Protection of shared objects for cooperative work

Coulouris, George; Dollimore, Jean

703

Protection of shared objects for cooperative work

George Coulouris and Jean Dollimore

Technical Report 703

Department of Computer Science
Queen Mary and Westfield College
University of London
Mile End Road London E1 4NS

Email: {George.Coulouris, Jean.Dollimore}@dcs.qmw.ac.uk
WWW: http://www.dcs.qmw.ac.uk/research/distrib/

Keywords: Cooperative work, group task, distributed objects, shared persistent objects, security, protection, access control.

Abstract

This paper discusses the design of a system for the protection of shared persistent information objects that are intended to provide a basis for building cooperative applications. A task-based model of cooperative work is adopted and user and task requirements are based on an earlier case study undertaken by the authors.

The key problem addressed is the mapping of user-level protection specifications onto groups of programming-level objects. A design for a two-level protection model to address this problem is outlined.

 September 5, 1995

1. Introduction

This paper discusses the requirements for the protection of information in cooperative work and describes an approach to the design of a protection system. The design is intended for use in an otherwise open software environment in which cooperative tasks are carried out using cooperative applications accessing shared long-lived information objects in a distributed environment. Application software is assumed to be object-oriented, but it carries no responsibility for the protection of the information objects that it accesses.

The protection schemes found in current operating systems such as Unix are often too limited to support cooperative work effectively. Most existing protection schemes are designed for use with single-user applications in which users carry out individual tasks making use of private objects which require protection against misuse by other users. In contrast, the users involved in a cooperative task require the ability to share objects in a more cooperative manner, providing one another with more access rights than would be desirable for individual tasks.

The design proposals in this paper are based on an earlier study of the requirements of security-critical applications [Coulouris and Dollimore 1994]. Our study suggests that it is useful to define the security requirements for the separate tasks in an organisation in a generic manner in terms of the roles of the users that participate in these tasks. The use of symbolic (unbound) names for roles enables tasks to be defined before it is known which users will assume the roles.

1.1. Aims

We are concerned with cooperative tasks that are carried out by people working in organisations and supported by a distributed computing system that enables persistent objects to be shared.

Our aims are strongly influenced by the fact that cooperative work is carried out in the context of organisations, which suggests that the way shared objects are protected should be determined by the security policies of the organisation. In addition, the users in the organisation must be provided with a convenient means for specifying how the security policies should be applied in the various tasks that arise.

Examples of cooperative tasks:

- Writing a joint report in which each participant is responsible for a different part. For example in a financial report, one participant might supply the financial data in the form of charts and graphs, while another would make future projections.
- Maintaining and using medical records in situations in which different fields are available for viewing or updating by the different people involved. For example, patients can see only their own records, receptionists can view and update the appointment times whereas doctors can view and update all the medical details.
- Preparing an examination paper; collecting and collating marks from several examinations; preparing a directory of courses for a university.
- Computer aided design of products whose new features are to be kept secret from competitors.

Limited trust

'It is a vice to trust all and equally a vice to trust none'. Seneca the younger (4BC – 65AD).

It should be possible to specify different rights for the various roles involved in a cooperative task. This is necessary to support cooperative work in which the participants are not equally trusted.

In addition, it should, where appropriate be possible to prevent users other than those playing the roles in a task from having access to the objects used by a task. In some cases it may be appropriate to provide limited access to the objects used in a task to users other than those with roles in that task.

1.2. Protection should support an organisation's security policy

Since cooperative work generally takes place in an organisation, the security policy of that organisation should be taken into account in defining the appropriate protection for the objects required to carry out a cooperative task. Security policies are often expressed at a high level; for example: 'no student can read an examination paper prior to the examination'.

Organisations have security policies that define the degree of security required for the information that they maintain and use. The policies state requirements for *secrecy* and *integrity*.

A security policy for a task with respect to secrecy may require that:

- no users other than those involved in the task should see the state of a particular object, or
- no users other than those involved in the task should be aware of the existence of a particular object.

A security policy with respect to integrity may require that:

- particular objects are created and altered only by the relevant roles and their delegates
- that a particular object must be signed (finalised) by some set of the roles.

Organisational security policies can include both generic and task-specific policies. Examples of generic policies include:

- a 'need-to-know' security policy: this allows members of an organisation to access just the information that they need in order to perform their jobs – in most organisations such a stringent policy would not be applied generically, but selectively in some security-critical tasks;
- a 'one-level delegation' policy: states that work may be delegated, but the delegates cannot further delegate it;
- a 'preselected delegates' policy: requires that delegates should be chosen from a specified group.

