REIHE INFORMATIK
TR-04-003

Conflict Visualization for Collaborative Multi-user Applications

Jürgen Vogel
Universität Mannheim
Praktische Informatik IV
L15, 16
D-68131 Mannheim

Conflict Visualization for Collaborative Multi-user Applications

Jürgen Vogel
Praktische Informatik IV, University of Mannheim, Germany
vogel@informatik.uni-mannheim.de

Abstract— **If a collaborative multi-user application allows concurrent or asynchronous manipulation of shared data, user actions may conflict. While conflicts can be resolved syntactically by concurrency control, the system is not able to interpret actions, and the determined result might violate the users' intentions. We propose a visual feedback scheme so that users are aware of critical situations and may react accordingly. In our approach, the action history is represented as an interactive timeline that can be analyzed by the user. A prototype was realized for a shared whiteboard application.**

Index Terms— **CSCW, Collaboration, Concurrency Control, Awareness.**

I. Introduction

Collaborative multi-user applications support remote collaboration of people through information sharing via computer networks. Examples are groupware and distributed virtual environments. In order to achieve high responsiveness for local user actions as well as a smooth multi-user interaction, collaborative applications often employ a replicated architecture with optimistic concurrency control [1], [2]. Thus, users are allowed to change the application's shared state anytime so that their actions may overlap and even conflict semantically (e.g., when two users want to move the same object simultaneously into different directions). The probability for such conflicts is particularly high if an application facilitates asynchronous collaboration [3]; here, a user may change data while being disconnected from the network, and the corresponding events can be consolidated only later on when going online. In the meantime, other users may modify the same data, leading to conflicts.

While concurrency control algorithms keep the shared state consistent at all sites, they are in general not able to determine which result would be preferred by the users (e.g., where the object in the example from above should be). Such situations could be resolved explicitly with user interaction (e.g., by voting), but this would interrupt the collaboration whenever a conflict is discovered. Instead, we propose to supplement automatic concurrency control with visual feedback so that users become aware of conflicts and may react accordingly. Before presenting our visualization scheme, we discuss concurrency control algorithms.

II. Concurrency Control

Collaborative applications with a replicated architecture maintain a copy of the shared application state at each site. Local user events can be executed immediately and are also transmitted to all other sites so that their data copies can be updated. Since multiple users may issue events concurrently (i.e., without having received each other's actions), a concurrency control mechanism ensures that all sites reach the same state after executing the same events, even when events are received in different orders, or after their scheduled execution times in the case of a real-time application [2].

The events exchanged in such a setting may *conflict semantically* if they modify correlated aspects of the shared state and if their execution times lie within a certain timespan T. For instance, a move event conflicts semantically with an event that deletes the same object since the users obviously disagree about the desired state of that object. T depends on the propagation delay of the events and the user's reaction time. For a shared whiteboard, preliminary experiments indicated that T is approximately one second. For asynchronous applications, T can be much higher [3].

Well-known concurrency control algorithms are serialization, operational transformation, and object duplication. Serialization executes events in a distinct order at all sites, e.g., based on their scheduled execution times. This order has to be restored if a received event would violate it when appended to the history. The timewarp algorithm described in [2] establishes an order based on the event history: Each

site saves a history of local and remote events together with periodic state snapshots. If necessary, the state is recalculated using an older snapshot and the subsequent events. For events that conflict semantically, this means that only the effects of the last event are reflected in the updated state.

Operational transformation also establishes consistency using a local history [1]: Events are transformed against the events executed before so that they can be applied to the state in the order in which they are received. As in timewarp, a certain event is favored in situations where events are opposed semantically (e.g., two move events).

Object duplication generates a new version of an object when conflicting events occur, thus presenting multiple versions of the same object to the user so that all user intentions are preserved [4]. But this might be confusing, and there is no support for merging different versions and finding a joint state. Like operational transformation, object duplication is difficult to realize for real-time applications.

