In Proceedings of APCHI'98, July 15-17, Japan, IEEE CS Press.

Human Interaction Issues for User-configurable Collaborative Editing Components

John Grundy

Department of Computer Science, University of Waikato
Private Bag 3105, Hamilton, New Zealand
jgrundy@cs.waikato.ac.nz

Abstract

The ability to synchronously and asynchronously edit work artefacts has become very important in many editing tools. However, most tools usually only provide one kind of collaborative editing "level", or provide incompatible levels of collaborative editing. We describe our recent work in adding flexible, user-configurable collaborative editing facilities to component-based design environments, and focuses on the human interaction issues in such systems. We also briefly describe the engineering of such tools using a component-based approach, which allows user-configurable collaborative editing capabilities to be added to component-based tools without modifying the tool or collaboration component implementations.

1. Introduction

Users of many editing tools require facilities to support collaborative editing i.e. multi-user collaboration on the development of work artefacts. This is common when using tools such as CASE (Computer-Aided Software Engineering) tools, CAD (Computer-Aided Design) tools, document editors, and drawing packages. The "level" of collaborative editing supported by the tools is often either synchronous i.e. "What You See Is What I See", where as one user changes an artefact other users see the exact same changes in "real time", or asynchronous, where users independently modify alternative versions of an artefact, then merge their changes at a later date. Examples of synchronous collaborative editing tools include CoolTalk [15], MS NetMeeting [13], and GroupKit [16]. Asynchronous collaborative work is supported by BSCW [2], wOrlds [3] and Mjølner [12]. Some systems provide integrated support for both levels of collaboration, for example TeamRooms [17], Oz [1], SPADE/ImagineDesk [4], and W4 [6]. However, most such systems completely separate support for different editing modes, and only a few, such as SEPIA [19], allow the same work artefact to be synchronously or asynchronous edited with users able to change mode of collaboration at will.

We have found in our work developing CAD, CASE and other design tools that a range of collaborative editing facilities are generally required by users, and users should be able to seamlessly move between levels of collaborative editing. For example, consider the JComposer OO CASE tool we have developed, a view (diagram) from which is shown in Figure 1 [7, 9]. This is an example of specifying an ER modeller meta-model, and an ER editing tool will be generated from this model (and other views specifying different aspects of the tool). Users of JComposer views often want to independently (asynchronously) edit versions of the same view, then have one user merge the changes to produce a new OO diagram. Sometimes users want to synchronously edit the views and discuss changes via audio and/or text chat synchronous communication. At other times users want to be informed of changes made to another version of a view by other developers, but retain control over applying them to his/her version of the view incrementally. At other times, users simply want to be informed when a particular change or sequence of changes are made by another user to a shared view.

From a tool architecture perspective, such collaborative editing capabilities are useful in many JComposer-like environments. However, groupware capabilities are usually hard-coded into most systems, with the decision to support collaborative work having to be made early in the system's development lifecycle [10]. Some proposals for pluggable, reusable groupware facilities, for example using CORBA-style remote object management, have been made [11, 5], or using custom architectures [18], but usually such proposals don't support the degree of user-configurable, flexible facilities we desire. We have developed pluggable components which provide the kind of groupware editing capabilities described above, and can be reused in different environments which have a component-based software architecture, like that of JComposer. We have reused these groupware components in an ER modelling tool and a process modelling environment, as well as in JComposer, without having to change the implementation of any of these environments or the groupware components themselves.

