An architecture for tailoring cooperative multi-user displays

Richard Bentley, Tom Rodden, Peter Sawyer and Ian Sommerville

CSCW Research Centre
Computing Department
Lancaster University
Lancaster LA1 4YR.
U.K.
Email: {dik, tom, sawyer, is}@comp.lancs.ac.uk
Phone: (+44) 524 65201
Fax: (+44) 524 381707

ABSTRACT
A range of architectures have emerged which support real-time cooperative user interfaces. These architectures have tended to centralise the management of the interface and thus provide only limited support for user-centred development and interface tailoring. This paper considers the problems associated with the development of tailorable cooperative interfaces and proposes an architecture which allows such interfaces to be developed using an incremental, user-centred approach.

The architecture presented in this paper has emerged within the context of a project investigating cooperative interface development for UK air traffic control. We conclude that the architecture is equally applicable to other Command and Control domains, where a shared information space forms the focus for the work taking place.

KEYWORDS
Multi-user interfaces, CSCW architectures, Command and Control systems, Database visualisation.

INTRODUCTION
The merging of computer and communications technology and the development of network based windowing systems has seen the development of a range of real-time cooperative systems. These systems often exploit multi-user interfaces which allow application sharing across a number of workstations and many examples have been reported within the CSCW literature [1,2].

The approach taken by the majority of these systems has been to extend existing windowing technology to provide replication of output on a number of displays. This approach has been most successfully applied within shared window or desktop conferencing systems. A number of more recent examples of real-time cooperative systems have considered the development of architectures to support multi-user applications with more control over the user interface [3,4].

Most existing approaches have tended to consider user interface construction as a task for application developers and have provided only limited support for user-centred development and tailoring. In contrast, this paper presents an architecture which supports rapid user-centred development of multi-user displays and its realisation within a prototype system for generating cooperative interfaces.

The development of the architecture and system described in this paper has taken place within the context of an on-going multi-disciplinary project investigating the application of user interface technology to support the work of air traffic controllers. The project involves the extensive use of ethnographic observation with the intent to inform the development of cooperative systems to support the controlling process [5].

SYSTEM RATIONALE AND REQUIREMENTS
Air traffic control (ATC) systems are a type of Command and Control (C^2) system and, like many classes of C^2 system, involve controllers cooperating using a shared information space which reflects the external processes being controlled. This shared space is updated by controllers and by input from external sensors such as radars, transponders, etc. (figure 1). Each controller maintains and interacts with their own view of the shared space, with cooperation between controllers supplemented by direct audio communication facilities.

Currently, most C^2 systems are hybrid systems where a computer-maintained database is supplemented by paper

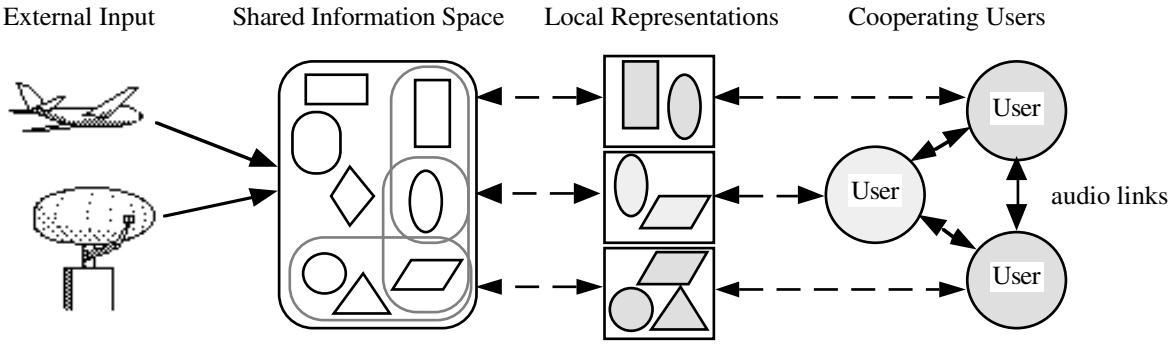

Figure 1 Cooperation via a shared space

work and a variety of manual procedures to form the shared information space shown in figure 1. Controllers evolve a complex and subtle way of working which often involves coordinating their decisions with other controllers. In making decisions, they must often anticipate how other controllers will react to particular configurations of the system being controlled. In order to reflect this complexity we are attempting to use an on-going ethnographic study [6] to inform the derivation of user requirements.