For an example of task-specific policies, we refer to the task of preparing an examination paper, in which two of the rules might be: 'The paper and comments on it must be created and altered only by the relevant examiners and their delegates' and 'The finalised exam paper must be designated as such by the relevant examiners and transmitted to the examination without subsequent alteration'.

A security policy for a task specifies the secrecy and integrity requirements for each of the information objects required to carry out the task. A policy can be stated in terms of the roles

in the organisation who will carry out the task and their rights to access the information. The rights specify which parts of the information they are allowed to create, to see or to update in various ways.

1.3. Protection should be specified in terms of generic operations

Users can only specify a security policy in terms of operations that they can recognise.

The detailed structure of the shared objects used in groupware applications and their interfaces (the sets of operations that they support) are not always known to users. For example, each paragraph of a document may have its own operations such as *display*, *change style*, *replace word*. However users of a shared editing application will generally have a different, larger-grained perception of the available operations. Our aim is to design a protection model in which users need not be aware of the operations on the individual shared objects used in cooperative work, but in which each object can be protected individually.

The security policies for a task may need to be described before the applications for carrying out the task have been chosen. Besides, these policies may be described by people who have little knowledge of the details of the operations in the available applications.

We therefore suggest that the use of 'generic operations' would enable security requirements to be specified at a level that is independent of the applications used. In addition, the use of a common set of generic operations to specify security requirements for a variety of tasks should be a simplifying factor and a basis for a common approach.

Examples of generic operations for a wide variety of applications based on editing activities might be *Edit*, *Format* and *Read*. A different set of generic operations might be required for other types of applications (for example a graphical application might require operations such as *Scale* or *Rotate*).

We shall address the distinction between the generic operations and the specific operations of objects by adopting a two-level model for protection. Each groupware application must include a specification of the mapping between a set of generic operations of which the users are aware and the actual operations in the interfaces of individual objects. Users specify the protection required for a task in terms of the generic operations. These are subsequently translated to access rights for actual operations on the relevant objects.

1.4. Outline of the paper

In Section 2 we define our assumptions about the system environment that provides protected objects. Section 3 gives some definitions and outlines related work on security models for cooperative work.

Section 4 outlines the security requirements we identified in our case studies and explains how tasks, roles and delegation were derived from them. In Section 5 we describe how object protection may be specified in a security template in terms of generic operations. In Section 6 we describe a two-level model for security and discuss the implementation of the two-level model in a client-server architecture. Section 7 draws conclusions.

2. Assumptions about the system environment

Security policies are defined in terms of *principals**, *objects* and their *operations*. They are designed to answer the question: 'Should a principal be allowed to perform a particular operation on a given object?' We adopt the access control model of security [Lampson 1991], in which each object has a *guard* that decides whether to grant a request to perform an operation.

2.1. A programming model that supports protected objects

This work assumes that groupware applications are implemented in a software environment in which shared persistent objects encapsulate their state and provide interfaces through which their operations may be executed. Objects protect themselves according to a predefined set of *access rights* that define the operations on the object that should be permitted to each principal.

Protected objects can be implemented in client-server distributed systems by applying access control to the operations in the interface of a service. This approach can be extended to apply to services that maintain multiple objects with their own interfaces.

Protected objects can also be provided within systems designed to support access to shared and persistent distributed objects in which all objects are potentially sharable. Sharing in systems of this type can be achieved either by remote invocation (Arjuna [Shrivastata et al. 1991], SOS [Shapiro et al. 1989] and Distributed Smalltalk[Bennett 1990]) or by means of distributed shared memory (Guide [Balter et al. 1994], Opal [Chase et al. 1992]). In the case of remote invocation, access control may be applied by the operation dispatcher. We have discussed the implementation of our protection model in a distributed shared memory model elsewhere [Coulouris and Dollimore 1994a]. In the latter case, the memory mapping mechanism can be used to apply access control at the level of read and write operations.

In principle, a shared persistent object environment based on groups of object replicas should also be able to support protected objects. We are currently investigating the protection of object replicas [Achmatowicz and Kindberg, 1995].

Our model of protection for cooperative work is applicable to all the above styles of implementation for shared objects.

2.2. Applications are built from protected objects

Cooperative tasks may be carried out by users employing a variety of applications such as editors, spreadsheets, databases and drawing packages. We assume that all of these applications make use of shared protected objects.