Even though semantic conflicts can be resolved by these concurrency control mechanisms, they do not understand the users' intentions and are not able to determine the desired state. Thus, information about conflicts should be provided so that the users become aware of critical situations and are able to analyze and resolve them.

III. Design Considerations

Besides notifying the users about a conflict, detailed awareness information is required in conflict situations: Which objects are modified and how? Which participants are involved? What is their current task? Which past actions are related? And what would an alternative state look like?

Design options to visualize this information concern its placement (where), representation (how), trigger (when), and duration (how long). For instance, the information can be displayed in the shared workspace next to the objects concerned. While providing a direct reference, this might be distracting or conceal other content. Alternatively, a separate window can be used. Second, information can be represented graphically or as text. With graphical representations (e.g., icons), information can be encoded using different colors, shapes, and sizes. This requires that the user learns the different meanings, which is demanding if many categories exist. If encoded as textual descriptions, information is easy to understand but might require more time to take in. Third, information can be displayed automatically or on the explicit request of the user which is a tradeoff between effort and distraction. And fourth, the data might be visible permanently or only temporarily.

General design goals have to be considered as well: First, the mechanism should be easy to use. Second, while there should be sufficient information available, an overload should be prevented. Finally, information should be accessible anytime, and its analysis should not disturb remote users.

IV. Related Work

Even though awareness information is vital for collaborative applications, existing work mostly focuses on general aspects and does not address conflict management in particular. [5] provides mechanisms to demonstrate the evolution of the current state in case the user did not observe the joint editing process or wants to review it: A history lists textual descriptions of past actions (e.g., "Alice creates an oval"), a trailing function shows the path of moved objects, and a replay mechanism repeats the course of action. The replay is very successful when controlled via the history. But since it is text-based, this history becomes complex in large sessions.

A scheme with multiple, alternative event histories is presented in [6]: Similar to object duplication, events are not merged into a single history but form a directed graph with paths that may split and join. Each node of this graph reflects a different application state. The user has direct access to past states and may navigate within the graph to explore different versions. Moreover, the user is allowed to add new events at arbitrary positions in the graph. Different paths may also be merged in order to create a joint state. While such a history graph is very flexible, it may become complex during a session. Keeping track of these multiple "realities" is difficult, especially since users can work on different parts of the graph (i.e., different points in time) simultaneously. Adding events to the inner graph is likely to cause more conflicts instead of resolving them. Finally, there is no support to merge conflicting paths into a single and non-conflicting shared state which should be the ultimate goal of a collaborative multi-user application.

V. Visualizing the Event History

Our goal is a visualization mechanism that shows how the shared state of a collaborative application evolved, especially when events conflict. Such a mechanism supports the user in changing the current state

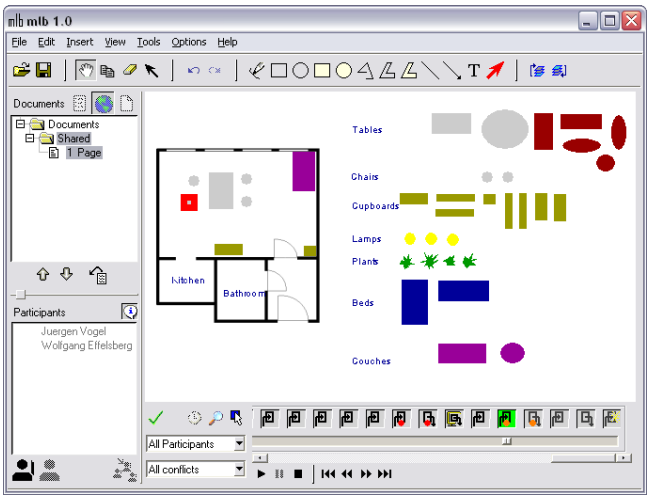

Fig. 1. The mlb with event history

to the semantically desired state and in coordinating future actions.