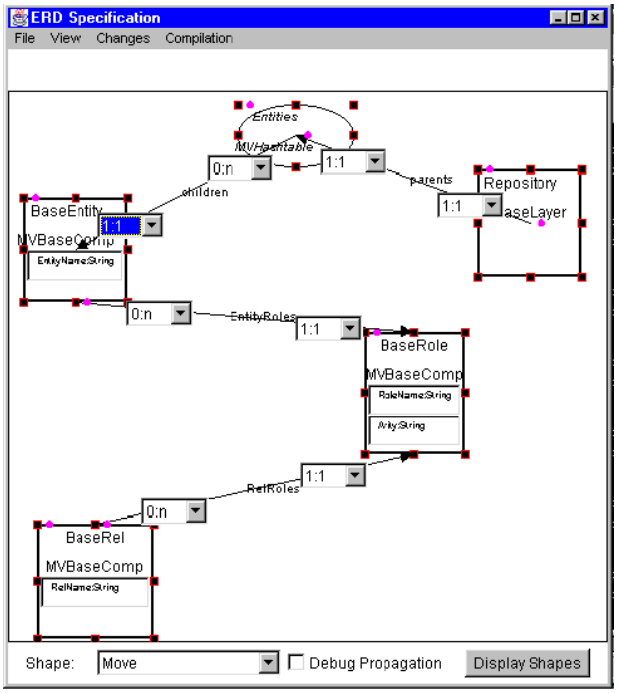
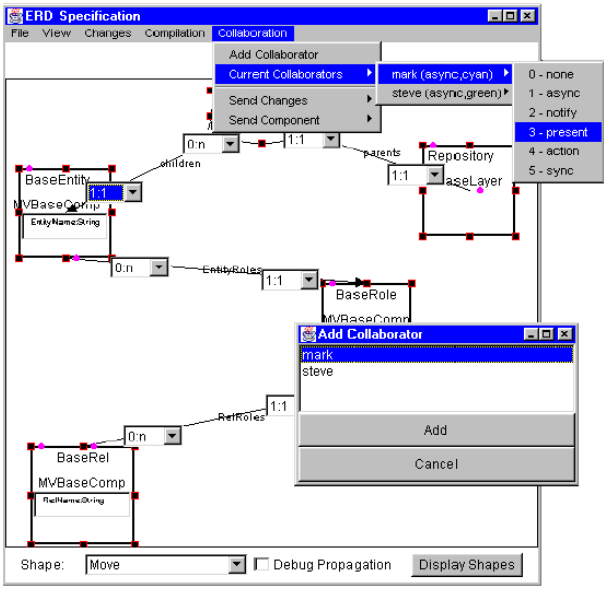


Figure 1. Example view from JComposer.

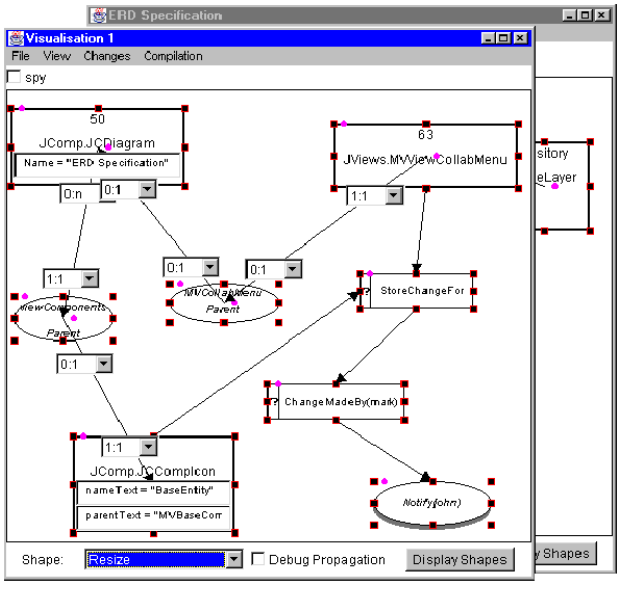
2. Specifying Collaboration Levels

The groupware components we have plugged into JComposer use a "Collaboration" menu item to allow users to configure the kind of collaborative editing for a view. We chose this approach as it allows users to easily configure collaboration with others from one place, and provides some basic awareness capabilities (informing the user who is registered for collaboration and at what level). Figure 2 (a) shows user "John" who is editing the ER modeller specification from Figure 1 in JComposer. John has specified two people he wants to collaboratively edit this view with via the "Add Collaborator" item, and is viewing the collaboration level of these people. John can change the level by selecting a "level" from the range 0 (none - i.e. remove the collaborator) to 5 (fully synchronous editing). John (or one of his collaborators) can change the collaboration level with a particular user at any, as required, and when John changes the collaboration level for this view for a specified user, the collaboration level for the user's view with "John" is set to the same level. Different views can be collaboratively edited at different levels as desired.

