Proceedings of the Third European Conference on Computer-Supported Cooperative Work
13-17 September, 1993, Milan, Italy
G De Michelis, C. Simone and K. Schmidt (Editors)

COLA: A Lightweight Platform for CSCW

Jonathan Trevor, Tom Rodden, Gordon Blair
Department of Computing, Lancaster University, U.K.

ABSTRACT: Despite the reliance of cooperative applications on the facilities provided by distributed systems, little consideration is given by these systems to the support of cooperative work. This paper examines the provision of appropriate mechanisms to represent cooperative work within a distributed platform. Based upon a examination of existing models of cooperative activity and the experiences of their use, a lightweight model of activities is suggested as the basis for the supporting platform. Rather than concentrate on the exchange of information, this lightweight model focus on the mechanisms of sharing of objects. This focus enables a clear separation between the mechanisms provided by the distributed platform and the policy which is the responsibility of the cooperative applications.

1. Introduction

The technologies currently exploited to construct cooperative applications were designed and developed prior to the emergence of CSCW and its associated applications. The facilities provided by these systems and the manner in which they are presented seldom sit easily with cooperative applications (Rodden, 1992). The needs of the application domain, the nature of the work and the mechanisms to support cooperation are intertwined within the development process. The developers of CSCW systems are forced to juggle all these factors in an attempt to realise a cooperative system. This is a painfully slow and problematic task resulting in a set of similar services replicated across a collection of applications in a manner that is confusing to both developers and users alike.

It our belief that a key requirement for developers of CSCW applications is the need for specific support for the development of these applications. Applications programmers should be free to concentrate on the semantics of the application and what should be provided. The programmer should not need to worry about how the mechanisms employed obtain the required result. The development of this support is essential to the future of CSCW and we would agree with (Patterson, 1991) when he argues:-

"If multi-user applications are to flourish in the future, then programmers will require support for building these applications" .

This paper presents a platform to support group working by providing mechanisms for sharing Cooperating Objects in Lightweight Activities (COLA). The COLA platform provides the means to allow applications to externalise appropriate features of cooperative activities in such a way that these can be shared across applications. Previous activity models have adopted a modelling perspective based on communication and its influences on work flow or document flow. In contrast, our approach is based on mediating the sharing of information rather than controlling its exchange. Consequently, the emphasis is on the provision of suitable supporting mechanisms within the platform. This approach allow the details of policy and the associated control of this policy to be administered by the applications being supported.

2. The Nature of Support

The need for support has been addressed within computing in a variety of ways relevant to CSCW. At the lowest level, existing distributed systems provide support for many features of cooperative applications, for example, object migration and location transparency. In addition, some distributed system designers are considering more advanced features which could prove useful to CSCW. These include multimedia objects, group multicasting and high performance networks.

Distributed systems are far removed from cooperative applications which embody a set of assumptions of why people work together and often characterise how they should work. These applications contain considerable information about the domain in which they are applied. A number of CSCW *application toolkits* have emerged to support application development in different cooperative domains. These toolkits provide a set of existing facilities for a particular domain which can be reused by developers in the construction of new applications. Examples of this form include RENDEZVOUS (Patterson et al., 1990) and LIZA (Gibbs, 1989) which support the development of multi-user interfaces and DISTEDIT (Knister and Prakash, 1990) a development toolkit to support the construction of shared editing systems.

These toolkits provide little or no support for representing the cooperation taking place. However, a number of *cooperative environments* exist which focus on representing cooperative work and how these representations can support the work taking place. Tremendous variety exists within these environments and a number of different classes can be identified. Some indication of the classes and the diversity of cooperative environments embrace are shown in figure 2 .

