Integrating Real-World and Computer-Supported Collaboration in the Presence of Mobility

John Bates, Mark D Spiteri, David Halls and Jean Bacon
University of Cambridge Computer Laboratory
{John.Bates, Mark.Spiteri, David.Halls, Jean.Bacon}@cl.cam.ac.uk

Abstract

We have developed an environment in which Real-World Collaborative Working (RWCW) and Computer-Supported Collaborative Working (CSCW) can be seamlessly integrated. Real-world collaboration usually involves mobile individuals in casual or pre-arranged meetings with colleagues. On the other hand computer-supported collaboration involves using computer-based tools to cooperate, e.g. multimedia conferencing and shared design tools. Our environment generalizes activities from the real-world and computer-supported collaboration as ***events****. This enables computer-based applications to be built around a mixture of both scenarios, e.g. a user, mobile in the real-world, may be using computer-supported tools to collaborate with other users. If the user moves between locations in the physical world, the computer-based tools he/she is using follow, being transported to the nearest, or a selected workstation. Our environment supports the automatic storage of events, both real-world and computer-supported, so that collaborative activities can be later replayed, queried, analyzed and visualized. It is our thesis that this paradigm effectively removes the limitations that each form of collaboration has in isolation, and applications like the ones we have been able to develop using this infra-structure can enhance individual and group productivity.*

1 Introduction

This paper briefly describes the systems infrastructure we have developed to support our collaboration environment in which Computer-Support Collaborative Working (CSCW) is seamlessly integrated with Real-World Collaborative Working (RWCW).

A unifying mechanism for communicating activity, relevant to both CSCW and RWCW, was first required. Activity in CSCW applications takes forms such as text messages, shared drawing strokes and occurrences within live video and audio sessions. RWCW activities require the capturing of as much information as possible from the environment, such as the physical locations of users (e.g. so we can find out when users are together) and what they are doing, e.g. discussing a certain document. To capture these widely different activities, we used the event-based paradigm we have developed in previous work [1], [2], and illustrated in brief in Section 2. This is more powerful than other event environments like CORBA's [3] in that it also provides support for higher level services like trading, brokerage and composition of new events through specification of event sequences. We are particularly interested in location-awareness, since collaborating users may also be mobile. We have thus made components mobile to enable them to follow their mobile users. An architecture for supporting the mobility of real-world users' computing sessions is described in Section 3. A service for the storage and retrieval of such collaborative activities is then outlined in Section 4.

To our knowledge, there is no one previous piece of work that has addressed all of the issues that motivated our work. However, there are several relevant projects that served as inspiration, amongst which are *Media Spaces* [4], [5], groupware toolkits to facilitate collaborative application authoring like *Groupkit* [6] and *SCOOT* [7], and the work on capture, indexing and playback tools done at Xerox PARC [8], [9]. Our mobile paradigm goes further than *Teleporting* [10] in that it does not rely on any windowing paradigm and in contrast to *Migratory Applications* [11] it does not imply the building of monolithic applications. We have developed a system for event storage and retrieval that is sufficiently general to meet any application's requirements, and our mobile system allows stream re-connection after application movement.

1.1 Collaborative test environment

To demonstrate our technologies we built a case-study collaborative environment. This provides familiar

CSCW features including video conferencing, text channels and shared drawing, and a shared web browsing system. We also support the monitoring of RWCW activities, including tracking the location of users, monitoring usage of phones and printers, books being taken out of a library etc. In addition, our video and audio equipment is deployed widely and video movement detection (with some amount of image recognition, and audio speech recognition) is available. Various workstation activities, like logging in and executing applications, are also monitored. Applications were built by gluing together existing components and specifying what actions to take in response to activities. This is also the mechanism for tailoring existing applications. Our applications can dynamically use any activity as a trigger to perform some actions, for example, if two users are seen together in the same real-world location, then play them a video message on the nearest workstation briefing them on the progress of a project.

