THE ACTIVITY MODEL ENVIRONMENT: AN OBJECT-ORIENTED FRAMEWORK FOR DESCRIBING ORGANISATIONAL COMMUNICATION

H.T. Smith, P.A. Hennessy, G.A. Lunt
Communications Research Group,
Department of Computer Science,
Nottingham University
Nottingham, NG7 2RD, England.

Abstract

This paper outlines a framework for modelling organisational communication. It also describes an object oriented environment (the AME) that has been used to explore such models. The AME consists of a database that holds a description of the structure of an organisation and its on-going activities. A rule interpreter makes use of the database to assist in the processing of activity related messages and their routing between organisational roles.

1. Introduction

Over the last decade there has been considerable research interest in co-operative working, much of it driven by the increasingly sophisticated facilities offered by computer systems and networks. Recently this work has focused on the development of abstract frameworks that can be used to describe recurring patterns of group communication. The background of such research has been varied, and the resulting frameworks approach the problem from a wide range of viewpoints (e.g. [Smit89] [Pank89] [Bowe88]). A comparison of several approaches can be found in [Henn89].

The MacAll project [Smit87] resulted in the outline of one such framework, a model of *organisational structure* which included objects such as departments, people, roles, workspaces, file cabinets, folders, and documents. The model was focused on organisational roles, each person in the organisation being bound to one or more roles, each role being performed in a distinct workspace.

This paper describes the follow-on project (MacAll II) which incorporated formal descriptions of organisational knowledge and patterns of communication into the original framework. This was achieved by adding the following components to the framework: activities, messages, information units (iunits), rules, and functions.

A prototype object oriented tool – the Activity Model Environment (AME) – has been developed to explore models of organisations based on this framework. The AME prototype consists of a database and an associated rule-based formalism for

---

The work described in this paper was supported by Digital.

representing activities and organisational states. Users interact with the organisational model embedded in the AME by creating and playing roles. Activity related communication proceeds via the exchange of messages between roles, where messages are 'persistent' objects.

This paper is structured as follows: Section 2 gives a description of the framework as it is realised in the AME prototype. Section 3 gives an example of an activity that has been modelled in the AME prototype. Section 4 outlines the implementation of the AME prototype. Section 5 concludes the paper by summarising important modelling and implementation issues, and identifying areas for future research.

2. The conceptual framework

The conceptual framework encompasses both the structural elements of an organisation, and activities that these elements may be involved in. The MacAll I framework provided the basis for the structural aspects of the model, and the procedural aspects (i.e. specification of activities) are loosely based on work undertaken by the AMIGO Advanced group [Pan89].

Eight components of the framework can be identified:

- **Activities** – specified sets of tasks that are performed by groupings of role instances in order to achieve a common goal.
- **People** – placeholders that represent the actual individuals working within the organisation, and their capabilities.
- **Roles** – specifications of the responsibilities and duties assumed by any person that plays the role.
- **Workspaces** – conceptual work areas containing resources associated with particular roles.
- **Messages** – persistent objects that flow between the role instances associated with an activity.
- **Information units** (iunits) – 'atomic' units of information that are used in building messages and other objects.
- **Rules** – used to constrain the behaviour of other components.
- **Functions** – operations carried out by roles and messages as part of an activity.

In the AME prototype, roles, people, workspaces, iunits and messages are all represented as objects. Thus, particular role *instances* combine to take part in a given activity. General and organisation-specific definitions of these component classes are stored in the *Organisational Manual* – a database which acts as a reference both for users of the AME, and for the AME itself.

The remainder of this section describes the components and their interrelationship as they are represented in the AME prototype.

2.1 *Activities*

These define the basic work processes of an organisation. The undertaking of a specific organisational activity results in the performance of a series of tasks by a group of roles in order to bring about a particular state of the world. The formal description of an activity details the start-state and end-state(s) as well as the roles and messages that will be involved in its performance. The description lists the duties of each role and potentially the sequence of actions that the role instance is expected to undertake. However, an explicit role co-ordination 'script' is not included in this component; co-ordination is left to the role instances.