Thus, the applications are not responsible for the security of the objects that they manipulate. Instead the protection is applied at the level of the protected objects from which applications are constructed. This ensures that applications cannot disobey the protection defined by the access rights of protected objects and enables different applications to share objects without endangering their protection.

This approach is similar to the situation in a UNIX environment in which the only shared persistent objects are files. Files are protected objects whose operations can be invoked only

---

* We follow the common practice in computer security work of using the term *principal* to refer to an active agent that performs operations on information objects. A principal may be a person or program (such as a service or an application). Principals may also be defined indirectly, for example as a member of a group, or as a delegate of another principal (see Section 5).

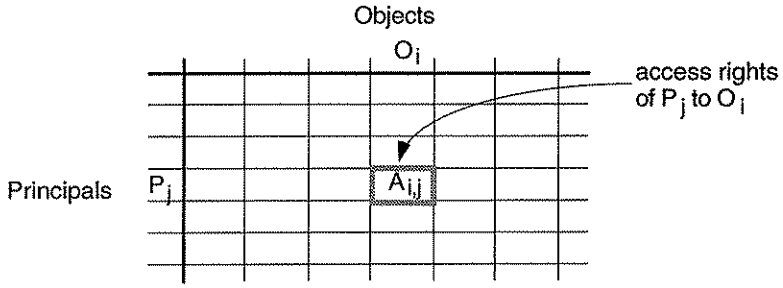

Figure 1. The form of an access matrix.

if they are permitted by the access rights. All applications must be constructed from files and applications that share files are subjected to the same access rights.

3. Definitions and related work

In this section we explain the *access matrix* representation that is often used to define protection models in computer systems and we review the existing models for security in cooperative work. We shall later introduce a variant of the access matrix, called a *security template*, as a basis for the specification of security policies in cooperative work.

3.1. Access matrices

The access matrix was originally introduced as an abstract model for the description of protection in operating systems. An access matrix contains the rights of principals to access protected objects (Figure 1). Each column is labelled with an object and each row with a principal. A cell in the matrix contains the rights of the principal associated with the row to access the object associated with the column. The rights specify which of the object's operations the principal is permitted to perform.

The access matrix is generally too large and too sparse for use as a concrete representation of access rights. Two alternative data structures have been used to represent the information that it contains: access controls lists (ACLs) and capabilities. The ACL of an object specifies the principals that may access it and the operations allowed to each one. A capability specifies an object that a principal may access and a set of permitted operations.

3.2. Protection models for cooperative work

Greif and Sarin [1986] discuss the requirements for access control amongst other issues of sharing data in group work. They were perhaps the earliest to suggest that group working requires a more sophisticated means for defining principals and permitted operations than that provided by the UNIX model. In particular, they propose that:

i) access rights should be associated with a user's role when performing an operation, and

ii) access rights should be expressed in terms of the operations in the interface of an object instead of just the file operations read and write.

They also suggest that access rights may need to be related to (a) the contents of an object (e.g. entering in a diary a meeting that conflicts with existing meetings) or (b) the relationship between the user or role requesting an operation and the properties of an object whose values are user or role identifiers (e.g. who created it). A requirement similar to (b) also arises in

[Dollimore and Wang 1993]. For example, only the user who created an entry in a diary for a meeting should be allowed to alter or cancel it.

Objects typically have many operations, so the variety of possible rights can be huge. To manage complexity and improve performance, in Guide [Hagimont 1994, Balter et al. 1994] the operations available on an object are grouped together into 'views' to which principals may gain or be denied access. Views are similar to the views used in databases to grant users the right to access selected information. Views are defined statically as part of the class of an object. This does not allow views to be defined as required during use in a particular application as suggested in [Dollimore and Wang 1993].

In a review of issues in groupware, Ellis, Gibbs and Rein [1991] state that access control must take into account that the likelihood of conflict between users for the use of shared objects is higher than in individual work. An important factor is that access rights are not static: they must be able to be granted and revoked. It should be easy for users to specify changes and to negotiate with one another for rights. The ability to negotiate for rights to use parts of a document is supported in the Griffon shared structured editor [Decouchant et al. 1993].

In cooperative work it may be convenient to specify access rights for groups of users instead of individuals. A problem that generally arises is that the users eventually define a very large number of different groups. The Andrew File System [Satyanaraynan 1989] which is designed for large numbers of users rather than cooperative working attempts to address this issue by allowing group membership to be transitive.