All collaborative-working sessions are automatically indexed, thus allowing later replay and/or querying of a session. For example, it is possible to automate minute taking for the CSCW version of our research group's weekly meeting. We can replay a meeting from any point to any other point within the collaboration. The replaying positions can be specified in temporal terms relative to the

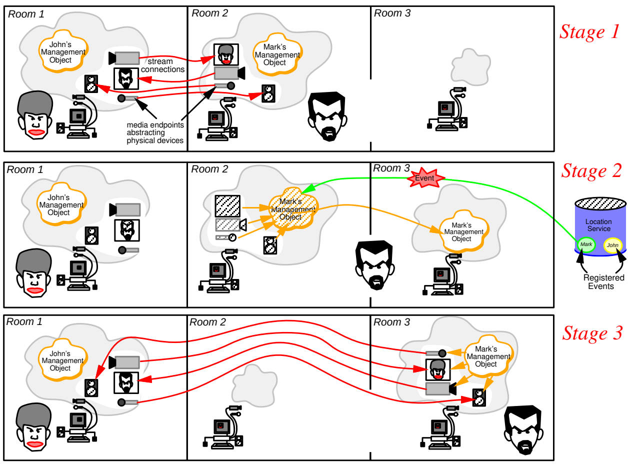

Figure 1. Application Movement

beginning of the meeting, or with respect to *any other event or sequence of events* within the meeting. We can perform high-level queries, such as "show me the URL of the slide that John showed after Giles joined the meeting" and "show me the name of the book I took out of the library last month, just before I met Pawel in the corridor".

Our collaborative working environment takes into account users' physical location. In our test environment, it is possible to track the location of all users through their Active Badges [12]. The environment can detect when a user moves, and if a user moves to a new location, the application detects this movement and his/her portion of the application follows them to the nearest, or a selected computer. This enables users to keep in touch with one another whilst on-the-move. When the application pops up at the new location, all state is preserved and any connections to other users are recreated. This is illustrated in Figure 1.

2 Building collaborative systems using events

The underlying framework to our system is that of distributed event-based programming. We have used this general-purpose paradigm to support many application areas but it is particularly suited to collaborative and mobile multimedia applications.

An event is an asynchronous message containing details of a collaboration activity that has occurred. By using events as a uniform activity metric, applications can be built around a generic event-response paradigm. Single or ordered combinations of events can be used to trigger further actions within the application. The uniform interchange of activity between the possibly distributed components making up an application simplifies the construction of a complex collaborative application. Also, since events can be manipulated generically, support for manipulating them in programs, and for storage and retrieval of them, can also be constructed, cutting down on application development overhead.

Event-based programming relies on two main concepts:

- Existing event-based components are used as the building blocks of applications. These may be distributed around the local network or Internet. They can inform interested users of pertinent events, such as two users meeting in RWCW or updates to a shared diagram in CSCW.
- In order to compose many event-based components into an application, a management component must be written. This is the bespoke part of the application. Its main functionality is composed of rules that describe what actions to take if certain events are received from components making up the application. Rules act as distributed systems integration *glue*, using input from one set of services to drive another set. Examples are moving a user interface to the application owner's new location in response to an event that the user has moved, or reconfiguring a multimedia application in response to an event detailing a change in bandwidth availability.

2.1 Event classification

One event source can generate events of one or more event classes. An event class has typed attributes, instances of which uniquely identify a captured activity. For example, an event source that provides information about the locations of users can offer monitoring facilities for the following class of event: `LocationEvent(Domain, Name, Type, Location)`. Class `LocationEvent` identifies that an entity (a person or equipment) has changed location in a specific domain.

Distributed event-based objects advertise the event classes they can monitor for by exporting them to an *event broker*. Clients can query the event broker, to search for appropriate event sources. Event brokers can be federated, to provide access to the event sources of other domains.

Inheritance can be used to create sub-classes of events. In our design, the event broker is the custodian of the class hierarchy for its local event domain. Querying the event broker for a particular class of event returns all event sources that can monitor for the class and all its sub-classes.