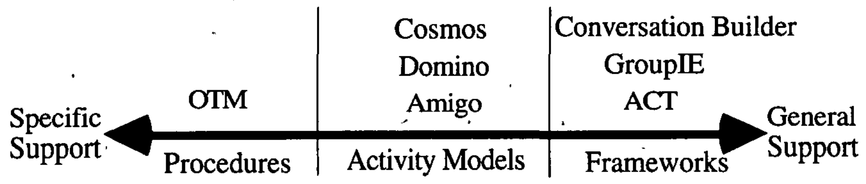

Figure 1. Spectrum of cooperative environments

In general, cooperative environments tend to be *goal oriented*; in that they include some conception of an activity or task which has some goal marking its completion. The more specific the support , the more specialised and narrow the goals. The majority of these systems are both computational and procedural in nature, However, *conceptual frameworks* exist where support is limited to the provision of models, outlines and frameworks upon which more specific instances can be built.

Procedural models describe well defined and understood tasks, almost exclusively in an office environment. The models control who takes part in a task, what operations they should perform in order for a task to be completed and what documents need to be exchanged. The model enforces what the user does and "runs" the procedure. Examples include OTM (Office Task Manager) (Lochovsky et al., 1988) and the systems currently being developed by action technologies based upon the COORDINATOR(Medina-Mora et al., 1992).

Activity models expand the horizons of procedural models by presenting a more general approach to task specification. In a similar manner to the procedure models, activity models have specific goals for the cooperation they are supporting. The main difference is that these models are general enough to be applicable to more than just those specific goals. The models concentrate on what and how information is exchanged between members of the activity and attach more significance to what these people can actually do. Many different systems offer these features, some of the most well known being COSMOS (Dollimore and Wilbur, 1991) , DOMINO-W (Kreifelts and Woetzel, 1987) and the AMIGO ACTIVITY MODEL (Danielson and Pankoke-Babatz, 1988).

Frameworks are the most general form of cooperative environment, they aim to provide support for cooperation, but without any specific domain in mind. The ACT (Activity CoordinaTion) model (Kreifelts et al., 1991) supports the coordination of activities in groups or teams. The ACT model differs from activity models (such as AMIGO and COSMOS which focus on coordinating the communication flow) by concentrating on coordinating the execution of actions. Other examples adopting particular perspectives on group work include GROUPIE (Rudebusch, 1991), CONVERSATION BUILDER (Kaplan et al., 1992) and OVAL (Malone et al., 1992).

2.1. Shortcomings of Existing Support

All applications, interfaces and tools rely, to a greater or lesser extent, on the underlying services provided. However, any problems with this support will inevitably be propagated upwards when the service is used. This is one of the major reasons why CSCW applications are so difficult to build. Not only do the applications contain many issues single user applications do not, but support can be unstable and often unsuitable for the demands that are placed on them.

The most obvious drawback of existing cooperative environments is that they are closed applications rather than open platforms and do not provide facilities to support a number of different approaches to cooperative work. In addition, the particular problems of merely extending or augmenting existing support models include:

i)Unrealistic Models of the real-world

One of the biggest shortcomings of existing support for cooperation is that they start from a set of unrealistic assumptions about work. For example, OM-1 (Ishii and Ohkubo, 1991) and other activity models represent well structured cooperative work knowledge. These assume that the information necessary for a task is know in advance and that the work follows a set procedure. In reality, work is not well structured or defined (i.e. the handbook is not followed and procedures are used as a resource rather than merely interpreted (Suchman, 1983)). Ishii and Ohkubo (1991) have also found this to be true in their experience of office tasks:

> "we found that office workers made many short-cuts and modifications to the standard procedures defined in the handbook. Therefore, it was no easy task to determine the actual standard procedure, even when it was defined clearly in the handbook".

Given that cooperative applications are intended to support the actual work of groups, unrealistic assumptions about that work will have tremendous impact upon the success of CSCW systems. Many authors have commented on the variability of work within natural settings and the difficulty of modelling this; interested readers are referred to (Bannon and Schmidt, 1992) for a full discussion of the issues involved. To minimise these problems we wish to reduce the set of assumptions within our support platform to the minimal set necessary to support cooperation.

ii) Constraining models of control

One of the main stumbling blocks of many activity models are that they rely on people behaving methodically and working to some plan. However, by constraining their actions users are being restricted by the model intended to help them. In fact, users often circumvent procedures and do the unexpected (Schmidt, 1991). This variability needs to be reflected in the support structure to allow applications to cope with variance from the expected norm. The handling of procedural exceptions in existing approaches is symptomatic of the problems of control. If exceptions are allowed, they can at best only be handled in a very general way. This leads to prescriptive models, which eliminate the possibility of exceptions and increase the burden upon the user.