At times users want changes made by other users automatically processed. This is particularly true in the "notify" level (2), where changes made to a view are sent to collaborators, but nothing explicitly is done with them (vs. presentation, level 3, where the changes are presented in a dialogue box to other users). Tools built with the same architecture as JComposer provide an event handling language that allows users to configure such behaviour (see [8] for details of this language). In Figure 2 (b) "John" has specified that if any changes made by "Mark" to the view component named "BaseEntity" are received, John should be notified by a message. The Visualisation diagram where this has been specified shows running components implementing the "ERD Specification" view in JComposer (e.g. component number 55 is the ERD Specification diagram component).



(a) Configuring collaboration levels for views.



(b) Configuring handling of broadcast changes.

Figure 2. Configuring the collaboration "level" with other and the handling of broadcast changes.

3. Interface Issues During Editing

In this section we illustrate the different levels supported by our collaborative editing components, and their user interface characteristics. We have at this stage opted for simple interface widgets and group awareness capabilities, all of which could be extended in the future.

3.1. Level 1 - Asynchronous Editing

Asynchronous editing is illustrated in Figure 3, where "John" has made some changes to a view in JComposer named "ERD View Spec #1" (a). The two bottom items in the Collaboration menu allow John to "send" the changes he has made to other users, or to send the whole view specification itself. John has sent the view using the "Send Component" menu to user "Mark", then sent changes made to this view since the view was sent to Mark. These changes are being presented to Mark in a dialogue (b). Mark can choose to have all the changes made to his version of this view, by selecting them all then clicking the "redo" button. Or he can selectively apply some of the changes only to his view, merging in a subset of the changes made by John. He can also make other changes manually to the view, then send these to John with his "Send Changes" collaboration menu item. Any changes which can't be made (e.g. Mark has deleted a view component John has edited) are shown by Marked "invalid". Users can then discuss how to handle such view merging inconsistencies.

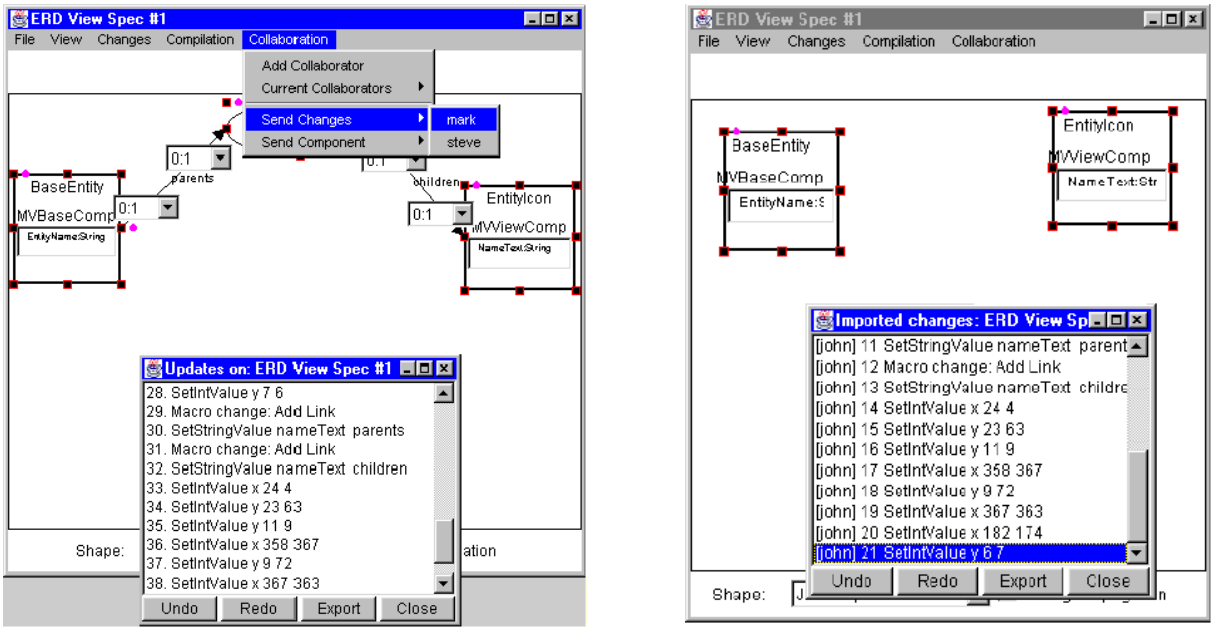
Note changes are not broadcast to another user's environment unless the sender explicitly requests this via the Send Changes item. We chose this approach to give the person making changes control over their distribution, but could extend our collaborative editing components to allow e.g. Mark to request all new changes from John's view, without having to ask John to send them via separate email, chat or audio communication channels.