This approach to the modelling of activities has been described as a *role-oriented* approach. It contrasts with a *mediator-oriented* approach, where the activity is represented by an event script which an autonomous agent uses to direct communication – see [Pank89]. Our major concern was to vest responsibility for the performance of an activity with the role(s) involved in taking part in it. The rationale was that many organisational co-ordination activities either do not yet admit of a precise formal description or, for reasons of maintainability/flexibility, choose not to support such a description. It therefore makes sense to associate the responsibility for activity co-ordination with specific role instances rather than abstract entities. Thus, users ultimately have to be responsible for making decisions concerning the performance or non-performance of parts of an AME activity.

2.2 *People*

Each person in the organisation being modelled has a person object entry in the organisational manual. This specifies the organisational roles that they are allowed to play (each person has a 'personal' role in addition to those allocated for organisational activities).

An AME user interacts with the organisational model by playing a role. This is achieved by creating an instance of the appropriate role class and binding that instance to the person. One person may be associated with several different role instances at any one time.

2.3 *Roles*

A role defines a group of duties and responsibilities that may be taken on by one or more people. Roles typically correspond to particular job titles within organisations, e.g. programmer, manager, secretary. For a given organisation, the set of role classes used is defined by the contents of the organisational manual. Each class definition includes a set of generic role rules and also sets of activity-specific rules.

A role is undertaken by a role instance, which consists of the person instance playing the role, the set of role rules, and a role agent. The role agent interprets the role rules under the direction of the person instance. The person instance playing a role may set a *division of responsibility* (DOR) for each activity that a role instance is participating in; this determines the extent of the role agent's use of the

role rules in the performance of the role. For example, a secretary role may allow the role agent to authorise stationery requests normally, but when budgets are low the person performing the role may wish to authorise requests on merit.

In the present prototype, the DOR is associated with an integer value. All rules that are invoked by role instances have a number associated with them; if this number is less than the DOR of the current ruleset it is the role agent's responsibility to perform the required action(s) – see section three.

2.4 Workspaces

A workspace is a (conceptual) work area that is associated with *one* particular role class, and is used by all instances of that class (i.e. all the people playing that role). It contains three message handling resources: an in-tray, an in-progress tray, and an out-tray; and other resources (e.g. tools and services) including filing cabinets for the storage of iunits and completed messages. Typically, use of workspace resources and stored information is restricted to instances of the associated role. However, subject to specified constraints, the contents of the workspace may be viewed by instances of other roles.

The in-tray is where a message arrives at the workspace. Once a message arrives, the appropriate role instances are notified and asked to process the message. A message may be directed at all instances of a role or to one particular instance (i.e. person), in the latter case only that role instance is notified. When a message is being processed it is transferred to the in-progress tray, which effectively locks the message and prevents other role instances from processing it.

2.5 Messages

A message exists for the purpose of collecting information associated with activities and transferring it between roles. Unlike the world of electronic mail, messages exist for the lifetime of an activity (during this time they may be passed around and processed by many roles). Several standard types of messages are defined in the organisational manual, (e.g. memos, notices, forms) and other types may be added. A message consists of one or more component iunits and a set of rules that are to be invoked when the message is complete, that is when a given set of iunits have been processed.

2.6 Iunits

Iunits are 'atomic' information objects. An iunit has a name, one or more fields, and a set of completion rules. Fields within an iunit are typed, and must be filled with values of the given type. (The iunit is typically structured as a group of *role related* fields.)

Message forms in the AME prototype are composed from groups of iunits, although other objects (e.g. documents) can also be composed of iunits. In principle, the iunits that are 'transferred' in a message body may be accessed concurrently or shared by other messages. In practice, the nature of the activity in which an iunit is used will determine whether shared access is allowed.

On completion of an iunit, its completion ruleset is invoked. The ruleset verifies the contents, determines the next iunit for processing and directs the parent message to the appropriate role. (The iunit may determine that the message is complete (i.e. no further iunits need to be filled in) in which case it informs the message (see 2.6).

2.7 Rules

The rule components of the organisational model define and constrain the behaviour of other components (specifically roles, messages, and iunits) under particular conditions. A rule has a label, an antecedent part consisting of zero or more conditions, and a consequent part consisting of one or more actions.

