Representing Change in Persistent Groupware Environments

Lorin McCaffrey (with course supervisor Saul Greenberg)
Department of Computer Science
University of Calgary, Calgary, Alberta
Canada T2N 1N4

Abstract

This project deals with the display of change information in object-oriented, graphical groupware. A review of current literature was performed, and important ideas incorporated into a conceptual framework. The framework highlighted specific questions that a groupware user may ask from a change management system. It was used to develop three change display mechanisms that went through several design and testing iterations. These include a symbolic approach following a sand trails metaphor, a literal replay mechanism, and a separately displayed change index. Successes and shortcomings of these mechanisms, as revealed by testing them with a simple testbed application, are discussed. In particular, the symbolic sandtrails scheme gave the best overview of changes, but only when the total number of changes was small and when the order of changes was not important. The replay and change index schemes were found to work best when coupled together, thereby allowing the sorting of change information on several attributes, and literal replay of the past user's actions. Users of this scheme, however, had some trouble associating index entries (presented in a separate change index window) with associated objects in the main application screen.

McCaffrey, L. (1998) **Representing Change in Persistent Groupware Environments**. *Grouplab report*, Department of Computer Science, University of Calgary, Alberta, Canada. January.

http://www.cpsc.ucalgary.ca/grouplab/papers/index.html.

1 Introduction

Groupware involves the development of virtual environments in which more than one person may work simultaneously. Often, this is done through an artificial worksurface, sometimes mimicking real-world worksurfaces such as whiteboards or tabletops in a conference room. The use of groupware is often seen in distributed environments where coworkers may not be in close proximity, but perhaps across the continent. Unfortunately, most current groupware systems mimicking real-world workspaces receive a failing grade. Often, this is due to the lack of support for many natural forms of communication that exist in real-world group work. This communication would enable co-workers to keep up-to-date on each other's work, an aspect that has been termed *workspace awareness*. For example, when people work cooperatively in the real world, they are often seated near each other, perhaps around a table. This alone supports many unconscious actions that aid collaboration: a worker can quickly glance at another's work, hear them moving objects about, and so on. In this context, workspace awareness can be defined as the collection of up-to-the-minute knowledge a person holds about the state of another's interaction with the workspace (Gutwin, Greenberg, & Roseman. 1996).

Up to this point, most commercial attempts at groupware systems leave a lot to be desired, partially because they do not support workspace awareness. Mechanisms have been developed, however, that restore some of the information lost when moving from a real-world setting to a virtual one; some of the mechanisms designed by Gutwin (1997) for example, are discussed later in this paper. While Gutwin concentrated on supplying awareness clues for concurrent collaboration, TeamWave (Roseman, 1998) is an example of a commercial groupware environment which, because of its persistent nature (the environment is always available, the users come and go as they please), brings to light some workspace awareness problems associated with past actions of multiple groupware users.

When a user returns to a previously left task, document, or application, they usually rely on their memory to track the significant changes done between editing sessions. This is not usually a problem, because users are often only concentrating on the current state of a document, and are not worried about how it arrived in that state. In a groupware environment, however, a user may work on a document and return to it later to find it changed due to the actions of other users. In this case, they need to be aware that changes have been made, and told the relevant information about those changes. I believe that the computer should, in such a case, support the users by identifying the relevant changes to them and by informing them of the information about those changes that they want to know.

I have investigated how change information can be displayed to users in an object-oriented graphical environment. Graphical environments have been targeted because much work has already been done in the domain of displaying changes in text based applications, some of which will be mentioned in the literature review section that follows. I have developed several graphical display mechanisms, and have done some exploratory usability testing on each of them. This work could be a valuable source of information for groupware designers when the development of change management systems is required, and I give some recommendations about the use of these types of mechanisms. In summary, the goal of the project is twofold: to generate ideas about how to display change information in object-oriented groupware, and to discuss the successes and shortcomings of these ideas.

The paper roughly follows a four-step process of information gathering, translation, display, and evaluation. Information is gathered in the literature section that follows. This knowledge is translated into a conceptual framework in section 3, while the means of display discussed initially in section 4. Finally, section 5 presents the evaluation

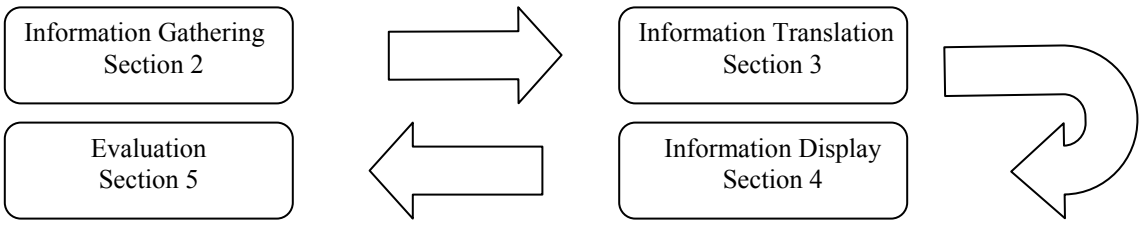

Figure 1 – The research process

results.

2 Literature Review

My project advances previous work done in the areas of workspace awareness (e.g. Gutwin, Greenberg, and Roseman, 1996; Gutwin, 1997), and in information visualization techniques that display change information in text-only environments (e.g. Hill & Hollan, 1992; Neuwirth et al., 1992).

2.1 Text-only change management systems

AT&T Bell Laboratories, demonstrated a system that visualizes the change history of a set of program source code files (AT&T Bell Laboratories, 1993). The system was able to demonstrate a colossal amount of change data including additions, changes, and deletions of text on a per-line basis. It can track the people who performed the changes, and lets the user build customized views of date ranges. It then displays this information by using colours and an index showing how the spectrum relates to age of the text (Figure 2). Users of the system were able to simultaneously see an overview of the source code files as well as use a detailed view window to read individual sections of code, both with the color coded change information present.

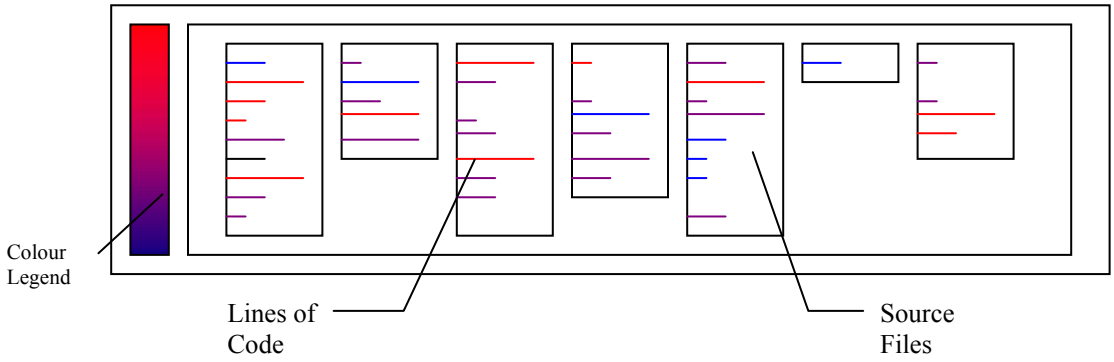

Figure 2 - a mockup of AT&T's "Program Change History" viewer

In addition, Hill & Hollan (1992) introduced the concept of *read wear* and *edit wear*. The original technique involved mapping horizontal lines, whose length was proportional to the amount of changes or reading of the corresponding line of text, into the scrollbar of a text document. As users read and/or edited lines of text, the corresponding lines in the scrollbar would lengthen to represent the occurrence of change. The authors compared this mechanism with the wear that objects receive in real life. As Hill & Hollan noted, "The best recipe cards in a stack are often dogged-eared and stained." As with the system from AT&T, this system let you filter out some of the recorded data so that you could ask specific questions, such as "How long did Bill spend reading this section?"