Services and support tools should assist and augment higher levels of abstraction, and not prescribe particular viewpoints. The purpose of a support platform is to aid both the programmer and user, not to force them to do things in a particular manner. Consequently, a platform needs to provide the *mechanisms* necessary to represent cooperation. Policy surrounding control and coordination remains the exclusive responsibility of users or where appropriate applications.

iiii) Lack awareness of "groups"
With the possible exception of a few CSCW environments, existing computer platforms, tools and services provide only limited awareness of others, thus users are unaware of who is cooperating on what. It is often the case that people gain new insights and ideas from others and there is no reason why this shouldn't be the case when people cooperate together on a computer. A supporting platform should provide a high level of "*group awareness*", with users aware of the actions of others.

iv) Limited support for "sharing"
Cooperative work relies on people *sharing* information (ideas, files, etc.). However, the majority of activity models attempt to coordinate people in cooperative work through a model of cooperation based on asynchronous message passing (for example, forms based systems). Little support is provided for the sharing of objects to support the cooperation taking place.

We would therefore, characterise much of the existing support as heavyweight with a high degree of application specific semantics encoded, and enforced, by a model based on message passing. We wish to consider the development of a lightweight model which adopts a perspective to cooperation based on sharing.

3. A Lightweight Approach

Future CSCW applications will desire a great deal more than current support can provide. Using the problems above as a basis, we can identify three major desirable characteristics for cooperative support platforms:

i) A lightweight and flexible representation of cooperation

ii) A separation of the application semantics from the support features

iii) The provision of increased group awareness

Our approach is to design a platform which directly addresses these requirements. This will allow additional services to develop more realistic models of cooperation and allow further study into suitable and realistic underlying support for cooperation. In essence, this lightweight approach needs to provide useful *mechanisms* for describing cooperative situations, while relying only on *minimal semantic knowledge* in order to interpret these mechanisms.

Our focus is on the representation of cooperation which augments existing distributed services and communication systems. The COLA (Cooperating Objects in Lightweight Activities) platform provides a lightweight service which *aids users and applications in the cooperative use of objects.* The central part of the platform is a lightweight activity model which is used to provide a context in which objects can be shared. Unlike many cooperative environments, the bias of this platform is towards providing mechanisms to support sharing but with limited semantics. This means the platform acts as a "veneer" between semantically laden cooperative environments and distributed systems.

COLA presents mechanisms to cooperative applications and environments building upon the general mechanisms provided by distributed systems. With this approach:

- the only semantic information put into the activity are the events that change the state of the activity and the stages that an activity goes through.
- an activity can move forward and backwards or jump to any stage. The lightweight model does not specify when this occurs.
- objects can be shared amongst activities, and can be accessed from many different contexts (even from outside any activities)
- objects are context dependent and can present different interfaces to different people at different times.
- every change in an activity produces an event. Events are delivered to anyone who has specified some interest in it. Therefore, anyone in the system can be as aware as they like.
- users can move between activities as they wish and may undertake some roles local to an activity.

Activity *policy* rests exclusively with the application - the enforcement of deadlines and other features of activity control are contained within the application. The COLA platform can be considered as providing a set of *policy free mechanisms* to allow different features of activities to be represented. With a clean mechanism and policy separation the platform not only allows existing models to be built on top of it, but enables users to circumvent applications and directly interact with the platform.

The platform provides two important interfaces. The first is through a defined service interface available to cooperative applications. This allows applications to register activities and update the information within the activity model using remote procedure calls. An equally important component is the lightweight activity browser which provides a user interface to the activity model which allows activity information to be directly accessed and manipulated by users.