Sets of rules (rulesets) are invoked at specified times during the lifetime of an object instance (e.g. when a role instance is notified of the arrival of a message). When a ruleset is invoked, the consequent part of each rule is evaluated. The actions in the consequent part of the rule are performed if the antecedent part evaluates to "true". (If there are zero conditions the default evaluation is 'true'.)

A ruleset consists of a list of rule names. The rules themselves are not pre-loaded when the object ruleset is created but are retrieved from the organisational manual just prior to evaluation ('late binding').

2.8 Functions

Functions are the operations performed within the group communication processes (e.g. instantiate-message, fill-field, send-selector). They are transactionally atomic (in the sense that they cannot be partially performed), and they must be executed entirely by one processor (either a role instance or the role agent).

2.9 Relations between components of the organisational model

Figure 1 provides an illustration of the components and their inter-relationships. The organisational model consists of the static description held in the organisational manual and the (dynamic) set of on-going activities. One of the activities in the figure is 'expanded' to show its components. Two types of role are involved in this activity, one of the roles is being performed by *two role instances.* A different person is associated with each of these three role instances.

A role instance contains groups of rules that govern its behaviour with respect to different activities. The person associated with the instance may allow a role agent to execute the rules, or may undertake them personally.

The workspace for each role is shared by all its instances and contains resources and stored information objects. One of the workspaces in the diagram contains messages. Messages consist of a series of iunits and a completion ruleset. As shown in the diagram, iunits consist of a set of fields and a completion ruleset.

It should be noted that components involved in the performance of this activity may also be involved in other activities within the organisation.

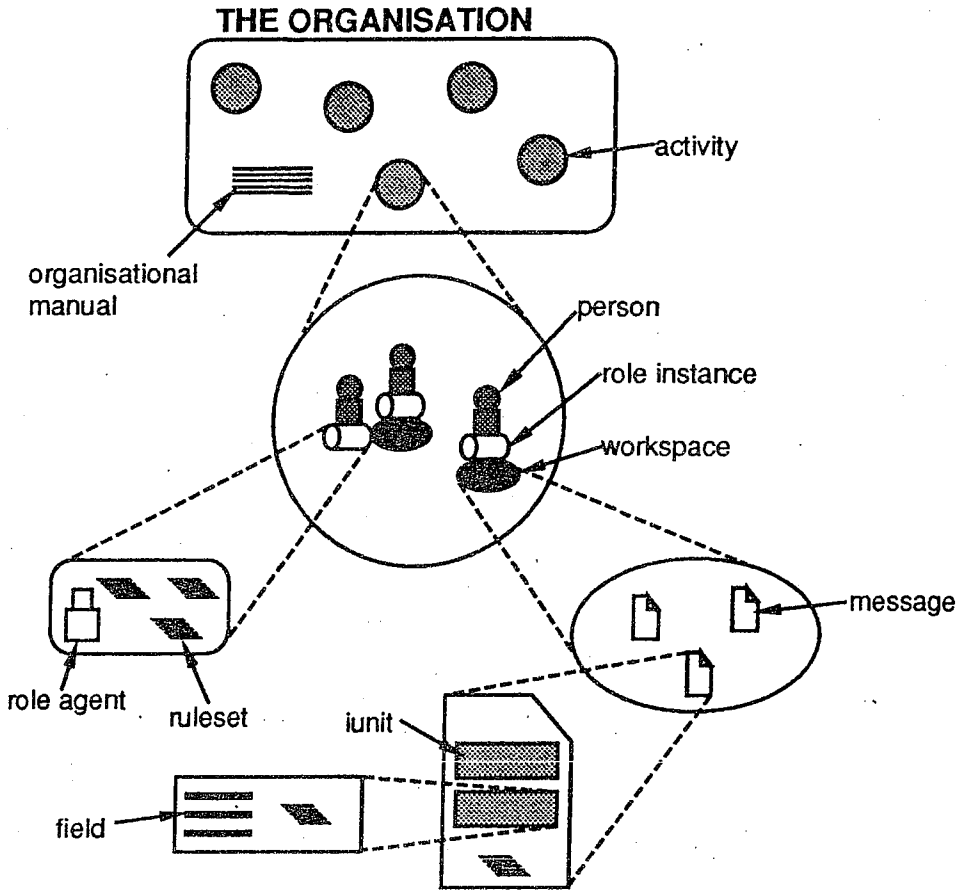

Figure 1: An overview of the AME components

The next section gives an example of the use of the AME and provides more details of its operation.