The concurrency control algorithms discussed above require each site to store at least parts of the event history. This history contains important awareness information: If visualized, it demonstrates the course of action and the correlations among events (actions). Our approach therefore provides the information identified above (i.e., who changed which object by which event conflicting with which other events) by means of a graphical representation of the history that the user can access and analyze if necessary. We realized a prototype of this visualization scheme for the shared whiteboard multimedia lecture board (mlb) [7] that employs timewarp [2] for concurrency control. As depicted in Figure 1, the event history is displayed underneath the mlb's shared workspace and visualizes information about events, objects, and participants, provides a replay function, and allows the user to explore past actions and alternative states.

The history is represented as a timeline with an icon for each event (see Figure 2). The most recent event is shown on the right, and new events are appended immediately. Each icon on the timeline encodes different information depending on its shape, color, and background: Events issued by the local user have an outgoing arrow (1), while all remote events show an incoming arrow (2). Events that are part of a conflict sequence are marked by a dot that is either red (3) or orange (4) in order to distinguish among different conflict sequences. In (3), one remote and one local event conflict semantically. The user can choose whether he wants to see all conflicts (5), always the last conflict, or a conflict that occurred at a certain

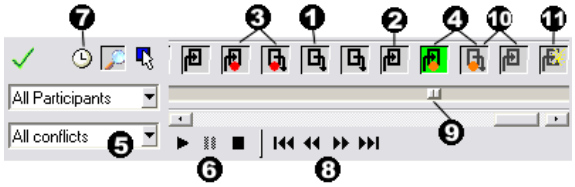

Fig. 2. Visualization of the event history

time (see Figure 3 (1)). If the mlb detects a new conflict, this is also indicated in the workspace window by attaching a temporary tool-tip window naming the users involved, and by highlighting the involved users in the participant window (see Figure 1).

When selecting the icon of an event that is part of a certain conflict sequence, the entire sequence is marked so that the history can be examined by the user. Also, the corresponding object is highlighted in the shared workspace so that the user can explore which object belongs to which event. Vice versa, all icons targeting a certain object are highlighted by a white background when the object is selected.

A. *Exploring Past States*

Our visualization scheme includes a replay function for reviewing the past course of action. It is controlled with a VCR-like interface (see Figure 2): After pressing play (6), events are replayed in the order given by the history, and the appropriate state is displayed in the workspace window. The replay can run either in the original time lapse or in a fast-forward mode (7). The skip buttons (8) change the current position in the history. The event that was executed last is marked by a green background (4 left), its target object is marked in the workspace, and the responsible user is named in a tool-tip window if desired. All events not yet executed are displayed with gray icons (10), instead of black ones (2).

The user can also browse through the history by means of a slider (9). The slider's position marks the last event executed. When the slider is moved, the content of the workspace is updated accordingly.

Exploring the event history has only effects on the local view and does not disturb remote users. While a past state is displayed, objects cannot be created or modified. All remote events received in the meantime are appended to the history, but their effect is not visible until the replay reaches their scheduled execution time. An event is marked as new (11) until it is executed for the first time so that the local user is aware of remote users' actions. After the latest event was executed, the local user is able to modify the state again.

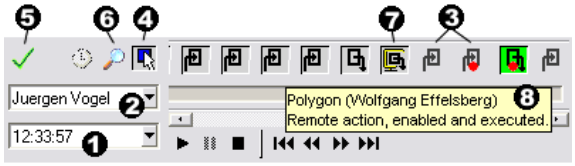

Fig. 3. Exploring alternative states

B. Exploring Alternative States

A user might not only be interested in how the current state came to be but also in alternative states. For instance, when several users issue concurrent and conflicting actions, it is useful to understand the intentions of a certain user. For this purpose, a dominating user can be selected as depicted in Figure 3 (2). This disables the events of all other users, starting from the event executed last. In the situation shown in Figure 3, some remote events are disabled, indicated by a flat icon (3) while enabled events are sunken. Starting the replay or browsing the history will now apply enabled events only to the workspace. Thus, the workspace shows the evolution of an alternative state. Similarly, the evolution of certain objects selected in the workspace can be tracked while all other objects remain unchanged.