4. Components of a Lightweight Model

The lightweight model adopted within the COLA platform focuses on providing mechanisms to allow the externalised nature of activities to be represented and shared. Consequently, the platform is less concerned with either the structure or intent of the cooperation taking place but more with representing external effects of the cooperation. A number of distinct components exist within the lightweight model.

- *Activities*, provide a structure for the cooperation.
- *Roles*, limit object access and presentation in an activity.
- *Objects*, which are unaware of their context and Object *Adapters*, which present objects in context sensitive manner.
- *Events*, enable any objects registered in the platform to be kept group aware.

This section explains these main aspects of the platform in the following sections, for each, a brief example is given to illustrate its use.

4.1 Activities

A lightweight activity is defined as *a process in which users and objects interact and exhibit a public state.* A number of people participate in activities and an activity has a *life-cycle*, subdivided into *stages,* which it moves through before completion. Advancement and retreating of stages in an activity is done by the applications involved in the activity through the activity service provided by the platform. There are several reasons why the conditions required for a change of stage are not recorded in the definition of the activity (the "activity template"):

- this information is the sort of semantic information which may not be known in advance
- exceptions may arise outside of the activity which means a stage may need to be skipped or retraced.
- often, even simple events are not fixed and are open to negotiation within a cooperative setting.

As an example, consider the setting of an examination associated with an undergraduate course. The question paper should be created and approved before the examination takes place. The paper may be submitted for approval and edited any number of times. The exam commences at a given time and lasts a set duration.

Answer papers cannot be created before the exam starts, after the exam is completed no further additions to the answer papers can be made.

In this scenario, heavyweight activity models would focus on the construction of this activity and its decomposition into constituent subtasks. A traditional activity model would attempt to capture the dependencies between sub tasks, the deadlines associated with different tasks and the behaviour exhibited by different roles. In contrast, the lightweight model within the COLA platform focuses on the external effects of this activity by defining only the stages the activity must move through and the objects and people associated with it. The stages to an examination setting activity could be described as:

CREATEPAPER STAGE

Purpose	To create an exam paper for students to complete
People involved	Writers (of the paper), Reader (to check the paper)
Objects	Exam paper and Sample Solutions

DOPAPER STAGE

Purpose	For students taking the course to read and complete an answer paper using the exam paper previously created
Roles involved	Student (of the course), Examiner (to solve problems with the paper)
Objects	Exam paper and Answer sheets.

MARKPAPER STAGE

Purpose	To mark all the answer papers produced by the students
Roles involved	Marker
Ends	Answer sheets and Sample Solutions

4.2 Roles

In an activity, people do different things to contribute to the work taking place. For example, within the examination activity their exists people who:

- set the exam paper.
- check/proof read the paper to make sure it is satisfactory.
- answer the questions
- mark the answer papers

In order to represent this variability, people can take on different *roles.* Some roles may be taken by more than one person, some may only be assumed by one. Previous activity models have used roles to explicitly describe the behaviour of different people, in contrast, roles are used within the lightweight model as a means of access control. This access control is sufficient to represent within the framework the profiles of the different people taking part in an activity. That is not to say that the application using the activity model cannot *prescribe* activity related actions or conditions on that role but this is not within the platform model which wishes to remain semantically neutral.

Within the examination activity a number of distinct roles can be identified:-

WRITER	The role concerned with writing and creating the question paper
READER	This role makes sure the paper is satisfactory.
STUDENT	The users taking this role create and write the answers during the exam.
MARKER	Anyone in this role marks the students answer papers.

Many of these roles may be occupied by the same person (e.g. in reality the writer of an examination paper and the marker are often the same) and at each stage any roles are free to participate. However, the roles within an activity need to be encoded in such a manner as to highlight illegal[1] access operations, such as a student reading the exam paper before the exam starts.