3. An Example

In order to provide an overview of how the above components interact within the AME, a model of part of a typical company activity is described – travel authorisation. The part of the activity that the example deals with is the processing of a Travel Requisition Form. The form is used by a would-be traveller to describe the purpose of the trip and it is circulated to various authoriser roles for 'signing-off'. The actual number of roles involved varies depending upon the location and cost of the trip. The AME sequence of events for part of this activity is shown in figure 2.

The numbers in the figure are keys to the textual explanations below. The starting point is that the traveller, John, has obtained informal permission for his trip. However, there are a fair number of forms which he must fill in before he can

actually go on the trip. He therefore asks the departmental secretary to start the processing of the relevant forms, including the Travel Requisition Form (TRF) message.

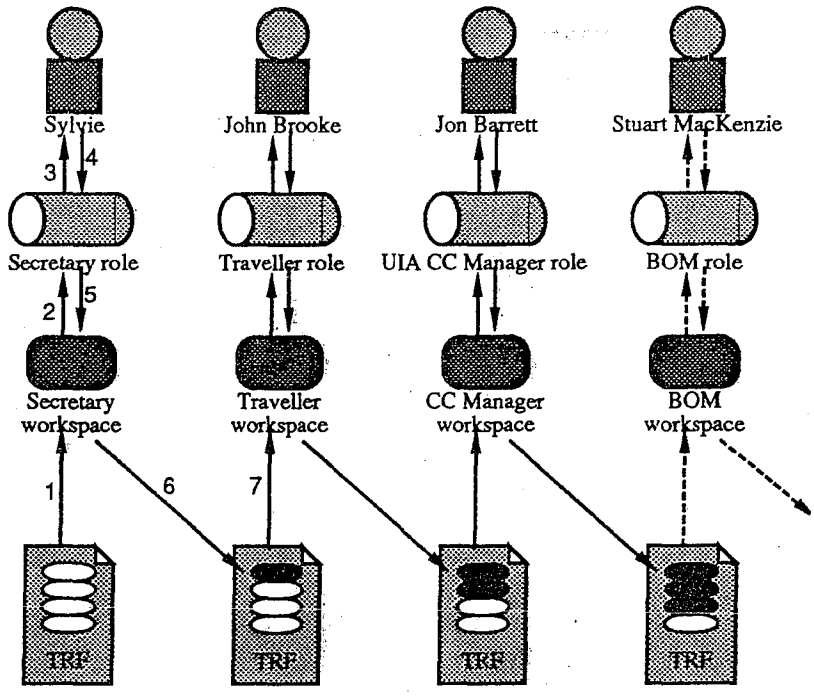

Figure 2: Part of the travel authorisation procedure

1. *The secretary, Sylvie, creates a new TRF message.*

The TRF message is created using a template from the organisational manual, thus ensuring that the form is as up-to-date as possible (e.g. it does not come from a pile of forms Sylvie has had since last December). As the message is created, it in turn creates its constituent iunits and rulesets; once again from templates contained within the organisational manual. When the message has been created, it records information connected with its creation; which person created it (Sylvie), which role they were playing (Secretary), and the time at which it was created. It also marks the first iunit in the message as the *current* iunit (i.e. the one that should be processed next). The message then routes itself to the workspace associated with the role that created it.

2. *The workspace notifies the role instances of the arrival of the message*

The Secretary workspace maintains a list of all the Secretary role instances currently being performed. When a message arrives in its in-tray, the workspace tries to determine which person should deal with it. Optionally, the message has an attribute called *Preferred Instance* which in this case is set to the value Sylvie. (This is a sensible default as Sylvie has just created the message.) The workspace

then informs the Secretary role instance that Sylvie is associated with that a message is awaiting attention.