2.2 Registration for filtering

In order to reduce event traffic a client of the event source can perform filtering to decide whether a particular event is of interest. Only events that match registered patterns are dispatched to the interested parties, thus reducing network traffic and filtering requirements.

To register interest with an event source, clients provide an *event template*. This is an event instance, with fields for exact match filled in and those for wildcard match expressed as variables. An example for class `LocationEvent` is as follows:

- `LocationEvent("cl.cam.ac.uk", "John.Bates", "Person", L)`

  Report wherever John is seen in the Computer Lab.

When event instances are notified to clients, values for all class attributes are given. As a comparison to another event-based component, the drawing board advertises event class `LineDrawn(user,x1,x2,y1,y2)`. This allows us to find out about lines that are drawn on the board. We will show how this is useful shortly.

2.3 Management of event injection

Event-based components also have an interface for event injection. This allows a client of a component to inject events into it, which the component must respond to as if the events had happened internally. The principle behind building cooperative applications using events is that the controlling part of one user's collaborative application updates the local state in response to notification of registered events.

An event-based component need not be designed to work specifically with any other component. But as long as it has event registration and injection interfaces it can be

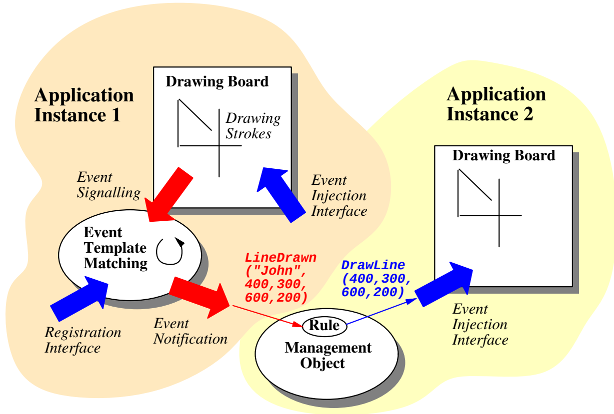

Figure 2. Event Injection in shared drawing board

used in a multitude of ways. In the case of a shared drawing board, a management component can register interest in drawing events with many different types of drawing board. When it is notified it can inject the event into the local board, thus creating a shared drawing application. Similarly, other users register with our local board and we send out events to all of them whenever we draw on the board. The stand-alone drawing boards can be used in many contexts with any complexity managed by a simple set of event-driven rules.

Of significance is that notification of *any* class of event can lead to the injection of *any* action. Consider the example of "when Jean and Mark are in the same location play them a video message". In this case, events associated with location trigger injection into a media presentation component on the nearest workstation.

2.4 Event sources

Systems support for event-based components. Our prototype event-based components were built initially using a locally developed distributed objects system called MSRPC, although we are now using a CORBA platform. We have developed event libraries to assist the component builder, providing facilities for managing the processes of registration and notification, and managing data structures for registered event templates. Each event source is responsible for monitoring for raw event occurrences and then signaling them to the event subsystem.

Stand-alone interaction components. We have described how drawing boards can be interfaced, using events, to create shared drawing boards. Many such stand-alone reusable components can be envisaged. For our experiments we built the drawing board, a text I/O component that generates events when sentences are typed and which can accept injected sentences, and a *Web Projector* applet. When one user selects a URL, rules are triggered that inject events to cause the other users' browsers to go to the same page.

Location-awareness. The physical realities of detecting users and equipment is through electronic tagging. In our experiments we use Active Badges that transmit an infra-red signature every 5 seconds. A network of detectors in rooms receive the signals and report the badge's position to the location service.

Other event sources. Some event sources are relevant to both RWCW and CSCW. An example is monitoring user work patterns on a workstation and recording what commands he/she is running. It is thus possible to deduce that since John was using an editor and LaTeX, he was working on a document. A phone exchange can be monitored to generate events when phones in specific locations are off the hook, on the hook, or ringing. An example application of this service is, if the phone rings in a room where a meeting is taking place, redirect the call to voicemail.

