The use of adapters to support cooperative sharing

Jonathan Trevor, Tom Rodden and John Mariani

Centre for Research in CSCW

Research report: CSCW/7/1994
The use of adapters to support cooperative sharing

Jonathan Trevor, Tom Rodden, John Mariani
Computing Department
Lancaster University
Lancaster LA1 4YR.
UK.
E-mail: {jam, tam, jonathan}@comp.lancs.ac.uk

ABSTRACT
This paper examines the importance of providing effective management of sharing in cooperative systems and argues for a specialised service to support the cooperative aspects of information sharing. The relationship between features of the cooperative shared object service and existing services is briefly examined. A number of management services of particular importance to CSCW systems are identified. The paper presents a technique of realising a shared object service by augmenting existing object facilities to provide management of their cooperative use. These facilities are realised through object adapters that provide additional cooperative facilities and greater control over the supporting infrastructure.

KEYWORDS: Information Sharing, Distributed Systems Support, Cooperative Systems Infrastructure.

INTRODUCTION
Shared information plays a central role in CSCW systems. It is often the primary means used to develop a shared understanding across a number of users working together. The nature of this sharing and the forms of cooperation it facilitates vary greatly from application to application. For example, systems such as Ensemble [18] allow a number of potentially distributed users to work together in real time while annotation systems such as QUILT [10] allow a group of users to work in a time-independent manner.

Information sharing is strongly dependant on the facilities provided by the supporting distributed infrastructure. However, a mismatch often exists between the manner in which facilities are provided and the diverse nature of cooperative applications. This is most visible in the balance of control between users, applications and the management of the facilities provided by the supporting platform. Distributed systems have traditionally employed a prescriptive approach to management and control. Successful control has focused on hiding the problems associated with distributed systems from users. For example, both the LOCUS [25] and ANSA [1] project identify a number of distinct areas of control, and associated with each a transparency that allows the feature to be hidden as necessary.

Unfortunately, the combination of prescriptive control based on transparency and the desire to create a single user view of the system can hide the existence of users from each other, actively prohibiting cooperation [21]. This conflicts with the inherent nature of CSCW systems which requires users to work together often using common tools and information. In contrast to the existing system view of control, CSCW systems and environments need a user oriented view of control. This control should seek to promote direct involvement of users in managing the cooperative aspects of the infrastructure.

The aim of this paper is to examine the management and sharing requirements placed on underlying platforms by CSCW applications, and to report new mechanisms developed to support the sharing required from these infrastructures. The work reported here is the result of the development of a set of services to manage information sharing across cooperative systems. Many of the facilities provided in the service have been informed from a social consideration of the use of documents [6]. Our particular focus within this paper is on the manner in which we can augment existing distributed systems with cooperative facilities. These facilities are provided in a manner that promotes user involvement in the management of sharing.

A SHARED OBJECT SERVICE
The intent of a shared object service is to provide the necessary platform for sharing in cooperative systems. Thus, rather than consider the problems of CSCW as being essentially a user interface issue, we are examining the more fundamental and generic information storage implications that emerge from a consideration of cooperative work. To do this we need to make a clear separation between the facilities required for shared objects, and those which need to be provided to support ‘real-time’ shared interface facilities. This separation is realised by the provision of a number of closely related services that manage a particular set of facilities. Two services of particular note are the shared interface service and the associated shared object service. The shared interface service makes considerable use of the facilities provided by the
shared object service. The relationship between these services and the different user applications that make use of them is shown in figure 1. The diagram is intended only to illustrate the functional relationships within the service rather than to suggest any particular architectural structure or separation for the service.

The explicit aim of the shared object service is to provide a set of services that allow objects to be cooperatively shared by a community of users. Previous services such as those provided in the COSMOS project [2] have focused on the provision of multi-user storage facilities. The distinguishing feature from existing multi-user storage services is the removal of prescriptive control and the provision of mechanisms which support the management of cooperative sharing. The shared object service provides a set of facilities for a group of users that abstracts from the properties of underlying infrastructure to provide a well-defined abstract set of cooperative services.

![Diagram of shared object service components]

Figure 1: Role of the shared object service

Our particularly concern in this paper is the shared object service and the means by which services are realised within the shared object service. The separation between the shared interface service and the shared object service allows us to highlight the close relationship between a shared user interface and the form of support provided for cooperative work. For example, one form of shared interface service could be workstation based visualisation facilities such as those provided by the Rendezvous [20] and MEAD [4] systems. Other alternatives include the shared hypertext facilities provided by the Sepia [14] and the Knowledge Net [17], or the spatial representation and interaction provided by a VR presentation [3]. The point is not that any of these is particularly appropriate but that all can make use of some form of shared object service. The provision of a service thus provides us with a platform from which to consider supporting cooperation through shared objects and integrating a number of approaches and applications to cooperation in a relatively seamless manner.