If, for some reason, Sylvie had stopped playing the Secretary role when the message arrived, the workspace, upon detecting that the message's *Preferred Instance* was not available to deal with the message, would then notify all other instances of the role. For simplicity, the diagram does not show more than one role instance, though others may exist.

The rulesets within the Secretary role instance are executed in order to determine if it can be processed. At this point the role instance moves the message from the in-tray into the pending-tray to indicate that it is being dealt with.

3. *The Secretary role determines if it can deal with the message.*

By default, the person performing the role (Sylvie) would fill-in the background travel details in the current TRF iunit. However, depending on the division of responsibility set on role instance rulesets, some of the fields may be processed automatically by the role agent.

If the role agent requires the user to deal with (part of) the iunit, Sylvie is requested to fill-in fields of the current iunit.

4. & 5. *The person processes the required iunit*

Sylvie fills-in the appropriate fields and then notifies the message that the iunit is complete. The message places itself in the out-tray and informs the current iunit that it is complete.

6. *The iunit checks its fields and determines further actions.*

The iunit fires its completion rulesets which generally check the validity of the field values, if necessary put in any default values, then determine which will be the new 'current' iunit and which role should process it (Traveller). The message then routes itself to the relevant workspace.

A similar sequence of events now occurs with the traveller supplying details of the trip, and the TRF message is then passed on to the first of the authoriser roles.

The whole process is then repeated for each of the other roles that need to be queried for information before the TRF message can be considered complete. (The number of roles involved will vary depending upon the travel details, or the state of the budget, and will be determined by the appropriate rulesets.)

At some point an iunit completion ruleset will determine that the message is complete and will tell the TRF message to fire its completion rulesets. This will result in the Traveller receiving notification of trip acceptance/rejection and a copy of the message being added to the secretary role archives.

A number of details have been omitted from this example in the interests of brevity. However, it should give the reader an understanding of how organisational activities are modelled. The next section discusses how the AME prototype that modelled this activity was implemented.

4. AME Implementation Issues

Early in the project it was decided to base the architecture on *object oriented* principles. The use of an object oriented approach allowed the implementation to be both elegant and adaptable.

Initial prototyping work was carried out in the Poplog Flavours environment. However, the project requirements dictated that the prototype did not run in an embedded environment. Accordingly, the final prototype was implemented in C++. An important factor in the choice of C++ was the availability of an object class library called OOPS [Gorl88]. These classes are loosely modelled on those provided within the Smalltalk-80 language.

The components that make up the model map almost directly into object classes. An important part of the implementation is concerned with the design of the constructors for these classes.

4.1 Constructors

The constructors for all AME classes have the same basic functions:

- Read and parse a description file.
- Create any other necessary objects.
- Assign the object a unique identifier.
- Register the object with any objects that need to know of its existence.
- Register the object with the global dictionary of all objects.

After the description file has been parsed it may be necessary for the constructor to make instances of other classes within the model. For example, the message constructor must create the appropriate rulesets and iunits as described in the description file – see figure 3.

The diagram shows how complex initialisation of a given instance may be. A message must create instances of rulesets and iunits, and iunits in turn will need to create further instances of rulesets. (The corollary is that the deletion of instances by destructors is quite complicated.) The rulesets (AME Class) that are associated with the message are stored in a dictionary (OOPS Class).

Every object in the diagram that is actually attached to the message object, e.g. type, comp_rulesets, is actually an instance variable and its constructor is called as part of the message constructor. After reading its description file the message constructor assigns values to the String instances and creates the required objects (Rulesets and Iunits) and inserts them into the appropriate OOPS variable.