The ethnographic observation of controllers and control rooms has been a central theme within CSCW and has been reported elsewhere by [7,8,9]. Our work aims to extend this previous history by using the ethnographic study to inform the construction of a computer system. In many ways this is a new field for both computer scientists and sociologists with methods of working and respective roles in the requirements capture process still being established.

Given the lack of any established working method we use a prototyping approach to system development. The focus of our work has been the development of a prototyping system which allows specific instances of cooperative interfaces to be rapidly constructed in conjunction with the results of on-going studies [10].

Architecture Requirements

The overall objective of the architecture, therefore, is to support an environment for the prototyping of interfaces for users cooperating via a shared information space. The architecture should be flexible enough to allow the interactive development of these interfaces by informal consultation between user, interface developer and ethnographer. The role of the prototype generator and initial observations highlight a number of requirements for the underlying architecture.

1) Support for multi-user displays

Cooperating users access the shared information store through individual workstations. This form of access will often be supplemented by the informal cooperation gained from screens being located within a common control room. Given the reliance on controllers' awareness of the activities of others, the supporting architecture should :-

- Allow the construction of different user display formats.
- Allow displays to be shared across different workstations.
- Support the visualisation of shared entities within user displays.
- Allow the manipulation of shared entities on different workstations.
- Allow the set of cooperative displays to be dynamically changed with workstations being easily added and removed.

2) Support for different views

Cooperating users may wish information to be presented in different ways. For example, in ATC a particular controller's view of an aircraft could be different to that of an officer concerned with controlling the flow of traffic. The architecture should support different views of entities within the shared information space so must :-

- Allow the definition of different interactive representations of shared entities.
- Maintain these different representations as underlying entities change.
- Allow the updating of entities through interaction with these representations.

To promote rapid interface construction the architecture should allow these facilities to be provided in a highly visible and accessible manner.

3) Support for tailorability

Our focus is on the construction of cooperative interfaces in conjunction with the observation of users and it is difficult to predict the exact nature of these interfaces in advance. It is important that the architecture allows interfaces to be easily constructed and tailored by domain experts and developers in line with the results of observational studies. In particular the architecture should :-

- Allow system features to be presented in such a way that details are readily understood.
- Allow displays to be easily tailored to support different working practices.
- Support the personalised presentation of shared information to different users.

An end-user could be given complete freedom to tailor the appearance of his or her display by the architecture. However, results of initial observations have suggested that

end-user tailorability needs to be bounded within a cooperative setting. For example, in ATC it is important that controllers understand each others displays at a glance. In fact when controllers change at the end of shifts the in-coming controller monitors the work of the out-going controller he or she is replacing for some minutes prior to switching position. The limits and bounds of the tailorability eventually provided to end-users will be a matter of consultation between developer, controller, observer and management.

SHARED INTERFACE ARCHITECTURES

The merging of workstation technology and real-time computer conferencing has played an important role in CSCW. This merging has been termed *desktop conferencing* and an integral part of desktop conferencing is the use of *multi-user interfaces*. Lauwers [11] outlines two approaches for the development of multi-user interfaces. The development of special purpose applications which are *collaboration aware*, and the adapting of existing single user applications to provide *collaboration transparent* shared applications. Examples of work which exploit collaboration transparency include Vconf [12], Rapport [13], SharedX [14], and MMConf [15].

Most of the collaboration transparent synchronous interfaces exploit network based window systems such as X or NeWS. These systems assume a client-server or virtual terminal model of interaction where the application and the display are separate and connected by an interaction protocol. This approach allows local graphical events to be handled by each display with interaction details being passed to the application using this protocol.

The majority of shared window systems use a centralised agent to multiplex output from applications onto a number of display screens and to route input from each display to the appropriate applications. In this approach an application does not know that it is being shared between users. Consequentially it is impossible to support different views of the shared information within these systems.

An additional problem with this approach is that an assumed model of cooperation based on many readers but only one writer is embedded within the system. As a result it is difficult to customise these systems and any tailorability which exists has concentrated on the development of techniques to support different forms of turn taking protocols [16].

In contrast to the transparent approach outlined above, collaboration aware solutions provide facilities to explicitly manage the sharing of displays between different users. This approach has been adopted by a range of applications including Cognoter [1], Grove [17] and rIBIS [2].

As a result of the centralisation of user interface management, multi-user cooperative applications can support different user interfaces and many of them provide facilities which allow individual views to be supported. Recently researchers have started to examine how this arrangement can be generalised to provide supporting architectures for a variety of applications. Systems adopting this approach include the Rendezvous system [4] which has the ability to support different user views on application objects and interactions with these views. This is provided by maintaining a separation between *underlying objects*, which contain the application semantics and *interaction objects*, which maintain details of users' views on these objects.

