Experiences in Building a Configurable CSCW System

Jean Dollimore and Sylvia Wilbur
Department of Computer Science, Queen Mary College
University of London, Mile End Road, London E1 4NS, England.

Electronic mail addresses: jean@cs.qmc.ac.uk, sylvia@cs.qmc.ac.uk,

August 1989

Abstract

The purpose of the Cosmos project is to design a system to support co-operative tasks based on asynchronous message passing. The real-world model for our design is based on the idea of each user participating in group tasks by playing one or more roles within activities. One of the primary aims was to provide a configurable system, based on the notion of communication structure, and we have developed two notations for abstract definition of the structural elements of group activities. The functionality of a particular Cosmos system will be determined by the communication structures represented in it, each structure describing a particular class of group activity. By establishing a library of suitable structures, the system may configured to meet the needs of the organisation in which it operates.

Cosmos system design models are described, including the system model and user's conceptual model on which the User Interface design is based. The Cosmos software has been designed as a set of components that mirror the properties of group communication activities reflected in the design model. A working prototype based on these models has been built, and we have completed trials based on some structures we have created. The project ends in November of this year, and we take this opportunity to reflect on our design approach. Finally, some areas are discussed where further research and development is required.

1. Introduction

The main aim of the COSMOS\(^1\) project [Wilbur and Young 88] is to design a messaging system that supports structured group working. The Cosmos prototype provides automatic support for the participants in co-operative tasks by prompting on actions appropriate to the current state of their joint task and providing contextual information.

\(^1\)The work described in this paper was undertaken as a part of the Alvey COSMOS Project, a co-operative research project involving groups at Queen Mary College, University of London; Manchester University; Nottingham University; British Telecom and Computer Sciences Company Ltd, funded under the Alvey Programme by the Department of Industry and the Science and Engineering Research Council.)
This paper is written from the viewpoint of the group who developed the prototype, but with acknowledgment of the ideas contributed by other members of the Cosmos team. Section 2 describes the design of the Cosmos prototype with reference to the theoretical work in communication structures, task analysis and the development of a user's conceptual model. It outlines the system model and our approaches to configurability.

Section 3 discusses the prototype, the experiences we have had in developing it and what we would have done differently in the light of our current experience and knowledge of recent developments.

2. Cosmos design models

We have designed the Cosmos system to support asynchronous interactions between the members of groups of people involved in structured joint tasks. The real world model for our design is based on the idea that each user is involved in several activities – an activity involves other people and is associated with some task to be done. Examples of activities include conferences, committee work, voting, remote tutoring, joint authoring, editorial work, project management. The Cosmos idea of activity has much in common with the Amigo activity model [Danielson and Pankoke-Babatz 1987] in that users and particular types of information objects are associated with activities.

A second important feature of our model is the notion of role - within an activity, each user assumes a particular role. For example in editorial work, roles could include editor, author and reviewer. Roles in Cosmos are similar to those described by Tsischritsis and Gibbs [Tsischritsis and Gibbs 87] in that they correspond to the roles that people play and are related to the destination of messages; but in Cosmos, roles are within activities and relate to all of the user's behaviour (including creating and sending messages as well as receiving them).

2.1 Configurability

One of the primary aims of the project was to provide a configurable system and we were encouraged by the existence of Emacs [Stallman 81] as an example of a configurable system. We proposed the notion of communication structure as a basis for configurability - a communication structure is an abstract description of a class of communication activity, such as those mentioned above, and includes a specification of roles, actions and their ordering, the sorts of messages to be composed and exchanged and the organisation of the messages received.

We have designed a system that includes a library of reusable communication structure definitions from which activities may be instantiated. Instantiation includes the selection of users to play some roles in an activity, although further users may generally assume roles throughout the course of an activity. The system supports its users by interpreting the definition of the communication structure as an activity progresses, and by using the information in the definition to prompt the users as to the appropriate behaviour for their particular roles. The system stores information about the current state of the activity in terms of the progress of each
role player and databases of messages exchanged, both by exchanging messages and by using shared databases (e.g. like bulletin boards).