Shared Object Service Requirements

Many of the facilities provided by the shared object service are motivated by a number of previous studies of work [6]. These combine with a number of technical limitations to outline a set of general requirement for shared objects in the cooperative object service. The most notable of these include:-

1) Lightweight semantics through a policy and mechanism separation

The dynamic nature of cooperative work means that the manner in which shared information is accessed and used at any given point can change quite dramatically. Given the variable nature of this context of use it is important that the information content of objects is preserved independently of their use. To ensure this, an object should not contain any semantics regarding the cooperation setting in which it is used. It is thus essential to allow policy to be imposed separately from basic objects which, while functionally complete, may require additional facilities within a cooperative setting. This separation also allows existing single-user objects to be used in a shared manner by overlaying new cooperative facilities.

2) Awareness

To promote cooperative sharing in the service, objects will need to make their use public across the service. The key to allowing this form of object use is the provision of awareness facilities in the service. These facilities need to combine an awareness of:

- The users accessing the object
- The environment in which the object operates
- Users’ interests in the information contained in the object

The provision of this awareness within the service is intended to balance the existing transparency facilities provided by distributed infrastructures.

3) Flexible access control and dynamic presentation

CSCW systems and the activities that they support can change rapidly and quite dramatically. Access to objects and the operations that they provide must be flexible and dynamic enough to reflect changes in the state of the group activity. Greif and Sarin [13] also outline the need to present shared data differently, depending on the access criteria allowed to the client. Consequently, the supporting infrastructure needs to allow a number of alternative presentations of objects to different clients.

4) Adjustable and extendible semantics

There is a need for both existing and new objects to cope with the extra demands, both functionally and semantically, placed upon them by multiple cooperating clients. For example, if a normal file object is used by multiple clients, some of its methods need to be extended to incorporate the knowledge necessary to preserve the consistency of the data in the file. New methods must be attached to objects to allow clients to make use of these existing, but extended, operations.
These requirements apply to the cooperative sharing of objects in general and need to be addressed at an appropriately general level. By developing a cooperative shared object service, support comes from a combination of the additional cooperative services and the underlying distributed system. A key to the provision of these services is the way in which objects are represented and managed in the shared object service.

The use of objects within the service is directly influenced by existing object models provided by popular distributed systems such as ANSA [1] (based upon ODP [15]) and CORBA [19] (based upon the OMG object model). Objects within the shared object service are abstract units that embody both state and procedure. Objects managed by the service publish an interface that makes the behaviour they provide available to other users of the service. The shared object service provides a set of user oriented facilities to manage the cooperative features of shared objects. Before considering the nature of the facilities provided by the shared object service it is worth briefly reviewing previous approaches to object management in distributed platforms.

MANAGING OBJECTS IN DISTRIBUTED SYSTEMS

A common abstraction within distributed systems is the notion of an object which encapsulates both state and behaviour. An object model associated with the distributed system defines common object semantics for specifying the externally visible characteristics of objects in an implementation-independent way. The externally visible characteristics of objects are described by an interface which consists of operation signatures. These model the external view of both object behaviour and object state. Objects make public these interfaces to the distributed infrastructure through some process of registration with a central trader or object broker. Clients exploit the services provided by objects by calling an operation associated with the object. This often makes use of specialised distributed system techniques such as remote procedure calls (RPC). The general arrangement is shown in figure 2.

![Figure 2: Objects in a distributed System](image)

Management of objects within distributed systems should be flexible enough to allow a wide variety of control. However, Davies et al. [8] indicate that systems which aim to be flexible often tend to tie the system into supporting only the design environments for which they were configured initially. This makes evolution, or the provision of support for multiple environments difficult. A key reason for this paradoxical situation is that the management mechanisms include semantic information concerning the domain of application. In contrast, we would suggest that flexible management is dependant upon the provision of an appropriate set of lightweight mechanisms. The need for flexibility in any general distributed system has generated a number of mechanisms which can be used as the basis of flexible management. The most notable of these include:-

- **Dynamic run-time binding** allows a client to invoke an operation on an object without requiring the object’s interface (typically inserted as a stub into the client) to be known when the client is first compiled. This type of binding allows dynamic presentation of objects and extendible interface semantics, since interfaces no longer need to be fixed permanently at any given time.
- **A separation of mechanism and policy.** Many distributed systems separate the mechanisms which the client employs (provided by one or more servers) from the policies which determine what those mechanisms will do. This separation minimises the amount of semantics which needs to be embedded in an object. Finally, provision of group mechanisms in a distributed system enables objects, and groups of objects, to be controlled and manipulated as a whole or as individuals [5].

These techniques can offer significant support for managing distributed objects. Further, additional techniques may also be employed by the system or application to augment existing management features. These often make use of application domain knowledge in object management. Within the shared object service we focus on the provision of facilities to manage the cooperative aspects of shared information.

