Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work, September 10–14, Stockholm, Sweden
H. Marmolin, Y. Sundblad, and K. Schmidt (Editors)

Medium versus mechanism: Supporting collaboration through customisation

Richard Bentley
German National Research Centre for Computer Science (GMD FIT.CSCW)

Paul Dourish
Rank Xerox Research Centre, Cambridge Laboratory (EuroPARC) and Department of Computer Science, University College, London

Abstract The study of cooperative work as a socially-situated activity has led to a focus on providing 'mechanisms' that more closely resonate with existing work practice. In this paper we challenge this approach and suggest the flexibly organised nature of work is better supported when systems provide a 'medium' which can be tailored to suit each participant's needs and organised around the detail of their work. This orientation towards 'medium' rather than 'mechanism' has consequences for cooperative system design, highlighting a need to allow participants to adapt details of policy currently embedded in the heart of the systems we build We describe an approach which allows users to perform such 'deep customisation' through direct manipulation of user interface representations

Introduction

A principal tenet of CSCW is that systems intended to support cooperative work should be sensitive to the context in which they will be used. A number of recent studies of cooperative work have highlighted a disparity between descriptions of the working context as given in job description manuals, process-oriented accounts, and so on, and the *actual* pattern of use, which is often far more flexible and contingent on particular knowledge, skills and local decision making (see Anderson *et al.* 1989, for example). These observations have led to general agreement across the CSCW community on the desirability of a better understanding of the way in which work is carried out in groups, in order to inform the development of more effective cooperative systems.

One interpretation of this is a need for better representations and models of group work to enable development of systems that resonate more closely with actual work practice. Models such as the Milan Conversation Model (De Michelis and Grasso 1994), for example, are intended to provide representations of cooperation at a level of detail suitable for the needs of system design. However there are a number of problems with this approach, as a gulf exists between the kinds of information required by system designers and that provided by methods of analysis that purport to capture actual working context (Sommerville *et al.* 1994). It has been suggested that this gulf can be overcome by adapting methods of analysis such as ethnography to better meet the needs of designers (Hughes *et al.* 1994); however others contend that by adapting methods in this way, many of the benefits in terms of provision of contextualised, situated accounts are lost in the compromise (Anderson 1994).

We believe that these modelling problems, and the resulting debates, are artefacts of an inappropriate focus of design activity and reflect a narrow interpretation of the notion of 'computer support' for cooperative work. In this paper we are concerned with an alternative approach; one which does not regard the creation of ever more intricate and more detailed representations of group work as the main route towards more effective cooperative systems. Rather, our approach promotes the view of a cooperative system as one whose behaviour can be adapted through high-level customisation to meet the needs of its users, and that effective 'support' arises from precisely this openness and flexibility. We characterise this distinction as a contrast between systems which provide 'mechanisms' to structure collaboration, and those that provide a 'medium' which can be shaped by the users rather than the technology, in and through which collaboration occurs.

The context for this discussion is a project at GMD which is developing a simple shared workspace system to provide basic facilities for cooperative work across the Internet. Our focus in this paper is the development of the client interface which allows users to cooperate through a representation of a shared space, and more specifically on the general issues which underpin its design. Motivated by the different requirements that users may have for interacting with a shared workspace, we have developed a client prototype which supports customisation of underlying system behaviour through high-level user interface manipulation.

The rest of the paper is structured as follows. The next section presents in more detail our distinction between a system-as-medium and system-as-mechanism, and suggests a need for participants to configure systems to meet their requirements for support. We then show how this configuration requires a much 'deeper' notion of customisation than is evident with most tailorable systems, and the problems that this poses in terms of the 'language' with which customisation requirements are expressed. We describe a possible solution that allows participants to express their requirements *incrementally* using high-level customisation operations, and illustrate this approach with examples from our shared workspace client. We conclude by highlighting the general implications of our approach for the development of CSCW system architectures.

Medium versus mechanism

The distinction between medium and mechanism is concerned with the notion of 'support' a system provides for cooperative work, and specifically how the support required for individual participants and activities is derived. Although the concept of a system-as-mechanism has previously been equated with systems that are built around some social model of interaction (Greenberg 1991), we argue that the distinction is rather concerned with how such models structure the activities of the users, both *within* and *around* the system. This is reflected in the extent to which systems can be 're-purposed' to support activities that they were not designed for, and to provide support in a manner not envisaged by their designers.