Continuous media components. Continuous Media (CM) components include video sources, video windows, audio sources, audio sinks, video image recognizers and audio speech recognizers. These differ from the other types of event-based components in that they can also be connected to other compatible components using streams. CM components also have associated event classes, e.g. a video window notifies a client if an area of video is clicked on and a video source can inform a client of the frame number currently being presented. These CM components are relevant to both CSCW and RWCW. In CSCW, CM components provide the building blocks for multi-party video conferencing. In RWCW, linked with location-awareness, CM components enable a user to instigate a video link-up to an existing meeting.

Locating/creating event sources: Some event sources are long-lived, such as the location service. Others are created on-the-fly for a single user, such as continuous media components. This latter case often requires a component to be created on a specific host, e.g. to see video from the meeting room requires a video source component to be created on the machine that manages the camera hardware. To this end, factories manage resources, such as cameras, and support the dynamic creation and destruction of components. Factories are deployed throughout our environment.

3 Mobile collaborative applications

Mobile collaborative applications assume that users are interested in being physically mobile whilst using collaborative applications. Taking a simple example of a video conference, each user has his/her own set of components managing his/her part of the application. If a user moves his/her physical location, then the application management component receives one or more events describing his/her movements and his/her final location. It is then able to reconfigure the user's components so that they pop up on the nearest, or a selected workstation. This section describes the additional infra-structure to make such an application possible.

Location information database. We provide a database service, containing the following information; the names of rooms and their geography, the equipment stored in each room, and the names and capabilities of pieces of equipment. If an application is following a user or is dispatched to a new location, it can check that the required facilities are available there. If the application requires video and audio and these facilities are not supported in a particular location then it can adapt and use text-only conferencing instead.

Mobile components. It can be necessary to move some or all of a user's collaboration components from one computer to another. For each component, this involves moving any associated state, such as the current state of a drawing board surface. It also involves hand-over of any communications channels; e.g. the video streams from a user's camera component, which are connected to other users' video windows, must be changed to originate from a camera at his/her new location. Using information from a location event, the location database can be queried to find a workstation with the required facilities in the location. We implemented event-based collaboration components with lightweight state-saving and management components using mobile agents [13]. Collaboration components do not usually have complex state; their associated state is usually limited to user interface and communications state. Thus, lightweight state-saving provides all the state-saving required, while avoiding the overhead of employing an interpreted agent language. Agents are more appropriate for management components, since they support complex state saving and move more easily inter-domain. In our system, the mobile management component controls the process of moving collaboration components, by saving their state and re-instantiating them.

Naming and hand-over. When components move, they may be communicating with other components, e.g. a video source representing one user's camera transmitting video to another user's video window. These connections are suspended and, after movement, restored. In the video example, a new video source object on another workstation will take over the function of Mark's video source, but the new source still should be contactable as Mark's video source.

To locate objects, we have built a trading service that supports a rich object naming scheme, based on SGML tags. An object advertises its functions with a trading service. Clients of the trader can query objects associatively. By using information from traders to update large information repositories it is possible to search and locate event services worldwide, based on their characteristics.

If an object moves, a mechanism to alert applications that may try to contact it is required. In our implementation, an object in the process of moving can mark its reference in the trader as `mobile`. When an object has moved and the hand-over process is complete, the reference can be updated with the new object location. If an object has moved to another domain, it converts its trader entry in the old domain to a *tombstone* (forwarding reference).

4 Storage and retrieval of events

In this section we briefly describe the event store we have designed and constructed to allow the storage and retrieval of sessions of events. The motivation here is that events represent indexing points into activities. They enable an activity to be reconstructed. Taking the example of a multimedia conference, it is possible to replay the conference by examining the events that occurred in the session from beginning to end. Figure 3 illustrates the architecture of the event store.