Neuwirth and Chandhok (1992) modified the existing PREP text editor (a tool for collaborative writing) so that it annotated changes made by other group members. In so doing, they brought to light some important facts about group authorship that have had an effect on my work. For example, it was found that because there are different roles within the group (editor vs. writer) and because the needs of the group change over time (first draft vs. final draft), it was desirable to have variability within the change representation mechanism so that the user could match the change display to his needs at any one time. This was incorporated with their idea of change threshold and resulted in a system in which, in addition to many other features, the user could select the granularity of changes to display in order to reduce negative distractions caused by displaying trivial change information.

2.2 Graphical workspace awareness and change representation techniques

Carl Gutwin (1997) recently completed a comprehensive thesis on workspace awareness in distributed groupware. In it, Gutwin examines the underlying cognitive aspects of situation awareness, collaboration, and shared workspaces, and then tackles the problem of supporting these processes in groupware. A framework was developed based on a modified version of Neisser's (1976) perception-action cycle. This cycle stipulates that human awareness

follows a threefold iterative process of gathering perceptual information, interpreting it, and then determining what to look for next.

Workspace awareness techniques developed using this framework were then categorized using the matrix of Table 1– Dimensions of design space. This organization makes apparent the difference between *situated* and *separate* placement of awareness information, as well as the form of presentation: either *literal* or *symbolic*:

- **Situated** placement means to display the information at the workspace location where it originated.
- **Separate** placement refers to displaying the information in a separate part of the interface.
- **Literal** presentation implies that information is shown in the same form in which it was gathered.
- **Symbolic** presentations extract and display explicitly only particular information.

Table 1 - Dimensions of design space

		Placement	
		Situated	Separate
Presentation	Literal		
	Symbolic		

As well, Gutwin operationalizes the elements of workspace awareness, and is able to group them into categories labeled with the 5 W's (and one H): Who, What, When, Where, Why, and How. When discussing these, he noted in particular that some of these categories relate only to past occurrences (such as How and When), and so the thesis touches lightly upon the subject of change display mechanisms, but falls short of suggesting specific solutions. We will visit these questions in more detail when discussing my conceptual framework in section 3. Gutwin's conceptual and organizational framework has been extremely valuable to this project, because it provided a convenient means by which I can organize and evaluate my own ideas.

In addition to Gutwin's work on workspace awareness, the beginnings of change representation have surfaced in work such as the 'Editable Graphical Histories' system by Kurlander and Feiner (1988). They realized that graphical histories require more of a *temporal browsing* method than do their text-only cousins. To do this, they made use of a comic strip metaphor, Kurlander and Feiner used a set of heuristics to determine which graphical 'snapshots' to use in a series of separately displayed before-and-after panels showing the history of a graphical editor document. The true power of the system came into play when a user 'undid' several actions by selecting a past frame of the history, made modifications, and then 'redid' all the subsequent frames, propagating the changes all the way through.

2.3 Resulting problem domain

As a result of my review of previous work, I estimate that my problem lies within the shaded area of Figure 3. Specifically, it shares concerns with research focusing on workspace awareness as well as research on change management. It does not include text-only change management systems, however.

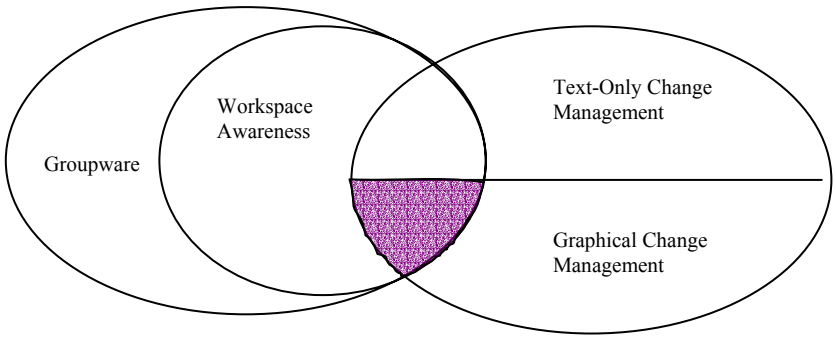

Figure 3 - Resulting problem domain

3 Conceptual Framework

In order to begin experimenting with change display systems, it was necessary to define the conceptual framework in which to work. I began with a simplified approach where I focused mainly on representing changes that can come about on a single generic object, no matter what the type. The term 'changes' was defined as a small list of fundamental actions that can be done on any object. Table lists the basic operations that were finally chosen: object creation, object deletion, object movement, and any combination of these three. These fundamental actions are common amongst most object-oriented graphical environments, and so the hope is that this work will be generalizable. The work is restrictive, however, as this approach will not consider relationships between objects, nor take into account any domain knowledge behind changes. As well, I have not experimented with displaying other actions such as resizing, change in colour and shading, etc. In essence, this means that applications that require change awareness such as groupware drawing packages (which probably have such advanced features), and applications such as groupware concept map editors (where relations among objects is paramount) will only be partially covered.

Table 2 - Basic change actions

Object creation
Object deletion
Object movement
Any combination of the above

I limited the size of the project by focusing on representing the changes since the last time the user was in the system, that is, the last time the user saw a particular set of objects. The research can be summarized by the following questions. The first two I will cover in this section while the last two are left for sections 4 and 5, respectively:

- What change information do people really need?
- How do we capture this change information, and translate into the constraints of groupware and digital storage?
- How are we going to display this information to the user?
- How can we evaluate whether the displayed information is useful in practice?

3.1 What change information do people really need?

Fairly early in the research process it became clear that a definitive answer to this question is not possible, mainly because it depends heavily on the task domain. It is quite apparent that displaying all change information is often inappropriate (Nachbar 1988 as cited in Neuwirth, 1992), such as when we compare changes in white space and comments in two versions of source code even though they have no effect on the operation of the compiled product. In fact, a central idea in Neuwirth's (1988) *flexible diff* system was indeed flexibility, as she realized that people play different roles in a writing group, and that the needs of collaborative writing groups change over time. Neuwirth maintains that the only changes that should be displayed are those that reduce negative distraction and increase positive distraction (positive distraction being information that the user finds informative), but this depends on the reader and the task. Nevertheless, for any groupware environment, the desired level of history awareness will probably be a combination of the following elements, which I've grouped into the "who, what, where, when, why, and how" questions as suggested by Gutwin (1997, p.34). Further distinctions within these categories was made by asking specific questions which, in theory, could be used to guide groupware designers to find the prioritization of elements that fit their particular task domain. As will be seen later, some of these questions were indeed used during the usability testing of the mechanisms that I have developed.

3.1.1 What were the changes?

Most of the time, this category will be quite important to any change display mechanism since, in my opinion, it will often be the primary motivation for implementing a mechanism in the first place. However, this component is affected by the *significancy* complication: what magnitude of a change is required to pass the threshold of negative to positive distraction? In some forms of authorship of text documents, for example, it was found that a mechanism for simply counting the number of times changes have occurred was sufficient (e.g. Hill & Hollan, 1992).

Questions include:

- What object was the person working on?
- What action (in my case, one of creation, deletion, or movement) was the person doing to/with that object?

3.1.2 Where did the changes occur?

Once again, I have classified this element as required information, due to the importance of the location of an object in relation to other objects is important in graphical workspaces. Mechanisms that display this element will be entirely different in the graphical domain than they would be in a textual one, due to the fact that we can "attach importance to the actual physical screen locations at which input and output occur, modifying in place what is being displayed" (Kurlander, 1988).

Questions include:
- Where has a person been (what did he/she look at)?
- Where did the person make a change?

