or its binding in the initial naming context.

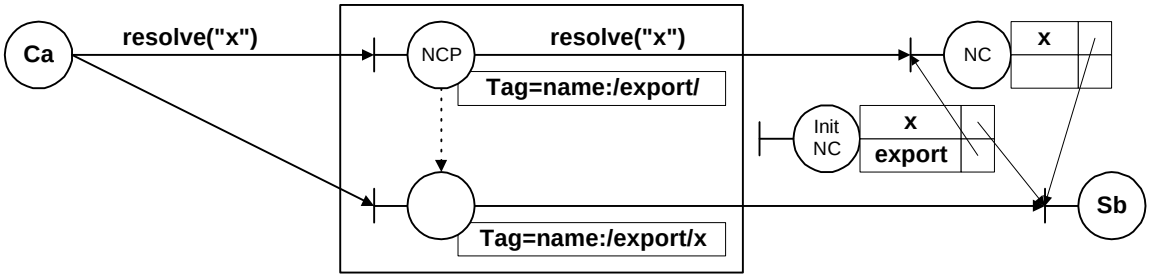

Figure 12: Naming context proxy

Figure 12 shows how a naming context proxy is used to create new tags. The naming context proxy NCP takes special action when mapping the reference that it returns as the result of the “resolve” operation. It generates a new tag by adding the name being resolved to its own tag. The effect is that access to the objects bound in the context for which NCP is the proxy can be revoked independently. If the external clients are given the proxy for the ‘export’ context as their initial naming context, it becomes very easy to control which objects are visible. An object is made visible by creating a binding for it in the ‘export’ context; access is revoked by deleting that binding.

The structure of the gateway allows other application specific proxies to generate different kinds of tags, and register themselves as the validation mechanism for those tags.

Statelessness and reference embedding

In CORBA systems, it is normal practice to create short-lived objects for some purpose, and pass their references to other objects. The Object Transaction Service uses objects of this kind. If the gateway is to run for any extended period, it must not consume additional resources for each object reference that passes through, nor must it accumulate additional proxies.

The reference mapping system implemented in the gateway puts all of the information needed to create a proxy into each mapped reference. Proxies are created on demand from the information in the incoming requests.

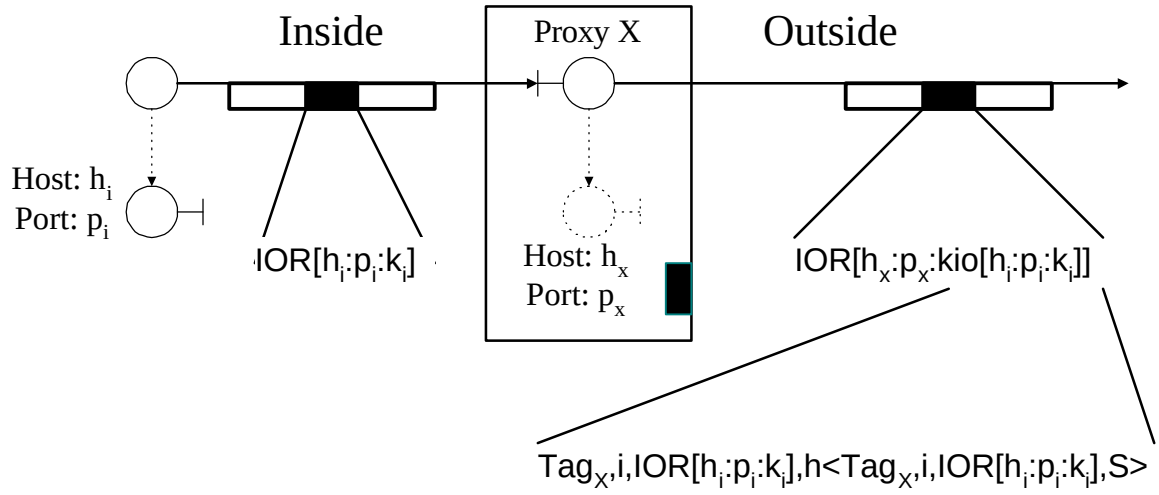

Figure 13: Structure of mapped references

Figure 13 shows the structure of mapped references, and how they relate to the original target reference. The object key in a mapped reference contains the information needed to create the proxy: the original target reference, the tag for the proxy, and a flag to indicate whether the proxy is inbound or outbound. The object key also contains a secure hash of this information and a secret known only to the gateway.

