ISSN 1369-1961

Department of
Computer Science

Technical Report No. 748

The Protection of Migrating Objects in a Virtual Enterprise

Jean Dollimore,
Marcus Roberts and
George Coulouris

QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

April 1998

The Protection of Migrating Objects in a Virtual Enterprise

Jean Dollimore, Marcus Roberts, George Coulouris.

Technical Report No 748. Department of Computer Science, QMW London

April 20th 1998.

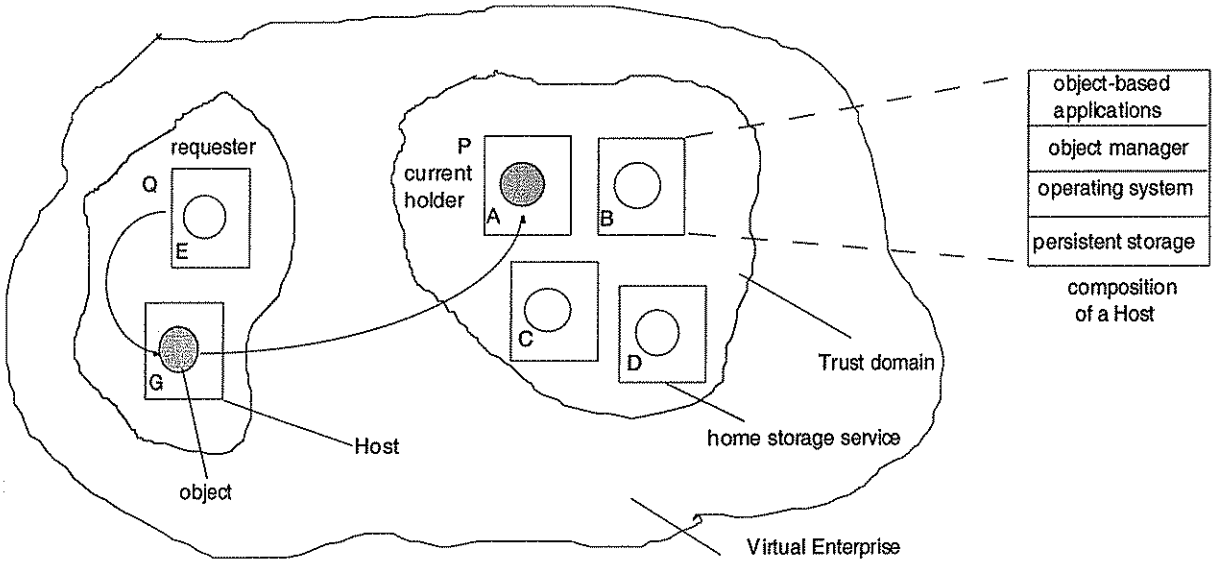

Figure 1. Object sharing in a Virtual Enterprise.

Introduction

The problems discussed in this paper have come to light during our work on the PerDiS project. (ESPRIT Basic Research Project No. 22533). For the purposes of the workshop we have re-stated the problems in a more general way in terms of migrating objects.

The context of this work is a Virtual Enterprise (VE) that consists of a group of Organisations that have agreed to cooperate with one another to carry out a particular joint task, but generally will not wish to expose information unconnected to the task to one another or allow one another to make unauthorised changes to such information. The users in the organisations in a Virtual Enterprise support their work on a joint task by sharing a number of persistent objects.

Each organisation may be the owner of some of the shared persistent objects. Ownership of objects entails two responsibilities: to be responsible for the safe storage of the objects and to specify the access rights of users to the objects. An organisation therefore has a storage service in which it keeps persistent copies and access control lists for all of the objects it owns. The storage service at the organisation that owns an object is called its *home storage service*. Objects are migrated to the local hosts of users when they need them for their work, so as to provide an effective interactive response. Figure 1 shows that each host can run several object-based applications and an *object manager* over its operating system. The object managers at the various hosts are collectively responsible for the migration, safe sharing and persistency of objects. They apply concurrency control (e.g. by using single writer/many readers locks) and access control before allowing an object to be mapped into an application.