3.1.3 Who did these changes?

This is often needed as a secondary piece of information, or one that can be used to filter displayed changes. For example, while a co-author may not be interested in seeing every change that a trusted co-author made, he or she may want to check every change from a less trustworthy editor (Neuwirth, 1998). It may not be necessary for this information to appear obvious at first-glance, however.

Questions include:
- Who produced this change?
- Who was here, and when?

3.1.4 How did the user make these changes?

It quickly became apparent that in some situations, the series of actions a user takes to enact a change on the environment helps an observer to understand the action as a whole. As this project is concerned with the goupware implementation of change display mechanisms, it is important to present object *feedthrough* (Dix et al 1993, as cited in Gutwin, 1997): "when artifacts are manipulated, they give off information, and what would normally be feedback to the person performing the action can also inform others who are watching" (Gutwin, 1997). Therefore, it is often useful to display all intermediate steps seen when a change is occurring, essentially showing *how* the user made the change in a very explicit way. Contrast this with the previously mentioned start-state versus end-state approach as demonstrated by Kurlander & Feiner (1988), where intermediate steps were filtered out because both the actor and observer are the same and did not need this information.

Questions include:
- How did that operation happen?
- How did this object come to be in this state?

3.1.5 When did these changes take place?

Consideration of this question is partially alleviated by the constraints on this project. In our scope, we are only interested in changes that have occurred since the last time that the user was in the system. As well, sometimes the order of changes is important, while in other domains it is not[1].

Questions include:
- When did that change occur?
- In what order did a sequence of changes take place?

[1] As will be seen in section 4, our choice of testbed applications for this project was one in which the sequence of changes was not very important.

3.1.6 Why did these changes occur?

Unfortunately, this question assumes some knowledge of the problem domain. Unless we want to implement AI techniques, it is hard to guess what the user is thinking when he or she is making a change. However, it may be sufficient to allow the user to attach short notes to his or her changes to notify subsequent users of their motives in an explicit way. Perhaps the only other method would be to rely on the future user's knowledge of context to sort out the motivations of the previous person.

Questions include:

- What motivated that person to make this change?
- What is the new meaning of the environment after this change (i.e. the person wanted to communicate something or accomplish something, what was it)?

As a caveat to this discussion, it should be noted that, while helpful to me in designing the first versions of change display mechanisms, my own prioritization of these elements will not always be optimal since they could change over time (i.e. Neuwirth et al. 1992) and could be extremely domain dependant. For example, if there have been many changes since a user last logged in to an environment, answering the 'where' question at a detailed level may be quite distracting and certainly not be as valuable as it would be if there have been only a few changes.

3.2 How do we capture this change information, and translate it into the constraints of groupware and digital storage?

Neuwirth et. al's concerns in 1992 when developing the Flexible Diff program included efficiency and storage space for change histories. Because of the rapidly decreasing cost of digital storage and increasing speed of computers, I am willing to discount this issue. In order to make this work as generalizable as possible, I have chosen to use the following data capturing scheme as the basis for all my change display mechanisms. Individual groupware designers are then free to increase or decrease the amount of data capturing to support their specific cases. For the most part, this information is kept in a central 'change-database' rather than being attached to the objects themselves. This arrangement was found to simplify the coding of some change display mechanisms, but they are not dependant on it.

For **adding** objects, record:

- The location in the environment where the object was introduced
- The time the object was introduced
- The user that introduced the object
- A pointer to the object

For **deleting** objects, record:

- The location in the environment where the object was deleted
- The time the object was deleted
- The user that deleted the object
- A pointer to the object

For **moving** objects, record:

- The path (probably as a series of x,y coordinates) the object took when moving through the environment, with a granularity high enough to be able to reproduce an approximation of it's path at a later time
- The time the move was completed. Because a move operation often will take more time than the others, it would be more accurate to attach a time stamp to every entry in the path description. This issue is discussed further in the section describing the 'replay' mechanism below.
- The user that moved the object
- A pointer to the object

4 Initial Design of Three Change Display Mechanisms

A small testbed application was first designed in order to test future change display mechanisms. The testbed is a single user application allowing the adding, deleting, and moving of arbitrary objects around a small environment. It was coded using the Tcl/Tk programming language (Osterhout 1996). The first version of this environment allowed

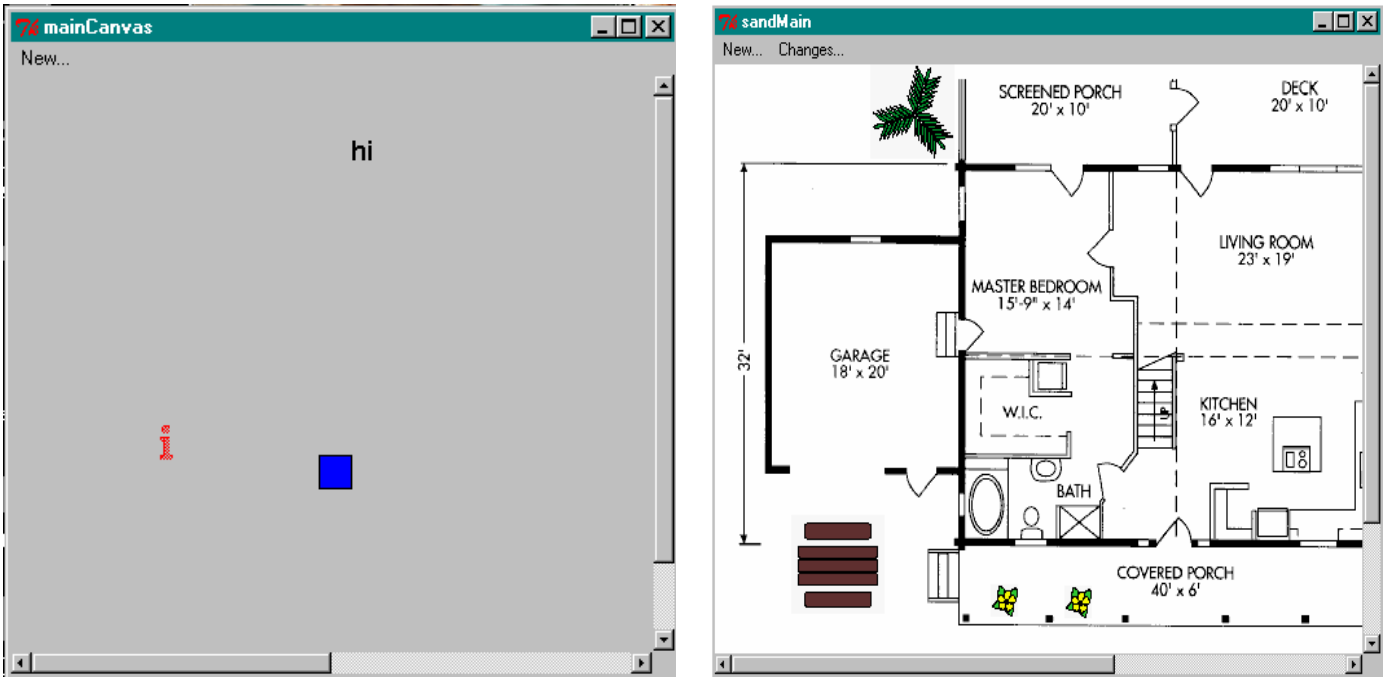

Figure 4 - Version 1 & Version 2 of the testbed

the creation, move, and delete operations on representative text and bitmap objects. However, when coding of change mechanisms began using this testbed as a base, it was deemed too unrealistic since it did not reflect any real-world task. Version 2 of the testbed was therefore created to resemble the floorplan of a building on which plants and other objects may be moved about, as if the user was planning the landscaping of his or her house. Both versions of the testbed are visible in Figure 4. In spite of their quite different appearance, the possible actions are identical. The testbed application uses the data capturing scheme mentioned above, but is independent of any change display mechanism.