Collaboration aware applications require a significant degree of logical centralisation. Systems which are physically centralised and directly support a number of displays have been criticised for their inability to provide the responsive interfaces required by cooperative systems [11]. Replicated architectures exist for collaboration aware applications [16] which significantly improve this performance. However, many of these systems involve an additional overhead in maintaining consistency between application processes and often require significant re-configuration when adding or removing displays. As a direct result of our need to provide responsive user interfaces whilst also supporting the addition and removal of displays we have adopted a hybrid approach in our architecture.

Existing architectures for collaboration aware systems do not concentrate on tailoring. They embed user interface management facilities within the cooperative application. For example, in Rendezvous each user's interaction is managed by lightweight processes embedded within the application process. Facilities tend to be application specific and provide only limited tailorability. In contrast, our architecture attempts to localise and make visible many features of user displays in order to support and encourage tailorability by exploiting independent, active display agents.

USER DISPLAY AGENTS

Traditional models of interaction consider the user interface as a presentation of an application with users interacting through some form of dialogue. User interface systems have reinforced this model by supporting the development of user displays as fixed system entities. User interface details are often closely coupled with application semantics with the result that tailorability is inhibited.

In contrast to previous dialogue models our approach is based upon the application of shared interaction objects. The state of these objects characterise the information presented to the user by the application. Users interact directly with these objects with state changes directly affecting the state of the application. The interaction semantics of the application are thus captured by the set of objects presented to the user and how the system reacts to changes in these objects.

Our architecture considers user displays as active entities with individual properties which can be tailored by systems designers in conjunction with users. We refer to these active

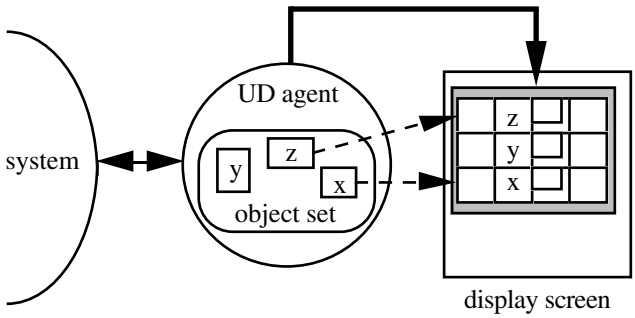

Figure 2 Detaching interaction semantics with a User Display agent

entities as *User Display Agents* (UD agents) and the parts of the user's screen managed by an agent is called a *User Display* (UD). These agents maintain state and use this state to determine details of interaction with the user. This state information includes the interaction and presentation details of the shared interaction objects relevant to a User Display. Each User Display agent allows this information to be directly manipulated with changes occurring dynamically. This arrangement represents an extension of the separation between interaction and underlying objects supported by systems such as Rendezvous.

Under this arrangement, details of the user interface can be held within the User Display agent rather than the system (figure 2). This allows users to access the details of the user interface semantics through mechanisms provided by the User Display agent. This approach is analogous to the difference between deep and surface object interaction highlighted by Took [18].

The User Display Agent Model

Our architecture exploits active User Display agents as the primary mechanism of interaction with a shared information store. The following properties are central to this interaction:

FOCUS: Cooperating users are only interested in a subset of entities within the information store. As the state of an entity changes, its relevance to each user may change. Consequently a representation of that entity may have to be added to or removed from existing User Displays.

VISUALISATION: Whilst cooperating users may require representations of the same information, the form of these representations may vary depending upon the user and task the representation is supporting. Representations should therefore reflect the user-task context, and changes in the state of represented entities may require changes in the form of representation used.

POSITION: The spatial arrangement of entity representations on the user's screen may provide information concerning relationships between those entities. As the state of represented entities change, it may be necessary to adjust the spatial arrangement to reflect changes in these relationships.

Our model of a User Display separates the notions of the entities to be displayed, their representations and their spatial arrangement. A User Display has three separate components: a *Selection* representing focus, a *Presentation* representing visualisation and a *Composition* representing position. A User Display agent manages the affects of entity updates on each component of its User Display.

A *Selection* is a set of entities which are dynamically chosen from the information store according to *Selection criteria*. Selection criteria are predicates over entity attributes and are specified by the interface designer. The Selection criteria filter entities to select which should be included in the User Display. As entities' states change, the selection changes accordingly.