System-as-medium and system-as-mechanism

A good example of a system that provides a medium for cooperation rather than a mechanism is electronic mail. Email applications are (by far) the most successful CSCW technologies developed to date, despite (or perhaps because of) the fact that they are amongst the simplest to design and implement. A characteristic of email use reported in a number of studies is the extent to which email supports different users' requirements, including the kind of information sent and received, frequency of reading email, different methods of notification of new mail and so on. Sproull and Kiesler (1991), for example, discussing the introduction of email systems into large organisations, show how users adapted the technology to their particular needs and exploited the opportunities which it introduced in ways not envisaged by management. So how does an email system provide support for such a wide range of potential activities and behaviours?

Consider the ways that email systems can be configured to save copies of your mail. For incoming mail, most mail programs provide options for saving mail in a special folder or file, storing in a default file for mail which has been read, leaving it in the 'spool' file and dumping it to the printer. In the same way most mailers offer a number of options for saving copies of outgoing mail, such as storing in a file, carbon copying to your spool file and so on. The decision to file a copy of a mail message, and which method to use, is contingent on many factors including (and this is just a small sample), who the mail is from (or to), what it is about, your attitude to email communication - whether it is communication 'in writing' or more 'verbal' - and even such issues as the trust you have in the system's reliability.

Email provides a *medium* for action because it can be adapted to support participants' specific requirements, contingent on any of the above factors and more besides. It provides a framework within which activity can take place, rather than structuring activities themselves. Whether to save a copy of a mail and which option to use are decisions for the user alone; the system does not attempt to determine what should be done with each mail message, but simply does what it is told. It is hard to imagine how another arrangement could be successful—after all, how could an email program know that I will find a particular message humorous and save it to my 'funnies' folder?

This may not seem so startling an observation, but it highlights a key issue to do with the extent and use of *representations* in computer systems in general and in cooperative systems in particular. An electronic mail system has no representation of the activities it is supporting, how the users' interactions relate to those activities, or what the requirements of those activities might be, but provides a service which can be adapted to the needs of each participant. We will return to this question of computational representations later in this section.

The alternative to providing an adaptable medium is characterised by a focus on providing mechanisms which directly regulate and manage participants' activity. In theory, participants are then free to concentrate on aspects of the task which require their attention, and the responsibility for the accomplishment of work is shared by the participants and the computer system. The key to this division of labour is the formalised description of 'regularities' in participants' activity; regularities which provide opportunities for automation of elements of work that would otherwise be performed by participants themselves.

An example of this approach is given by the class of Workflow systems[1] which aim to separate task *performance* from task *coordination*. Regularisations in the patterns of participants' activities are encoded in 'working processes', allowing the system to take over elements of coordination so that participants can concentrate on aspects of the activity, the performance of individual tasks, requiring their particular attention. This approach, then, is characterised by the *mechanisms* embodied in the system for the achievement of aspects (specifically, coordination aspects) of the supported activity. The problems arise when users wish to step outside the regularised patterns of behaviour which the mechanisms prescribe.

Structure, representation and 'support' for cooperative work

On the surface this distinction between medium and mechanism might be seen as one of *structure*, or the extent to which interaction is guided and/or constrained by policies embedded within the system. In this sense, email systems provide little in the way of structure, while for Workflow applications this structure is implicit in the goals of the technology. However, interaction with any computer system is structured in some sense, and it is not the existence of such structure which is at issue here, but rather the methods by which it is derived, employed, and how it can be manipulated to support different user requirements.

1 "Workflow" is a wide-ranging term, we use it here only in the broadest sense

Mackay (1990) reports on a study of the Information Lens system (Malone *et al.* 1987), which supports filtering of *semi-structured* email messages on the basis of user-specified rules. She describes how some users working with a version of the system exploited a little-used debugging feature to allow them to filter their mail for archiving purposes *after* they had read it—a use of the system not envisaged by its designers, who assumed that the value of the system lay in filtering messages *before* they got to the user. It is interesting to note that after this 'feature' was made more explicit in the following release of the software, a number of users who had rejected the technology as unsuitable for their methods of working then adopted the system and found it useful (Mackay, 1990).

This example shows that even highly-structured systems can be re-purposed to support activities not considered by their designers. A system's ability to adapt to different user requirements is determined by the ease with which the structures and policies embedded in the system can be customised. Customisation in this sense implies not only the ability to mould and manipulate structures within the system, but also the ability to appropriate them and use them in new ways; support for customisation is support for innovation. This position echoes the arguments of Greenberg (1991) who suggests that we should be developing 'personalizable' groupware, which can be adapted to the needs of individual users. Contrast this with the idea of a system which governs its behaviour according to some model or representation of its users' working contexts, where the objective of the representation is to enforce roles and commitments to ensure the group is 'efficient and effective' (Greenberg 1991).