Management in the Shared Object Service

There are a number of ways the management functionality provided by a shared object service can be realised and presented. Each approach offers different potential benefits and drawbacks for the service. To illustrate the point consider locking, a fundamental component of any multi-user system. Locking can be provided in one of a number of ways:

**As a basic mechanism of the service**

Traditionally, within database management systems, this would be implemented as a fundamental component of the DBMS service.
As part of each object
Within the object-oriented database world, this functionality might be implemented as part of the methods and state associated with the most basic object type. Thereafter, the user can safely assume that every object type possesses the ability to lock access to (portions of) its state.

Using multiple inheritance
A further approach might be the use of an associated object linked using multiple inheritance to the object being locked. For example, we may have a “person” object and a “lock” object. By creating a new shared subtype, we can create a “lockable person” type.

Using object adapters
An object-oriented alternative would be to use interface adapters that dynamically handle objects’ invocations to provide the additional functionality required.

The philosophy adopted by the shared object service is to provide all aspects of the service as objects that can be interacted with using a similar set of mechanisms. Thus both objects within the services and the facilities provided by the service can be used by clients in an orthogonal manner. Given this orthogonality it is important that we maintain a separation between the behavioural semantics of the objects within the service and the management facilities provided by the service. For example, the use of multiple inheritance in the case of locking, mixes together management and behaviour semantics and potentially overloads the class hierarchy. We have chosen to use object adapters as a means of maintaining this separation.

INTERFACE ADAPTERS
Two concepts are important in considering interaction with an object in the shared object service, interfaces and interface adapters. Interfaces describe the means by which a client interacts with the services provided by the object. Interface adapters provide facilities that abstract over these services to provide different services to users based upon context. The service provides mechanisms to create and manipulate both interfaces and interface adapters. An application or user can interact with an object by:-

- Access to the basic interface provided by the object.
- Access through an interface adapter which abstracts away from the object.

The relationship between these different forms of interaction is one of abstraction. Thus an interface represents an object and an interface adapter replaces the interface it represents. Object interfaces within the shared object service are analogous to the use of interfaces within distributed platforms such as ODP and ANSA. Interface details are described using an Interface Definition Language similar to that suggested by either ODP or OMG. We would expect this definition language to have strong similarities to those provided by most distributed systems. We do not see the shared object service becoming overly concerned with the exact details of how methods and parameter details are defined and leave this as an issue for supporting platforms.

For the purposes of the shared object service a sufficient overview of an interface is to consider it as an access list. Thus each node requiring services from the shared object service contains information about methods used, invocation protocols, parameters and different methods’ location.

Adapters provide a level of indirection between object interfaces and object users. This level of indirection allows interface adapters to provide additional facilities to manage the invocation of methods depending upon the context within which they are defined. Thus object adapters can be used to support access, coordination of users, and task sensitivity. The basic structure of object adapters is as a set of mappings which map method names from those presented to users to those used within the object interface. This overview of interface adaptation within the shared object service is shown in figure 3.

The use of adapters builds upon a number of previous themes in distributed and object oriented systems. We are particularly influenced by the use of delegation in object oriented systems [16], adapters in OMG and active values in Smalltalk [11]. Delegation allows objects to provide specialised behaviour by forwarding messages to associated objects. Object adapters within OMG clad units of functionality provided by different programs to allow them to appear as objects and in Smalltalk active values are first class objects adding specialised behaviour by intercepting object messages.

The use of adapters maintains a clear separation between behavioural and management semantics. Adapters in the shared object service build upon this separation to allow us to focus on the cooperative sharing of objects. We provide a set of interface adapter objects that provide facilities to manage and support the cooperative features of shared objects. The use of adapters in this way provides a number of benefits.

- A clear separation between the behavioural semantics of objects and the management of objects.
- An explicit identification of the cooperative aspects of shared objects.
- The ability to experiment in order to investigate the correct requirements for object sharing in a cooperative setting.
• A simple mechanism that allows the shared object platform to evolve to support new functionality as it is identified.

At no point is an underlying object interface directly visible to a client, and the only way of invoking the operations is through an object adapter. This allows adapters to manage dynamic access control within the service. Interface adapters also dynamically enhance (or reduce) the underlying object interface to provide cooperative object presentation. This allows objects to present the set of services they offer in a variety of ways depending on who is using the object.

Dynamic access control
Multi user systems have used access mechanisms to control the different action of users on the shared systems. A variety of different models exist which offer different facilities to different application communities. Most of these access models are derived from the Access Matrix Model [26]. Cooperative applications present a number of distinct problems for the access matrix [23,24]. In particular, the assumption that access is set up and only occasionally altered by a single administrator has resulted in a static view of access control. However, access models within CSCW systems must provide alternative facilities. In particular, they should provide features which:

- Make access control more dynamic because access is altering frequently,
- Allow access rights to be inferred from the state of some activity or a users role,
- Allow a group of users to manage changes to access,
- Offer a finer granularity of access control than that offered by the access matrix.