The functionality for a particular Cosmos system will be determined by the communication structures represented in it and may be configured to suit the needs of an organisation. We have provided configurability in the form of a language for defining communication structures that may also be interpreted by the system. But to make it possible for ordinary users to configure the system, this language must be accessible to them.

2.2 Languages for defining Communication Structures

The project has developed two separate notations for the description of communication structures. The Structure Definition Language (SDL) [Bowers & Churcher, 88] is an extensible rule-based notation for the formal description of a wide range of human communication activities, without constraints deriving from the particular implementation environment. The Script Language (SL) is a procedural language with concurrency constructs designed to be executed in an interactive environment that supports group working – to support these requirements, it includes user sanctioned operations and persistent environments.

In the Script Language a communication structure is defined in terms of the scripts for roles that may be played by the different classes of users participating in a group task. A script specifies procedures in terms of actions such as composing messages, sending them to a particular role player, receiving messages and making decisions [Dollimore et al 89].

2.3 Cosmos System Model

The Cosmos software has been designed as a set of components that mirror the properties of group communication activities according to the design model outlined earlier in this section. We have selected system components or data objects to correspond to the main entities in the real world and attempted to relate them to one another in the same way. The entities that are represented by the data objects in our model include communication structures, scripts, users, activities, roles, messages, queues of messages, lists of received messages, message headers and bodies. Each data object has a unique identifier by which it may be referenced within other data objects, for example, a communication structure refers to a set of scripts. An activity refers to a set of roles. A role refers to the set of users that play it.
Figure 1: Objects and references

Some example of simplified versions of these data objects are shown in Figure 1. We described the system in terms of this model and used the model to help us in building the system.

This system model contains all data objects required by all of the users in a particular location and is structured so that each user has an individual view of the structure. For example, each user is related to the activities in which they participate.

2.4 The user's conceptual model

The project has defined a user's conceptual model for this type of group working [Buckley 88], based on a preceding task analysis of group communication tasks. In this model, the user is presented with a role-specific view of the each activity, within which the relevant objects are arranged in related subsets. The user's conceptual model is based on a hierarchy of four different types of context described briefly in the next two paragraphs.

In the top level context, the objects available to users are the activities in which they play roles. The second level is the activity context – it contains scripted actions currently appropriate to the role player, together with item boxes and subactivities. An item is typically something sent in a message from one role player to another. An item box is a container for a related set of items, for example a list of messages. Item boxes may also be used as private workspace. The main operation available in the activity context is to carry out one of the scripted actions – these will cause new items to be created and sent to other role players. Items are sent to item boxes in activities where the recipients may find them.
The other two contexts are the item box context and the item context. An item box can correspond to a list of messages shared by several users (e.g. messages on a bulletin board), or it may be private to a single user (e.g. a personal mail box).

Figure 2. Conceptual view

2.5 Cosmos system support for Group Work

A user performs two sorts of co-operative actions in Cosmos: those that relate to information search (browsing) and administration; and those that support structured group communication. This is illustrated in Figure 2.

The former actions are supported by the Cosmos Information Service that provides coordinated access to a shared database that contains many of the data objects in the system model, for example, users, communication structures, and activities. Users browse the database in order to get information to enable them to contact users, join activities or start new activities. The Cosmos Information Service also provides browsing access to an X.500 directory service.

The user performs group working tasks involving the explicit creation and communication of objects relevant to joint tasks with the support of the Script Interpreter. Figure 3 illustrates how the script interpreter enables a user to play a role in an activity. The script interpreter works in an environment that include the script for the role and provides the other data objects relevant to the current state of the activity. These include a queue of incoming messages, a database of messages already received in the activity, the position in the script, the other role players. The user is presented with some options suggested by the script and when the user selects an operation, the interpreter performs the operation. This results in changes to the user’s environment, for example, messages may be received from the user’s queue or sent to another role player.
Figure 3. The Script Interpreter interacts with the user

All of the communications between the people involved are sent via an X400 message delivery service [CCITT/ISO 88a].

The Cosmos system is built from a number of software modules including the Cosmos Information Service, the Script Interpreter and the Message Delivery Service [Araujo et al. 88]. These modules are supported by a name mapping service and an object server [Dollimore 88]. The object server provides a common storage module used by all the other modules to store the entities mentioned above. This server includes support for transactions, thereby enabling the other modules to share the entities in a controlled manner.