The modelling of working context and its representation to govern the behaviour of cooperative systems is a central focus in current CSCW research. There are a number of reasons to doubt that this approach, which we have referred to as an attempt to provide 'mechanism' rather than 'medium', will be successful. We have already alluded to the problem of describing contextualised cooperative activities at a level of abstraction suitable for system design. In addition, a focus on existing work context would seem to inhibit the exploration of more innovative approaches to supporting cooperation, and also preclude innovation in the way that users employ systems to support their activities (as was seen to be a factor that influenced the successful adoption of the Information Lens system).

It is crucial to recognise that the forms of group support which are embodied in a cooperative system affect not only the establishment of working behaviour *within* the system (those defined within the systems' terms), but also those *around* it. It is frequently the behaviours around rather than within the system which are important determinants of its acceptability. In a study of a collaborative text editor, Dourish and Bellotti (1992) emphasise the way in which 'shared feedback' within a shared workspace—essentially a 'non-structure' in comparison to more formalised models of collaborative writing—is key to the emergence of a range of coordinating mechanisms by writing groups. Not only do the coordination activities of these groups arise around, rather than within, the shared system, but, arguably, it would have been much more difficult for such naturalistic coordination to occur had mechanisms been directly embodied in the tool.

The question of medium and mechanism is really one of representation—what is to be represented in the system, and how are the representations going to be used. We have tried to show that it is not necessary that a cooperative system has access to a representation of the activities it is supporting in order to provide 'support'. This should be obvious from the non-computational domain. For example, a piece of paper does not have access to a model of the writing process yet it still supports that activity. Not only does it support writing, with the requirements of different styles, smooth transition between writing and drawing, concurrent and serial access and so on, but it can also be used to soak up spilt coffee. As with the email example discussed earlier, the lack of embedded representations allows great flexibility in supporting different styles and behaviours.

This should by no means be interpreted as an argument *against* representation. After all, representation and formalisation lie at the heart of computational design. There is however a great deal of variability in how computational representations are *interpreted* in supporting human activity. The danger is that we may be seduced by the representational qualities of software systems and begin to confuse the representations for the activities they represent. This confusion between reality and representation lies at the heart of debates in the CSCW community such as that highlighted by Suchman and Winograd (Suchman 1993, Winograd 1993). Our proposal here embodies an attempt to view computational representations in CSCW systems as objects of, rather than proxies for, user activity.

So one motivation behind the call for a better understanding of the way that group work is actually carried out is to allow development of systems that more closely resonate with existing work practice. We agree that such an understanding is indeed essential for the development of more effective systems, but that the value of this understanding is in revealing the flexibility required by groups of participants with different individual requirements for support, rather than in yielding up ever-more-detailed fare for representation. This position places the emphasis on developing systems which users can adapt to meet their requirements, rather than systems that constrain interaction to some model of how they perform their work—manipulating representations instead of being manipulated by them.

Customising system behaviour

Our focus on medium rather than mechanism has suggested an approach to the design of cooperative systems that places the emphasis on customisation rather than rigid structures and policies governing system behaviour. Details of individuals' working contexts, which determine their requirements for system support, are so contingent on factors like individual, local and organisational knowledge, as well as the tasks being supported and personal preferences, that trying to model these factors at a level suitable for system design is unlikely to be successful. Indeed, the very variability of these factors, and their highly individual nature, suggests that we should take an alternative approach rooted in the recognition of open-ended variation, rather than an attempt to close it under some fixed representation.

Customisation is often advocated for systems which must support users with different working practices, levels of expertise and personal preferences. However, although many systems provide facilities for tailoring of *surface* interface features, few allow aspects of *deeper* system behaviour to be customised at the point of use. The conventional wisdom is that this separation of surface and deep—interface and application—is a good thing, both for developers and users. In particular, details of implementation decisions made within a system—the policies which determine how distributed data is managed in a collaborative application, or how information regarding user activities is made public—should be hidden from users who are focused on carrying out their work. The realities are that such 'low-level' policy decisions have important consequences for how users do their work—Greenberg and Marwood (1994), for example, show how different methods of concurrency control have a major impact on the interaction that can be supported by a system.

It is these details of system policy or behaviour that we have argued should be flexible and open to customisation, rather than hidden under some representation of the activities the system is supporting. In traditional systems however there is a *gulf* between the ability to customise aspects of process and functionality as opposed to interface and presentation. If possible at all, the former usually requires much more knowledge of system internals and the ability to express customisation requirements in a programming language. We examine the characteristics of this 'customisation gulf' below, before discussing an approach which attempts to bridge this gulf and provide support for both surface and deep system customisation.