3.2. Level 2 - Notification of Changes

Changes made to a view are broadcast to another user's view, where they are then broadcast to any "listeners" of the view i.e. other components interested in changes made to the view. The changes are not presented to the user or actioned in any way, but filter/action event handlers, like the one shown in Figure 2 (a), may have been specified by the other user to detect and act on the change. Changes broadcast from another users environment are "tagged" with the other user's name, distinguishing them from the user's changes.

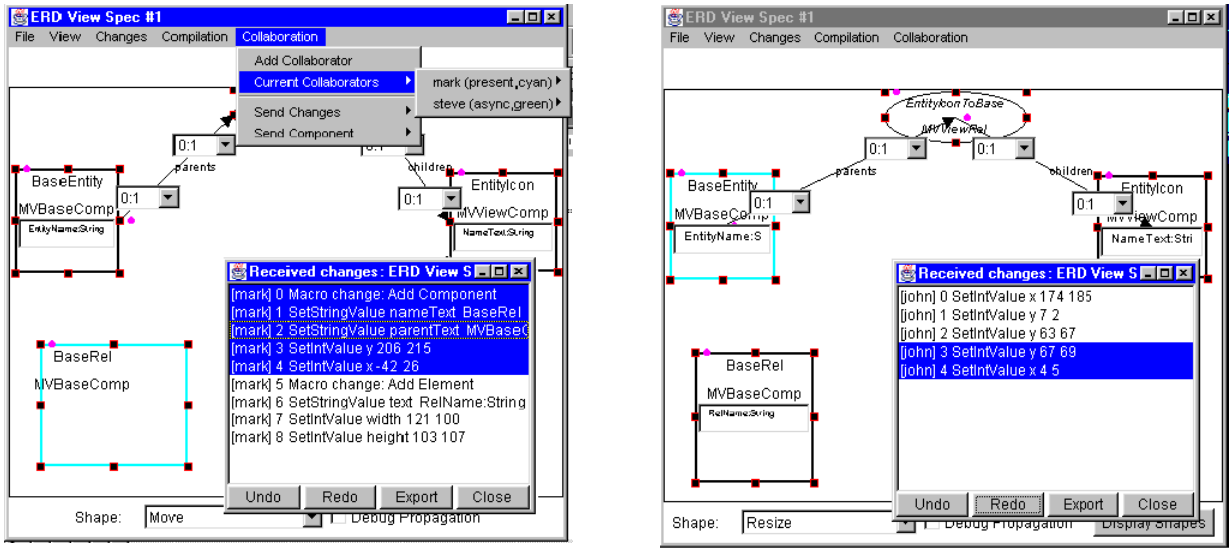
3.3. Level 3 - Presenting Changes

Presenting changes is a mode of collaborative editing "between" asynchronous and synchronous editing. Figure 4 shows John and Mark now editing the ERD View Spec #1 view, with changes made by other collaborators presented in a dialogue box, annotated with the name of the other user who made the change. The user can select particular changes or sets of changes and have them performed on his/her view by clicking on the "redo" button. Clicking on the "undo" button will reverse the changes if they turn out to be inappropriate.



(a) John's view. (b) Mark's view.

Figure 3. Asynchronous editing of a JComposer view.



(a) John's view.

(b). Mark's view.

Figure 4. Presenting changes as they are made by other users.

When a change is applied to the user's view in this manner, the icon affected by the change is coloured to match the colour associated with the collaborator in the Collaboration menu. This assists users to identify parts of a view last modified by collaborator actions.

This mode of collaboration we have found very useful when a "looser" form of collaboration is required by collaborators, and when version merging is to be performed by more than one user at the same time. It has advantages over asynchronous editing in that users are kept aware of changes made by other collaborators as they occur. It has advantages over synchronous editing in that users' views are not automatically changed when a collaborator changes the view, allowing users to "incrementally" merge changes into a view.