On top of the testbed, change display systems were then implemented. This section discusses the initial design of three completed systems, *sandtrails, replay*, and the *change index*. In the next section, I will discuss the improvement of these mechanisms as they progressed through several design iterations.

Table 3 – The three display mechanisms as classified by design space

		Placement	
		Situated	Separate
Presentation	Literal	*Replay*	
	Symbolic	*Sandtrails*	*Change Index*

Sandtrails

The *sandtrails* display mechanism follows the example put forth by Hill & Hollan's (1992) *Editwear and Readwear* by attempting to make the objects reflect the changes done to them. Sandtrails adheres to the metaphor of objects on a beach, leaving trails as they are moved about. It is classified as a situated, symbolic method as can be seen in Table 3. Sandtrails are situated since they are drawn at the locations that the actions took place, but symbolic since they do not present a literal representation of the action. Figure 5 presents two snapshots of the sandtrails method after a user made several 'changes' to the environment. Here is how the sandtrails method handles each of the basic change actions (respective annotations can be found on Figure 5):

Object creation:

Object creation is represented by the object itself existing in the environment, but with a star placed on top of it, made to resemble somewhat of a gleam of light off a shiny new object. This can be seen in Figure 5 on top of the round tree at the top-right of the screen.

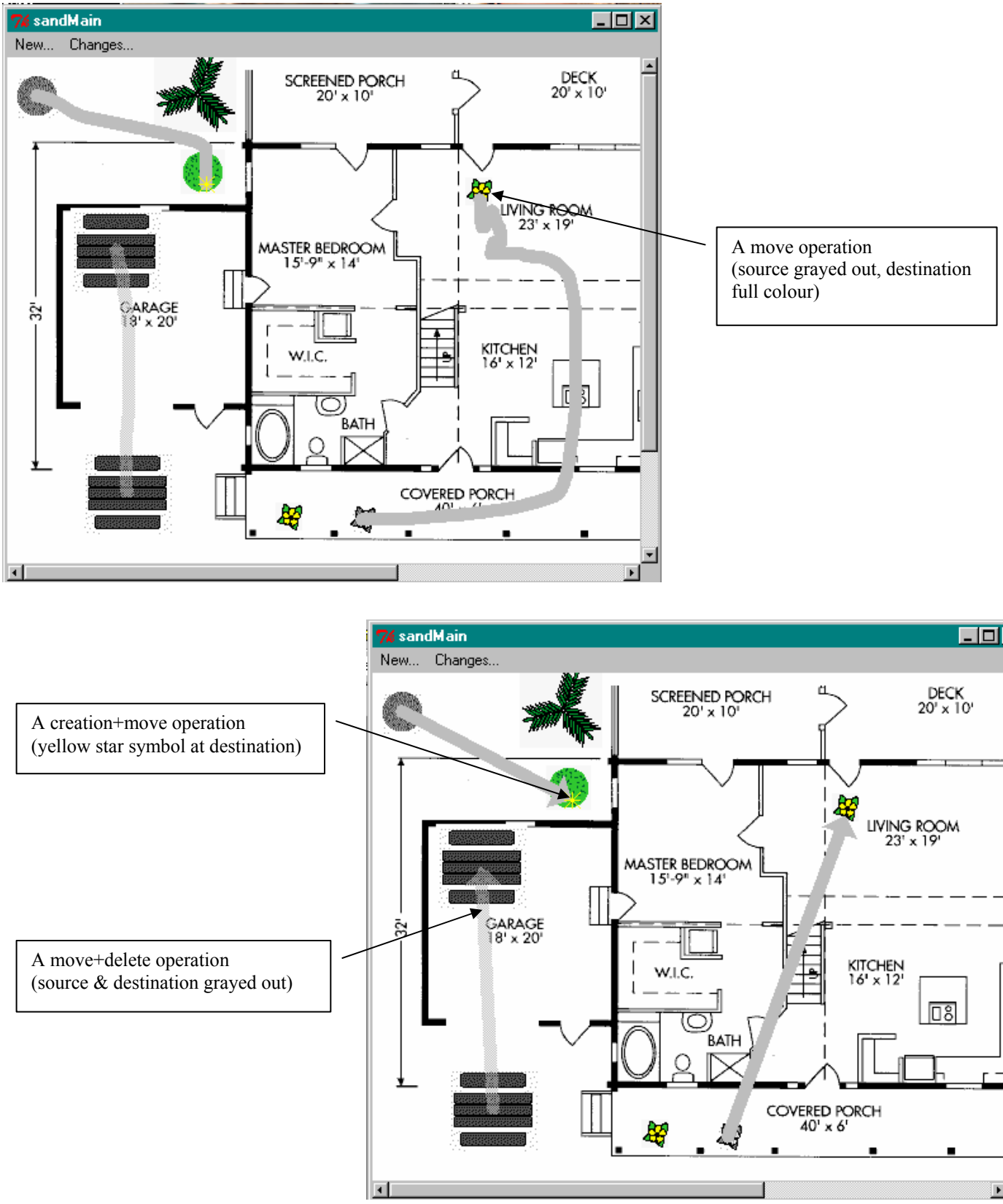

Figure 5 - Curved line and straight line versions of *Sandtrails*

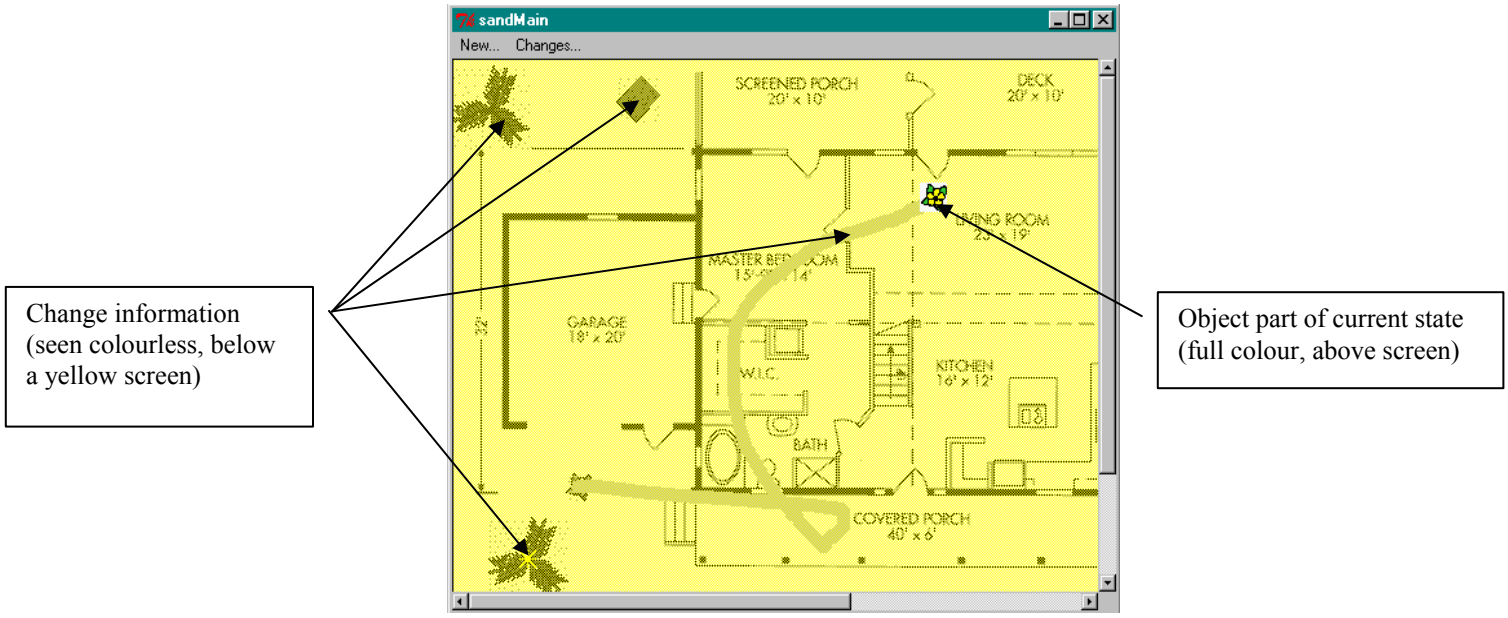

Figure 6 – The Sandtrails mechanism used in conjunction with a transparent screen.