Shen and Dewan [1992] have proposed an access control model for collaboration in which users can assume multiple roles, in the context of editing shared objects. To simplify the management of organisational roles, they are arranged in a hierarchy through which the rights of roles are inherited. They note that there can be many different sets of rights in shared editing, for example if one user formats some text, which other users can view the result? Their model uses inheritance to simplify the specification of access rights. A fixed set of editing operations is arranged as a hierarchy of views. In addition, negative rights are provided to simplify the definition of policies.

Note that negative rights have been used elsewhere, for example in the Andrew File System [Satyanaraynan 1989] and in BirLix [Kowalski and Hartig 1990]. The designers of the latter systems state that their motivation for including negative rights is to provide the ability to revoke access to sensitive objects rapidly and selectively. In the absence of negative rights, revocation can be done by removing a principal from all of the relevant ACLs and user groups but this process may be time-consuming in cases where the principal belongs several groups.

4. Case studies

In our case studies [Coulouris and Dollimore 1994], we analyse the security requirements of two cooperative tasks, both of which come from a university setting. The first is a study of the preparation of an examination paper, in which the security of the shared information is highly critical. The second study concerns the preparation of the college course directory with cooperation between the academic registrar and the lecturers in the departments.
In each of these studies we used an approach in which:

i) we identified the roles of the users and the information objects required to carry out the task;

ii) we identified the security threats and specified the security policy with respect to the secrecy (who is allowed to see or be aware of the existence of certain information) and integrity (who is allowed to create, alter or finalise certain information);

iii) specified the security requirements for each task in the form of an access matrix.

An example of roles and objects

The roles for the examination task include a member of the Exam Board and its Chair (*Board* and *Chair*), the first examiner (*Ex1*) the second examiner and the external examiner. The objects include a series of versions of the paper created by *Ex1* and the comments on them created by *Chair* and the other examiners. The examiners need to be able to read the versions of the paper and the comments in order to produce the paper. The members of the board also need to be able to read them to maintain consistency with other papers. The paper is finalised by the *Chair* and the other examiners.

Security policies

In the examination case study the security policy states that no student must see any version of the examination paper before the date of the examination, that the paper must be finalise by Ex1 and that delegates must be taken from a set of clerical staff selected in advance by the Chair. In addition, delegated tasks cannot be further delegated.

An access matrix to specify security requirements

We specified the security requirements in the case studies with respect to integrity and secrecy for a task in the form of an access matrix, which shows the rights of each of the roles to perform operations (e.g. create, edit, read or finalise) on an object.

The access matrix is defined in terms of generic roles, (rather than names of users). When the same security requirements apply to a series of similar tasks (e.g. the other examination papers) the same access matrix can be used for each task, but with different users bound to each of the roles. For example there were five rows for the five roles in the access matrix for the examination task.

Objects with similar rights are specified in the same column of the access matrix, for example in the examination task all the comments created by the various examiners require the same rights, irrespective of who created them. The access matrix for the examination task used three columns - for initial and later versions of the paper and for the comments.

4.1. Requirements derived from the case studies

In the case studies we identified the following clear requirements:

Roles: As the tasks involved in administrative work are generic, their security requirements should be defined without using the names of specific principals (e.g. users). This can be achieved by stating security requirements in terms of the rights of roles to access the objects required for a task. An access matrix defined in terms of roles can be used for several instances of the same task.

Rights are role dependent: Each of the roles involved in a task may require quite different access rights.

Delegation of rights: In both of the case studies, delegation of work to clerical staff occurs on a substantial scale and in both cases it is important that the rights conferred on delegates

should be restricted to the task at hand. The first case study suggests that a delegate should not be allowed to delegate the rights further.

Objects should be created by particular roles: It should be ensured that each of the objects used in a task is created by a role with the right to do so.

We also identified other requirements likely to arise in some tasks, which included the following:

Concealment: Some objects should be concealed from principals other than those playing roles in the relevant tasks. The actual existence of an object can sometimes convey information to unauthorised people, even though they cannot see its contents.

Organisational roles versus task roles: Organisational roles are often assumed by the same users in several different tasks. But a principal's rights within a task do not include the rights held by the same role when performing other tasks.

Dynamic rights: The specification of access rights changes with time and in response to events.

Finalising an object: Finalising prevents further modification, even by users who are allowed to update the object, and attaches the authority of the role to the 'finalised' object. This can be done by attaching a digital signature to the object.