The approach adopted by the shared object service is to use interface adapters to provide access management facilities. Access adapters contain a table of interface users and a set of rules that determine the access of users to the object. The access rules within the adapters also make use of specialised state information to determine access.

Clients of the shared object service may be people, services, or applications. Each of these are clients of the service and have an associated context. This context information can be used to represent external activities surrounding the shared object store. For example, within a lightweight activity model [26] supported by the service this context is described by a triple of the form:-

<name, role, activity_condition>

To perform access control, the client’s context must be presented to the adapter when an operation is invoked. Each adapter contains a set of rules that determine which operations are available to a client. When a client invokes an operation on the object, through the adapter, the client’s context determines the client’s access rights for the operations presented by the adapter. The resulting “rights” are an access level, and a ladder reference number. Access levels are a less cumbersome way of defining which operations can be performed on an object. Instead of listing all the operations with each potential access rule, the object(s) operations are listed in one or more “ladders” of importance. Each ladder is identified by a ladder reference number. The greater the allowable access, the “higher” up on the ladder a client can reach and the more operations can be used. This arrangement closely ties access to the cooperative use of objects. The access model currently provided by our shared object service is associated with a lightweight activity model [26].

Cooperative Object Presentation
Object adapters can also selectively present and change the interfaces of one or more underlying objects. Operations presented by an adapter are mapped onto an underlying object interface and may filter out operations provided by an object. Unlike interfaces within distributed systems which cannot change once they have been assigned, an object adapter provides a set of rules based on the client current status within the system. These rules dictate which operations a user can invoke at any time during the activity’s life cycle. Two particular presentation techniques are exploited to promote cooperation within the shared object service.

Extending Object Semantics
Adapters can extend the operational semantics of the objects they adapt to support cooperative behaviour. These additional operations masquerade as standard operations to the client. Unlike object operations, these new methods have side effects which are necessary to change the underlying object into a more cooperative one. For example, an adapter can extend the single user oriented write operation of a file object, by checking that the client invoking the operation can do so and still preserve the consistency of the file. The client can now use the write operation, presented and modified by the adapter, as though it was directly on the file object. However, unlike the original write operation, it can now fail gracefully if any consistency problems occur.

Extending Object Interfaces
Adapters can amend the presentation of more than one object interface. This means that an adapter can present a combination of more than one interface (and hence more than one object) to a client. For example, consider a document object shared between two users Tom and John. This is merely an extension of a simple file object, which would normally have read and write operations and the basic function of the file should remain the same when the object is shared. However, the semantic information regarding when the methods affecting the objects state, for example write, can be applied must change to accommodate multiple users.

The document object is realised in the service as an adapter which combines two object interfaces, the basic file object and a special schedule object, which determine when it is appropriate for a client to be refused write permission (figure 4). Therefore while the adapter “read”, “append” and “write” operations come from the file object, the “Request write permission” and “Release write permission” methods come from the schedule object.
Additional specialised operations are also provided by the adapter which use and act upon knowledge of how the object is supposed to be shared. These additional operations characterise different classes of adapters which provide different management facilities with the shared object service. A couple of these are examined in the following section.

ADAPTERS WITHIN THE SERVICE
A variety of different adapters can be supported by the service, each allowing different aspects of the service to be made available to users for management. Rather than consider all of the different aspects of the service in full we wish to focus on two particular uses of adapters within the service:

Awareness
Current multi-user sharing systems have adopted an approach of strict access transparency. In contrast, cooperative systems need to promote awareness across the user community. The service provides a set of adapters designed to promote just this awareness.

Locking
Effective CSCW systems require a close linking of the locking mechanisms provided by the shared infrastructure and the co-ordination and cooperation needs of user groups.

Each of these different aspects of the shared object services is examined in the following sections.

Awareness in the object service
The shared object service is distinguished from other multi-user storage systems by its explicit goal of providing an awareness of the action of others across the service. Thus, when an object is accessed or altered in the shared object service, the system informs relevant users the action has occurred. In addition, the shared object service provides facilities to allow users of the service to make others aware of actions of interest to them. The principle mechanism used to provide this form of awareness is events. The shared object service provides facilities to allow events to be defined, created and related to others.

In supporting co-operation among a wide set of different users and platforms, possibly with different interfaces to applications and even diverse language bindings, a uniform and clean event mechanism is needed. In the shared object service an *event handler* provides a central facility responsible for event propagation. This portion of the service also maintains a record for the clients of which types of events are of interest to them.

A special form of object adapter called an event adapter is used to inform others of changes to an object in the service. In essence this object converts existing objects which are unaware of the existence of others to cooperation aware objects (figure 5). An event adapter knows about interest relationships through the event handler. If an object declares interest in events to or from another object or agent then an event adapter “filters” events to and from the interested object. If an event is of “public” interest that event is also propagated.