Moreover, single events can be disabled or enabled by switching to the change mode (4) and clicking on the icons. In case the execution time of a disabled event lies before the time of the current state, the workspace is updated immediately. In some cases, it might also be necessary to disable subsequent events together with the one disabled by the user. For instance, when disabling a create event, all other events for the same object need to be disabled as well.

Disabling certain events affects only the local view, i.e., the current shared state is not affected. But it might happen that the user creates a state that she actually prefers. For instance, she might not have issued a certain event if she had known about the modifications of another user. Or perhaps events were issued on the basis of a state that suffered from a short-term inconsistency [2]. For such cases, a convenient way to alter the shared state is provided: The state displayed in the user's local view can be finalized by pressing the apply button (5). Then, the local state becomes the new shared state for all users and is propagated to all sites.

Summing up, icons indicate whether an event is local or remote, conflicting or not, is executed or not, is the last event executed, is enabled or not, and is selected or not. Although it would be possible to encode even more information into an icon such as the event's type (e.g., create, move, etc.) or the responsible user, this would increase the representation's complexity. Instead, textual descriptions for each event are provided via a tool-tip window (see Figure 3 (8)).

C. Implementation Issues

For the mlb, we had two possibilities to implement the replay mechanism: Using the event history, past states can be calculated either on the basis of the concurrency control algorithm or by employing the mlb's undo scheme.

In the first case, a timewarp is executed when a state older than the current one is to be displayed (e.g., when moving the slider to the left or when jumping to the history's beginning). When moving forward in time, the respective next state can be calculated by applying the next event to the current state at the appropriate execution time. This execution time is determined on the basis of the current time and the offset between the execution times of the current and the next event (divided by the fast forward factor if in the fast forward mode). The main advantage of the timewarp-based approach is its easy implementation if the application uses timewarp for concurrency control, as is the case for the mlb. Moreover, it is applicable to all applications, including real-time applications. But executing a timewarp is costly in terms of processing power [2], which might be critical when browsing quickly through the history.

For the mlb, we therefore decided to realize the replay by means of undo and redo operations: When moving backward in time, the last event executed is undone by applying the appropriate undo event, and when moving forward in time, events are redone. Jumping to a position (e.g., to the end of the history) is realized by executing an entire sequence of undo or redo events. The mlb generates undo and redo events semantically, i.e., an undo event restores the attributes changed by the corresponding event to their original state. For instance, the undo for a move event encodes the original position of the moved object, and the undo for a delete event holds the complete state of the deleted object. While this approach requires that the application is able to undo all events (as it is the case for the mlb but might be rather difficult for real-time applications), it achieves a very good performance resulting in smooth state updates even when skimming quickly through the history.

Disabling events, when exploring alternative states as described above, again can be realized either by executing a timewarp on the basis of the changed his-

tory or, as in the case of the mlb, by undoing the concerned events.

When finalizing a modified state, we also use the application's undo functionality: For each disabled event, the corresponding semantic undo is created and distributed to all sites. Thus, the original event history is not modified but new events are appended that are handled just like regular events: They are displayed in the history's representation and can be analyzed or undone if desired. Implementing the propagation of a modified state via timewarp instead would require to notify all remote sites about the events to be undone so that these can execute a timewarp on the changed history. This is considerably more complex than the semantic undo approach and has the severe drawback that all sites need to store the same parts of the event history. And in order to redo undone events at a later point in time, they would have to be retained by all sites.