The customisation gulf

The gulf between surface and deep customisation in most current systems reflects the separation in system architectures between interface and application functionality (Dourish 1995a). This gulf is an artefact of software engineering practice which does not reflect the customisation requirements of systems' users. An example of surface or interface customisation is the option provided by some word processors to switch between menus with an abridged and the full command set, where the list of commands available with each option is pre-defined by the system developer. An example of deep customisation is the ability to integrate the word processor with a foreign language translation program, when this operation was not pre-programmed by the system developer.

Although many systems are advertised as being highly-customisable, very few support flexible mechanisms for deep customisation. For example, some systems extend the idea of selection from different menu sets to allow users to change menu labels, key bindings, menu composition and even to create 'macros', so that one keystroke or menu selection causes execution of multiple commands. In all these cases however the user is constrained to using the functions the system developer has provided, and cannot customise the actual behaviour of the system. Indeed, in a study of word processor and spreadsheet packages, Oppermann (1994) states "few packages offer options for redefining their current functionality in such a way that a task- and user-specific adaptation can be achieved" (page 18).

A deeper model of customisation is provided by systems whose functionality is *parameterised*; that is, users can configure system behaviour by selecting from lists of alternative functions. However, rather than a smooth progression in the level of complexity and the degree of expertise required to perform customisation operations with such systems, there is often a 'steep incline' to be climbed before the user can isolate the functions that should be modified and appropriate values for parameters (Maclean *et al.* 1990). The 'parameters of interaction' provided by a version of the PREP collaborative editor (Neuwirth *et al.* 1994), for example, allow users to set parameters to control frequency of update propagation, granularity of updates and so on. This greater flexibility is however bought at the cost of increased expertise required of the user in knowing how these functions relate to the system behaviour, and how to set them to serve the requirements of the current working context.

This increased expertise is partly explained by the shift in the *language* through which customisation is performed. Basic macros begin to form such a language by providing the facility to *compose* or combine existing operations to form new ones, and more complex extension facilities add other linguistic features such as conditional operations and iterators. To customise surface features, users can often use techniques such as demonstration or menu selection that are part of the usual operating language for the system. Deeper customisation, however, must typically be performed in a different language which is oriented more towards the system developer than the application user, and a step function is often involved in acquiring the skills to begin to express customisation requirements in this language. This is particularly true for systems which allow users to extend system behaviour by creating new functions or modifying existing ones, often via an *Application Programmer Interface* (API), which open up the levels of customisation involved at the cost of the reusability of existing system components.

The customisation gulf is therefore characterised by two inter-related problems. The first one is the *level of customisation* possible, and with most systems this lies above the functionality of the application, rather than within it. The second problem is the *language of customisation*, and traditional systems provide limited facilities to express customisation requirements using the skills users already have, requiring the learning of new languages to describe new system behaviours. Both of these problems combine to give users no way to reach into the system and customise the way in which functionality (and not simply the interface to the functionality) relates to their accomplishment of work.

Incremental customisation

Issues of level and language of customisation form a barrier to users wishing to customise the behaviour of their systems. Studies carried out in the HCI community have shown that users are often unwilling to invest the time and effort required to surmount this barrier and acquire customisation knowledge, even if acquiring such skills would allow them to accomplish their work more easily (Carroll and Rosson 1987, for example). These observations have led some to suggest that users should be guided and encouraged in adopting a more exploratory approach (Oppermann 1994), and a 'tailoring culture' should be established which encourages users to share their expertise and results of customisation operations (Maclean *et al.* 1990). Trigg and Bødker (1994) observe that in some organisations such a tailoring culture is actively promoted by assigning responsibilities for customisation and establishing procedures to discuss proposed modifications.

The establishment of a tailoring culture does not require users to become skilled system designers, as there will inevitably be differences in users' willingness and abilities to acquire new skills. However it is important that users are aware of the possibilities for change which exist with their systems, to allow them to express their needs to more skilled customisers (Maclean *et al.* 1990). The barrier formed by separation of surface and deep system details, and use of different languages to customise features of each, inhibits users from even envisaging what can be done to tailor their environments, regardless of whether or not they themselves perform the customisation. This suggests that an approach is required that explicitly recognises that users need to customise, or envisage customisation of, both surface and deep system details, and that the separation currently enforced by different customisation languages is not a useful one.