Sessions provide a context to associate related event instances, particularly useful for limiting scope of retrieval and replay. Any class of event can be stored in a session, and an event instance can appear in multiple sessions.

Selection operations can be applied to a session, for example, to find out the URL that Oliver displayed at the end of a conference yesterday. The event store can provide immediate responses to multiple concurrent queries and it can also replay in real-time. This involves outputting events in the order in which they occurred, and with the appropriate relative temporal delays between successive events, thus allowing a user to re-view a session as it actually happened.

Event stores are lightweight services, allowing each user to run his/her own instance, rather than relying on a centralized repository. Some users may be interested in storing different classes of events from others, e.g. one user may wish to retain only the names of documents displayed, whereas another may record every event to allow him/her to replay the session as it happened. Allowing users to have individual event stores can also be useful if users become mobile or get disconnected. Media streams associated with a collaborative session can also be stored; for example, video and audio streams, of at least the person who is speaking, can be stored in a continuous media repository. Events can provide references to locate media objects in repositories, as well as annotations that act as indexing points into the object's content. For instance, a video recording component can generate an event specifying that it is now recording the current speaker, and containing the output file name. This event can be used later to locate the raw video data.

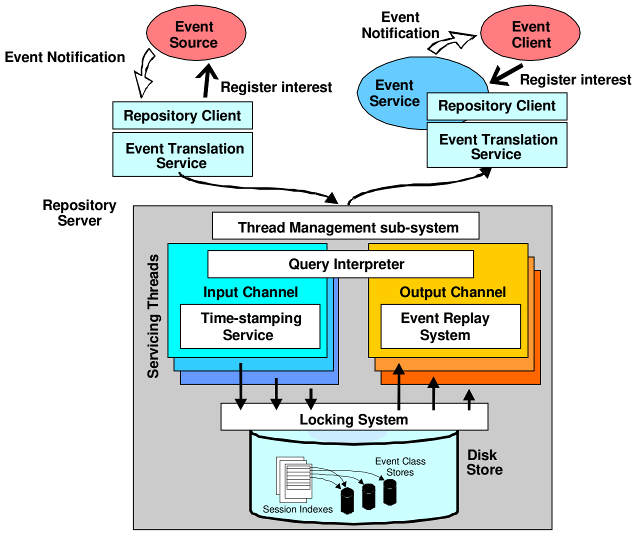

Figure 3. Event Store Architecture

4.1 Replay of missed events and support for disconnected operation

Event loss can occur due to object mobility or network partitioning. To address this problem in a multi-party conference, each participant runs an instance of the event store. Once connection of a disconnected participant is re-established, his/her management object can replay the events generated during the disconnected period by contacting event stores that were able to record the session events. In order to avoid the problem of inconsistency between users' event stores, the re-connected user can request each individual user's store to replay only the events generated locally by that user. An enhancement, to show a user what happened whilst he/she was disconnected, is to create exact copies of all local objects and to use the replayed events to update the copies. The user can then decide whether or not to add these updates to his/her session. If the disconnection was due to network partitioning and not mobility, then this approach applies to all users, since disconnected working in one or more partitions may have occurred. In this case, all users can discuss what happened in the conference partitions, and decide which of the event sequences they wish to be added to the session.

4.2 Querying and replaying

We provide a language to manipulate stored event classes and instances, as well as event sessions. Queries can be composed through a query interface provided by the event store, either interactively, in advance, or dynamically, by application components. Special keywords and new constructs are provided to express contextual-temporal relationships and notions of temporal intervals in which one or more sequences of events might have occurred. For example;