VI. Discussion

We evaluated our visualization scheme in preliminary experiments with two users. The task was to place furniture from a given set in a small apartment (see Figure 1). In this setting, it is likely that two users move the same piece of furniture and raise a semantic conflict. By introducing artificial network delays, the probability of conflicts could be increased. The experiments indicate that the visualization mechanism provides sufficient information to notice and analyze conflicts. Especially the slider-based browsing of the event history together with the possibility to review the actions of a certain user proved to be very efficient. In this context, we plan a feature that allows to save different versions of the state so that the user can quickly compare them.

However, some shortcomings were discovered as well. First, the user has to acquaint himself with the different visualizations. In an earlier prototype, the history was displayed only when a conflict occurred or when requested by the user. But we discovered that a permanent view of the history together with immediate updates and tool-tip descriptions of icons accelerated the user's learning process.

When the remote users continue to issue events, analyzing the history might be too slow so that the local user permanently lags behind. An analysis might also be difficult when multiple events conflict. Furthermore, changing the shared state by disabling events is problematic when several users do this at the same time. In such situations, it is likely that new conflicts emerge, and that the resulting state violates the users' intentions once more. In order to lower the probability for repeated conflicts, it is not allowed to disable remote events when finalizing a state. The risk for new conflicts might also be reduced by indicating which users are currently reviewing the history so that users can coordinate.

In sessions with many members and a high activity, the history might become large and difficult to analyze. In order to shorten the history, it is possible to find a more compact representation for some event sequences without losing vital information. For instance, when creating a freehand line on an mlb slide, each point is propagated singly in order to achieve good responsiveness. But for later examination, this event sequence can be substituted with a single state containing the complete line which shortens the history significantly. If the history is still too long, its granularity can be reduced by switching to an overview mode (see Figure 3 (6)) where homogeneous event sequences are subsumed and displayed as a single icon (7). A sequence is homogeneous if all events from a certain user target the same object and do not raise a conflict, e.g., subsequent move events.

VII. Conclusions and Future Work

We have presented a novel visualization scheme for collaborative multi-user applications that supplements the application's concurrency control mechanism and provides extensive awareness information about semantic conflicts in user actions, and about past and alternative states. Our graphical representation of the event history allows to quickly detect and analyze conflicts, which may also help to prevent future ones. The viability of our approach was demonstrated for a shared whiteboard. But the visualization scheme is generic, and we plan to integrate it into other collaborative applications as well. Our main interest lies in real-time applications, where the representation should also reflect the correct time lapse of events, and in applications for asynchronous collaboration, where conflicts occur more frequently. So far, no large-scale user study was conducted but we plan to evaluate different conflict scenarios.

References

[1] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving Convergence, Causality Preservation and Intention Preservation in Real-Time Cooperative Editing Systems," *ACM Transactions on Computer-Human Interaction*, vol. 5, no. 1, pp. 63–108, 1998.

[2] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications," *IEEE Transactions on Multimedia*, vol. 6, no. 1, pp. 45–57, 2004.

[3] W. Geyer, J. Vogel, L.-T. Cheng, and M. Muller, "Supporting Activity-centric Collaboration through Peer-to-Peer Shared Objects," in *Proc. ACM SIGGROUP, Sanibel Island, FL, USA*, Nov. 2003, pp. 115–124.

[4] C. Sun and D. Chen, "Consistency Maintenance in Real-Time Collaborative Editing Systems," *ACM Transactions on Computer-Human Interaction*, vol. 9, no. 1, pp. 1–41, 2002.

[5] L. McCaffrey, "Representing Change in Persistent Groupware Environments," Tech. Rep., Grouplab Report, Department of Computer Science, University of Calgary, Canada, 1998.

[6] W.K. Edwards and E.D. Mynatt, "Timewarp: Techniques for Autonomous Collaboration," in *Proc. ACM SIGCHI, Atlanta, GA, USA*, Mar. 1997, pp. 218–225.

[7] J. Vogel, "multimedia lecture board (mlb)," URL http://www.informatik.uni-mannheim.de/informatik/pi4/-projects/mlb/, 2004.