In effect this calls for a more *incremental* approach to customisation, where users can express their requirements for support as much as possible using the skills they already possess. This is a similar technique to the support in more advanced word processors which allow customisation of menu contents using direct manipulation techniques, but here we are discussing its application to details of deeper system behaviour and not just the surface user interface. More complex customisations which cannot be performed using the same language should require a minimal increase in the level of expertise required. This should then reduce the size of the customisation gulf, both in terms of the language and level of customisation, and equate increases in tailoring power with proportionate increases in knowledge and skill required.

At the lowest level it may be possible to support customisation of some aspects of system behaviour through high-level user interface tailoring. An example of this is provided by the MEAD system (Bentley *et al.* 1993) which supports construction of different *User Displays* (UDs) of information from a shared space. With MEAD it is possible to develop a multi-user interface for a shared information system that allows users to select the UD representations which suit their requirements, and to switch between UDs as these requirements change. Although MEAD only supports selection of UDs from a developer-specified list, the customisation possibilities go beyond surface tailoring to aspects of system behaviour.

Underlying MEAD is an event service which distributes notifications of changes in the shared information space to interface clients. It maintains a list of the kinds of updates each client is interested in, and uses this list to selectively propagate updates to clients that should be informed. When users select alternative UDs to represent information from the shared space, the client automatically informs the notification service of the change in its interests to ensure it can maintain consistency between UDs and shared information. Thus MEAD supports limited customisation of the notification policy embedded in the system, but without requiring users to do extra work or learn a new customisation language.

So this approach begins to address issues of deeper customisation, and does not make the usual distinction between user interface adaptation and changes in system behaviour. Customisation is not a separate activity from normal interaction with the system, but rather users express their requirements for support from the system in terms of the information displays they require, and in doing so adapt the details of deep system policy with regard to update notification. What MEAD doesn't support is the ability to go beyond adapting the kinds of event notifications that users are interested in to more flexible strategies for configuring event granularity, different representations of events, event importance and so on.

To support incremental customisation requires techniques which allow users to move up the 'customisation curve', gaining more customisation power at the cost of a minimal increase in complexity. The next section describes a system to illustrate this incremental approach, showing how the basic language of system interaction is enhanced with facilities to customise details of system behaviour.

Incremental customisation in the BSCW client

The Basic Support for Cooperative Work (BSCW) project at GMD is concerned with the development of a simple shared workspace system that runs on top of the Internet. Our target user population is the academic research community as a whole, and specifically academics involved in large research projects. Such projects often consist of a number of project partners who have different organisational concerns, computing infrastructures and so on, but have a requirement to share information and work collaboratively.

The BSCW client is an application program which allows users to interact with a shared workspace. It provides basic functionality such as browsing the contents of a workspace, adding and removing documents, examining change histories and so on. The volume of information that can be stored in a shared workspace is large, for example, each document in the workspace can have associated annotations, version information, creator/modifier details, current status and so on. It is not practical to display all this information at the same time to every user; moreover, it is not useful to do so, as each user may have different reasons for interacting with the workspace at any point in time, and much of the information it contains may not be relevant to the current working context.

To address this problem the client prototype supports incremental customisation of the policies which determine the visualisation, interaction and change notification properties of the shared workspace. Thus, rather than using some representation of users' activities (such as 'roles') to determine their requirements for support, the client provides facilities which allow users to adapt system behaviour to reflect the requirements of their current working contexts. The aim is to provide a medium for cooperation which can be customised as much as possible using skills users already possess. This approach is illustrated below.

Customising system policy with Attachments

User customisation of the workspace is performed by associating *Attachments* with basic representations of workspace artefacts (figure 1). An Attachment carries with it (customisable) methods which adapt the visualisation, interaction and notification policies associated with items in the workspace for an individual user. In figure 1, for example, two users have tailored their representations of a simple workspace by associating different Attachments with a document called *ECSCWpaper*. User Paul is currently responsible for editing the next draft of the document, and Dik wants to proof-read it when Paul has finished editing. The different Attachments associated with the document reflect the different orientations of the users to the workspace. Paul, possibly trying to locate text from an old draft, has placed an Attachment over the document allowing him to open previous versions by clicking on their names, while Dik has added an Attachment to show who is currently editing the document.

In this example, the system reveals information to the users in response to their customisation operations, and allows them to perform different actions (such as opening a previous version). The system does not have a representation of the two users' activities, which they have negotiated externally and can re-negotiate at any time (if the authors decide a further draft is needed, for example). The process of customisation through addition, removal and tailoring of Attachments is therefore an ongoing one, reflecting the fluid and highly dynamic manner in which roles and responsibilities are negotiated in group work (Anderson *et al.* 1989).