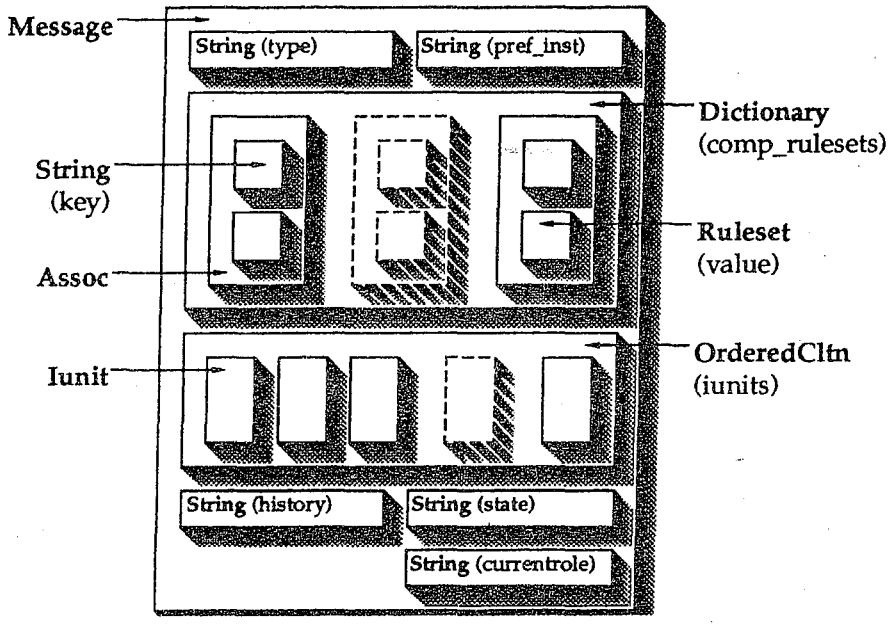

Figure 3. Object instances within a message instance.

4.2 Identifiers

A major goal of the AME was to provide facilities to browse the current state of the organisational model. This requires a simple mechanism for listing and displaying all the currently active objects in the system. However, the name of an instance cannot be guaranteed to be unique as, for example, it is possible to have more than one instance of a TRF form active in the system at any one time. Each instance therefore needs to be assigned a unique identifier.

The command interface uses these identifiers to access the various objects. The unique identifier is not absolutely necessary for all classes as the name of some objects can be guaranteed to be unique – e.g. there can only be one workspace with a given name.

Registration with Interested Parties

When an instance is created, various other instances already in the system need to be informed. For example, when a new role is created it needs to *register* with the workspace associated with that role (so that the new role can be informed when there are messages to be dealt with).

Therefore the constructor needs to know where to register each instance it creates.

Registration with the Global Dictionary

Each instance of an object that is created must also be registered with the global dictionary. This enables two important actions to be carried out:

- It enables the user, via the Command Interface, to list and inspect all objects within the model simply and easily.
- It enables the functions of various classes similar access to all the components within the system. For example, when a role registers with a workspace the role constructor must find the workspace instance with the name specified in the role description file.

The Global Dictionary is actually a Dictionary of Dictionaries. The top level dictionary consists of a series of strings which are the keys to other dictionaries. There is a separate dictionary for each of the AME Classes. The majority of Class specific Dictionaries are indexed on the identifier of the instance they contain.

4.3 Command Interface

The AME prototype does not contain a WYSIWYG visual interface. Rather, it was envisaged that the prototype should provide a simple generic command interface that could have direct manipulation interfaces layered on top of it. Therefore AME provides a simple ASCII based command line interface that allows the user, or an interface component, to communicate with the model.

The interface commands can be divided into three categories:

- browsing – commands issued by the user to interrogate the state of the model and components within the model.
- updating – commands that the user uses to update the state of the model.
- AME generated – the AME needs to inform the user of any necessary changes in state e.g. an Iunit needs completing.

An example of the use of one of the browsing commands is given in the appendix 1.

Each command needs to be preceded by a user signature, and the role he/she is currently playing, to the model. This is necessary for two reasons:

- identification – e.g. to ensure that the person taking on a new role is allowed to perform that role.
- to allow multiple users to access the model.

A signature takes the form [Person, Role]. For example:

[Sylvie Harris, UIA Secretary] new role "Traveller"

may generate the response

RESPONSE: *[Sylvie Harris, UIA Secretary]*

ROLE: "Traveller" 21 "Sylvie Harris"

ENDRESPONSE:

A sophisticated UI component may maintain separate windows for each role a person is performing and thus be able to update the appropriate window. Alternatively, a 'mediator' component may route the message to one of many other UI components representing different users.

5. Conclusion

This paper has described the development of a framework for modelling organisational processes and a prototype execution environment – the AME.