Lock Adapters

The locking of information within a multi-user setting has two distinct effects. It ensures the integrity of the information and reduces the possibility of error caused by simultaneous alteration. Locking also provides a means of mediating and controlling the work taking place around the shared information. Potentially, locking provides a means of co-ordinating access to information, sequencing the use of information and identifying the parties using the information. Investigating the use of locking in a cooperative setting requires us to separate the locking mechanism, realising the locks upon information, from the locking policies, that control how that information is locked. The intent within the shared object service is to maintain this separation and provide a means of making the locking policy explicit.
If an object requires the locking services, an interface adapter is placed in front of the object itself. Requests that were originally targeted to the underlying object are now directed to the adapter. The adapter contains “state” information relative to the locking functionality it provides. Moreover, it contains the “algorithm” or “protocol” used by that particular locking regime. Traditional locking strategies have long been recognised as unsuitable for CSCW applications [9, 13, 21]. Embodying the locking mechanism(s) in a set of adapters allows the flexibility to prototype and experiment with a set of different mechanisms. Thus interface adapters contain an interface description, locking state information and an appropriate locking policy. The arrangement is shown in figure 6.

Lockable interface adapters provide a set of management facilities that allow the locking policy to be modified. Often this policy can be represented externally with reference to a particular policy object. As in the case of access, locking adapters may use representations of the status of activities outside the service to infer locking conditions [26]. For example, a lock adapter may use the stage of a project to determine the locking policy to be applied.

While events promote ‘real time’ awareness, locks can be used to promote awareness in a time independent manner. The shared object service supports the notion of “identified locks” within adapters. This means when a user (or an application executing on behalf of the user) locks an object, the user’s identification is part of the lock. This provides information regarding “who” is using the object. In certain applications, this may be complemented by recording a reason for the object being locked within the adapter. This combination of “who” and “why” provides a useful form of awareness across the service.

We have shown how the use of architectures within the shared object service allows the addition of mechanisms to support cooperation. The following sections consider the architecture needed to support object adapters and the particular adapters provided within a shared object service realised as part of a lightweight activity platform.

ARCHITECTURE

A number of components of the service architecture are relevant in supporting object adapters. These include:

- a trader, which enables clients to locate, and services to register, interfaces.
- an object factory, to create objects.
- an adapter manager, which acts as a central repository of adapter information and creates the objects required by an adapter.
- a binder, which obtains an object’s interface reference for clients, after binding all the objects needed by adapters.

Each of these components are realised directly as high level managers specific to the service. These map down onto the object services provided by the distributed system, such as a low-level object binder and trader. For example, the binder in the shared object service manages the binding process based upon the cooperative environment it is in. However, it still uses a lower level system binder to perform the actual binding. Likewise, the shared object services trader makes uses of the mechanisms provided by a system level trader while overlaying specific knowledge and functionality.

Trader

The trading service imports and exports interface references for objects within the shared object service. Upon creation, objects export their interface descriptions to the trader which are rendered unusable without being passed to the binding service. Interfaces are described using an Interface Definition Language (IDL) derived from the existing OMG IDL. Clients obtain these interface references by requesting any interface which matches an interface description passed with the call. Any description which matches this interface description has its reference returned to the client. Interface descriptions can be either constructed, or selected from the result of a more general trader query.

Adapter Manager

The adapter manager acts as the central point of coordination and reference for the adapters and contains information about the object adapters. This includes:-

- Basic adapter definitions, exported to the factory when the adapter object is created.
- Invocation path lists. These are lists of object requirements for adapters which describe the necessary invocation path that an adapter requires. Path relationships are expressed as:

  \text{Adapter A, interface I \text{adapts} Object B, with interface J}

  This indicates that the adapter object A is an adapting object over object B, which may be a simple object or an adapter itself.

Binder

The binder’s function is to set-up the connection between a client object and the interface of a service object. A client passes an interface reference to the binder, which may be to a normal object or an adapter object. The interface reference contains its network address. The binding process consists of assessing whether the required binding is legal and converting the interface reference to an identifier. The actual binding itself takes place by calling a system level binder. Upon completion, the binder returns an interface identifier to the client which can then be used to invoke operations on the object.

Object Factory

The object factory holds a list of object templates and corresponding object definitions for the objects in the platform. The definitions for adapters and masters are exported to the factory by adapter managers upon their creation.
In order to create an object, a client passes an object template to the factory. The template is tested against the factories template list. If a successful match is made, the object, stored along with the matched template, is created from its definition, and the resulting reference to the new object is returned to the client.

**MANAGING, INITIALISING, BINDING AND INVOKING SHARED OBJECTS**

A client may use any adapter or object within the service, but to do so they must first request that the service bind them to the required object or adapter. If the object does not yet exist the client must first ask the service to create it. Creation of objects can also occur during the binding process if the object being bound is an adapter. Successful completion of the binding process on an adapter object forms an *invocation path* between the client and the underlying object(s). Adapters are then subsequently managed through an associated management interface.