In addition to visualisation and interaction details, associating Attachments with workspace artefacts may customise the underlying notification service. This service uses an expression of each users' interests in a similar manner to the MEAD system discussed above to select the events to be propagated, the method of propagation to use, and how events are represented at the user interface. In figure 1 the addition of the *current editors* Attachment registers an interest for Dik in changes to the set of editors for ECSCWpaper. When Paul finishes editing, his name will be removed from the Attachment, providing feedback on his activities. This feedback is generated within the context of the workspace and the context of users' activities, as the interests users hold are derived from the adaptations they make to suit their individual situations. Thus Attachments are not just 'views' but carry with them relevant interest patterns and behaviours.

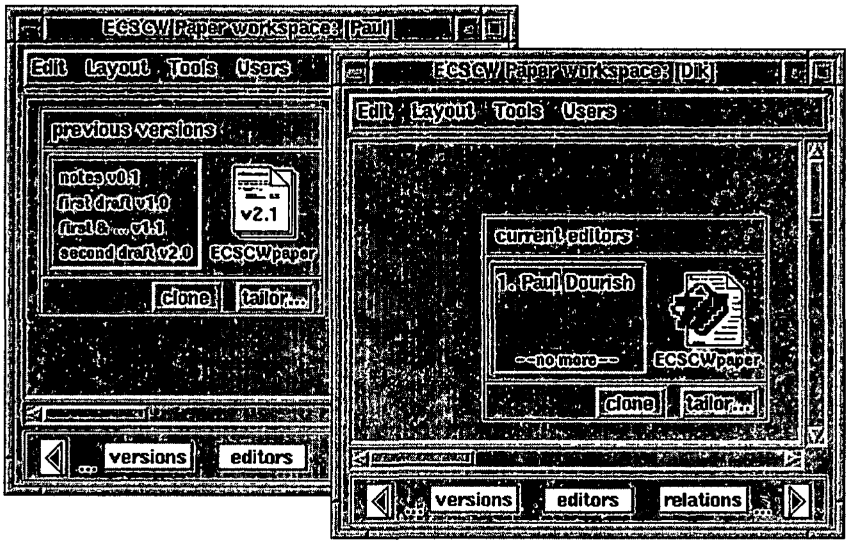

Figure 1 Alternate representations of a shared workspace using Attachments

The basic customisation of a shared workspace offered by adding and removing Attachments does not enforce a separation between surface interface features and deeper aspects of system behaviour. In addition to addressing some issues of *level* at which customisation is supported, users customise their workspaces using the same *language* of customisation, as association and removal of Attachments are part of the ordinary language for interacting with the client. However, interacting with a workspace using only pre-specified Attachments has limitations. For example, the indication of an interest in a workspace artefact may not be enough; users may hold many such interests at any one time, but their requirements for event notification may vary, depending on the relevance and importance of events for their working contexts. Thus there is a need to support customisation of details of the Attachments themselves.

Moving up the customisation curve: Customising Attachments

The bottom of the customisation curve is characterised by association of pre-defined Attachments with representations of workspace artefacts. A number of Attachments have been developed in the first version of the workspace client like those described above. In addition, the client provides a range of facilities for customising details of the basic Attachments to give users more flexibility in expressing their requirements for support from the system. Each of these methods requires slightly more skill on the part of the customiser, but brings with it a proportionate increase in the power available to perform customisation operations.

The emphasis here is on providing facilities for flexible customisation of system behaviour at the point of use that build incrementally on knowledge and skills that users already have. The techniques that have been implemented are described below in order of increasing customisation power.

- *Sharing Attachments*: The importance of customisation as a cooperative activity has been highlighted by a number of studies (Mackay 1990, Trigg and Bødker 1994). To support cooperative customisation it is possible to add Attachments to the shared workspace for others to retrieve and use Thus users can share the expertise they have acquired with the system, and others become familiar with the possibilities for customisation offered. This is similar to the strategy used by Maclean *et al.* (1990) to exchange 'Buttons' by email, but here the Attachments are treated just like other information in the workspace, and can be added and retrieved in just the same way as other documents. Therefore users do not have to acquire new skills in order to share the Attachments developed by others.

- *Tuning Attachment properties*: To customise existing Attachments requires a slight increase in knowledge about their construction. Attachments consist of a number of methods that comprise the visualisation, interaction and notification properties of each Attachment component. Each component is similar to the concept of an *Icon Region* in Iconographer (Gray *et al.* 1990), but unlike Iconographer each property can be configured without programming. In figure 1, for example, it is possible to customise the notification properties of the *current editors* Attachment to provide audio feedback, put up a dialogue box, send an email and so on when someone finishes editing a workspace item. These can be combined—a 'finish editing' event might remove the editor's name and sound a chime, to bring the event to a user's attention even if the current orientation is not to the workspace. This example of event notification emphasises our focus on medium rather than mechanism. Rather than derive 'importance' or 'relevance' of events from some representation of activity, users can configure the system to provide notifications that are suitable. This is similar to the approach of Khronika (Lövstrand 1991), which supports a range of flexible practices by a focus on information rather than action (Dourish *et al.* 1993).