If users at different hosts participate in an activity then the shared activity data must be available to all of them. When message hosts are connected by a wide area network, it is not practical for users to access data from another message host. Instead of this, the data is replicated at the user’s local host.

3 Progress so far

3.1 State of the Cosmos prototype

We have a working prototype comprising a user interface to the Script Interpreter and Cosmos Information Service with supporting software including the object server and message delivery service.

Members of the Cosmos project team have been using this system - for example, we have an "expert advice" structure instantiated so that users of the initial text-based user interface can send helpful comments to the designer. We have also completed two trials, each with about a dozen
users; the first was a "brainstorming" activity; the second was a "structure defining activity" and each activity was based on a set of scripts designed for the purpose. Although personal mail may be sent to users at other sites, we are not yet able to run activities that involve users at several sites.

The text based user interface was used in both of these trials as no graphical user interface was ready at the time and it enables access to the system via a modem. The user interface design relates strongly to the users conceptual model for group working and is supported by the semantic level, but it has not yet taken on board the new ideas related to activity control and defining structures.

The programming of direct manipulation user interfaces has proved to be a very difficult task and progress has been disappointing. Both graphical user interfaces now include access to most of the group working aspects of Cosmos, but neither has attempted to implement browsing in the Cosmos Information Service.

3.2 Configurability by ordinary users

Our aim was to provide a system that could be configured by all of its users. It is true that we have a configurable system, in that new communication structures may be defined as they are needed.

In order to define a new communication structure, we propose several steps: users do a task analysis of their activity and write an informal definition of its procedures. Following this, they define the communication structure either in SDL or in Script Language. If SDL is chosen, the definition is translated into Script Language. The Script Language definition is augmented and installed in the system as described in the previous section.

The method of defining communication structures is unfortunately not very accessible to an ordinary user and requires the help of structure builders. Although there has been discussion in the project of graphical user interfaces for structure definition and activity control, we are far from the sort of configurability by users that is available with Hypercard – the Macintosh hypertext tool.

The Hypercard example is an interesting one because it shows that ordinary users can make use of a simple programming language – Hypertalk – that is not unlike our Script Language. Perhaps we could come nearer to our aim of configurability by the ordinary user if we provided editable scripts easily accessible in the user interface instead of attempting to design a graphical Structure Editor.

Another related problem is the requirement to make progressive modifications to the definitions of the scripts of an activity whilst it is in progress. One factor that seems to contribute to our failure in this respect is that we had the idea of editing a communication structure or a set of scripts - a very centralised approach. In order to give power to the individual user to build their own tools [Ehn 88], we require a decentralised approach in which a user edits their own script. Referring again to the Hypercard model, we envisage a user adding a message to their script and
the receiving script reporting an exception and displaying the script to the recipient who might insert the necessary receive instruction.

A factor that induced the designers of the Script Language to think in a centralised manner is the knowledge that parallel programs require careful design to ensure the correct synchronisation and to avoid problems of deadlock. Although we planned that scripts should be interpreted independently, we were unable to extend this idea to editing them.

3.3 Designer's and User's Conceptual models

Our design model is similar to but not identical with the user's conceptual model. The main difference is that the design model does not include item boxes and scripted actions. The two models were developed at the same time and it may have been more helpful to converge on an identical model.

The reason for the divergence is that entities such as item boxes are handled directly by the script interpreter from its own private environment and the entire environment is saved as an uninterpreted chunk. We have decided to keep this arrangement for reasons of performance.

The use of the same model for designers and users would enable the programmers to have the same conceptual model of the system as the users will have when the system is built. This would also be useful to enable new programmers to understand the system for future maintenance and extension.

3.4 Specification of Software Modules

We decided to design and implement each of the modules (e.g. Cosmos Information Service, Object Server) as a separate service (in the distributed systems sense) with a well defined interface. These interfaces were defined in the draft specification and have changed very little.

Each server is implemented as a separate process and could run in a separate computer. Modules communicate only via remote procedure calling. We have found this an excellent starting point for development of software modules by different programmers. It has the advantage that each module can be sure of the services of the other modules and that there are no side effects between modules.