**Invocation Paths**

An *invocation path* is the route an invocation can take from a client, through zero or more adapters, to a basic object. Since the method for binding objects is recursive, enabling an adapter to use another adapter (as adapters are only a special form of object), paths of any size can be created. Information about which objects belong in a path, and how to construct them, is stored in the master adapter. Figure 7 shows a client, John, bound to an extended version of a shared file adapter. To provide a high degree of flexibility in deciding when someone may write to the file object, the scheduler is not considered as a single object, but as an adapter with a number of separate policy objects. The initial binding has caused the creation, and subsequent binding, of the objects who belong to the invocation path from the client to the scheduling policy objects or file.

From John’s point of view, this path is transparent. The only “visible” section of the path is to the shared file adapter. This transparency holds for any adapter (who is effectively a client of the next adapter in the path) with each client being only aware of the subsequent object within the path they are invoking and no more. Figure 7 highlights the various different views of the invocation path using an extended version of our shared file example. The client (John) is only aware of the shared file adapter, the shared file adapter can only see the file object and the scheduler, which itself is only aware of the policy objects to which it has access.

Once created, the entire path may be extended by the addition of further adapter objects on the first original adapter. This extension of the path causes no access conflicts with the existing adapters as a new adapter cannot bypass the old.

In order for a client to bind successfully to an object in the shared object service, the requested object must not be contained *within* the original invocation path. This path integrity condition allows a client to bind either to an existing interface at the start of the original path (the shared file adapter above), or to adapters which have extended the original path.

In addition to the path integrity condition, we place a final restriction on adapters and their relationships with objects. An object may only be presented by one adapter type at any one time. This condition may be relaxed if the different adapters do not invoke conflicting underlying object methods that alter some state. Otherwise, it is necessary in order to preserve the consistency and security of the underlying object state.

Finally, it is the responsibility of the underlying distributed system to make sure that shared service objects (those being used by multiple clients) are isolated and cannot be accessed from outside of the service.

**Adapter Management**

Each adapter contains a management interface which is used to manage the adapter and the adapted objects. The interface may or may not be used depending upon the adapters position in the path. The head adapter of the original invocation path and all subsequent adapters added to the path externalise the interface but those contained within the original path do not. This adapter interface provides a number of important functions:

- It manages and co-ordinates the group of adapters under its control. This means, for example, that locks can be freed if an adapter locks an object and then fails.
- It holds associated adaptation information, such as who may access the underlying object, which adapters can be attached to it and which objects it can abstract over.
- It participates in the creation of the adapters to be under its control.
- It enables the state of the adapter to be changed during the adapters life-time, such as the access rules and ladders.

**Binding Process**

A client asks to bind to an interface using an encrypted interface reference. This reference is decomposed by the binding service into the physical network address of the interface and the object type. If the object is not shared (i.e. it is not adapted) then the binder proceeds to bind, using the distributed system binder where available, and returns a usable interface reference to the client. If the object is an adapter, the associated management interface is consulted to
check what position the interface occupies within an invocation path.

If the interface is unavailable then the binder aborts as the client fails to satisfy the path integrity condition. Otherwise, the binding proceeds and a usable interface reference is returned.

**Object Creation Process**

In order to create a shared object a client initially calls an adapter manager in the service with a template which describes the object they wish to create. The service attempts to match the template with one of an internal list of adapter object templates. If no match can be made then the object is assumed not to be an adapter and the object factory service is used with the template. The factory subsequently creates any matching object definition and returns the new object reference.

If the adapter exists, the adapter manager calls the factory to create the adapter. Creation of all other related and required objects is done by calling the adapter manager itself. This recursive creation procedure enables adapters to “adapt” over other adapters, whose nature will remain hidden from the client adapter.

If a client wished to create an adapter to extend an existing path, the adapter manager would be invoked with the object template that they wish to add to the path, and the interface reference of some top level interface in an existing invocation path. A similar process to the above then takes place.

**REALISING ADAPTERS WITHIN COLA**

Our realisation of the shared object service has taken place as part of a more general platform to support for cooperative work called COLA [26]. The platform is motivated by the need to promote a more open approach to supporting cooperation. Existing purpose built CSCW support tools and environments allow little access from outside the system itself. In addition, many include a constraining amount of semantic information about the cooperation being supported. This heavyweight and closed approach make them unsuitable in providing CSCW systems within real-world heterogeneous environments.

COLA to act as the supporting “veneer” between semantically laden cooperative environments and distributed systems (figure 8). The platform is underpinned by a lightweight model of activities [26]. The model exploits the nature of the platform to employ specialised object adapters as the basis for a lightweight activity facility which relies on information sharing rather than exchange.