Since all of the information used to create a proxy is in the mapped reference, it is important to ensure that an attacker cannot make a successful invocation with some forged object key. An attacker can obtain the host and port of the gateway from any valid mapped reference. An attacker can also discover or guess a valid tag easily. If the attacker also discovers or guesses an object reference for some internal object that should not be accessible from the outside, the attacker must be prevented from combining this information in such a way that it can make an invocation that the gateway will forward to the target object. It is the secure hash that prevents the attacker from constructing an object key acceptable to the gateway.

Invocation requests arriving at the gateway include an object key, and this will designate a proxy for some target object. If the proxy exists, the invocation can proceed. If the proxy does not exist, an activator is called to create the necessary proxy. It is this activator that checks the secure hash and the validity of the tag in the object key, and creates the proxy only if both tests pass.

Transactions through the gateway

The object gateway described above has the mechanisms necessary to support the interaction patterns that occur in distributed transactions across firewalls. To complete the picture, the

appropriate mechanisms must be triggered by the interactions that can occur in practice when a transaction takes place.

The simpler case is where the transaction context information is passed by explicit propagation. In this case, an additional parameter is added to each operation on the transactional objects. These additional parameters appear in the IDL, and all of the type definitions associated with the transaction support objects must be explicitly available. As far as the gateway is concerned, all of the transaction support is just part of the type, and needs no special processing. The existing generic proxies can discover all that they need from the description registered in the interface repository; application specific proxies can also be used. The proxies do not take an active part in the transaction, they simply pass on the invocations.

Transaction context information may also be passed by implicit propagation. In this case, the IDL does not have any additional parameters to the operations. The necessary transaction context information is passed in the invocation request and response messages, separately from the parameters. The CORBA General Inter-ORB Protocol defines a place for this information in the messages, but there is no standardized way to use this facility. The Object Gateway must include special processing for the implicitly propagated transaction information.

Related Work

There are some products that can be used to allow some kinds of CORBA interactions across firewalls. There is also a specification, recently adopted by OMG, that sets out to address this issue. These will be described briefly, with emphasis on the different objectives of the various approaches.

OMG CORBA/Firewall Security Specification

The Object Management Group has recently adopted a specification for a mechanism to allow CORBA interactions to cross firewalls. The primary focus of this specification is to define how to change CORBA so as to allow interactions across current conventional firewall components. A new GIOP proxy firewall component is also described in the specification.

The specification describes changes to Interoperable Object References (IORs) to include additional data, as well as changes to the GIOP protocol, and the Portable Object Adapter. All of these involve changes to ORB implementations.

The OMG specification describes three kinds of firewall: TCP firewall, SOCKS proxy firewall and GIOP proxy firewall. For each of these, a particular type of component must be added to the profile in the IOR if the target object is to be invokable across that type of firewall. A profile includes a list of these components in order to support interaction across multiple firewalls.

Unlike CORBAgate, the OMG specification is based upon having each object create its own references containing the components that describe all of the inbound firewalls that need to be traversed to reach the object. There are two issues with this approach:

- Every object that creates a reference that might need to be used from an outer enclave must have been configured to know about all of the inbound firewalls. There is no description of how this configuration is to be done, and the issue of managing the configurations of all the servers is not discussed.
- There is no anti-forgery mechanism, and, since every server object can create references, there is no plausible way to add such a mechanism. Neither the TCP nor the SOCKS

mechanisms inspect any CORBA specific data, so they cannot perform any CORBA reference validation. A GIOP proxy can inspect the CORBA specific data, but it has no way to verify that the target object included the inbound proxy data in the reference. If the proxy is to reject any incoming invocations, it must have an access control mechanism separate from the reference. The OMG specification does not discuss the issues involved in creating an access control mechanism that can permit invocations of dynamically created objects.

The OMG specification also requires that a client inside an enclave be configured to know its outbound proxy. The configuration management issues are not discussed.

Clients, including proxies acting as clients, are presented with references containing a list of firewall components and a target reference. It is up to the client to determine where it is relative to the firewalls in the list, the target object, and its own outbound firewall. The reference contains all the information needed to invoke the target from anywhere, and it is up to the client to determine how much of that information it needs.