A *Presentation* is a set of *Views* used to represent entities in the selection. A View is a graphical representation which defines the appearance, attribute positioning and interaction mechanisms for an individual entity. Views are selected for each entity dynamically, according to the *Presentation criteria*. These define a guard for each View which an entity must pass if it is to be represented by that View. As an entity's state changes, it may be necessary to change the View used to represent it in the User Display.

A *Composition* is a set of positions which represent the spatial arrangement of the Views in the User Display. These positions can be absolute or relative to other Views. As an entity's state changes, the positions of the Views also change, to remain consistent with the arrangement defined in the *Composition criteria*.

This abstraction of User Displays as autonomous entities allows cooperative systems to be constructed of a federation of agents interacting and responding to changes in a shared information store. Each User Display agent contains sufficient information to allow the presentation aspects of shared entities central to cooperation to be readily tailored without system re-configuration.

COOPERATIVE USER DISPLAY AGENTS

A characteristic of the kind of cooperative system we are considering is that shared data provides the focus for the cooperation taking place; different users working at separate screens interact with shared information entities. In our cooperative setting, users accessing a shared information store each have a working set of User Display agents which manage different displays of the shared information for them (figure 3). Each agent can present the User Display it manages in one or more screen windows, and agents can be added to and removed from the working set as required.

Changes in the state of entities in the information space potentially affect the selection, presentation and composition components of each User Display. For example, a change in the longitude of an aircraft will require a change in composition in a User Display which presents a radar (ie the radar blip for the aircraft will be seen to move). Thus the architecture must allow relevant User Display

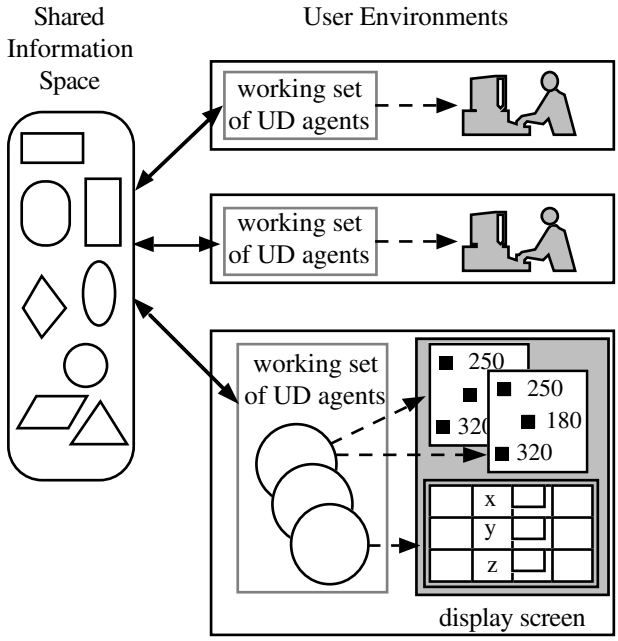

Figure 3 User Display agents cooperating via a shared information space

agents to be informed of such changes, these agents to compute the effects on their User Displays and all windows presenting that User Display to be brought up to date.

Maintaining User Displays

A requirement for our architecture was the need to allow the same User Display to be presented on separate display screens. Referring to figure 3, this means that users may hold copies of the same User Display agent as part of their working set. However, changes in the state of entities in the information space will affect each of these copies in the same way, and thus the effects of such changes on selection, presentation and composition need only be calculated once.

To support this, each user may hold a copy (or *Surrogate*) of each of the User Display agents as part of their working set. These Surrogate agents are minimised, in that each holds only the current state of the User Display; the definition of the User Display is held by the *Master* User Display agent. This receives notification of relevant changes in the state of entities, computes the effects on the User Display (using the selection, presentation and composition criteria), and informs each of its Surrogates of the new state (figure 4).

This factoring out of display semantics supports local tailorability of information displays without requiring system re-configuration. Local tailoring operations performed on a display, such as selection of alternative Views for represented entities, are retained by the Surrogate agent. Machines can be added and removed at any time during system execution; added machines will contact the Master User Display agents and create the Surrogates required, establishing communication links to receive the

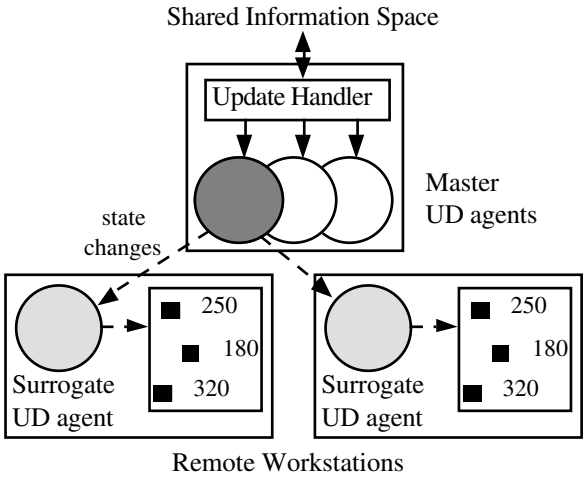

Figure 4 Surrogate User Display arrangement

current state of the User Display and future update information.