4.3 Objects

Objects present the largest problem in a lightweight activity model. They can be accessed in several ways, from within an activity where they were created, from inside another activity or from outside of any activity. In each case the object may present a slightly different interface to reflect the invoking users rights. Objects in a lightweight activity model may also present a different set of operations (an interface) to each role during an activity. Thus an object exhibits a degree of *activity sensitivity*. However the basic object itself may still be activity unaware, but is merely presented in different ways to different roles and at different stages of the activity.

Accessing an object from within the context of an activity, through a role, means that the object must respond differently depending on the accessing role and the stage an activity is in. Accessing an object from inside another activity requires more rules for different roles for the second activity. More importantly the security of the object could be easily compromised as two activities may have no knowledge of the restrictions each has placed on the objects. Clearly it is cumbersome in the extreme, if not impossible, to define huge sets of rules on each object operation for every potential eventuality. The solution adopted within the COLA platform is the use of *Object Adapters.* which provide a clear interface between the objects within the platform and the entities which can access them.

4.3.1 Object Adapters

Object adapters have attempted to capture aspects of both object presentation and access control, and as such, have much in common with many methods in both of the presentation and access areas (such as capabilities and proxy objects (Shapiro, 1986)). However, object adaptors in COLA embody a set of concepts concerning the cooperative sharing of the object. Every object has a basic set of operations.

[1]Although appearing illegal, access might not necessarily be invalid. For example, it could be a matter of policy that dyslexic students are allow to read exam scripts in advance

When an object is created from an activity that activity is given an object adapter that abstracts the basic object into a set of activity meaningful operations, together with a set of access rules for that object. The object adapter is initially defined as part of the activity and is stored, along with existing pre-defined activity templates (such as an exam activity) in an information store.

When an object is used, an instance of an object adapter is created which as far as the user within an activity is aware acts as the object and presents an appropriate interface to the them (as illustrated in figure 2). The object adapter provides several operations beyond normal interface filtering:

- *Extended operational semantics:* These methods masquerade as standard object interface operations insofar as the client would view the object interface without the adapter. However these new methods have deliberate side effects which change the underlying object into an object which is contextually sensitive. e.g. a normal "write" operation on a file will look the same to a client as the "write" operation on the underlying object, but will also perform some locking information.
- *Dynamic Contextual Filtering*: Unlike a normal interface to an object, which cannot change once it has been assigned, an object adapter provides a set of rules based on an associated activity's state and the client's current status within the system. These rules dictate what operations a user can invoke at any time during the activities life cycle.
- *Extendible Interfaces*: These are new operations specially provided by the adapter which use and act upon the semantic knowledge of what the object is supposed to represent with the activity.
- *Interface amalgamation*: No assumption is made that an adapter only presents 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. The client only sees one object through the adapter

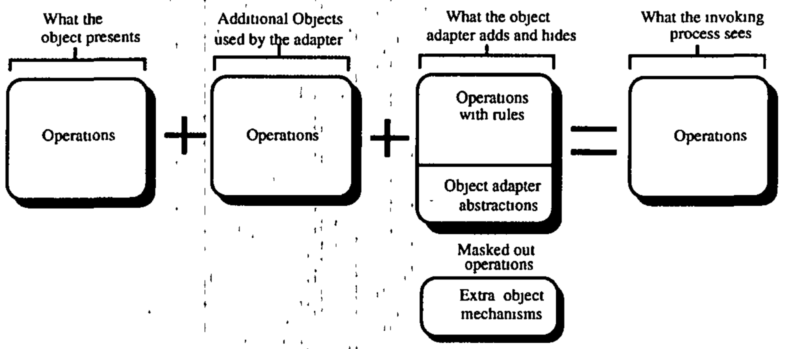
What the object presents
Operations
Additional Objects used by the adapter
Operations
What the object adapter adds and hides
Operations with rules
Object adapter abstractions
Masked out operations
Extra object mechanisms
What the invoking process sees
Operations

Figure 2. Presentation of an object

Each set of basic adapters and objects are managed and controlled by a master adapter, or simply a master. In effect the master adapter distinguishes the objects owner. This "ownership" can be transferred to any other activity or user at any time.