Participation of objects in multiple tasks: Objects are shared (rather than copied) between tasks in order to ensure that up-to-date versions are available in all of the tasks sharing them.

5. Object protection in cooperative tasks

In this section we discuss a view of principals that we believe is appropriate for security in cooperative work and then describe how a security template can be used to specify the requirements for object protection in terms of generic operations.

We assume that the principals accessing the protected objects are acting within a task structure. Therefore the access rights of objects are defined according to the needs of the roles within a task. In this context, a principal without a task has no access rights.

In addition, a principal with a role in a task can delegate its access rights to another principal. Our discussion of principals is derived from, and intended to be compatible with the work on authentication and compound principals described in [Lampson et al. 1992 and Wobber *et al*. 1994]. In that work, the notion of *compound principal* is fully formalised, and the authority of principals is encapsulated in the form of a set of unforgeable credentials. In this paper we shall use some of Lampson *et al*'s notions, but we present our usage of them in an informal manner.

5.1. Principals

A program acts with the access rights of the principal on whose behalf it is running. In our model for cooperative work, access rights are associated with roles in tasks. This section first discusses roles in tasks as principals and then discusses how delegation fits into this framework.

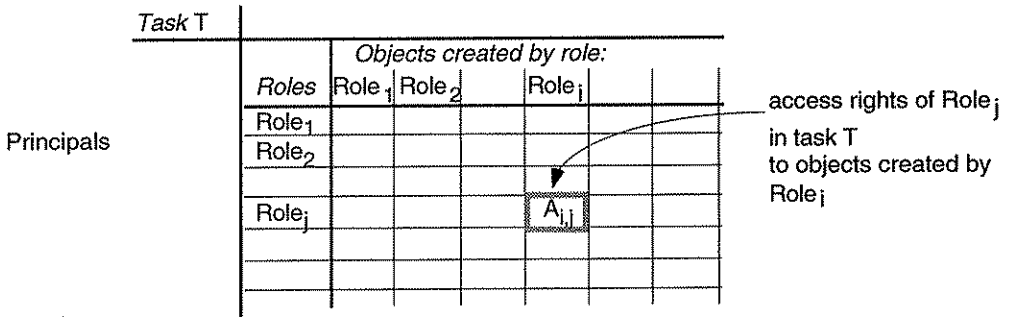

Figure 2. Security template specifying access rights of roles on objects

Playing a role in a task

Access rights for the objects used in a task are defined in terms of the roles involved in that task. Users can participate in cooperative tasks by adopting the roles for which they are authorised. Several users may be authorised to adopt the same role (for example, 'member of the examining board'). The access rights of a user playing a particular role in a particular task do not include any additional rights available in other roles they can adopt, for example in other tasks.

We refer to a principal whose access rights are just those of a user playing a role in a task as: 'role *R* in task *T*'. We abbreviate: 'role *R* in task *T*' as *R* **in** *T*. This principal has the properties:

- it corresponds to a group of one or more authorised principals (e.g. users) and
- it can be adopted by a user, *U* provided that *U* is a member of the group of authorised principals.

Delegation

Note that we relate access rights to tasks and that we intend the rights conferred by delegation to be restricted to a work undertaken within a specific task. Any principal *U* that is allowed to adopt '*R* **in** *T*' can transfer the corresponding access rights to another principal. The mechanism for doing so may include a check that the recipient comes from a specified set of principals (for a 'preselected delegate' policy). We suggest that as in [Lampson et al. 1992], delegation requires the participation of both parties - the delegator delegates and the delegate accepts the delegation by quoting the delegator in all its requests.

5.2. Specification of protection by users

In this section we describe the *security template* for a family of tasks of the same type, for example the security template for the examination task can be used for the task of preparing each of the papers. A security template is similar to an access matrix in that it specifies the access rights of each of the roles in a task to perform operations on objects in a task, but access rights are specified for generic operations and the objects are grouped, initially according to the role that created them.

Figure 2 illustrates the general form of a security template for a task. The rows correspond to roles and the columns correspond to the sets of objects created by each of the roles that may create objects. Elements of the security template specify the rights of the roles to access the objects in each of the sets in terms of generic operations.

The grouping of objects is necessary because there are likely to be many shared objects in an object-oriented groupware application. In general, we must retain the ability to control access to individual shared objects (just as Unix provides control over access to individual files), but in order to reduce the complexity of the security template, we wish to group objects with similar access rights. It would be convenient if the same access rights could be applied to all of the shared objects used in a cooperative task, but our case studies show that there are cooperative tasks for which this is not so.