All access to the information store is made via a well-defined interface, which informs the Update Handler component (figure 4) of any changes made by external (ie non-user) sensors. In addition, the Update Handler must pass on user-sourced changes to the information store using this interface. Thus, our architecture can support a range of storage technologies and places no requirements on the information store for broadcasting update information.

The Update Handler is responsible for passing update information on to those Master User Display agents potentially affected by an update. For this, the Update Handler uses information regarding each agent's *interest set*, which is registered when the User Display the agent manages is defined. (For example, an agent managing a radar Display will register changes in longitude and latitude as part of its interest set). In the event of a change in definition of a User Display, the User Display agent must register the new interest set with the Update Handler as well as computing the new state.

Figure 5 shows the *dispatching* mechanism of the Update Handler, which minimises the communication (and so the cost) involved in informing User Display agents of updates.

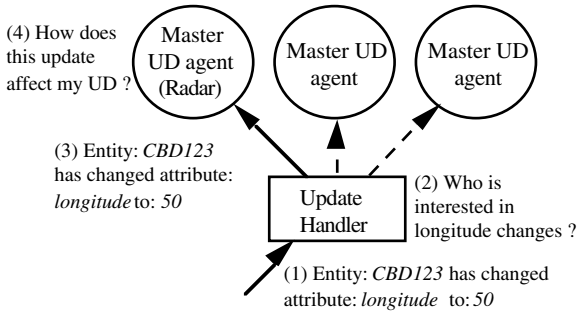

Figure 5 The dispatching role of the Update Handler

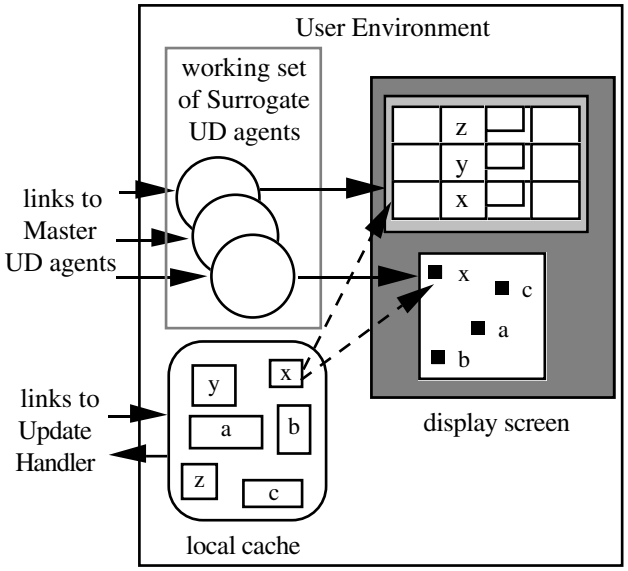

Figure 6 Local caching mechanism

It is possible (and quite likely) that entities in the information store may be represented in different User Displays by different Views. As updates occur, these Views must also be updated to remain consistent with the entities they represent. Routing such updates through the User Display agents would require each agent which maintains such a View to be informed of the update, even though the update may not affect the selection, presentation and composition components of that agent's User Display. These agents would then have to inform their Surrogates of the change, potentially resulting in many communications to the same workstation to update the Views.

To minimise this communication, we have adopted a caching mechanism where replicated subsets of the information store are held locally (figure 6). It is the Update Handler's responsibility to broadcast changes in the state of entities to all caches which maintain replicated copies of these entities. Views are connected directly to these cached entities, removing the need for complex mechanisms to maintain entity-View links across workstations. The links between the Update Handler and caches are bi-directional to allow the Update Handler to receive updates in the state of a cached entity caused by user interaction with one of that entities Views. The Update Handler can then inform the information store of such updates, selectively multicast updates to caches which maintain a copy of the updated entity and inform relevant User Display agents in case the update requires changes in the state of any User Displays.

IMPLEMENTATION

Our implementation of this architecture forms the basis of a prototyping environment which allows the construction and refinement of cooperative displays [10]. We have found that this implementation, developed in Smalltalk-80 for Sun workstations, meets our performance requirements for real-time updating.

