

## Consistency Management for Distributed Collaboration

JONATHAN REES, SARAH FERGUSON, and SANKAR VIRDHAGRISWARAN

*Crystaliz, Inc., 9 Pond Lane, Concord, MA 01742 <sv@crystaliz.com>*

Collaborative enterprises, such as the development of complex software systems or web sites, are vulnerable to accidental conflicts between participants' activities. They face the twin threats of information loss, where one person accidentally erases or overwrites another's work, and inconsistency, where one person alters the *context* of another's work so as to make that work invalid. The primary purpose of collaboration support tools should be to enable participants in a collaboration to proceed with confidence, knowing that their work is safe from direct and indirect threats, and that they are not accidentally compromising the work of the other participants.

The general problem of concurrency control and consistency management in such collaborative projects is addressed by *configuration management* (CM) systems, which have been the subject of two decades of academic research and industry experience [Feiler 91]. Most CM systems assume central or tightly coupled object repositories, a high degree of trust between participants, and/or implicitly understood consistency criteria. These conditions that fail to apply in many situations in which distributed collaboration is called for.

We believe that a new kind of CM system needs to emerge if the full potential of the Internet to facilitate collaborative work is to be realized. Many kinds of collaboration, such as

---

The work reported here was supported in part by DARPA under contracts N6001-96-C-8512 and DAAH01-96-C-R190.

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 1999 ACM 0360-0300/99/1200

those between supplier and customer or between multiple contractors working on a single project, are not well served by a centralized or tightly-coupled approach. New CM systems must support loose coupling between collaborators, who will have independently evolving, independently managed repositories that may be mobile or intermittently disconnected.

We advance the concept of *change proposal* as a way to manage change in a distributed collaborative environment. A change proposal is a structured object that describes changes to a set of other objects, together with consistency conditions that should be met in order for the changes to be considered sensible. A change proposal may be applied directly to a shared or private object repository, submitted to an editor or manager for review and possible modification, or published for use by multiple recipients.

### CHANGE PROPOSALS

CM systems may generally be classified as *version-oriented* or *change-oriented*. In version-oriented CM, typified by RCS [Tichy 1985], the focus is primarily on controlling revisions of individual items (files, images) and only secondarily on relating revisions to one another. In change-oriented CM, item revisions are always considered as part of a logically coherent *change set*. Version-oriented CM is dominant in practice at present, but the virtues of change-oriented CM have been promoted vigorously [McKay 95], and have been adopted by at least one commercial CM system (Aide de Camp/TrueSoft). We advocate change-oriented CM because it is more natural and robust than version-oriented CM in the absence of central control and reliable coordination.

The change set idea only describes change; it does not directly address the question of when it is meaningful to make a change. For example, a change to web page A, such as the addition of a link to page B ("see page B for a description of the nutria"), may only be known to make sense if some property holds of page B

(e.g. that it includes the text “the nutria is a medium-sized rodent”). Similarly, the installation of a set of software patches may be only known to work when NotScope version 9.6 is installed.

To capture such dependencies, we extend the change set concept slightly. Define a *change proposal* to be a change set together with a set of conditions that must hold before the change set can be meaningfully applied. When a change proposal is received by an instance of the CM system, the conditions are checked against the local repository and, if the conditions are met, the change set is applied. The condition checking and change set application occur as an atomic action.

Conditions may involve either items to be modified by the change set or other items that somehow affect the integrity of the change. For simple items, a condition might be simply that the item be in a particular state (e.g. a file must hold revision 17 of that file), while compound items such as tables and directories may have more fine-grained conditions, involving the state of an individual row or entry (as opposed to the state of the compound item as a whole).

Change proposals prevent lost work in the same way that they promote consistency. When a change proposal is prepared, an item X is initially read from a repository in state  $S_1$ , and a proposed new state  $S_2$  is prepared. The change proposal will assert that for X to change to  $S_2$ , it must start out in state  $S_1$ . If the target repository’s X is in some other state, say  $S_3$ , then a concurrent change has taken X from  $S_1$  to  $S_3$ , and  $S_3$  probably holds information that would be lost if X were overwritten by  $S_2$ . The proposal’s conditions aren’t met, so its change set is not applied, and the information in  $S_3$  is safe. Extra work is then needed in order to reconcile  $S_2$  with  $S_3$ , for example by merging them to make a new state  $S_4$ , but at least  $S_3$  has not been lost.

Item creation and deletion are incorporated into change proposals in a natural way. If item X is to be created, then a precondition (necessary to avoid loss of work) is that X not exist beforehand; similarly, for deletion, it is necessary to specify that a recognized state is to be deleted, lest a state containing useful information written in the meantime be received in the meantime.