- *Composing Attachments*: Attachments can be associated with other Attachments to create composites. The model we use is that each Attachment has an empty *slot* which will represent the workspace artefact it is associated with. In figure 1 for example the slots of both the Attachments shown are occupied by the workspace artefact called ECSCWpaper. As Attachments are workspace artefacts just like any other (can be added and retrieved from the workspace and so on), they can occupy slots in the same way as representations of documents.[2]

- *Editing Attachments and creating new events*: It is possible to create new kinds of Attachments or re-configure properties of existing Attachments using a graphical Attachment editor. This supports definition of simple components which present information from a workspace, perform actions when sequences of interactions are recognised, and can re-configure when represented information changes. We are currently developing this tool to support more powerful visualisations and interaction techniques, one of which is the ability to create new kinds of events, more descriptive and oriented towards the activities users are performing. Most notification services tend to propagate events oriented more towards the system, as information such as 'file added', 'user logged in' and so on is easy to capture. However, if users can define new events, conditions under which they are generated, and means to represent them in the user interface, it is not necessary that these are described in system-oriented terms (as with, for example, the 'pub-call' event provided by the Khronika system—Lövstrand 1991). This approach is consistent with our emphasis on the system as a medium to support activities without representations of what those activities might be.

Thus the shared workspace client supports incremental customisation of surface and deep system features with a range of techniques. These techniques provide greater flexibility and power without the common 'step function' in terms of the expertise required to perform more complex customisation operations. The techniques show one way in which a deeper model of customisation can be provided which utilises existing skills and integrates customisation behaviour as part of normal interaction with the system.

Conclusions

Much effort in CSCW development has been aimed at creating detailed models of collaborative activity which can be used as the basis for computational design. The application of such models, however, has often been highly problematic. In this paper we have outlined an alternative approach which emphasises a CSCW system as a *medium* through which collaborative work occurs, rather than an embodiment of *mechanisms* representing perceived regularities in collaborative activity. This perspective, recognising the emergence of patterns of collaborative behaviour both

2 It is not possible or appropriate to go into detail in this paper In brief, we use standard object-oriented techniques to send messages to representations of workspace objects, which either ignore the messages or re-configure their presentation, interaction and/or dynamic properties As the Attachments are also workspace objects, they can respond to these messages, allowing them to be occupants of slots in other Attachments

within and *around* technology, necessitates an examination of the ways in which customisation and adaptation are supported in CSCW systems.

We have pointed to a gulf in many current systems between the customisation of surface features and deeper functionality. This 'customisation gulf' forms a barrier to the forms of emergent, adaptive behaviour which we advocate. However, the use of incremental techniques provide a means for effective deep customisation through the manipulation of high-level interface components, as illustrated by the prototype client for the BSCW system currently under development at GMD.

Customisation is not simply a method for individuals to adapt technology to meet their own needs; it is, fundamentally, a means by which users can *construct* their working patterns, individually or as groups, from the basic materials provided. We believe that facilities of this sort are critical in CSCW, and motivate re-consideration of how systems are developed. In particular, this focus highlights a need for more *open* system architectures (Dourish 1995b), where details of deep system policy are 'visible, accessible and tailorable' at the point of use (Bentley 1994). One approach to providing such architectures which builds on the use of reflective, 'meta-level' architectures for CSCW system construction is described by Dourish (1995b). In general, however, we look towards the use of customisation techniques as more effective than explicit representation in bridging between the expertise of social science and the needs of computational design.

Acknowledgements

The prototype of the BSCW client was implemented by Markus Wasserschaff The authors would also like to thank David England and Thilo Horstmann for fruitful discussions about many of the issues dealt with in this paper.

References

Anderson, R., Hughes, J. and Sharrock, W. (1989): *Working for Profit: The Social Organisation of Calculability in an Entrepreneurial Firm*, Aldershot, Avebury, 1989.

Anderson, R. (1994). Representations and requirements: The value of ethnography in system design, in *Human-Computer Interaction*, 9, 1994, pp 151-182.

Bentley, R , Rodden, T , Sawyer, I and Sommerville, I. (1993): Architectural support for cooperative multi-user interfaces, in *IEEE Computer*, 27(5), May 1994, pp 37-46.