The prototyping environment makes visible our model of a User Display, whilst hiding the complexity of the Update

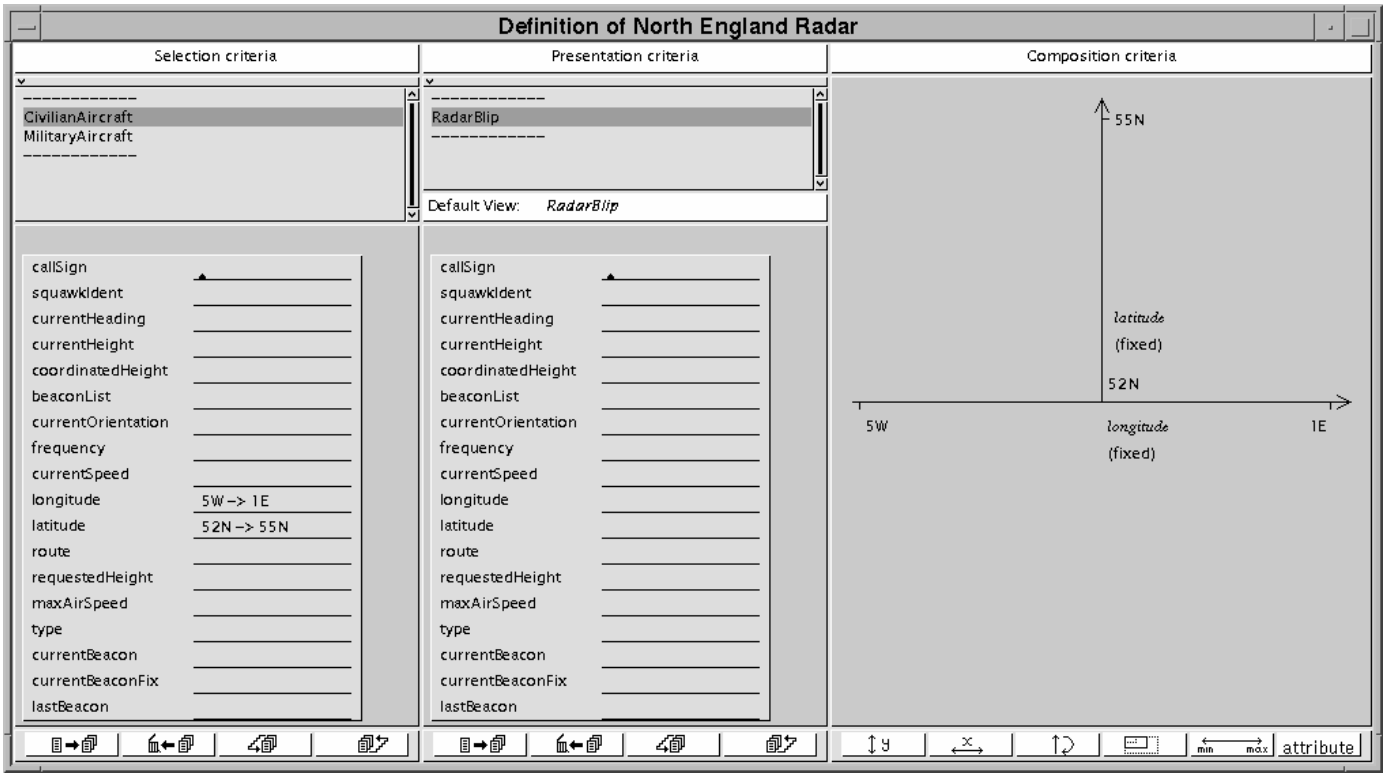

Figure 7 The User Display definition tool

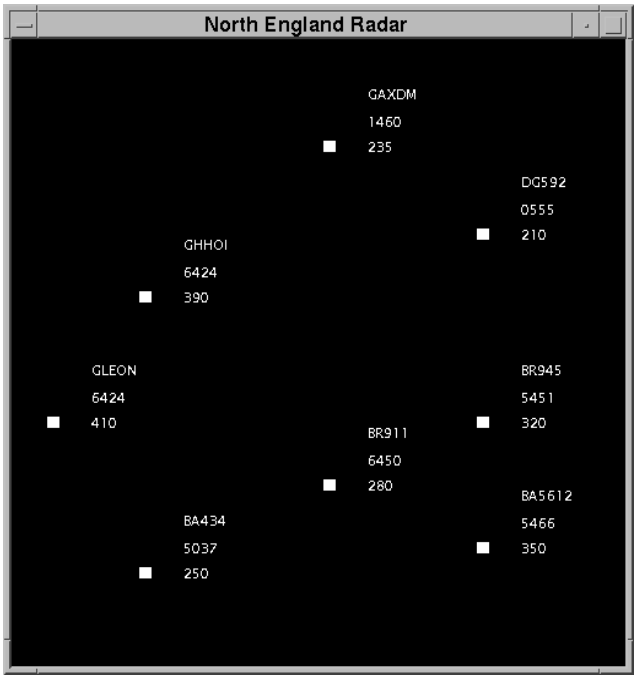

Figure 8 A window presenting the 'North England Radar' User Display

Handler, Surrogate agent and local caching mechanisms. The interface designer, in consultation with a domain expert and/or end-user, defines the selection, presentation and composition criteria of each User Display. Libraries of such criteria can be constructed and re-used, shortening the time required to construct new User Displays. Our tool used to capture these criteria is presented in figure 7, which shows the definition of a radar User Display.

The task of defining a User Display is partitioned, so that the interface designer completes a separate 'form' for each of the selection, presentation and composition criteria. For the selection and presentation criteria, these forms are represented as Pattern templates [19] which can be used to capture the criteria in a by-example manner.

A pattern defines a guard and an entity must meet all conditions specified in the pattern to pass that guard. (A pattern with no conditions implies all entities pass that pattern's guard). To meet the selection criteria of the User Display defined in figure 7 (and thus be represented in that display), the values of longitude and latitude attributes for 'CivilianAircraft' type entities must be between 5°W-1°E and 52°-55° N respectively.

The presentation criteria are defined by specifying the conditions under which different Views will be used to represent entities of each type. View templates are defined using a different tool, which captures each template's appearance, attribute positioning and interaction mechanisms. Templates are associated with one or more entity types and entities of these types can be represented by directly manipulatable instantiations of these templates. Both User Display and View definition tools are integrated, so that the User Display definition tool knows which Views are associated with which entity types. In figure 7, entities of type 'CivilianAircraft' will use a 'Radar Blip' representation.

For each type of entity which can be represented in the User Display, the composition criteria define how entity representations will be arranged. The model of composition is relatively simple, but allows Views to be positioned in two dimensions, absolutely or relative to other Views in the User Display. For example, the composition criteria defined in figure 7 will position Views of 'CivilianAircraft' type entities absolutely, depending on the values of their longitude and latitude attributes.

Figure 8 shows a window presenting the User Display defined in figure 7, as it would appear on the user's display screen. Aircraft represented are taken from a small example information store created from actual flight plan data.