Any user who has appropriate access rights may obtain an object and a write lock for it and may then use an application to update the local state of the object. Updated versions of objects are always sent as soon as possible to the home storage service. But to enhance the overall performance, objects are migrated from one host to another, following their use. In Figure 1, D is an object's home storage service and E is a host whose object manager is requesting the object, having supplied it previously to the OM at host G. But after E's OM supplied the object, G's OM passed it on to the OM at host A. Therefore the object can be supplied to E's OM by the current holder of the object at A.

A request for an object is sent initially to the host where the object was last heard of and then passed on until it reaches the host that currently holds the object. If the requester does not know where to ask, they send the request to the home storage service for passing on to the current holder. In our example, the user at host E requires the object, so E will ask host G for it and G will forward the request to host P.

This paper outlines a trust model for a virtual enterprise, discusses the basic security issues for migrating objects and then goes on to discuss the problems that arise when attempting to secure migrating objects.

Trust Model for a VE

The design of secure systems has generally been based on Lampson's [Lampson xx] *trusted computing base* as it is essential to have some trusted components on which to build the rest of the system. For PerDiS we derived a trust model to fit the following characteristics:

1. VE: each member of a VE is a separate organisation. Thus, the VE consists of a number of *Trust Domains* – a collection of hosts that can trust one another more than they trust hosts outside their own domain.
2. Hosts:
   - users can run applications that are not guaranteed to apply the correct protection to the shared objects that are migrated there. Therefore applications are not trusted for the purpose of access control.
   - certain **object managers** and **operating systems** are known to be **'secure'** in the sense that they can be trusted to apply the correct protection to shared objects.
3. Users: the users responsible for a particular task are trusted to behave responsibly with objects supplied to them according to their access rights. This implies that they are trusted to run a secure object manager and operating system. Users with different rights should not share a workstation.
4. Communication: authentication by means of digital signatures can be trusted to establish the originator of an authenticated message.
5. Home storage: an organisation trusts the integrity of its own local persistent storage, but not that of others. The home storage service is always trusted as a source of an up-to-date and correct copy of an object .

Within a Trust Domain.

The management and users within a trust domain believe that all hosts within that domain run a secure object manager and operating system. Therefore the object managers within a trust domain trust one another:

- to provide correct information as to the principal behind a request
- and to apply access control correctly.

Application of Object Protection and Security

We assume that each object has:

- an access control list (ACL) specifying the rights of principals to carry out its operations.
- the ACL for an object can be obtained at any site that requires it.

As our trust model says that applications are not trusted for the purpose of access control, this must be applied outside the application, either by the local object manager or by the remote object manager that has been asked to supply an object.

Auhentication and access control between trust domains

An object manager does not trust object managers outside its trust domain to apply access control correctly. We consider the migration requests and update requests separately:

1. *Migration requests*. When an object manager sends a migration request to another object manager, the latter must check whether the requesting principal has sufficient access rights before granting the request. In our example, the object manager at host A checks that Q has sufficient rights. A principal requesting an object migration needs sufficient rights to allow them to view any part of the state of the object, since the user's applications can potentially view any part of the state of an object is migrated to their OM which is not trusted to apply access control.
2. *Updates*. An update to an object is accepted only if the principal that made it has sufficient rights on the object. This rule is applied in two cases:
   - by any home storage service receiving updates;
   - by any object manager receiving a copy of an object in response to a request for it. For example, in Figure 1, the OM at E should check the access rights of the principal P at A, whose OM is the current holder.

- In the case that there is no current holder or the requester cannot accept an object from its supposed current holder, the requesting site may resort to the home storage service.

A principal behind an update request needs sufficient rights to change any part of the object's state, since their applications can potentially update any part of the state of the object and their object manager is not trusted to apply access control.