The emphasis of the OMG specification is on changing CORBA in order to live with current popular firewall mechanisms, and other security restrictions. In particular, some of the mechanisms are specifically aimed at applets that are to act as CORBA clients and servers despite both the applet security model imposed by the browser running the applet, and a client side firewall which has not had any CORBA-specific functionality added. These objectives are different from the CORBAgate design goals.

Orbix Wonderwall

Orbix Wonderwall is perhaps the best known product designed to support IIOP across firewalls [Iona]. Unfortunately, there is no publicly available detailed information available about the current version. Therefore the remarks here have to be based upon the white paper dated November 1996, and the Wonderwall Administrators guide dated May 1997.

Clients wishing to access servers through Wonderwall must be given 'proxified' IORs. Since IORs with the OMG specified firewall components are not yet available, the proxified IOR cannot contain all the necessary information for servers implemented on arbitrary ORBs. References created by Orbix and OrbixWeb have an internal structure understood by Wonderwall, which contains sufficient information to discover the target object. For non-Orbix servers, Wonderwall must be explicitly configured to know the target IOR for each individual object.

Orbix and OrbixWeb servers can create proxified references, but mechanisms provided seem to have an all or nothing approach. Either all references created by the server are proxified, or none are. This is a problem for any server that is to be used from within an enclave as well as from outside.

Wonderwall can provide access control at the granularity of individual objects for Orbix and OrbixWeb objects, provided that the references are well-known, and specified in the Wonderwall configuration. If objects are created dynamically, access control must be based on the host and port, rather than the individual object.

None of the information available to us mentions 'callbacks' - invocations from what was initially the server side of the firewall. Support for this interaction pattern is essential for transactions across the firewall.

The GIOP proxy part of the OMG specification addresses many of the problems described here. Since Iona Technologies, the suppliers of Wonderwall, were one of the submitters of the proposal adopted by OMG, it is likely that Wonderwall will evolve in that direction.

VisiBroker Gatekeeper

VisiBroker Gatekeeper is primarily intended to support Java applets that connect back to server objects after being downloaded into a browser [Inprise]. Much if the emphasis is on providing ways for applets to operate despite the security restrictions imposed by a browser. Gatekeeper also makes it possible to interact across a firewall.

Some experiments suggest that the client must be using the VisiBroker ORB in order to use Gatekeeper successfully, but the target server may be using a different ORB. The client needs some configuration information in order to know where to find Gatekeeper. This is well matched to the downloaded applet scenario; all connections must go back to the download host, and configuration parameters can be supplied in the download. The absence of any mention of 'proxification' or any other reference modification also suggests that the client must be using something other than the information in the IOR.

Gatekeeper is not intended to provide any kind of access control. There is no apparent mechanism to limit the objects that can be invoked through Gatekeeper. Although this may be a security issue, it means that there is no problem in accessing dynamically created objects.

Gatekeeper does support callbacks, but it is not clear if this means that every object that acts as a client must be using VisiBroker.

There is no mention of nested enclaves, or crossing multiple firewalls, in any of the Gatekeeper documentation.

Although this has not been verified experimentally, Gatekeeper seems to have the mechanisms necessary to support transactions across a firewall, provided that all the participants use VisiBroker, and there is no requirement for access control at the firewall.

Implementation notes

The CORBAgate object gateway has been implemented using the ORBacus ORB from Object Oriented Concepts [OOC]. Some of the functionality, such as support for the stateless proxies, was only possible because of the availability of the ORBacus source code. The transaction service used was OTSArjuna from Arjuna Solutions. OTSArjuna is an OTS-compliant transaction service that is portable across many ORB products including ORBacus. Some interoperability testing was also carried out using Visibroker from Inprise. The platforms used were HP-UX and Windows NT.

Conclusions

Transactions and security (in particular firewalls) are essential elements of the e-commerce framework. The objects that make up a transaction (application objects, transaction infrastructure objects such as recovery co-ordinators) are deployed over both sides of the firewall and will need to be visible to each other in a controlled way through the firewall.

The configuration required for transactions poses problems for firewall design. In particular, the case that client-side objects may be invoked by other objects involved in the transaction,

and may in fact not even reside in the same process causes significant problems with current firewall offerings.

CORBAgate demonstrates that currently available ORB technology is sufficient to create a firewall component that can support the required interactions, without unnecessary exposure on unrelated objects.