We have considered several approaches to grouping objects:

According to the class of the underlying protected objects: We rejected this idea for two reasons: (i) because each object mentioned in the security template can be implemented in terms of several objects with different classes and (ii) objects with different access rights such as 'comments' and 'versions' in the exam case study might be of the same class;

Associate objects explicitly with named categories: (e.g. 'comments' and 'versions'). We felt that this approach would be inconvenient to implement because there is no straightforward way of specifying the category of an object at its time of creation;

According to the identity of the role that created the object: This is our chosen alternative, illustrated in Figure 2. It is attractive because it enables us to specify groups of objects without introducing any new concepts. It is convenient because the role is known when an object is created. Also it follows the suggestion in [Greif and Sarin 1986] that the rights of roles to access objects may be related to the roles that created the objects.

Another requirement from the case studies is that particular objects should be created by particular roles. In the representation of Figure 2, only those roles with the right to create objects will appear in the columns of the security template. This avoids the need to specify the rights of a role to *Create* an object in a cell of the security template.

The following additional aspects of security requirements for cooperative tasks can also be specified in the security template:

Finalising objects: The roles that are allowed to 'finalise' an object used in a task can be indicated by *Finalise* operations in the appropriate cell of the a security template.

Controlling delegation: In many tasks, recursive delegation is undesirable. The permitted number of levels of delegation can be specified in the security template either for each group of objects or for all objects. Levels can be encoded as numbers - e.g. as zero or more, where zero means no delegation and one means no recursive delegation.

6. A two-level model

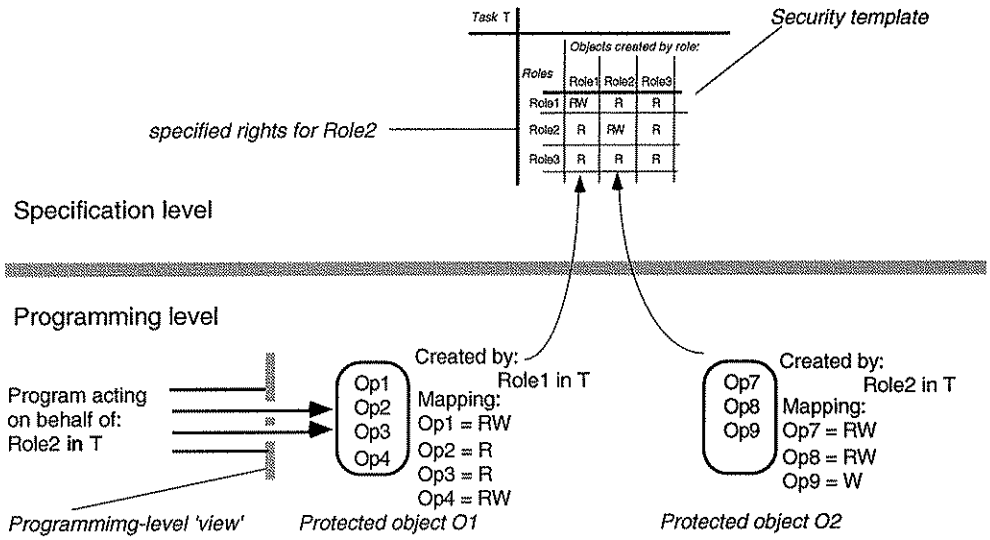

Figure 3. The two levels of the model.

The differing needs of the users and the system environment lead us to propose the two-level model shown in Figure 3:

At the specification level: users specify access rights on the objects created by each role in terms of a set of generic operations in a security template for each task.

At the programming level: access control is applied to the operations of the objects in the execution environment. Access rights for the programming-level objects are derived from the security templates defined at the higher level.

The major issue to be addressed in such a model is how access control specifications in terms of generic operations can be translated to access control lists for objects with programming-level operations.

For example, in Figure 3 a program acts on behalf of a user that has adopted *Role2* in task *T*. The specification level shows a security template for task *T* with the generic operations *Read* (R) and *Write*(W). It shows the generic operations permitted to each of the roles on the groups of objects created by *Role1*, *Role2* and *Role3*. For example, within this task, *Role2* may perform the generic operation *Read* but not *Write* on objects created by *Role1*.

At the programming level, each object is accompanied by the following information:

- the *R* **in** *T* that created it;
- a mapping from the operations of the object onto the generic operations {*Read* and *Edit*}. This mapping lists for each operation of the object the corresponding generic operations.