Bentley, R (1994) *Supporting Multi-user Interface Development for Cooperative Systems*, PhD thesis, Computing Department, Lancaster University, June 1994 Available by anonymous FTP from Lancaster University at "ftp://ftp.comp lancs ac.uk/pub/reports/ThesisRB.ps.Z"

Carroll, J. and Rosson, M (1987). Paradox of the active user, in J M Carroll (ed), *Interfacing Thought*, MIT Press, 1987, pp 80-111

De Michelis, G and Grasso, M A (1994): Situating conversations within the Language/Action perspective: The Milan Conversation Model, in *Proceedings of CSCW'94*, Chapel Hill, ACM Press, 22-26 Oct 1994, pp 89-100

Dourish, P (1995a): Accounting for system behaviour: Representation, reflection and resourceful action, to appear in *Proceedings of Computers in Context (CIC'95)*, Aarhus, Denmark, 14-18 Aug. 1995.

Dourish, P. (1995b): Developing a reflective model of collaborative systems, to appear in *ACM Transactions on Computer-Human Interaction*, 1995 (in press)

Dourish, P and Bellotti, V. (1992). Awareness and coordination in shared workspaces, in *Proceedings of CSCW'92*, Toronto, ACM Press, 31 Oct.-4 Nov. 1992, pp 107-114

Dourish, P., Bellotti, V., Mackay, W and Ma, C. (1993): Information and context: Lessons from a study of two shared information systems, in *Proceedings of COOCS'93*, Milpetas, California, 1-4 Nov. 1993, pp 42-51

Gray, P , Waite, K. and Draper, S. (1990): Do-it-yourself iconic displays: Reconfigurable iconic representations of application objects, in *Proceedings of INTERACT'90*, 1990, pp 639-644

Greenberg, S (1991): Personalizable groupware: Accommodating individual roles and group differences, in *Proceedings of ECSCW'91*, Amsterdam, Kluwer Academic Publishers, Sept 1991, pp 17-31.

Greenberg, S. and Marwood, D (1994). Real time groupware as a distributed system. Concurrency control and its effect on the interface, in *Proceedings of CSCW'94*, Chapel Hill, ACM Press, 22-26 Oct 1994, pp 207-217

Hughes, J., King, V , Rodden, T. and Andersen, H. (1994): Moving out from the control room: Ethnography in system design, in *Proceedings of CSCW'94*, Chapel Hill, ACM Press, 22-26 Oct 1994, pp 429-439

Lovstrand, L (1991): Being selectively aware with the Khronika system, in *Proceedings of ECSCW'91*, Amsterdam, Kluwer Academic Publishers, Sept. 1991, pp 17-31

Mackay, W. (1990): Patterns of sharing customizable software, in *Proceedings of CSCW'90*, Los Angeles, ACM Press, 7-10 Oct 1990, pp 209-221.

Maclean, A., Carter, K., Lovstrand, L. and Moran, T. (1990): User-tailorable systems. Pressing the issues with Buttons, in *Proceedings of CHI'90*, Seattle, ACM Press, 1-5 April 1990, pp 175-182

Malone, T , Grant, K , Turbak, R., Brobst, S and Cohen, M. (1987): Intelligent information-sharing systems, in *Communications of the ACM*, 30, 1987, pp 484-497

Neuwirth, C., Kaufer, D , Chandhok, R. and Morris, J (1994) Computer support for distributed collaborative writing: Defining parameters of interaction, in *Proceedings of CSCW'94*, Chapel Hill, ACM Press, 22-26 Oct 1994, pp 145-152.

Opperman, R and Simm, H (1994): Adaptability: User-initiated individualization, in R Oppermann (ed), *Adaptive User Support*, Lawrence Earlbaum, 1994.

Sommerville, I., Bentley, R , Rodden, T. and Sawyer, P (1994) Cooperative systems design, in *The Computer Journal*, 37(5), 1994

Sproull, L and Kiesler, S. (1991): *Connections: New Ways of Working in the Networked Organisation*, MIT Press, Cambridge, Mass , 1991.

Suchman, L (1993): Do categories have politics? The language/action perspective reconsidered, in *Computer-Supported Cooperative Work*, 2(3), 1993, pp 177–90.

Trigg, R. and Bødker, S (1994): From implementation to design. Tailoring and the emergence of systematization in CSCW, in *Proceedings of CSCW'94*, Chapel Hill, ACM Press, 22-26 Oct. 1994, pp 45-54

Winograd, T. (1993). Categories, disciplines and social coordination, in *Computer-Supported Cooperative Work*, 2(3), 1993, pp 191–198