Summary Review
Timewarp: Techniques for Autonomous Collaboration
Edwards, W. Keith


Reviewed by:
Jennifer Bevan
Joerg Meyer

Summary:
Edwards and Mynatt define autonomous collaboration as, "periods in which groups of users work independently on a loosely-shared artifact." As an example, they offer the case where an author distributes copies of a document for review. The reviewers read the document and make suggestions which then must be integrated. As the author integrates changes, the resulting document becomes progressively more dissimilar to the document on which the recommendations were based. To understand and integrate further suggestions, the author may need to be able to work with the document as it existed when the changes were recommended.

Timewarp is a framework which promotes this type of collaboration by increasing user awareness of changes, and providing the flexibility to see and modify documents at any point in their development history. The ability to see and change "earlier" document states allows changes to be seamlessly propagate to all subsequent versions of a document. The Timewarp framework is a toolkit with which to develop specific applications. The toolkit provides five modes for editing timelines (typically called version histories in other systems). The first, split mode, behaves like branching in many other systems. It leaves the existing timeline alone, and creates a new timeline rooted at the version where the split was added. Split mode is complemented by join mode. Join mode is often referred to in other systems as merging. It allows separate timelines for an artifact to be combined into a single result. Application builders must define conflict resolution strategies. Insert mode and retained mode are unique to Timewarp. They both allow changes to be inserted anywhere in the timeline. Changes are propagated to all "younger" versions of the artifact. Insert mode makes the change without preserving the previous timeline. Retained mode creates a copy of the previous timeline, and makes the change in the copy, effectively resulting in a split at some prior point in time. The final mode, couple mode, is referenced on page 221 of the Timewarp paper, but is never discussed.

Ratings:
Importance mean: 4
Importance mode: 4
Writing mean: 4.2
Writing mode: 4
Commentary:

Usefulness of technique:
Reviewers of the Timewarp article expressed many concerns about the usefulness of the Timewarp technique. The two primary areas of concern were automatic change propagation and overwriting of "earlier" versions.

Automatic change propagation as described by Edwards and Mynatt is dangerous because erroneous changes would be automatically propagated to "current" versions of a document. Furthermore, changes which produce correct results in the version in which they are made may not produce correct results in the "later" versions to which they are propagated. For example, suppose that Mary decides to rename a variable in version X of some code which is a predecessor to version Y. In version Y, however, there is some additional code which uses the variable. Unfortunately, only the code shared by version X and version Y is updated, so version Y is left with undeclared variables in the section it does not share with version X.

Although Timewarp provides facilities for defining conflict resolution policies, it relies on the developers of applications to define those policies for their applications. As we have seen in other papers, algorithms for automatically resolving semantic conflicts rapidly become unwieldy as the number of possible operations increases. This suggests that it may not be possible to define reasonable conflict resolution policies for real-world applications. As the preceding example illustrates, use of the Timewarp techniques (especially insert mode) without adequate conflict resolution is extremely risky and relies heavily on humans to avoid errors.

Despite the potential problems, several reviewers suggested situations in which a Timewarp-based application could be effective.

One reviewer proposed pair-programming as an effective situation in which to utilize a Timewarp-based application. For example, he proposes that using a Timewarp-based editor, one member of the pair could be working on detailed coding, as the other works on building the application framework. Although technically pair-programming requires that only one member of the pair at a time actually works on the code, a Timewarp based application could certainly be effective in allowing multiple people to work on different aspects of the code simultaneously. Combining Timewarp functionality with a concurrency control algorithm similar to that proposed in "Concurrency Control in Groupware Systems" could be an effective approach.

Another reviewer proposes a policy for utilizing Timewarp modes which would reduce the risk. She suggests using retained mode for large-scale changes, while allowing insert

---

mode to be used for small-scale, tightly-coupled changes. A variant on this policy would, in a sense, create joint workspaces. Small working groups could extract files from the main repository using either split or retained mode. They could work collaboratively using any modes they chose. When ready, they would commit back to the main repository using either join or retain mode (depending on whether they want to merge, or maintain a separate development line). This could be an effective approach in very large-scale development efforts, where careful control is necessary on the main body of code, but where development on individual pieces happens in smaller groups in a collaborative fashion.

**Scalability:**
In addition to general concerns about the robustness of the Timewarp techniques, several reviewers had technical concerns. The primary technical concern was scalability. The Timewarp article does not really address scalability of the framework, except to indicate that caching is used to help reduce the impact of slow network connections. Several reviewers were concerned about the impact of increasing the number of users of an application. Since the Timewarp article does not discuss its time-synchronization techniques in detail, it is impossible to assess the impact on response as the number of users increases. Reviewers also question the scalability relative to geography. How do Timewarp applications handle tightly-coupled collaborative sessions across continents?

**Usability and UI:**
The final significant area of concern is around usability of Timewarp applications. Reviewers expressed concern that propagation changes to users in real-time would be distracting. Also, most people perceive history as immutable. It may be quite difficult for users to get used to the notion that history can change.

Although the notion of changing history made many reviewers nervous, several of them liked the notion of scrolling through history. One reviewer even proposed user interface elements that would help visualize changes over time. He proposed that as the user scrolls through a document's timeline, changes are animated (e.g. deleted section fade out, added section fade in, moved sections visibly move). The visualization would help users understand how the document changed over time. He goes further to propose that metadata be attached to changes so that as the user watches the document evolution, he or she can see additional information about the changes.

Animation of changes could be combined with the "magic lens" idea to provide a visualization of how a document was affected by particular changes. For example, it would be possible to visualize how a particular author has impacted a document over time, or how changes made to address a particular defect impacted the document.

**Relations to other systems:**
One reviewer suggested that traditional branch and merge techniques can be used to simulate all of Timewarp's modes. Obviously split and join modes are Timewarp's terminology for traditional branch and merge operations. Retained mode is essentially like retroactive splitting. It could be simulated using branching and merging by creating
a branch, copying all files on the original branch onto the new branch, making a change at the branch point, then propagating the change to the copies on one branch and not to the copies on the other branch. Insert mode can be simulated as per the following example. Suppose there are currently three consecutive versions of a file, version 0, version 1, and version 2. We want to insert a new version, called version 3, after version 1. To do so using branching and merging we first make our change to version 1 and save it to a new branch as version 1.1. We merge 1.1 into version 2 to create version 3. Next, delete version 2 and rename version 3 to version 2. Finally, rename version 1.1 to version 3. Obviously this is rather cumbersome, and is dependent on the ability to modify the version tree directly, but it is conceivable.

Another reviewer suggested applications of the history-visualization and magic lens ideas to traditional systems. He suggests using the history-visualization technique to view valid configurations of source files containing meta-language conditionals. He also suggests using it in an object-oriented environment where fine-grained version control (e.g. at the method level, rather than the class or file level) is desirable. History-visualization could be used to represent the relationships between a current class and other method implementers (superclasses, subclasses, sibling classes) and even method invokers (senders). The concept of "magic lenses" would be useful in almost any collaborative authoring environment (even when serial authoring is enforced). It can enhance awareness for any dimension on which it's possible to collect metadata. This reviewer suggests that application of these techniques could be used in an object-oriented code browser (or, really any environment) to help with refactoring, version control and merging.

Reviews included in summary:
Guozheng Ge
Kai Pan
Kai Wang
Mike Baker
Rita Garcia
Teng Xu
Dorrit Gordon
Sonja Ellefson
Jennifer Bevan