To apply access control to migration, there is a need for mutual authentication between the two principals involved in the migration of an object. In our example, these are P and Q.

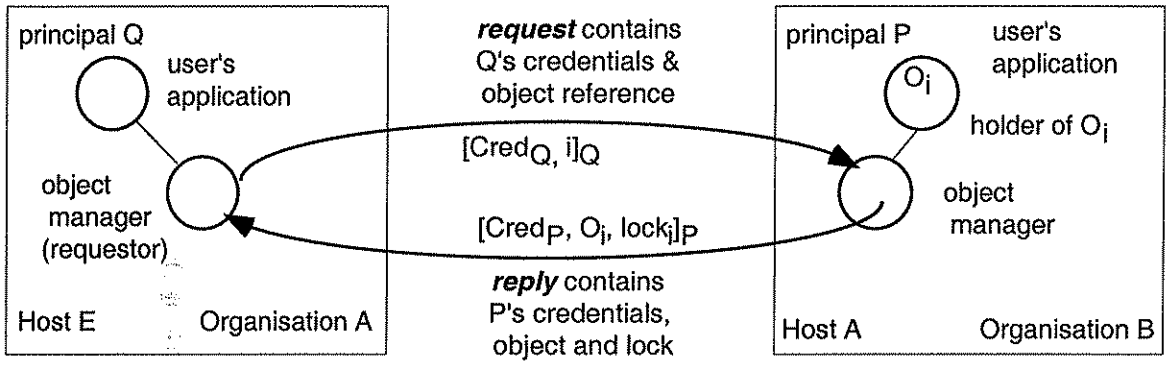

Figure 2. Migration between trust domains. [m]$_X$ denotes message *m* signed by principal *X*.

Authentication can be achieved by digital signatures signed by the two principals involved, as shown in Figure 2. The request message contains the credentials and is signed by the private key of the principal (Q) behind the request, which enables the object manager receiving the request to authenticate the requesting principal and then check their access rights. The reply contains the credentials he object and a lock and is signed by the private key of the principal (P) behind the current holder.. This enables the requesting object manager to authenticate the principal P and check that they have sufficient access rights to update the object.

Our trust model states that the home storage service of an object is always trusted as a source for that object. The home storage service in each organisation is regarded as a principal and has its own public, private key pair. This allows it to authenticate itself by signing replies to migration requests that reach the home sight.

The application of public key signatures on every request is quite expensive (e.g. signing takes about 30 ms and verifying a signature about 2-3 ms on Linux+PC.). For communication between two parties, the expense of signing can generally be reduced significantly by establishing a secure session between them as in SSL.[SSL]

Initial Problem statement for migration between Trust Domains

In a migration scheme which allows objects to flow freely between hosts as described above and to be obtained from the current holder, the requester does not know the identity of the principal that will eventually supply the object. Therefore, even if an object manager has established secure sessions between itself and the other object managers currently using the objects, it cannot decide which session key to use to sign the request.

Therefore every request from one object manager to another to migrate an object appears to require mutual authentication using public key cryptography.

Simplification for authentication and access control within a Trust Domain

Our trust model states that object managers in the same trust domain trust one another to apply access control correctly.

In Figure 3, the object manager at Host G sends a migration request to Host E in the same trust domain. Object manager G is trusted to check whether principal R is allowed to view the state of the object. Object manager E is trusted to check whether principal Q is allowed to update the state of the object.