Object deletion:

Deletion of the object causes the object to be displayed at the location of the deletion, but faded out to resemble the imprint it would have left in sand after being removed. Any other change information applying to this object (such as a 'new' symbol or a 'move line') is also faded. The picnic table in Figure 5 is an example of a delete action.

Object movement:

When a heavy object is dragged through sand, it leaves a trail behind it. The sandtrails mechanism therefore attempts to reproduce this effect by drawing lines along the movement path. Unfortunately, some users who were either fussy or indecisive about object placement may create long drag paths using this mechanism. Two choices are therefore presented to the user (as are displayed in Figure 5): curved lines which show the precise path the object took, or straight lines which show only the difference between starting and ending positions. Successive moves of the same object were concatenated together to produce one contiguous line in either case. The start position of a move action also has a faded imprint of the object, making a move action much like an object deletion and re-creation tied together with a line.

One of the positive features of the sandtrails mechanism is the ability to continue working when the changes are being displayed (in essence, the user sees what trails he or she is leaving for the next user to see). To take advantage of this feature even more, it seems necessary to make the change symbols less obvious, such as varying the colour of move trails from the background colour only slightly, or by using bitmapped techniques. An idea along these lines was indeed attempted; Figure 6 shows the preliminary version of the sandtrails techniques used in conjunction with a semi-transparent screen that was intended to visually separate the change information from the current state of the environment. It was not successful: the screen was too obvious to the user and (at least in this case) distracted her from the application itself as well as the change information. It was therefore rejected.

Replay

The *replay* mechanism represents a more literal display of change information than does sandtrails (Table 3). Like the name suggests, the mechanism, when requested, replays all of the change actions that have occurred since the last time the user was in the system. Firstly, the environment is reset to the state as the user last saw it. Then each action is replayed in order of occurrence and eventually brings the environment to its current state.

Object creation:

In the first implementation, the new object simply appears in the location it was created. Because the quickness of this action is hard to notice, a better idea would be to make use of *process feedthrough,* as discussed by

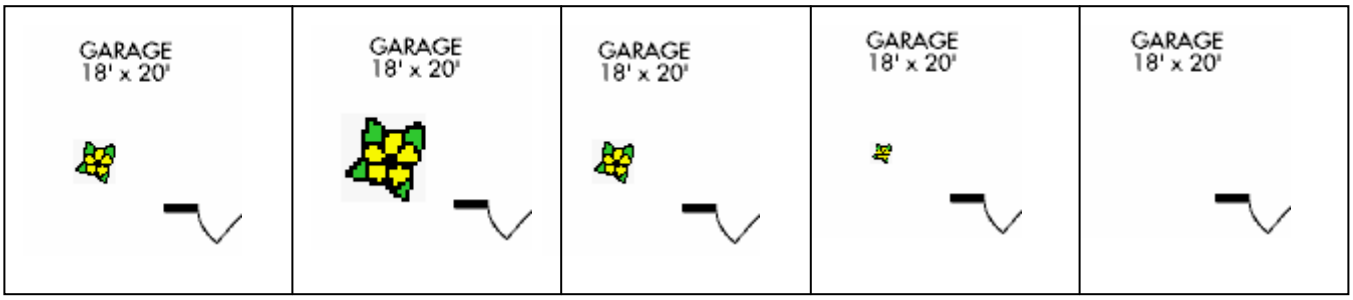

Figure 7 – "Supernova" animation of a delete action

Gutwin (1997). Such techniques would involve showing all intermediate steps of a past operation, including the user selecting creation commands from menus, filling in dialog boxes, etc, which aid an observer to piece together the actions that lead to the result (as well as call more attention to an otherwise instantaneous action). Indeed, this is what eventually was added. See the description of the 'supernova effect' used for object deletion (below) and the discussion of test results in section 4.

Object deletion:

When dealing with object deletion we once again often suffer from the fact that deletion of an object may not be entirely obvious to an observer since it occurs quickly and produces no action feedback except that of the object disappearing. As quoted by Gutwin (1997), Ellis, Gibbs, and Rein (1991) observed that "other's actions are inherently more difficult to interpret than your own, and that smooth animation of changes can aid interpretation." To remedy this situation, an animated 'supernova effect' was built into the object deletion event. Now, when an object is deleted, "it does not simply disappear, but swells up for a moment before gradually fading away" (Gutwin, 1997). Figure 7 shows how this technique has been applied here.

Object movement:

This is the most literal of all events: the object simply retraces the path of movement as if the user was dragging it.

One of the problems with this implementation of a replay feature is caused by the fact that we are working in a groupware environment. In this context, it is entirely possible that two users move about two objects at the same time. As discussed in section 3.2 above, our testbed does not timestamp intermediate steps of a move operation, and so during the replay, what were once simultaneous operations would instead appear one after another. Also, the current implementation does not allow partial replays; there is no way to ask the mechanism to show only the last few changes for example. This could be remedied by the use of a slidebar indexing the replay, such as those seen in many sound and video computer applications, or with the help of a textual index, such as the one discussed in the following section.

The Change Index

The change index is a separate window that summarizes for the user the changes that have occurred. Each change occupies one row in the index and is represented (at least in the first implementation) by an english sentence, making it a play list in chronological order. The summarized and detached nature of the change index places it in the symbolic, separate category as shown by Table 3. The first version of the change index can be seen in Figure 8. As shown in the figure, the only difference between the display of any of the different changes is the wording of the sentence.

The length of the index (i.e. how many entries it contains) gives a good indication of how many changes have occurred since the user was last in the environment. However, it was known from the outset that the change index used by itself would not be very useful for answering many of the other questions a user might ask, due to the effort it would take a him to connect change information to it's respective object in the main screen. It was decided, therefore, to link the change index with the replay mechanism: clicking on any entry in the change index causes the selected change to be replayed in the main screen.

5 User testing and design iterations

The problem of displaying change histories in object-oriented groupware spans many applications and domains. The generalist approach I am taking to the design and testing of change display mechanisms will not be