- "*Replay from the group's diary from when Mark entered room T12 and John was there.*" One way of expressing this is to find the period when John was in T12 and then see if Mark was seen within that period in the same room. If this is successful, then replay from that instance onwards.
```
replay fromsession OperaDiary
fromwhere (badge.Name = 'Mark.Spiteri' and
badge.location = 'T12')
during (: (badge.Name = 'John.Bates' and
badge.Location = 'T12')
until (badge.Name = 'John.Bates' and
badge.Location != 'T12') :)
```
- "*Retrieve when Jean walked from the tea-room to her office (T15) and started editing a document sometime last week.*" Instead of replaying this will return all event sequences that match the specification provided. A simple way of specifying this is:
```
retrieve fromsession JeanDiary
eventsOf( [:(badge.Name = 'Jean.Bacon' and
badge.Location = 'TeaRoom')
followedby (badge.Name = 'Jean.Bacon' and
badge.Location = 'T15')
followedby (workstation.documentedit.user =
'Jean.Bacon'):]
during (: 10/02/1998 to 17/02/1998 :) )
```

One can find events relative to other events that have occurred before or after them, as well as with respect to specified bounded intervals. Intervals can be operated on to determine overlapping periods, and if desired, output can be filtered so that only specific event types, or events with specific parameters, are returned.

Conclusion

We have built a collaboration environment in which applications can be built to support communication between groups of users, some of whom will be mobile, some computer-based, and some both. Allowing users to collaborate across the boundaries of computing worlds enhances flexibility of working and thus potential productivity. Our environment also recognises that an important role of meetings is the agreement of strategies and the definition of subsequent actions to be taken by the participants. To support this in our collaboration environment, activities can be stored by the system. Flexible query and replay of activities is available via a specially-developed language.

Using our infra-structure, we have built advanced co-operative applications for multiple users, involving shared drawing, text, video and audio conferencing and shared projection. On-the-fly application development is possible through users adding event-driven rules to their applications. Event sources to drive the rules include the collaborative components of the application as well as real-world events, such as changes in the location and activities of users. Experiments have demonstrated the benefits of mobile applications and the storage and retrieval of event sessions.

References

[1] J. Bacon, J. Bates, R. Hayton, and K. Moody, "Using events to build distributed applications", in *Proceedings of 2nd International Workshop on Services for Distributed and Networked Environments*, IEEE, 1995.

[2] J. Bates, "A framework to support large-scale active applications", in *Proceedings of SIGOPS European Workshop*, ACM, 1996.

[3] Object Management Group (OMG), "CORBA Services Specification", at http://www.omg.org/corba/sectran1.htm

[4] S. Bly et al., "Media spaces: Bringing people together in a video, audio and computing environment", *Communications of the ACM*, ACM, January 1993.

[5] R. Kraut, R. Fish, and B. Chalfonte, "The VideoWindows System in informal communication", in *Proceedings of the Conference on Computer-Supported Cooperative Work*, California, 1990.

[6] M. Roseman and S. Greenberg, "Building real-time Groupware with Groupkit", *ACM Transactions on CHI*, 3, 1995.

[7] E. Craighill, M. Fong, K. Skinner, R. Lang, and K. Gruenefeldt, "SCOOT: An object-oriented toolkit for multimedia collaboration", in *Proceedings of ACM Multimedia*, ACM, San Francisco, 1994.

[8] S. Minneman and S. Harrison, "Where Were We: Making and using near-synchronous pre-narrative video", in *Proceedings of ACM Multimedia*, 1993.

[9] S. Minneman, S. Harrison, B. Janssen, and G. Kurtenbach, "A confederation of tools for capturing and accessing collaborative activity", in *Proceedings of ACM Multimedia*, 1995.

[10] T. Richardson, F. Bennett, G. Mapp, and A. Hopper, "Teleporting in an X Window System Environment", *IEEE Personal Communications*, August 1994.

[11] K. Bharat and L. Cardelli, "Migratory Applications", in *Proceedings of ACM Symposium on User Interface Software and Technology*, Pittsburgh, USA, November 1995.

[12] A. Harter and A. Hopper, "A distributed location system for the active office", *IEEE Network*, IEEE, 8(1), 1994.

[13] J. Bates, D. Halls, and J. Bacon, "A Framework to Support Mobile Users of Multimedia Applications", *Mobile Networks and Nomadic Applications* (NOMAD), ACM, 1, 1996.