In Figure 3, the arrows from the programming level to the specification level indicate the connection between the role that created an object and a column in the security template for the task. Thus programming-level object *O1* (which was created by *Role1*) allows *Role2* **in** *T* to execute the operations *Op2* and *Op3* but which correspond to the generic *Read* operation.

6.2. Protection of objects in a client-server architecture

In this section we illustrate the two-level model by outlining how a server of persistent objects could implement protection according to the model.

Each request by a client to perform an operation in the interface of an object managed by a server contains the credentials for a principal such as a user *U* as '*R* **in** *T*'. The server must verify that *U* is allowed to play the role *R* **in** *T*. It does so either by checking with a trusted task manager (e.g. the Task Service outlined below) or by validating credentials supplied by the client to this effect.

Thus when a client creates an object in a persistent storage server, the server records the *R* in *T* submitted with the creation request as the identity of the principal that created the object.

Task service

A Task Service manages the secure storage and retrieval of the information related to the various tasks currently taking place in an organisation.

The Task Service maintains a security template for each type of task carried out in the organisation. When a new object is created at an object server by a principal *R* **in** *T*, the server uses the type of the task *T* in requesting the access rights for that task from the Task Service. It may either get the entire security template or just the column that specifies the rights for objects created by role *R*.

The column of the security template is effectively an ACL for the newly created object. The server may attach it directly to the new object or it may use an indirect connection enabling ACLs to be shared between objects. Note that with this arrangement, any changes to the security template will not affect the ACLs of the objects already created.

Service interface

The operations of the objects managed by a service are defined in the service interface, generally written in an interface definition language.

Each operation in the service interface may be annotated to show which of the generic operations it corresponds to.

Example

Suppose that the generic operations are Read (R), Write (W) and Format (F). Then an interface definition for two of the types of objects used in the examination task could be annotated as follows:

```
Interface: ExamPaper {
operations:
   ReadPaper    (R)
   AddQuestion  (R)
   EditRubric   (RW)
}
Interface: Question {
operations:
   ReadQuestion       (R)
   WriteQuestion      (W)
   FormatQuestion     (F)
}
```

When the server receives a client request to perform an operation in the interface of one of its objects it applies the rights of the principal '*R* **in** *T*' that made the request in deciding

whether to perform the operation. To do this it looks for '*R* in T ' in the ACL of the object and retrieves the generic operations that are permitted to that role. It then checks the permitted generic operations against the annotation of the operation requested.

The service interface is a good place to put the mapping between generic operations and operations of the service because clients and server share the information. This allows user interfaces to show on menus which operations are permitted (e.g. by greying out the others). Also the client can cooperate in carrying out the security policies, thus avoiding unnecessary contact with the server.

7. Discussion

In the first part of the paper we identified some problems that arise in the design of a protection scheme for persistent information objects when they are shared in cooperative work. The problems were derived from a case study of two cooperative tasks (reported in detail elsewhere) and some assumptions about a suitable software environment for the construction and use of cooperative applications ('groupware').

Two key design problems emerged:

Multiplicity of objects: There are likely to be many more shared objects in an object-oriented groupware application than there are in a similar Unix file-oriented application. In general, we must retain the ability to control access to individual shared objects (just as Unix provides control over access to individual files). But to avoid generating large amounts of clerical work, the access rights for most objects should be automatically generated.

Diversity of operations: Each class of objects used in a groupware application is equipped with its own set of programmable operations. The detailed nature of the shared program-level objects and the sets of operations that they support are generally not known to users. But our aim was to design a protection model in which each object can be protected (and therefore shared) individually.

The use of task-related security templates described in Section 5 and the two-level protection model described in Section 6 are our proposals for the resolution of these design problems.

The security template is designed to enable users to specify their security requirements for tasks. The grouping of objects in the security template according to their creator addresses the problem of object multiplicity. It is similar in some ways to the treatment of 'owners' in the Unix model of file protection, but it should be remembered that we are dealing with protection for specific tasks – if an object is shared between several tasks, its protection in the second and subsequent tasks in which it was made available might be specified in terms of the 'importing role'.

The use of generic operations instead of object operations addresses the problem of operation diversity, but at the cost of requiring an additional specification for object classes – the specification of the mapping between generic operations and the actual programming-level operations of object classes.