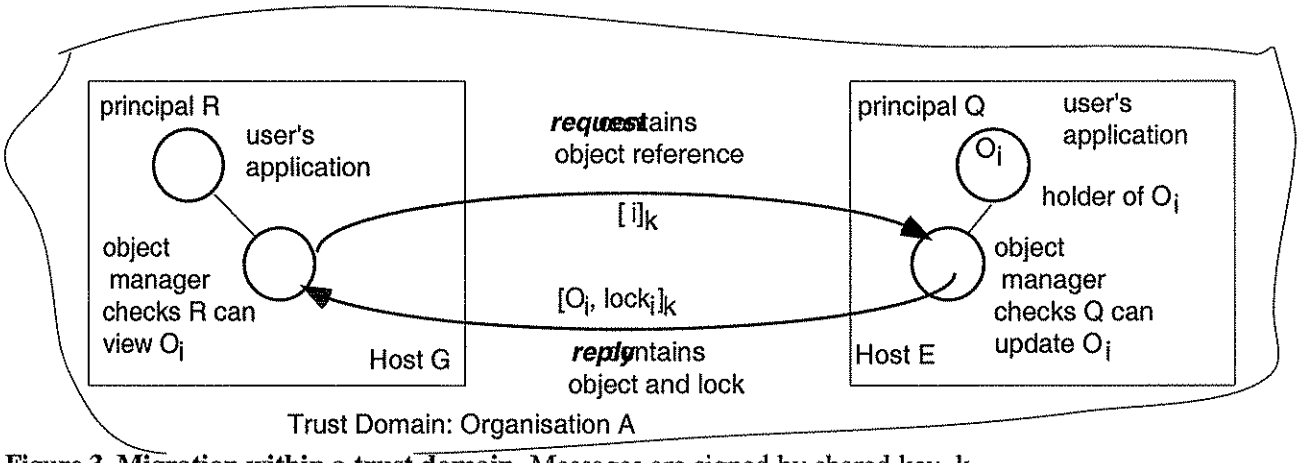

Figure 3. Migration within a trust domain. Messages are signed by shared key, k.

We have the following statements of trust between object managers in the same trust domain:

- an object manager that receives a migration request from another object manager within its trust domain can trust the requesting object manager to apply access control before allowing migrated objects to be mapped into its local user's applications.
- an object manager that receives a migration reply from within its trust domain can trust the replying object manager to have applied access control before allowing its local user's applications to update the objects.

Since this is the case, all that needs to be done is to make it possible for object managers that receive migration requests from within their own trust domain to authenticate the originator of the request as being an object manager in their own trust domain.

For this purpose, the object managers within a single trust domain establish a shared session key, (k in Figure 3) which they use to sign the messages containing migration requests and their replies. Any object manager within a trust domain can be sure that a message came from within the trust domain if it is signed by the shared session key.

Refined Problem statement for migration between and within Trust Domains

Unfortunately, the site that makes a migration request does not know the identity of the current holder. Nor does it know whether it is within its own trust domain.

Therefore, the requester cannot decide whether to use public key encryption to sign the message. Before discussing our proposed solutions, we give further details on the granularity of access control and of a possible consistency protocol.

Granularity of access rights specification and of access control

In general, an application uses a large number of objects and it is not convenient for users to specify access rights for every single object. We have therefore suggested that access rights should be applied to categories of objects [CD 94]. For example, an object category might be a set of architects plans for a building, which might be represented by thousands of programming level objects.

In PerDiS, objects are grouped into application related clusters and each object category is represented by one or more clusters. Thus access rights are specified at the cluster level.

When we said above: 'each object has an ACL', we referred to an ACL that is generally shared by a collection of objects. The access rights are specified for collections of related objects. In practice, applications will tend to make many successive accesses to related objects from the same category.

Access control granularity

We are sometimes asked why we can't apply access control to an entire category, once and for all, e.g. in PerDiS when a cluster is opened. In spite of the notion of categories or clusters, in practice, each object may be accessed independently of he others in the same category. Therefore every single migration request and reply must be authenticated and access control applied to them.

Another question is whether the initial access to a category can be used to reduce the overhead of object protection on the later requests. The requested principal will be authenticated and the ACL will be fetched

from the home storage service at the first access to an object category. Therefore there will be no further need to fetch the ACL until its specified validity expires.