The rules describing which operations are usable by who, and when, inside the basic adapter and master are defined using a combination of the rules based upon the activity context, the role of the client, and the clients status in the overall system. If the object is *activity dependent*, e.g. an exam paper, then the set of allowable operations in the basic adapter will be very small. If the object is highly *activity independent,* e.g. a shared whiteboard application, then this set of operations may be large. One basic adapter is created per client of an object in order to allow the adapters to customise themselves towards particular clients.

It is up to the original activity, through the master, to promote clients to allow access to higher privilege operations. For each combination of user, role and activity which may access the adapter there is a corresponding rule and *access level.* Levels are a less cumbersome way of defining which operations can be performed on an object. Instead of listing the operations with each potential access, they are listed one (or more times) in the adapter and are ordered into one or more "*ladders*" of importance. Therefore the greater the degree of access you have, the "higher" up on a particular ladder you are, and the more operations you can perform. When the client invokes an operation on the object, through the adapter, the first rule which can be applied to the user (starting at the most specific) is applied. If no rules are applicable then the invocation fails. It is up to the original activity, through the master, to promote users and other activities to use higher privileged operations.

From the examination example, we can identify three necessary objects; the question paper, the sample solutions and the answer scripts. The following tables define the question paper as an abstraction of a text file which, for example, can only be read by someone in a student role when the activity is in the "DoPaper" stage.

UNDERLYING OBJECT: TEXT FILE

OBJECT OPERATION LADDER: 1

Level	Operation(s)
1	Wnte
2	Append
3	Read

QUESTION PAPER RULE SET:

User	Role	Activity	Required condition	Level	Ladder
any[2]	Student	*local*	stage = DoPaper	3	1
any	Reader	*local*	stage = CreatePaper	2	1
any	Writer	*local*	*none*	1	1
any	*any*	*any*	stage>= MarkPaper	3	1

2Keywords in ıtalics have specıal meanings. "any" in the user column for ınstance matches any user.

4.4 Events

Events are small structured messages that are propagated around the environment in order to make environment objects aware of any changes of state. Events which the user or application are interested in and can generate are kept and managed by the platform. The awareness that events provide is primarily achieved by allowing users to register *interests* in certain types of event with the platform. When an event that matches this "interest" arrives at an activity it is delivered to the user (as well as the actual specified destinations). Events addressed to one activity from another, or events which are generated locally within an activity, are always seen by members of the activity.

An event is delivered according to three levels of addressing, the most general being to all the people registered to an activity. The next level is the role within an activity and finally the user themselves. It is not necessary to always specify the activity if a role is specified, nor a role or activity if a user is specified.

Events themselves all contain several standard fields. Each event must be *named* uniquely (within an activity) and contain a *description* that outlines what the purpose of the event is. Two addressing fields are used, the *source* and *destination.* Each of the addresses can be decomposed into *user* (which may be an object ID and therefore does not always refer to a human user), a *role* and an *activity.* The final part of an event is the *contents* field which can contain any number of text sub-fields which can have activities, roles and users tagged onto them to restricted reading and writing.

Any activity will have access to a core set of events. These include event which are commonly used between objects (e.g. for synchronisation) and by the user. One such event is the *notify* event, a very general message which can be passed between users, objects and a mixture of the two. Most events are a specialisation of this event. Within the examination example there are a few possible specialisations, for example:

PAPERREADY? EVENT

Purpose	To ask the READER role to check that the question paper is suitable of the exam.
Contents fields	The location / name of the question paper object
S'rce restrictions	Writer role
Dest' restrictions	Reader role

PAPERREADYREPLYEVENT

Purpose	To confirm to the READER that the paper is satisfactory or needs further work
Contents fields	Approval field and a text field indicating what work needs doing
S'rce restrictions	Reader role
Dest' restrictions	Writer role