Change sets and change proposals are intended to complement, not replace, other CM facilities such as locks, notifications, and long transactions [Kaiser 1994]. If users are aware that others are working on certain items, they may act to avoid the overhead of a merge: They may wait for a lock to be released or arrange to coordinate activity with the individual who holds the lock. Change proposals serve a different purpose: to handle situations in which locking and notification are infeasible, unreliable, or disallowed, or in which the value of proceeding with development of a parallel change outweighs the cost of merge.

A change proposal is a kind of document, and as such has first-class status – for example, it may be transmitted via email, or stored in a file system. If a proposal is sent to someone, the recipient may freely ignore it, return it to the sender for improvement, or subject it to scrutiny and testing before applying it to a repository. The recipient may make additional modification to the proposal before applying it, such as merging in concurrent changes, and may modify or ignore the acceptance conditions specified in the proposal. For example, a proposal that says that a patch is only known to work with NotScope 9.6 may also work with NotScope 9.7, and the recipient may test to see whether it does.

Change proposals resemble some recent package management (document or software distribution and update) schemes [e.g. Bailey 1997]. In a way, we are calling for a synthesis of CM and package management.

## DISCUSSION

The treatment of change proposals as explicitly manipulated items interacts synergistically with distributed collaboration and item replication. Change proposals support arbitrary hierarchical, multi-stage workflow regimes, serial handoffs, and comparison of alternative proposals in a principled way.

Our implementation prototype, which we call A4, centers around tools for proposal preparation and reception. To prepare a proposal, the user creates a workspace with an associated rule for obtaining initial item states (e.g. that they are to come from a specific reposi-

tory/workspace). Items are located using a browser and read according to the rule, and updated states (prepared using a standard application such as a word processor) are kept in a local workspace. A record is kept of reads and writes to the workspace, allowing a change proposal to be extracted at any time: states read (such as  $X=S_1$ ) become the proposal's preconditions, while states written (such as  $X:=S_2$ ) become its change set.

To reduce the size of change proposals, states are specified not as entire file contents, but as globally unique state identifiers and/or deltas, as appropriate. File checksums and/or modification timestamps may substitute for state identifiers when necessary.

A change proposal may be directed to a recipient, who again has a workspace. Incoming proposals may be checked for their effect on the consistency of the workspace, merged with inconsistent sibling proposals, and then accepted, perhaps on a trial basis. Proposals may be rejected, withdrawn, or even reordered – a set of proposals may be consistent when applied in one order but not when applied in another.

When merge is necessary, either the preparer or the recipient may do the necessary work. The change proposal may be modified using a new workspace, and its precondition set may be updated after merging individual items. This makes a resubmission more likely to succeed.

Distributed CM is an important and active area of research [van der Hoek 1995]. Constructing a useable CM system of any kind is a major undertaking, with much of the effort going to smooth integration with the host operating system and applications. While centralized CM systems are mature and well established (at least for software development and CAD), there is no widely accepted practice of distributed CM. It is likely that industry will adapt existing successful version-oriented CM models for Web use [Slein 1998], but true decentralized collaboration will likely require more radical approaches.

We have outlined one particular approach to distributed CM based on a change proposals concept. Change proposals prevent information loss by keeping track of what information is *expected* to be revised, and they help ensure

that consistency is preserved across application of change sets by stating consistency relationships explicitly. By separating change proposal standards and machinery from the orthogonal issues of concurrency control and repository management, we take a step toward realizing global internetworking's potential to support distributed collaboration.

## REFERENCES

- BAILEY, E. C. *Maximum RPM*. Red Hat Press/MacMillan, 1997.
- ESTUBLIER, J. (ed.). *Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops Selected Papers*. Springer-Verlag LNCS 1005, 1995.
- FEILER, P. H. Configuration management models in commercial environments. CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie-Mellon University, 1991.
- KAISER, G. E. Cooperative transactions for multi-user environments. In Won Kim (ed.), *Modern Database Systems: The Object Model, Interoperability, and Beyond*, pages 409-433, ACM Press, 1994.
- MCKAY, S. E. The state of the art in concurrent, distributed configuration management. In [Estublier 1995], pp. 180-193.
- SLEIN, J., et al. Requirements for a distributed authoring and versioning protocol for the World Wide Web. RFC 2291, Internet Engineering Task Force, February 1998.
- TICHY, W. F. RCS – A system for version control. *Software – Practice and Experience* 15(7):637-654, 1985.
- VAN DER HOEK, A. Does configuration management research have a future? In [Estublier 1995], pp. 305-309.