Another question is whether there would be any benefit from the home storage service providing the requesting principal's object manager with a capability to access objects in the category. For example, the capability might consist of:

> [object category, rights(R/W), principal, time limit] K_H^{-1}. signed by private key of home storage service

The capability would be passed with each migration request and reply and would require verification, but would remove the need to check the access rights. However, the requests and replies will still require authentication, so the use of capabilities does not appear to be a major way to save the overhead of object protection.

A coherency protocol

We describe here the details of a protocol used in PerDiS to provide entry consistency. The cooperating object managers use a single writer/many reader locking scheme. As applications attempt to access objects, they acquire the necessary locks via their object managers. Thus, an OM's migration request specifies the type of lock required (read or write). Sometimes an application may first read an object and subsequently attempt to write it. In such cases, the OM asks for a read lock with the migration request and subsequently requests that the read lock be promoted to a write lock.

In our protocol, all the requests for locks are sent to the *master* which is defined to be one of:

- the OM that is the current holder of the single write lock *or failing that*
- the OM that is the most recent holder of a single write lock (when there may be many readers) *or failing that*
- the home storage service.

If the lock cannot be granted immediately, the requesting OM must wait and is added to a FIFO queue at the master.

When a lock can be granted, either immediately or after a wait (and leaving the queue):

- for a *write lock* or lock promotion, the new holder becomes the master and the queue of waiting OMs is migrated to new master;
- for a *read lock* the requesting OM is added to a *read set* maintained by the master.

When an application releases a write lock, its OM is still the master and it grants the locks as described above, either immediately (if any other OM is waiting for the lock) or later when it receives a request.

Whenever a master with a non-empty read set receives a request for a write lock, it queues the request and sends a *callback* message to each of the OMs in its read set. The latter will record that they have received a callback.

When an application releases a read lock, its OM will send an *unlock* message to the master OM, either immediately (if it has already had a callback) or later when it does receive one.

We now consider the messages between the OMs involved in a migration request as shown in Figures 1-3, in light of the details of the coherency protocol. In the case of a migration request that asks for a write lock, there are just two messages as suggested above. However for a migration request that asks for a read lock, will result in one of the following patterns of communication:

- the OM requests a lock promotion which is granted and it becomes the new master;
- the OM holds the read lock until it no longer needs it and it has received a callback message, then the OM sends an unlock message.

The additional messages for promotion, callback and unlock all require authentication. Deadlock can sometimes occur, for example when a pair of OMs in the read set for a lock both request its promotion. We don't discuss here the authentication of messages related to deadlock.

When should the migration request be authenticated?

We have been asked why authentication and access control is not done before adding an OM to the queue of OMs waiting for a lock, so as to avoid delay in case of refusal. There are two reasons why not:

- the OM that eventually is able to grant the request for a lock must repeat the authentication and access control check if the request has come from outside its trust domain – we can't predict which OM will eventually rant the request;
- the OM that makes the request has the ACL and can check the rights before making the request. Therefore no lock request from a correct OM will be refused.

Solutions

In general, a group of object managers supporting applications that are sharing objects to support their users' cooperative task will communicate according to the following patterns:

- There will be a succession of migration requests from one OM to another for related objects in the same category. In PerDiS, access control is applied to clusters whilst migration is applied to pages. A cluster consists of several pages and a request to migrate one page in a cluster is often followed by subsequent requests for other pages in the same cluster.
- Objects and locks pass from one OM to another during the sharing, probably coming back to each OM from time to time; In the PerDiS case, a page, will typically be passed from one object manager to another as users take turns in acquiring it.
- messages concerning lock promotion, or callbacks and unlocks follow migration requests between each pair of OMs.

Any solution to the above authentication problem will take into account that the set of OMs whose applications are sharing the objects related to a task form a loosely coupled group in which there is generally a succession of messages passed between each pair of OMs.