Acknowledgements

This work has been partially funded by the European Union as part of Esprit Fourth Framework Programme Project No 26810 (MultiPLECX). David Ingham's contribution has also been partially funded by a grant from Hewlett-Packard through their External Research Programme.

The authors would like to thank Chris R. Dalton, Paula Dotti, Nigel Edwards, Mark Little, Santosh Shrivastava and Qun Zhong for their useful comments and suggestions during the writing of this paper.

URLs

[Arjuna] Arjuna Solutions homepage.
<URL:http://www.arjuna.com/>

[Inprise] Inprise homepage.
<URL:http://www.visigenic.com/>

[Iona] Iona homepage.
<URL:http://www.iona.com/>

[OOC] Object Oriented Concepts homepage.
<URL:http://www.ooc.com/>

References

[Bernstein87] P. A. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison Wesley, 1987.

[Chapman95] D. B. Chapman, E. D. Zwicky, "Building Internet Firewalls" O'Reilly & Associates 1995.

[DOD85] Department of Defence Standard, "DOD Trusted Computer System Evaluation Criteria," DoD 5200.28-STD, December 1985.
<URL:http://www.disa.mil/mls/info/orange/index.html>

[Edwards97] N. Edwards and O. Rees, "High Security Web Servers and Gateways," Computer Networks and ISDN Systems, 29(8-13), Proceedings of the 6th International World-Wide Web Conference, Santa Clara, USA, April 1997.
<URL:http://proceedings.www6conf.org/HyperNews/get/PAPER59.html>

[JPW91] J. P. L. Woodward, "Compartmented Mode Workstation Evaluation Criteria VERSION 1 (Final)," DDS-2600-6243-91, 1991.

[Little97] M. C. Little, S. K. Shrivastava, S. J. Caughey, and D. B. Ingham, "Constructing Reliable Web Applications Using Atomic Actions," Computer Networks and ISDN Systems, 29(8-13), pp. 1405-1416, Proceedings of the 6th International World-Wide Web Conference, pp. 563-571, Santa Clara, USA, April 1997.
<URL:http://proceedings.www6conf.org/HyperNews/get/PAPER12.html>

[Lomet77] D. B. Lomet, “Process Structure, Synchronisation and Recovery using Atomic Actions,” Proceedings of the ACM Conference on Language Design for Reliable Software, SIGPLAN Notices, vol. 12, no. 3, March 1977.

[Mastercard95] Mastercard and Visa, “Secure Electronic Transaction (SET) Specification, Book 2: Programmer’s Guide,” June 1996.
<URL:http://www.visa.com/cgi-bin/vee/sf/set/setprog.html>

[Matena96] V. Matena and R. Catell, “JTS: A Java Transaction Service API,” Sun Microsystems, December 96.
<URL:ftp://ftp.javasoft.com/pub/jts/jts.pdf>

[Microsoft97] Microsoft Corporation, “Microsoft Transaction Server White Paper,” 1997.
<URL:http://www.microsoft.com/transaction/learn/mtswp.htm>

[OMG95] Object Management Group, “CORBAservices: Common Object Services Specification,” OMG Document Number 95-3-31, March 1995.
<URL:http://www.omg.org/>

[OrbTP] Bull S.A., “OrbTP White Paper,” 1997.
<URL:http://www-frec.bull.com/dom/orbtp/doc/orbtpwp12.doc>

[OTM] IONA Technologies, plc., "The Orbix Object Transaction Monitor (OTM)," 1997.
<URL:http://www.iona.com/Developers/whitepaper/otm/otm_wp.html>

[Parrington95] G. D. Parrington et al., “The Design and Implementation of Arjuna,” USENIX Computing Systems Journal, vol.8, no. 3, pp. 253-306, Summer 1995.
<URL:http://arjuna.ncl.ac.uk/group/papers/p048.ps>

[Shrivastava97] S. K. Shrivastava, “Notes on Arjuna Transactions,” Dept. Computing Science, Newcastle University, UK, August 1997.
<URL:http://arjuna.ncl.ac.uk/Arjuna/OTSArjuna.ps>

[TIP] IETF Transaction Internet Protocol Working Group.
<URL:http://www.ietf.org/html.charters/tip-charter.html>

[Xopen96] X/Open Ltd., "X/Open Guide Distributed Transaction Processing: Reference Model. Version 3".