We have sketched a possible implementation in a client-server architecture. The model has yet to be tested and evaluated in practice. Several issues of detail remain to be resolved, including the following:

- Our case studies indicate that as tasks progress through their phases there may be a need to change the access rights at the end of each phase. The changing of phases might be managed by a workflow model.
- Other changes to security templates may be needed on a more *ad hoc* basis, due either to a change in policy or to the need to make a task easier to carry out. These changes might or might not be reflected in existing tasks.

Acknowledgements

We should like to thank our colleagues Tim Kindberg, Richard Achmatowicz and Andrew Rowley at QMW for discussions on the role and task model and the security of object replicas. Thanks are also due Victoria Bellotti and William Newman (Rank Xerox Cambridge Research Centre) for their comments on user perceptions of protection and security and to Sacha Krakowiak, Jacques Mossière, Xavier Rousset de Pina, Michel Riveill and other members of the Bull-IMAG Systèmes Laboratory in Grenoble for witnessing the birth-pangs and assisting with the delivery of these ideas. Responsiblity for the results, however, is entirely ours.

8. Bibliography

Achmatowicz, R. and Kindberg, T. 1995, Object Group sfor Groupware Applications: Application Requirements and Design Issues, ERSADS Workshop, Val d'Isere, April 1995.

Balter, R, Lacourte, S. and Riveill, M. 1994, The Guide Language, The Computer Journal, Vol. 37, No. 6, 1994.

Bennett, J.K. 1990, Experience with Distributed Smalltalk, Software - Practice and Experience, Vol 20(2), 157-180. Feb. 1990.

Chase, J.S., Levy, H.M., Baker-Harvey, M. and Lazowska, E.D. 1992, How to Use a 64-bit Virtual Address Space, Technical Report 92-03-02, University of Washington, Seattle 98195, March 1992.

Coulouris, G. and Dollimore, J. 1994, Requirements for security in cooperative work: two case studies, Technical Report 671, Department of Computer Science, Queen Mary and Westfield College, May 1994.

Coulouris, G. and J. Dollimore 1994a, Security requirements for cooperative work: a model and its system implications, Position paper for 6th ACM SIGOPS European Workshop, Dagstuhl, September 1994.

Decouchant, D. Quint, V., Riveill, M. and Vatton, I. 1993, I. Griffon: A Cooperative, Structured, Distributed Document Editor. Rapport Technique 20-93, Bull-IMAG Systèmes, Gieres, France, June 1993.

Dollimore, J. and Wang Xu 1993, The Private Acccess Channel: a Security Mechanism for Shared Distributed Objects. In TOOLS Europe 93. March 1993. Pages 211-222.

Ellis, C.A., Gibbs, S.J. and Rein, G.L. 1991, Groupware - Some Issues and Experiences, *Communications of the ACM*, 34, 1, January 1991. Pages 39-58.

Greif, I. and Sarin, S. 1986, Data Sharing in Group Work, In Proceedings of First Conference on Computer Supported Cooperative Work, (Austin Texas). ACM New York, December 1986. Pages 175-183.

Hagimont, D. 1994, Protection in the Guide object-oriented distributed system, ECOOP 94.

Kowalski, O. and H. Hartig 1990, Protection in the BirLix Operating System, PODCS-90. Pages 160-66.

Lampson, B.W. 1991, Requirements and Technology for Computer Security, Chapters 2 and 3 of *Computers at Risk: Safe Computing in the Information Age*, National Academy Press, Washington, DC, 1991.

Lampson, B.W., Abadi, M., Burrows, M. and Wobber, E. 1992, Authentication in Distributed Systems: Theory and Practice. *ACM Trans. on Computer Systems*, 10, 4, Nov. 1992. Pages 265-310.

Satyanarayanan, M. 1989, Security in a Large Distributed System, *ACM Trans. on Computer Systems*, 7, 3, August 1989. Pages 247-280.

Shapiro M., Gourhant, Y. and Habert, S. at al. 1989, SOS: an object-oriented operating system - assessment and perspective. *Computer Systems*, 2, pages 287-338.

Shen, H.H. and Dewan, P. 1992, Access Control for Collaborative Environments, Proceeding of ACM CSCW '92. Pages 51-58.

Shrivastava, S., Dixon, G.N. and Parrington, G.D. 1991, An Overview of the Arjuna Distributed programming System. IEEE Software, Jan. 1991, pp. 66-73.

Wobber, E., Abadi, M., Burrows, M. and Lampson, B.W. 1994, Authentication in the TAOS operating system, *ACM Trans. on Computer Systems*, 12, 1, Feb. 1994. Pages 3-32.