**The COLA model and platform**

A central part of the platform is a lightweight activity model which represents the cooperation of the users and provides a context in which the objects are shared. The activity model separates different goal oriented (however vague) forms of cooperation into separate activities. Each activity is managed by an associated manager who facilitates group awareness through the provision of an externalised state.

The platform is considered as providing a set of policy free mechanisms to allow a wide spectrum of different activity features to be represented. Therefore, activity policy, such as the enforcement of deadlines and role allocation, resides exclusively with the application and not COLA. It is the prerogative of the application or cooperative environment to decide how and when the cooperative mechanisms should be employed. This mechanism and policy separation not only allows a variety of existing cooperative models to be built on top of it, but also enables users to circumvent applications and directly interact with the platform.

Within each activity, objects (users and applications) can play many roles at once and a role may be played by any number of objects. No semantic attachment to the notion of a role is made beyond grouping objects with some commonality. If required, the application using the activity can use the role names as hooks to attach any semantic information they wish.

Users, applications and objects are kept aware of each others’ actions, inside and outside of their own activities, by event delivery. Events are delivered through subscription to an event facility. All these services are supported by the ISIS distributed toolkit which was chosen because of the provision of mechanisms for reliable message passing, group multicast, synchronisation and fault tolerance [5].

**COLA Object Adapters.** All COLA object location, binding and creation is performed by the a particular instantiation of the shared object service. The shared object service uses specialised COLA adapters to support the lightweight activity model. Because the information within an adapter is necessarily activity dependant, adapters are uniquely defined with respect to a given activity. This separation of the sharing policy from the basic object mechanisms allows all the underlying object definitions to be globally instantiated. The adapter definitions and requirements are stored in an activity manager provided as part of the COLA platform.

To understand the use of COLA adapters it is worth considering a simple scenario of cooperative work based on the use of the platform within an academic department. The scenario centres on the setting and marking of an
examination held electronically within the COLA platform. The examination has four basic phases: creation (of the paper), approval (that the paper is satisfactory), sitting (when the student actually take the paper) and marking (when the student answers are checked). Jonathan and Nigel are in charge of creating the exam paper, a shared object, and are both assigned the “creator” role within the activity. Tom must approve the paper and is assigned a “reader” role, while Bill and Ben are students who must take the paper and have a “student” role. Dave is a user of the service but has no role within the examination activity. The exam paper is being shared among a number of different people who have different reasons for using it.

COLA adapters exploit a user context realised as triples to manage the presentation of objects. These triples consist of a user, the set of roles which the user is playing, and the activity they are in. The platform ensures the triple is complete by making clients members of a default activity and in any given activity provide a default role. The defaults, while having no specific function other than for filling in a context, enable all objects to use, and be used by, the activities and services within the platform.

As in the general shared object case when a client invokes an operation through the COLA adapter the client’s context is checked against a set of access rules. The first rule to succeed results in an access level and a ladder reference number for the user. If no rules are applicable then the invocation fails.

Any client can execute a method on a COLA object by presenting the name of the method they wish to call, the parameters the call requires, and one of their assigned COLA context triples. The platform passes the information to the COLA adapter object which uses the context to decide what access level and ladder the client can reach with the current context. If the desired operation has an access level in excess of the one calculated then the operation fails.

Figure 9 shows a rule set for the examination paper COLA adapter used in the examination scenario. Any students, for example Bill and Ben, will only be able to read the paper during the Sitting stage of the activity, while anyone with the writer role, Nigel and Jonathan, will always be able to invoke every method on the operation ladder 1. If the activity is in any stage before Mark then clients without a specific role cannot invoke any operations.

As in the shared object service, to prevent tampering with a shared object, objects created using the COLA platform can only be used through the COLA platform. This still allows applications and environments outside of the activity owning the adapter to share the object. The degree of object sharing across different activities depends on how leniently the access rules for each object have been specified, and how activity dependant the objects are. For example, the access rules of a shared examination paper would make little sense outside of an examination activity because that object is highly activity dependant. Conversely, it may be desirable to allow access to information contained within an object representing a shared whiteboard across more than just one activity.

### SHARED EXAM PAPER RULE SET:

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
<th>Activity</th>
<th>Level</th>
<th>Ladder</th>
</tr>
</thead>
<tbody>
<tr>
<td>any</td>
<td>Student</td>
<td>stage = Sitting</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>any</td>
<td>Reader</td>
<td>stage &gt;= Approve</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>any</td>
<td>Writer</td>
<td>none</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>any</td>
<td>any</td>
<td>stage &gt;= Mark</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

### OPERATION LADDER: 1

<table>
<thead>
<tr>
<th>Level</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Write</td>
</tr>
<tr>
<td>1</td>
<td>Release Write Permission</td>
</tr>
<tr>
<td>1</td>
<td>Request Write Permission</td>
</tr>
<tr>
<td>2</td>
<td>Append</td>
</tr>
<tr>
<td>3</td>
<td>Read</td>
</tr>
</tbody>
</table>

Figure 9: A shared file ladder and rule set

### PRELIMINARY ISSUES AND EXPERIENCES

The managers and objects which make up the COLA platform and its shared object service have now been realised, together with several graphical browsers and a suite of library routines which support user and application interaction with these components. One of the first applications which has attempted to exploit many of the lightweight features provided by this platform, most notably object adaption, is a variant of the examination example presented above.