There are several key features of the framework. First, the responsibility for the co-ordination and management of an activity is vested in the roles involved in its performance. Second, roles are distinguished from people, thus allowing a separation of responsibilities between a role and the person playing it. Third, *persistent* messages provide the basic mechanism for collecting and transferring activity information between roles. Finally, the concept of *information sharing* (see [Henn89b]) has been addressed by (a) allowing role instances to share access to objects in workspaces, and (b) constructing all complex information structures from iunits.

In summary, the development of the AME prototype represents a first attempt at dealing with the difficult problem of modelling group communication processes in organisations. There are still many high level issues that need to be addressed. For example, the requirement for a meta-language for activity management (e.g. see [Benf89]) and an associated set of tools.

Furthermore, to conduct more realistic modelling of organisational activities, the system needs to be re-implemented in a truly distributed environment. Tentative plans have been made to do this using a commercial object oriented database which offers many of the necessary facilities.

Acknowledgements

The authors gratefully acknowledge the contribution to this work made by two groups from Digital, John Brooke and Jon Barrett of UIA-A/D and Paul Jacobs and Scott Bowie of IOSG. In addition, thanks are due to Alan Shepherd of the Communications Research Group at Nottingham who implemented the AME prototype rule interpreter.

References

[Benf89] S. Benford, Requirements of Activity Management, *The 1st European Conference on Computer Supported Cooperative Work*, 1989

[Bowe88] J. Bowers, J. Churcher, & T. Roberts, Structuring Computer-mediated Communication in COSMOS. In R. Speth (editor), *EUTECO '88 - Research into Networks and Distributed Applications*, pages 195-210, 1988.

[Gorl88] K. Gorlen, OOPS (Object Oriented Programming Support). Computer Systems Laboratory, National Institutes of Health, Maryland USA.

[Henn89] P. Hennessy, S. Benford, & J. Bowers, Modelling Group Communication Systems: Analysing four European projects, *Singapore International Conference on Networks*, pages 56-61, 1989.

[Henn89b] P. Hennessy, Information Domains in CSCW, *The 1st European Conference on Computer Supported Cooperative Work*, 1989

[Pank89] U. Pankoke-Babatz, *Computer Based Group Communication: The AMIGO Activity Model*. Ellis Horwood, 1989.

[Smit87] H.T. Smith, G. Lunt, D. Young & J. Lawrence, *The MacAll Project*. Communication Research Group, Nottingham University, 1987.

[Smit89] H.T. Smith, J.P. Onions, & S.D. Benford (editors), *Distributed Group Communication: The AMIGO Information Model*. Ellis Horwood, 1989.

Appendix 1

The following details the syntax of one of the browsing commands – Display

Syntax **display *type id***

Means Display the Object of *type* with *id*. Shallow descriptions of objects contained within the displayed object will be used – e.g., the description of a Ruleset within an Iunit will not be displayed. The *id* is an integer (the creation id). However, for the *types* Workspace and Person, it is permissible to use the "name".

Example display message 11; display workspace "UIA Secretary"

The first example may generate the output:

```
MESSAGE:
NAME:                   TRF
VERSION:                1.0
DESCRIPTION:
                        The traveller's request form
ENDDESCRIPTION:
CREATION TIME:          19-OCT-88  11:53:00 am
ID:                     11
TYPE:                   Form
STATE:                  INCOMPLETE
CURRENTROLE:            UIA Secretary
PREFEREDINSTANCE:       Sylvie Harris
INSTANTIATOR:           Sylvie Harris
INSTANTIATINGROLE:      UIA Secretary
IUNITS:
        IUNIT:          "Inst"  48  "ACTIVE"
        IUNIT:          "TRF Details"  50  "INACTIVE"
        IUNIT:          "CC Signoff"  52  "INACTIVE"
        IUNIT:          "BOM Signoff"  54  "INACTIVE"
ENDIUNITS:
RULESETS:
                        "message complete"  56
ENDRULESETS:
ENDMESSAGE:
```

Note that the components within the example Message e.g. Iunits, Rulesets, are not expanded; only a short version is provided. This cuts down on the amount of information passed to the interface. If further information is required about a component, another Display command is required. This aids the implementation of a hypertext-like browsing tool.