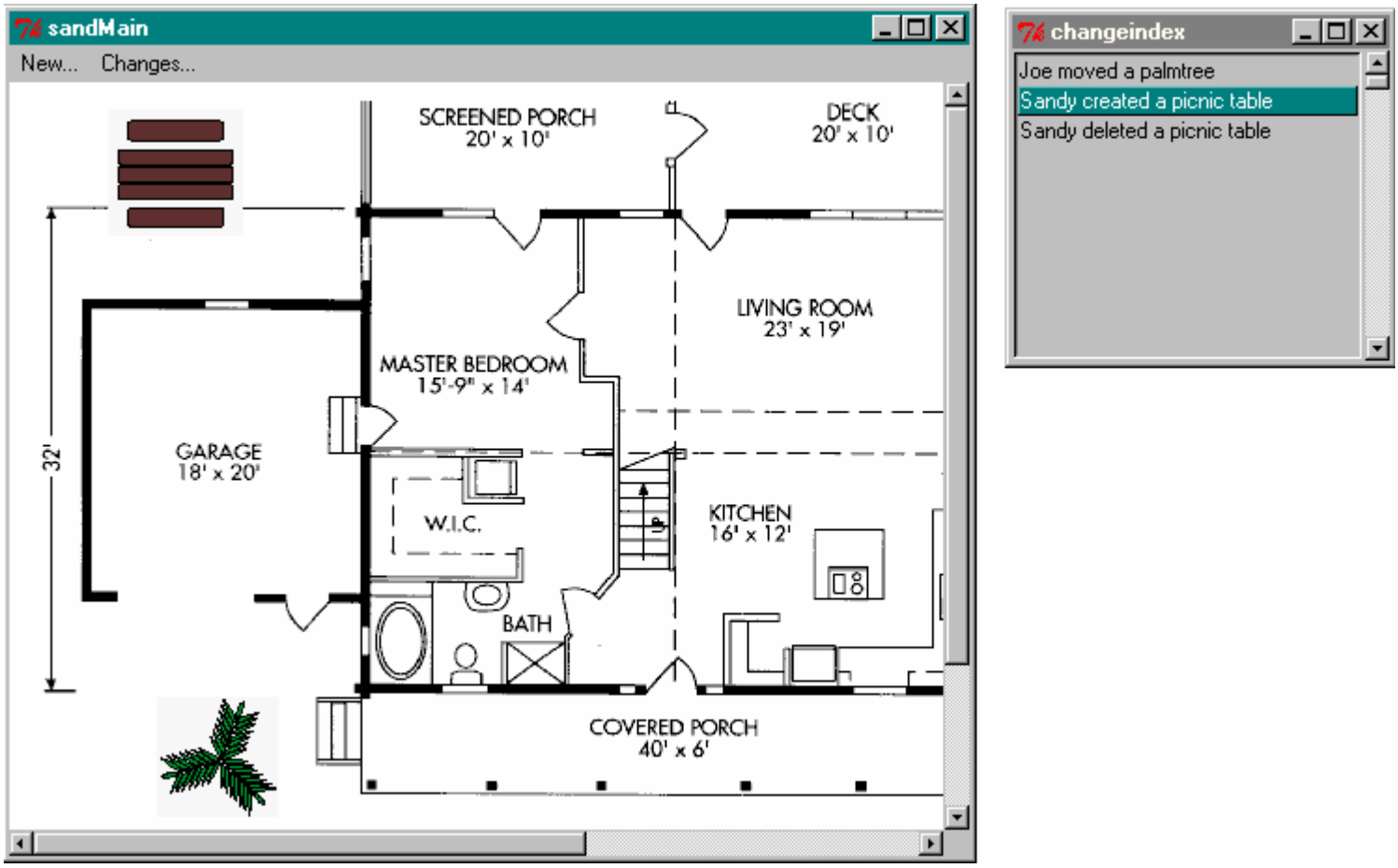

Figure 8 – The first version of the change index

useful in proving their suitability for any one application. Therefore, a formal testing procedure in which a mechanism can pass or fail is not terribly useful. Instead, I chose to follow a somewhat more informal method whereby the mechanism(s) could be improved between trials, thus incrementally fixing the problems detected by the usability tests. What follows is a summary of test results and improvements made to each of the change mechanisms that were introduced in the previous section.

Testing of the change display mechanisms was accomplished with the help of four volunteer subjects. Each subject was introduced to the landscaping program (the testbed application) and shown how to add, move, and delete objects from it. They were then asked to plan the landscaping of a home, by placing at least 8 objects onscreen and moving them to their desired locations. The subject then left the room while the experimenter, acting as another user in a groupware environment, changed the layout that the subject has completed by adding and deleting objects as well as moving existing ones. The subject returned, received some training about how to interact with the change display system being tested, and was then asked the questions listed under the Who, What, When, Where, Why, and How headings in section 4.1. The subject was encouraged to make use of the display mechanism to help answer the questions. The number of questions the user could answer about past events in the environment and the apparent ease of these answers were taken to be an indication of the success of the change mechanism. The user's memory plays a part in answering some questions (for example, Where did the person make a change?) but not in others (Who was here, and when?). This mixture will be present in any real-world applications as well.

In addition to the experimental sessions with subjects, general opinions about the look and feel of the display mechanisms were elicited from several colleagues and this provided many additional ideas and improvements.

Sandtrails

This testing procedure pointed out some flaws in the sandtrails mechanism. Firstly, the symbols were unproportioned in the amount of attention they attracted from the user. For example, the symbols for object creation (a shiny star) and for object deletion (a gray shadow of the original object) were sometimes overlooked, while the trails

caused by object movement were far too distracting. Secondly, neither straight-line nor curved line move representations seemed to get their point fully across. Some users have commented that the curved line display makes it more obvious that it is the path a previous user took to move an object, as the irregular curves don't look computer generated. However, the curved line is hard for the user to trace and produces too much screen clutter. No test subjects switched the line styles during questioning without some prompting from the experimenter, even though they all knew that it was possible. Thirdly, this mechanism does not communicate any information pertaining to the order in which the changes were produced. This is fine for a task such as moving furniture around a floorplan, but may be a critical oversight in other problem domains. Fourthly, the system performs much better when there are few changes than when there are many. The resulting number of symbols displayed when many changes have occurred is simply too much to handle.

Some modifications were made to the sandtrails mechanism to fix some of these problems as testing proceeded. The thick, gray lines used to display object movement were made to better follow the sand metaphor by bitmapping them and turning them yellow. Unfortunately, while indeed making long trails less overpowering, this change caused its own problems, namely that small movements were no longer noticed at all. Another change was the removal the shadowed object found at the start of a sandtrail due to comments that it was redundant (the confusion caused by this object is seen in Figure 9).

On the positive side, sandtrails gives a good overview of activity. It was very easy for test subjects to identify which areas of the environment had experienced the greatest activity. Support for multiple groupware users (the Who question) could be included by color-coding the symbols or by using textual or pictorial annotations (see Gutwin, 1997, pp. 111-124).

Replay

One of the replay's strong points is its literal nature, because it shows the user the same information that was available when the action was originally performed. Attention-getting devices such as the supernova effect also proved to be a good way to accommodate for the user not observing the correct area of the workspace to see the information she asked for. In fact, the supernova deletion effect worked so well it was decided to add a similar, but opposite, effect to the object creation action.

While sandtrails gave a good overview of activity, this is one of the replay's weakest spots. My first version of the replay mechanism would play changes from start to finish in the order that they were performed, which very often did not match the user's desire, for example to see all changes pertaining to one object. Some sort of organization was needed to help the user filter the changes in order to answer questions. This organization eventually came in the form of the change index.

Change Index

Originally designed as a method of overview and organization to be used with the replay mechanism, user testing revealed that the change index had much more promise. Clicking on entries within the change index window caused the selected change to be played out in the main window, and some summarization of changes was provided (for example, a creation followed by a move was classified under a single 'creation' entry in the index). It was noticed, however, that the subjects were performing mental sorting and filtering in order to use it as more than just an index. Often, for example, users would search down the list and click on each entry dealing with a certain named object in order to see the modifications made to it. Sometimes, they would click on an entry just to see which object it referred to on the main screen. All this clicking and filtering, however, seemed to be quite laborious and it was evident that the change information required a fair amount of time to absorb using this method. There was some question as to how much information should be collapsed into one index entry as well. This was evident when many test subjects would replay and be confused by an almost imperceptible move action in which the previous user had only adjusted the object's location by a few pixels. Also, having to split your attention between two windows was somewhat distracting.