The definitions of the selection, presentation and composition criteria are translated into directly evaluatable expressions by the User Display agent. The agent can use these expressions to determine the effects of updates on the User Display it manages. By using these form-based representations to capture the definition, the underlying complexity of these expressions is hidden from the designer, allowing rapid development and modification of different display formats. Our environment makes no distinction between design and run time, so User Display definitions can be modified and the effect on currently displayed information immediately observed.

CONCLUSIONS AND FUTURE WORK

We have presented an architecture to support the development of cooperative interfaces for C^2 systems which is intended to support the rapid prototyping of different formats of information display. The envisaged development model is that interfaces will be constructed in informal consultative sessions and informed by the results of observational studies. These interfaces can then be evaluated by controllers in actual control environments.

The architecture allows the construction of multi-user interfaces by the definition of User Displays which present information from a shared information space. User Displays can be shared across different workstations and a single workstation can support multiple User Displays. Each type of entity has an associated set of Views which can be used to represent entities of that type. Besides information presentation, these Views also define the interaction mechanisms which allow users to update the entities they represent. These updates are submitted to the shared information store and propagated to all other representations of the shared entity, as well as relevant User Displays.

Our approach to development has been to make as few assumptions as possible regarding the coordination the architecture must support. Rather, we have built general mechanisms which can be tailored to conform with the

results of observational studies. The propagation of change is an example of this philosophy; our update mechanism is 'policy free', with changes propagated to other users without interference from the architecture. More specialised update strategies will be tailored from this general mechanism to meet the specific needs of controllers.

Although our architecture has evolved to support the development and management of cooperative interfaces in the domain of ATC, we believe it is equally applicable to a wide range of C^2 systems. Within such systems, shared system entities provide the focus for cooperation, with changes to the state of these entities being the smallest granularity of update requiring propagation.