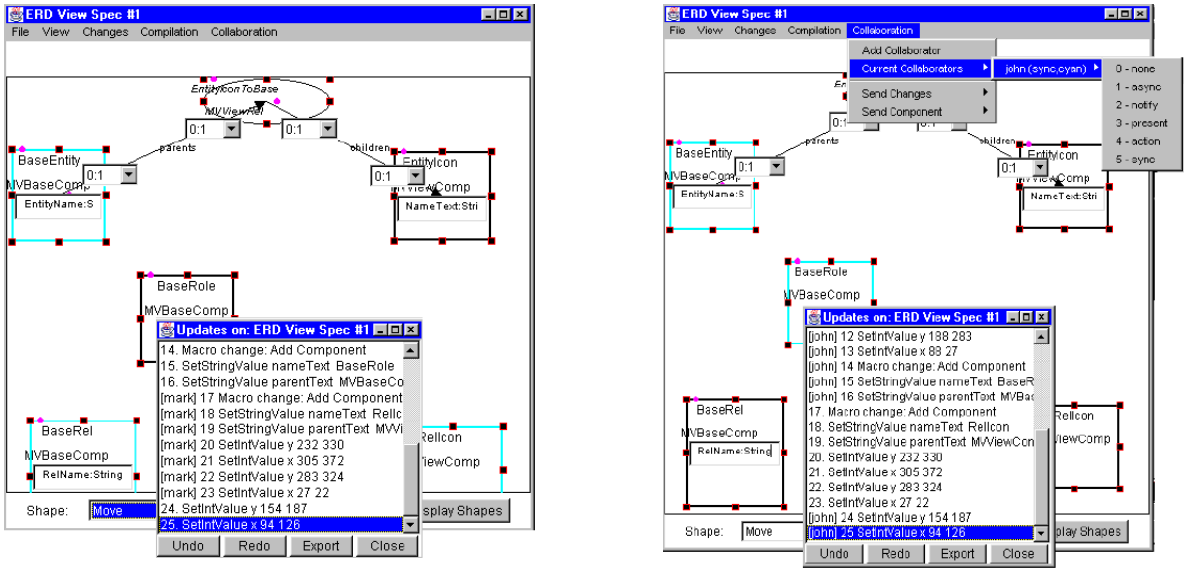
3.4. Levels 4 and 5 - Synchronous Editing

These levels of editing provide the typical "what you see is what I see" mode of editing common in many real-time groupware applications [16]. Actioning changes (level 4) is similar to presentation (level 3) except instead of storing and presenting received changes in a dialogue box, the changes are immediately applied to the receiving user(s) view. In this mode of operation, no locking of view components is used, meaning two (or more) collaborators can edit the same view artefact at the same time. This is resolved when the competing changes are received by other collaborators, and one is immediately superseded by another in a non-deterministic fashion. Changes are still stored in a dialogue box so any simultaneous edits can be reviewed and discussed further. This mode of editing works well when complemented by audio and/or text chat communication.

Level 5 (synchronous) editing, illustrated in Figure 5 adds locking of view components to ensure no simultaneous edits occur. Whenever a user attempts to modify a view component, a lock request is sent to other collaborators in "synchronous" edit mode on this view. If the view component is unlocked, all users environments allow the current user to edit it. If any other user has requested a lock simultaneously, all requests are rejected with the affected icon coloured red, and users must attempt to edit the component again. In both actioning and synchronous editing modes, affected icons are coloured, to indicate the last person to change the icon. Users can review the view change history, with name annotations on changes, as shown in Figure 5.

Mixed-levels of Editing Views

The different modes of collaborative editing we have illustrated above can be used on different views e.g. one view edited synchronously while another in presentation mode. In addition, different users can be editing the same view using different levels of collaboration, for example John and Mark at the synchronous level, but Mark and Steve editing the view in asynchronous or presentation levels. When Mark merges any changes from Steve, these are synchronously sent to and applied by John's view, due to the level of collaborative editing between Mark and John. John and Mark may then decide to move to a different level of editing for the view, or John to add Steve as a collaborator. This can be done simply by using the collaboration menu to change the mode of operation. This ability to easily mix "levels" of collaborative editing has proved to be very useful in design environments like JComposer.



(a). John's view. (b). Mark's view.