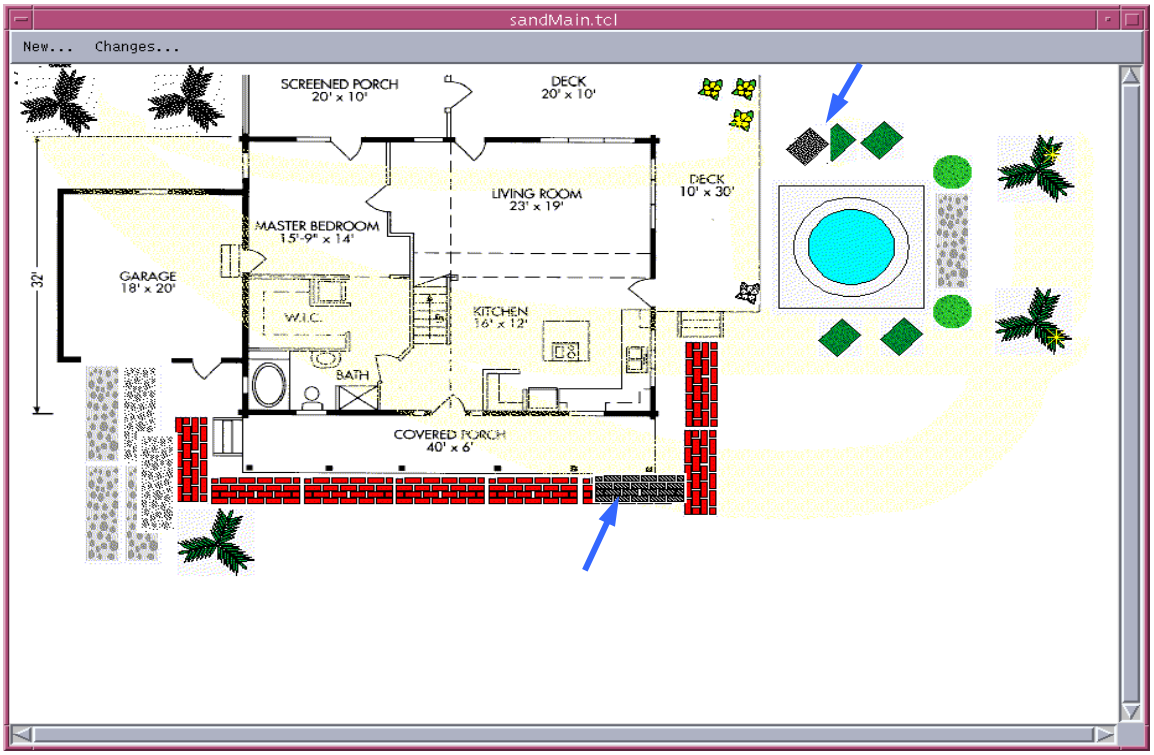

Figure 9 – This test subject was confused about the symbolism presented in areas where slight adjustments were made (see arrows). Eventually, the grayed 'source' objects were removed completely.

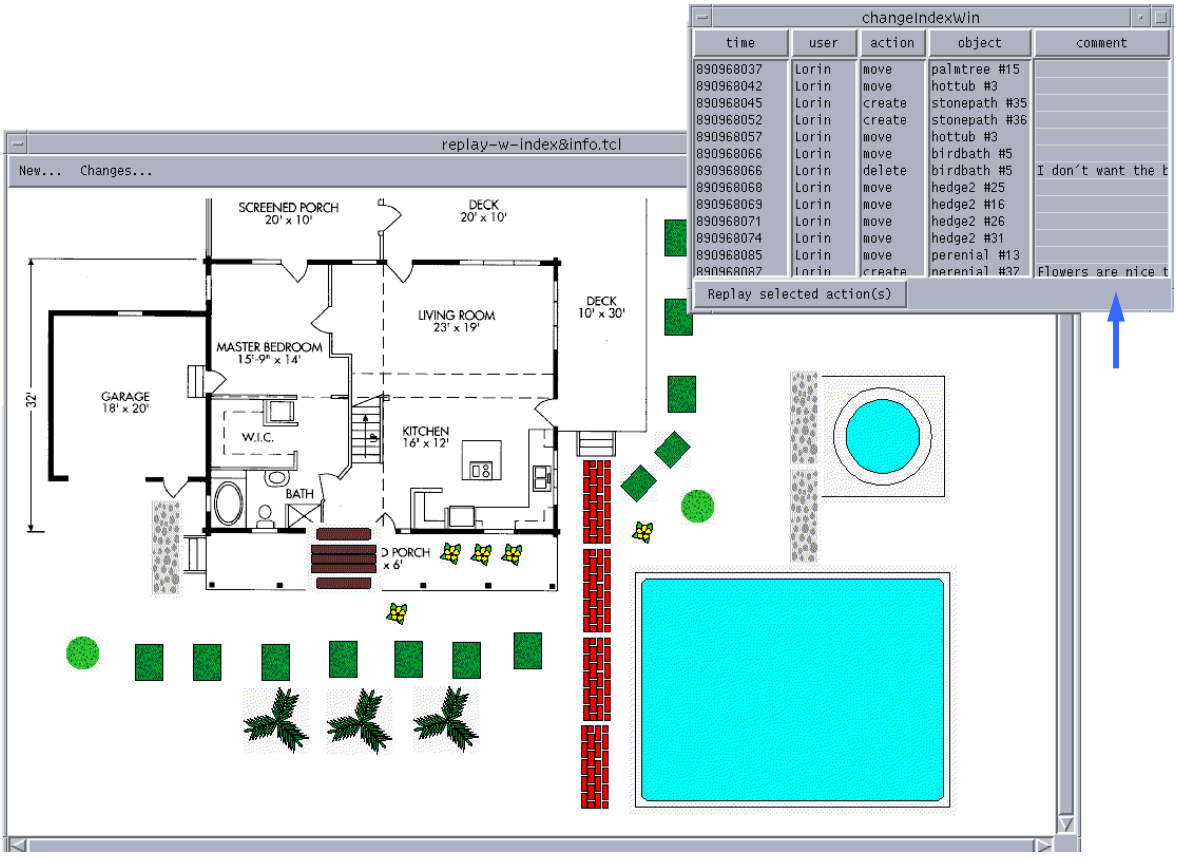

Figure 10 – This test subject had some help from the comment fields and understood why the last user had created so many flowers.