The domain that we have addressed allows us to consider the shared information store as devoid of embedded procedure, so that information is held as state, without considering methods which operate on that state. This simplifies our model of distribution and our local caching mechanisms. Future investigations will consider the problems of embedded procedure, such as method mutability, information migration and propagation of change within the information store.

REFERENCES

1. Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., Suchman, L., 'Beyond the chalkboard: computer support for collaboration and problem solving in Meetings', *Communications of the ACM*, 30(1), January 1987, pp 32-47.
2. Rein, G. L., Ellis, C. A., 'rIBIS: A real-time group hypertext system', *International Journal of Man Machine Studies*, 34(3), March 1991, pp 349-368.
3. Greenberg, S., 'Personalisable Groupware: Accommodating Individual Roles and Group Differences', in *Proceedings of ECSCW '91*, Bannon, L., Robinson, M., Schmidt, K. (eds), Sept 25-27, Kluwer, 1991, pp 17-31.
4. Patterson, J. F., Hill, R. D., Rohall, S. L., Meeks, W. S., 'Rendezvous: An architecture for synchronous multi-user applications', in *Proceedings of CSCW '90*, October 7-10, Los Angeles, Ca., ACM, 1990.
5. Bentley, R., Hughes, J.A., Randall, D., Rodden, T., Sawyer, P., Shapiro, D. and Sommerville, I., 'Ethnographically-informed systems design for air traffic control', in *Proceedings of CSCW '92*, November 1-4, Toronto, Canada, ACM, 1992.
6. Harper, R., Hughes, J. A., Shapiro, D. Z., 'Working in harmony: An examination of computer technology in air traffic control', In *Studies in Computer Supported Cooperative Work. Theory, Practice and Design,* eds J. M. Bowers and S. D. Benford, Amsterdam: North-Holland. 1991.
7. Heath, C., Luff, P., 'Collaborative Activity and Technological Design: Task Coordination in London Underground Control Rooms', in *Proceedings of ECSCW'91*, Bannon, L., Robinson, M., Schmidt, K. (eds), Sept 25-27, Kluwer, 1991, pp 65-80.
8. Suchman, L., 'The Situated Structuring of Cooperative Work', *Inaugural Lecture*, CSCW Centre, Lancaster University, October 3, 1991.
9. Hutchins, E., 'The technology of team navigation', in *Intellectual Teamwork: Social Foundations of Cooperative Work*, Galegher, J., Kraut, R. E., Egido, C. (eds), Hillsdale, New Jersey, Erlbaum, 1990, pp 191-220.
10. Bentley, R., Rodden, T., Sawyer, P., Sommerville, I., 'A prototyping environment for Dynamic Data Visualisation', in *Proceedings of the 5th IFIP TC2\WG2.7 Working Conference on Engineering for Human-Computer Interaction*, Ellivuori, Finland, 10-14 August, C. Unger and J. A. Larson (eds), Elsevier Science, 1992.
11. Lauwers, J. C., Lantz, K. A., 'Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems', in *Proceedings of CHI '90*, April 1-5, Seattle, Washington, ACM, 1990, pp 303-311.
12. Lantz, K. A., 'An experiment in integrated multimedia conferencing', in *Proceedings of CSCW '86*, Austin, Texas, December 1986.
13. Ahuja, S. R., Ensor, J. R., Horn, D. N., 'The Rapport Multimedia Conferencing system', in *Proceedings of the Conference on Office Information Systems* (COIS88), Allen R. B. (ed.), March 23-25, Palo Alto, Ca., 1988.
14. Gust, P., 'Shared X: X in a distributed group work environment', *presented at the 2nd Annual X conference*, MIT, Boston, January 1988.
15. Crowley, T., Milazzo, P., Baker, E., Forsdick H., Tomlinson R., 'MMConf: An infrastructure for building shared multimedia applications', in *Proceedings of CSCW '90*, October 7-10, Los Angeles, Ca., ACM, 1990, pp 329-342.
16. Greenberg, S., Bohnet, R., 'GroupSketch: A multi-user sketchpad for geographically-distributed small groups', in *Proceedings of Graphics Interface*, June 5-7, Calgary, Alberta, 1991.
17. Ellis, C., Gibbs, S. J., Rein, G., 'Design and use of a group editor', Technical report STP-263-88, MCC, September 1988.
18. Took, R., 'Surface Interaction: A paradigm and model for separating application and interface', in *Proceedings of CHI '90*, April 1-5, Seattle, Washington, ACM, 1990, pp 35-42.
19. Larson, J. A., 'A Visual Approach to Browsing in a Database Environment', *IEEE Computer*, 19(6), June 1986, pp 62-70.