Figure 5. Synchronous editing of JComposer views.

4. A Supporting Software Architecture

We did not want to make large-scale modifications to retro-fit collaborative facilities onto JComposer, or other design environments we have developed and are developing. Instead, as JComposer and our other design environments use a component-based software architecture, we built reusable, pluggable software components to embody these facilities. JComposer was developed using the JViews component-based software architecture [7], a specialisation of the Java Beans componentware API of Java 1.1 [14]. Software components are parts of software systems which can be readily "plugged" into different software applications, often when the software is actually running and in use, rather than being software libraries or class frameworks which are chosen at compile-time. Advantages of components-based approaches to building software is that they allow software to be incrementally extended and upgraded, and often allow end-users of software to reconfigure their systems using third-party reusable components.

Figure 7 shows the architecture of our JComposer environment, and the groupware components we added to JComposer to support the flexible collaborative editing facilities described in the previous section. The grey-coloured items are existing JComposer components and links [7], the black components and links are those we have added to support collaborative view editing. For each view which is to be collaboratively edited, a "Collaborative Menu" component is attached to the view component, and is sent changes from the view component and the view's "version record" component (where changes made to the view are stored). The Collaborative Menu component then sends changes to other collaborators' "collaboration server", as required.

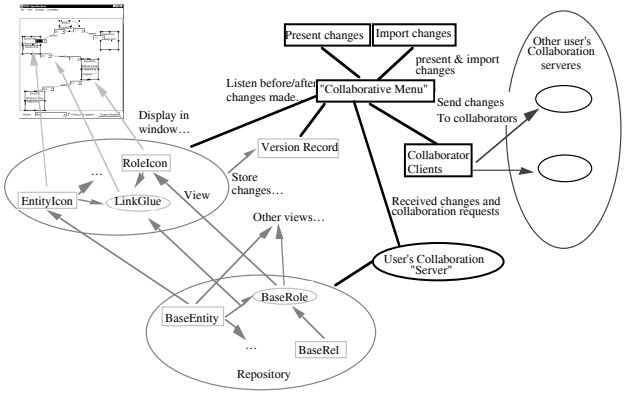


Figure 6. Groupware components in JComposer.

Each JComposer environment for a different user has its own "collaboration server", which responds to collaborative edits and collaboration requests by forwarding these to the appropriate view's collaboration menu component for actioning. This approach is used to provide robust, efficient propagation of changes (no central server is used, but rather selective point-to-point(s) broadcasting). The collaboration menu component for each view has a "present changes" and an "import changes" version record instance used to support asynchronous and presentation collaboration levels. Notify level changes are supported by the collaboration menu sending received changes to any user-defined component listening to it. Actioning and synchronous editing are supported by the collaboration menu actioning received changes, or broadcasting a lock message to collaborators then actioning received changes.

This component-based approach to adding collaborative editing support to JComposer (and other JViews-based environments) is possible due to the JViews approach of broadcasting change description objects to linked components before and after the change has been actioned. We also utilise JViews component persistency mechanisms to serialise view and view change description objects for transmission via socket connections to support

the broadcasting of changes and view descriptions. We have plugged these collaborative editing facilities into the JComposer OO design tool, an ER modeller, and a graphical process modelling tool. Neither these existing tools nor our collaborative editing components needed any modification to achieve this.

Difficulties in adopting this approach involved providing a locking mechanism for synchronous editing and for handling simultaneous edits for the action changes level of collaboration. We also encountered limitations in the presentation of change descriptions and the amount of highlighting of JComposer components that could be done without changing existing code. An unsolved issue occurs when a user is collaboratively editing different views in JViews-based environments using our collaborative facilities, but different levels of collaboration are used with different collaborators for these views. Better awareness indicating which components in a view are being editing in other views would be useful, but it is unclear just how this should be indicated.

5. Summary

We have described flexible, user configurable collaborative editing facilities that have been added in a seamless fashion to an existing OO design tool. These allow users to collaboratively edit using asynchronous, semi-synchronous and synchronous collaboration, and at any time to be able to change collaboration level, add new collaborators or remove collaborators. Multiple views can be simultaneously edited using different collaboration levels and collaborators. We have utilised a component-based approach to adding these facilities to design tools, which necessitates no modification to these tools. Our collaborative editing support components can themselves be upgraded and reused in diverse environments to support a flexible range of collaborative editing needs.