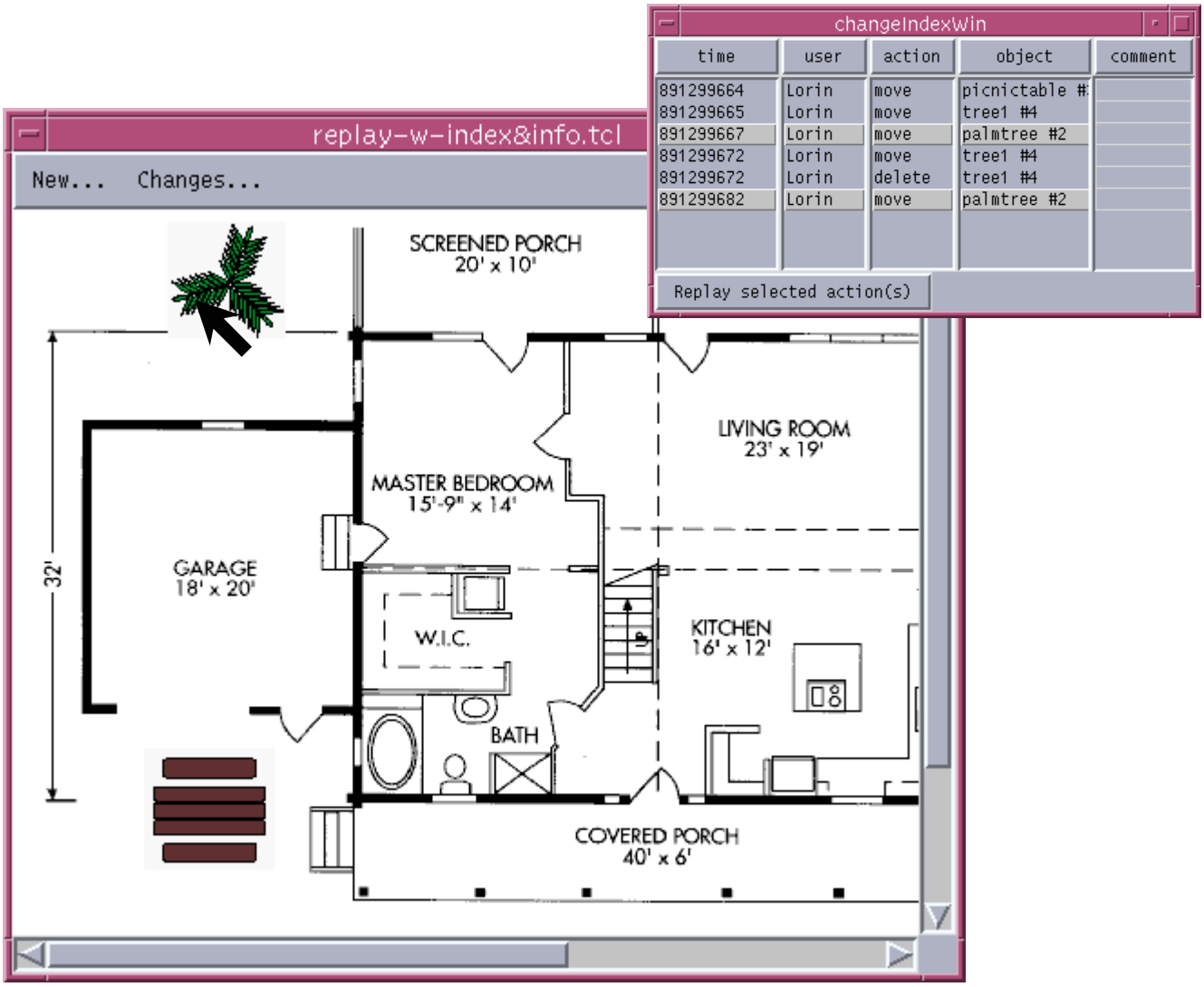

Figure 11 – Passing the mouse over objects highlights the corresponding changes in the change index.

Out of all the change display mechanisms, the change index experienced the most modification between testing situations. The english sentences were split up into columns and the columns were made sortable, giving the user the ability to sort the index on attributes such as type of change, user who made the change, object the change was performed upon, or the time of the change (see Figure 8). As well, the user could select and replay ranges of changes by dragging the mouse overtop of the desired range. These two features combined allowed the user to answer many more questions than before. For example, it was now easy to replay only the changes that had occurred to a particular object by first sorting on the object column and then replaying a range of entries.

The addition of a comment field now enables a user to annotate the changes he was making. This enables subsequent users to answer the Why question (see Section 4.1). As well, to reduce the amount of mental work the user must do to associate entries in the change index with their respective objects in the main window, the entries in the index now highlight themselves when the mouse is passed over objects in the main window (as seen in Figure 8), and conversely, the objects are annotated (currently with a shiny star) when the mouse is passed over any index entry that refers to it.

These improvements to the change index made it fairly versatile. As discussed by Neuwerth et al. (1992), the best change display mechanisms are the ones that let the user customize it for his or her needs at any time. The coupling of the change index with the replay mechanism seemed natural, but there is also no reason why it could not be combined with the sandtrails mechanism, although this was not attempted here.

	Sandtrails	Replay	Change Index	Change Index + Replay
Who?	Color coding or other annotations	None	User column in index	User column in index
What?	Different symbol for each change	Literal replay of changes	Textual description of change	Textual description, optional replay
When?	None	Sequence of changes only	Timestamp column in index	Timestamp column in index
Where?	Locations of symbols	Location of replayed actions	None	Location of replayed actions
Why?	None	None	Comment column	Comment column
How?	Some support, such as curved movement paths	Literal replay of changes + exaggerated creation and deletion (supernova)	None	Literal replay of changes + exaggerated creation and deletion (supernova)

Table 4 - How the change display mechanisms answer the 5 W's and 1 H.

6 Conclusion

The three display mechanisms discussed here have benefits and drawbacks that should be considered when designing similar systems for groupware applications. The sandtrails mechanism gave a good overview of changes but failed to be useful when the number of changes grew, or when the sequence of changes was important. The replay mechanism was not useful on its own since no method of selective replay was available. It was, however, the most literal presentation and, coupled with attention-attracting devices such as the supernova effect, the easiest to interpret. The change index could indeed be used by its self, but was most useful when coupled with another mechanism. The ability to sort by attribute and the collapsing of information into single entries are useful features. Mouse-over highlighting helps associate index entries with the corresponding objects in the main window and vice-versa. Extra columns on the index allow the addition of goodies like comment fields.

There are, however, some identifiable problems with the procedure used in this project that may have some impact on the reliability of the results. For instance, the sole use of the 'landscaping' testbed application for testing calls into question the generalizability of the results to other, more complicated situations. Although every effort was made to have the landscaping program be representative of any object-oriented environment, it would have been better to incorporate the same mechanisms into several other applications (especially true groupware) to see how they perform there. Time restraints prevented me from accomplishing this. Also, because the needs of the user will probably change over time as Neuwirth (1992) pointed out, the mechanism should have some user controllable flexibility to be the most useful. The mechanisms introduced here have limited flexibility from the user's point of view, but can be used as a starting point for more advanced schemes. The area of alternative visual organizations of change information (building on the success of the change index) seems an especially promising area that deserves more research.

It is important to remember that any one mechanism will not be valid across all domains. The goal here was to generate and test ideas about displaying changes in groupware applications, but it is doubtful that the same display methods will work equally well for a concept map editor as for a drawing package. Nonetheless, the more ideas available to the groupware designer, the more likely they will be to chance upon one that matches their application, and the people who use it will benefit as a result.

7 References

AT&T Bell Laboratories. 1993. Video. High Interaction Data Visualization - using Seesoft to Visualize Program Change History. In Computer Graphics, issue 88: Interchi '93 Technical Video Program.

Gutwin, C. 1997. Workspace Awareness in Real-Time Distributed Groupware. Ph.D. thesis, The University of Calgary

Gutwin, C. & Greenberg, S. Personal Interview. October 1st, 1997.

Gutwin, C., Greenberg, S. and Roseman, M. 1996. "Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation." *Proc. CHI'96 People and Computers XI*, pages 281-298.

Hill, W.C. & Hollan, J.D. 1992. "Edit Ware & Read Ware". *Proceedings of CHI '92*, pages 3-9.

Kurlander, D. & Feiner, S. 1998. "Editable Graphical Histories". *IEEE Visual Languages Workshop*.

Neisser, U. 1976. Cognition and Reality, W.H. Freeman, San Fransisco.

Neuwirth, C.M., et al. 1992. "Flexible Diff-ing In a Collaborative Writing System." *Proc. ACM 1992 Conference on Computer-Supported Cooperative Work*, pages 147-154.

Ousterhout, J. 1996. *Tcl and the Tk Toolkit*, Addison-Wesley.

Roseman, M. 1998. TeamWave (Version 3.1b2) [Computer program]. Calgary:TeamWave Software Ltd.

Roseman, M. and Greenberg, S., 1992. "GROUPKIT: A groupware Toolkit for Building Real-Time Conferencing Applications", *Proc. of the Conference on Computer-Supported Cooperative Work (CSCW'92),* pages 43-50.