We were aware of the advantages of the prototyping approach in the incremental development of software and have been able to use it in a limited way within each module and to extend the services provided by a module. We have not discovered how to use the prototyping approach between two modules, although it has been used in development of the user interface - which has no clients!

3.5 The relevance of persistent programming

We are using an object server to store the long-lived data representing the entities in the model of the system. This achieves its objective in providing a shared storage module accessible to all the other software modules. The transactions it supports seem to provide synchronisation of
shared access adequate for the purpose. It also provides a simple mechanism for mapping
persistent data between permanent storage and programs, and for replicating data where it was
required.

However, the object server is only a storage module and provides no application specific
semantics for its objects. The latter are provided by means of header files defining the attributes
and library procedures defining the behaviour of each object. The data handled in these
procedures comprising an object requires flattening or unflattening each time it is transferred
between the object server and the program. The use of a language such as Argus [Liskov 88] that
supports both the definition of abstract data types and reliable atomic storage of their instances
would simplify the work.

In particular, it would have provided a basis for sharing a model between the script interpreter
and the rest of the system. This in turn would provide easier access to the model from the semantic
level and facilitate the building of the user interface and a more natural approach to defining and
editing communication structures.

4 Suggestions for Further Research

The issue of configurability and the need for more work in this area have already been discussed
in 3.2. Another area which has been flagged for attention, but never satisfactorily resolved, is
the requirement within many group activities for communication peripheral to the main task.
That is, informal discussion and information exchange that take place around the main task,
whose exact patterns cannot be predicted, but which are nevertheless an important aspect of group
communication. The expression of this requirement within a communication structure is difficult
to achieve.

The whole area of designing useful structures is one where more experience is needed, in order to
develop methods and design approaches to aid the process. In some situations, a group task has a
well-established procedure that makes structure definition relatively straightforward; in other
cases, it may be difficult to decide on activity boundaries (which roles should be included, what
actions start and end the activity, etc), and to resolve inter-dependencies among activities. Our
experience in this area is limited so far, and there is considerable scope for investigation of the
whole area of structure definition.

From the systems perspective, we have built the prototype described earlier, but have not
addressed the problem of integration with other applications. In a real-world situation, users
would need to transfer data and documents between Cosmos and their favourite day-to-day
applications. Our architectural framework was designed to make this possible, but the required
mechanisms have not been implemented.

Finally, we believe that our aim of providing a system to support structured group-working has
been realised, but that there is still work to be done in meeting the full requirements of cooperating
groups. In this section we have outlined some of the areas where further work is required.
Acknowledgments

We would like to thank our colleagues in the Cosmos team.

In particular George Coulouris has provided continuous support and criticism - he suggested many innovations such as having an object server, a layered architecture and using Sun NeWS and MacApp for the user interface development. Hugh Smith has been a strong influence in the design of the software architecture and has provided the way towards integration with the X400 and X500 standards. John Bowers and John Churcher's SDL has provided us with an alternative approach to communication structures against which we could measure our own efforts in designing the Script language. Tony Rubin was a source of many early ideas in the user interface design and also was a lone voice stressing at an early stage the importance of ordinary users being able to define communication structures. Paul Buckley's work on the user's conceptual model has been invaluable in most levels of the design and particularly in the user interface.

Tim Roberts has designed and implemented the script interpreter and the semantic level of the user interface, Regina d'Araujo designed the Cosmos Information Service and Alan Shepard has integrated it with the X500 directory service. Julian Onions implemented the X400 message delivery service. Stathis Gikas, Andy Hockley and Bruce Clark designed and implemented the various forms of the user interface. Doug Steel implemented the replication service. Finally, Robert Young has provided invaluable support in assisting with technical coordination of the work, and through contributions to research on group task management.

References


[Dollimore et al. 88]: Dollimore, Roberts and Coulouris, Towards a Language for Defining Structure in Message-based Cooperative Working, Cosmos report 69.1, March 1989


[Stallman 81]: Stallman RM, EMACS, The extensible, customisable, self documenting display editor, Proc ACM Sigplan Sigoa, Symposium on text manipulation, Portland Oregon 81.