References

[1] Ben-Shaul, I.Z., Heineman, G.T., Popovich, S.S., Skopp, P.D. amd Tong, A.Z., and Valetto, G., "Integrating Groupware and Process Technologies in the Oz Environment," in *Proc. 9th International Software Process Workshop,* Ghezzi, C., IEEE CS Press, Airlie, VA, October 1994, pp. 114-116.

[2] Bentley, R., Horstmann, T., Sikkel, K., and Trevor, J., "Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system," in *Proc. of the 4th International WWW Conference,* Boston, MA, December 1995.

[3] Bogia, D.P. and Kaplan, S.M., "Flexibility and Control for Dynamic Workflows in the wOrlds Environment," in *Proc. of the Conference on Organisational Computing Systems,* ACM Press, Milpitas, CA, November 1995.

[4] Di Nitto, E. and Fuggetta, A., "Integrating process technology and CSCW," in *Proc. of 4th European Workshop on Software Process Technology,* LNCS, Springer-Verlag, Leiden, The Nederlands, April 1995.

[5] Emmerich, W., "An Architecture for Viewpoint Environments Based on OMG/CORBA," in *Proc. of Viewpoints'96* ACM Press, 1996, pp. 207-211.

[6] Gianoutsos, S. and Grundy, J., "Collaborative work with the World Wide Web: adding CSCW support to a Web browser," in *Proc. Oz-CSCW'96*, University of Queensland, Australia, August 1996, pp. 14-21.

[7] Grundy, J.C., Mugridge, W.B., and Hosking, J.G., "A Java-based toolkit for the construction of multi-view editing systems," in *Proc. 2nd Component Users Conference,* Munich, Germany, July 14-18 1997.

[8] Grundy, J.C. and Hosking, J.G., "Serendipity: integrated environment support for process modelling, enactment and work coordination," *Automated Software Engineering*, vol. 5, no. 1.

[9] Grundy, J.C., Hosking, J.G., and Mugridge, W.B., "Support for end user specification of workflows, work coordination and tool integration," *Journal of End User Computing*, vol. 10, no. 2.

[10] Hill, R.D., Brinck, T., Rohall, S.L., Patterson, J.F., and Wilner, W., "The Rendezvous Architecture and Language for Constructing Multi-User Applications," *ACM Transactions on Computer-Human Interaction*, vol. 1, no. 2, 81-125, June 1994.

[11] ter Hofte, H., Van de Lugt, H., and Bakker, H., "A CORBA Platform for Component Groupware," in *Proc. OZCHI96 Workshop on the Next Generation of CSCW Systems,* November 1996.

[12] Magnusson, B., Asklund, U., and Minör, S., "Fine-grained Revision Control for Collaborative Software Development ," in *Proc. of the1993 ACM SIGSOFT Conference on Foundations of Software Engineering,* Los Angeles CA, December 1993, pp. 7-10.

[13] Microsoft, Inc, *Microsoft NetMeeting 2.1*, 1998. See: http://www.microsoft.com/netmeeting/.

[14] Microsystems, S., "Java Beans 1.0 Specification," 1997. See: http://www.javasoft.com/

[15] Netscape, Inc, *CoolTalk for Netscape Navigator,* 1996. http://home.netscape.com/comprod/products/.

[16] Roseman, M. and Greenberg, S., "Building Real Time Groupware with GroupKit, A Groupware Toolkit," *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 1, 1-37, March 1996.

[17] Roseman, M. and Greenberg, S., "A Tour of Teamrooms," in *Video Proc. of ACM SIGCHI'97,* ACM Press, Atlanta, Georgia, March 22-27 1997.

[18] Roseman, M. and Greenberg, S., "Simplifying Component Development in an Integrated Groupware Environment. ," in *Proc. of the ACM UIST'97 Conference,* ACM Press, 1997.

[19] Streitz, S., Haake, J.M., Hannemann, J., Lemke, A., Schuler, W., Schütt, H., Thüring, M. SEPIA: A Cooperative Hypermedia Authoring Environment, *Proc. 4th ACM Conference on Hypertext*, Italy, November 30 - December 4, 1992.