Proposed Solution 1

Our initial idea was that every request must be signed using public key encryption and that the expense must be born. But another way was found! Each migration request is signed using the shared session key of the local trust domain. One of two alternatives can follow:

- the current holder is in the same trust domain and can deal with the request immediately;
- the current holder is outside the trust domain and acts as follows:
  - it uses public keys to establish a session key with the requester which is used to sign all the subsequent migration messages;
  - two messages supply the credentials of each of the two principals to the object manager in the other trust domain (e.g. in Figure 2, Q's credentials are supplied to Host A and P's credentials are supplied to Host E);
  - a third message supplies the object as a reply. (e.g. in Figure 2, O_i is sent to Host E.)

Once a session key has been established between a pair of object managers and the credentials of the principals behind requests have been exchanged, they may be re-used so long as the same pair of principals are involved and the session key is sufficiently fresh.

This method is currently under investigation in PerDis. The protocol includes the necessary nonces to prevent replay and an optimisation to allow an object manager that receives a subsequent requests from a same principal as before to re-use the established shared key and to avoid unnecessary exchange of credentials. The details of the protocol are given in our companion paper. [CDR 98]

Proposed Solution 2

The object managers whose principals are currently using an object become members of an *object interest group*. Each member of an object interest group knows the identity of the other current members and has a shared key for communicating with each of them.

For example, in Figure 1, suppose as before that object O_i came from the home storage service at D to Host E, then to Host G, then to Host A. At this stage the members of the object interest group for O_i are {D, E, G, A}. Each member has a key it shares with each of the other three members. For example, Host E has the following shared keys $\{k_{ED}, k_{EG}, k_{EA}\}$. The shared keys will as usual have a limited lifetime and will need to be refreshed.

Now suppose that as before, the object manager at E needs to make a migration request for O_i without knowing who the current holder is and whether it is in the same trust domain. In order to provide a signature that can be understood by any of the members of its current interest group, it digests the

message and signs it with each of the three shared keys and appends a vector of signatures to the message. [Rowley 97] showed that about 40 signatures with shared keys cost less than a single RSA signature. The reply message is signed only by the key shared between requester and current holder. In our example, this is k_{EA}.

An object interest group is created whenever the home storage service receives a migration request for an object that is not currently in use. The requesting object manager is supplied with a shared key for use with the home storage service.

Whenever an object manager asks the home storage service for an object that is already in use, and the object manager doesn't already belong to the interest group for that object, it will become a member. Whenever a member joins a group, it is given a key to share with each of the other members who are also notified about the shared keys.

This protocol will evict members who have not been active recently, so as to reduce the size of the vector of signatures. This protocol is about to undergo investigation as an MSc project.

Secrecy

If a shared object requires privacy, the migration reply messages between object managers are encrypted using shared keys. Since all of the users whose applications can share an object are allowed to view its state, we can allow their object managers to share a single key.

In either of our above proposed solutions, the home storage service for an object requiring privacy is responsible for issuing these shared keys to the object managers whose principals are allowed to view its state. In the second solution, the share key is associated with the object interest group.

Acknowledgements

Our work was done in the context of the PerDiS project and the problem identified here and the description of the coherency protocol resulted from several discussions with Olivier Fambon (INRIA Grenoble).

References

[PerDiS] TD 1.1.PerDiS Deliverable: Security Services Design. Available at the following url:: http://www.dcs.qmw.ac.uk/research/distrib/perdis/deliverables/TD1.1a.html

[Rowley 97] Secure Group Communication for Groupware Applications, Andrew Rowley and Jean Dollimore. ERSADs 1997.

[SSl] Netscape Corportation, 1996, Secure Sockets Layer (SSL) V3.0.

[Lampson] Trusted Computing base paper.

[CDR 98] Secure communication for distributed objects in non-uniform trust environments. Coulouris, G, Dollimore, J., Roberts, M. Submitted to ECOOP Workshop on Distributed Object Security.

[CD94] G. Coulouris and J. Dollimore, A Security Model for Cooperative Work Tech Report 674, Dept. of CS, Queen Mary and Westfield College, August 1994. ftp://ftp.dcs.qmw.ac.uk/pub/distrib/publications/TR674-Revised-Security-Model.ps.gz