Use of operational transform

David Barrett-Lennard
28 June 2007

Requirements

- Support for repository
  - Efficient translation of working set from one state to another. This means we need to be able to retrieve any delta for two given vector times.
  - Full compression before check-in. Compression ignores original (s,t) values.
  - Update while have local edits in progress
- Support for online collaboration
  - Allow for latecomers

IDEA: We allow for differ types of compression as follows

- There is only one non-lossy merging algorithm – which accounts for (s,t) values
- To perform lossy compression first recurse down through the operation and apply the same (s,t) values.

Lossy compression algorithm

Lossy compression can be applied to a given operation. It involves the following

- Discard all disabled move intervals. Note therefore that aliasing of extraction intervals is no longer required.
- If there are multiple deletes on a given interval then only keep one and discard all the rest.
- Apply same (s,t) to all intervals
- Coalesce adjacent create, delete or move intervals

Requirement for allowing repositories to merge

A crucial requirement is that (s,t) uniquely identifies a check-in to a repository, anywhere in the world. Therefore it is important to provide unique site identifies. A guid seems a good choice. Note that as required there is universal agreement on a total ordering on site ids.

However problems can arise, such as through the following scenario

1. An LSS file is copied so that there are two computers C1,C2 with the same siteid.
2. A user on C1 does some work and checks the changes into repository R1.
3. A user on C2 does some work and checks the changes into repository R2.
4. Later R1, R2 are merged but there is a failure due to repeated (s,t) values.

The following are some ways that repeated (s,t) can appear in the system

- A store is copied
- Just after a check-in, but before the operation is written to non-volatile storage there is a power failure.

- A user recovers a store from backup
- A user recovers an entire hard-disk from a mirror copy.

It is proposed that we avoid problems as follows

- It is assumed that the repositories have excellent network connectivity, reliability and durability of checked-in operations. It is far more likely for a client machine will roll back to an earlier state than for a repository to be missing check-ins.
- During check-in to a repository there is a validation test for a repeated (s,t). If a client store has been copied or rolled back to an earlier state the problem will be detected by the repository. If validation fails there is the option to assign the client a new site id. This makes good sense if the store has been copied. Another option, suitable if the client has rolled back to an earlier state is to require an update from the repository.
- Repositories can use an interactive session in order to get fast agreement on a vector time representing all check-ins to all repositories!

With this approach we will avoid a proliferation of siteids, because we assume the repository will tell us exactly when we really have to generate a new siteid because of client roll back.

Lazy loading of objects from the repository

At any given moment a working set is assumed to be associated with specific *base vector time*. This is used to retrieve objects on demand from one or more repositories.

We generally want to avoid interrupting the user during the retrieval of objects from the repository. Therefore it is necessary to display something suitable in place of an object while it is waiting to be downloaded. For example, a multi-res-image simply displays at a lower resolution while waiting for tiles to be downloaded. Many viewable components will simply be transparent while waiting for the content to appear.

prefs and lazy loading

We need a version of a pref that supports asynchronous loading of objects from disk, or downloading of objects from a repository. The PersistStore layer shouldn't know about repositories so a hook will be provided to allow the repository system to be called back in order to fault in objects for given OID on demand.

The asynchronous approach will work with the dependency graph system in order to

- establish the dependency on the pref so that when binding occurs, upstream nodes can be marked as dirty
- the active state of the pref as a dependent node causes it to asynchronously load the data. If the node transitions to inactive then the asynchronous download request will be cancelled.

Allowing for an interactive session on a branch

An interactive session is characterised by a particular base vector time. Any process invited to the session will receive this base vector time, and on that basis it can go to a repository (lazily) in order to retrieve the objects being edited in the session.

When an operation arrives and it needs to be executed asynchronously, it can cause objects to be retrieved from the repository. This is all part of the asynchronous process and doesn't interrupt the user's local edits.