Figure 10 illustrates a lecturer’s view of a particular examination activity. Each of the shared file objects which represent and hold the state of the various papers in the activity (sample, question, answer and marks) are adapted by
special locking adapters and presented through simple text widgets. These adapters intercept all the underlying file object operations and actively alter the write/insert/delete operations in order to apply the locking policy. A simple popup menu on each file’s widget (figure 10) interacts with the corresponding adapter and allows the application and user to select which particular locking policy they wish the adapter to apply. Read operations on the file, while still propagating through the adapters, remain unaffected by the actual adaption (except where the read may conflict with the access control rules which COLA enforces on the adapter).

In this application the use of adapters has allowed the successful separation of:

- the underlying file objects mechanisms and semantics from the application itself,
- the problems of multi-user access on a shared, single, object from the application and the file object.

This separation of application semantics from the multiuser issues and from the basic object functionality has allowed us to concentrate on each of these aspects individually without becoming concerned with the whole, and has made it far easier to change and adjust the application as new requirements become apparent.

One of the biggest potential drawbacks of object adaption is the performance cost of any invocation on an adapted object. While no detailed performance evaluation has yet been made, it is obvious that in a straight-forward implementation of the shared object service a normal invocation using adapters will be proportional to the length of the adapter’s invocation path. Indeed, this is the case in COLA. However, this cost can be reduced in a number of ways. Objects in the ANSA testbench which are related in some way can be grouped into “capsules” [1]. When one object in a capsule invokes another within the same capsule this invocation is mapped onto a more local procedure call and not a remote one. Because it is often the case that the objects being adapted and the adapters themselves are tightly coupled, such a technique can be applied, and would reduce the overhead of the number of remote, slow, calls to an adapted object. In the next worse case, where an adapter is being added to an existing object (the invocation path is being extended), the potential benefits of simply making the adapter local with respect to the adapted object is again worth exploiting as any reduction in network use will be beneficial (for performance). In the worst possible scenario, where the adapters must reside on a different location than the adapted object, the impact of advances in technology in the near-future should not be ignored. More sophisticated shared virtual memory areas and higher speed networks could dramatically increase the performance of a remote object invocation, therefore decreasing the current penalty that object adapters may currently incur.

CONCLUSION AND FURTHER WORK

Successful information sharing is critical if users and applications are to cooperate together. Previously, CSCW applications have relied on distributed systems to provide the neccessary management and control of the information which is to be shared between users. Unfortunately the traditional approach adopted by many distributed systems to management does not involve users and actively prohibits the cooperation and sharing they need to support.

By considering the generic implications that emerge from cooperative work, we have proposed a shared object service which explicity provides the necessary platform for sharing across any cooperative systems. The service supports a number of general requirements for shared objects which include:

- Lightweight semantics through a policy and mechanism separation.
- Awareness.
- Flexible access control and dynamic presentation.
- Adjustable and extendible semantics.

The management and creation of shared objects within the shared object service takes place through a variety of cooperation-oriented services, many of which augment those provided by the underlying distributed system.

This paper has concentrated on interface adapters within the service which help to satisy the requirements for shared objects. Adapters maintain a clear separation of the behavioral semantics of an object and their management by providing a level of indirection between a shared object interface and its users. This indirection enables adapters to provide dynamic access control and cooperative object presentation. We have shown how two types of adapters within the service, events and locking, provide an essential degree of cooperative awareness between objects and how the cooperative work can be mediated and controlled from the nature of that work itself.

While the focus of this paper has been on the use of object adapters to support cooperative sharing, object adaption can usefully be incorporated into other application domains. For example, the transparent presentation and modification of an object through an adapter may provide a very intuitive and workable approach to integrating heterogenous systems or objects. In addition, the ability to present a combined or amalgamated view of one or more objects can be applied to object oriented-databases, where the results of multiple queries may be “joined” using some form of adapters.

We have highlighted how many of the concepts and services introduced in this paper have been realised and built as an integral part of a more general platform, COLA, to support CSCW applications. Further work is required to see how successful adapters and other services provided by the shared object service through COLA are at supporting the needs of cooperative users for sharing objects.

ACKNOWLEDGEMENT

This work was partially funded by the COMIC project (ESPRIT project no 6225). Our thanks are due to our partners in the project.
REFERENCES

1. ANSA, ANSA: An Engineers Introduction to the Architecture, Projects Management Limited, Poseidon House, Castle Park, Cambridge, CB3 0RD UK, November, 1989


