# Study of tree conflict handling in selected modern version control systems, with the goal of providing guidelines for Subversion's new working copy library "wc-ng" (Introductory presentation)

Stefan Sperling

Free University of Berlin

June 13, 2008

# Who am I?

- Stefan Sperling
- Studying at inf.fu-berlin.de since 2004.
- Subversion developer since 2007.
  Been working on Subversion's "tree-conflicts" branch and other areas.

# Revision control basics

Does **everybody** know what these terms mean?

- working copy
- repository
- updating
- committing
- branching
  - feature branch, release branch
  - trunk a.k.a. “main line”
- merging
- text conflict

I can explain them now if you want me to. I **will** lose you after this slide if you don’t know this stuff!

# What are tree conflicts?

- Conflicts at the level of directory tree structure.
- If not handled properly by your revision control tool, they can:
  - Lead to local modifications being lost accidentally.
  - Lead to whole files or directories being lost accidentally.

# Example tree conflict scenario

- You **modify** a file in your working copy.

# Example tree conflict scenario

- You **modify** a file in your working copy.
- Sally **deletes** the same file in her working copy.

# Example tree conflict scenario

- You **modify** a file in your working copy.
- Sally **deletes** the same file in her working copy.
- Sally commits.

# Example tree conflict scenario

- You **modify** a file in your working copy.
- Sally **deletes** the same file in her working copy.
- Sally commits.
- You update.

# Example tree conflict scenario

- You **modify** a file in your working copy.
- Sally **deletes** the same file in her working copy.
- Sally commits.
- You update.
- What do you think should happen in your working copy now?

# Example tree conflict scenario

How does Mercurial 1.0 handle this case?

# Example tree conflict scenario

```
added 1 changesets with 0 changes to 0 files
(run 'hg update' to get a working copy)
$ hg update
 local changed foo.c which remote deleted
 use (c)hanged version or (d)elete?
```

It detects the tree conflict and asks us how to proceed.

# Example tree conflict scenario

How does Subversion 1.5 handle this case?

# Example tree conflict scenario

```
$ svn status
M foo.c
$ svn update
D    foo.c
Updated to revision 2.
$ svn status
?      foo.c
```

foo.c fell out of version control! Clearly, there is room for improvement.

# Another example

- Sally **renames** foo.c to bar.c in her working copy.

# Another example

- Sally **renames** foo.c to bar.c in her working copy.
- You **rename** foo.c to baz.c in your working copy.

# Another example

- Sally **renames** foo.c to bar.c in her working copy.
- You **rename** foo.c to baz.c in your working copy.
- Note that the target locations of both moves differ.

# Another example

- Sally **renames** foo.c to bar.c in her working copy.
- You **rename** foo.c to baz.c in your working copy.
- Note that the target locations of both moves differ.
- Sally commits.

# Another example

- Sally **renames** foo.c to bar.c in her working copy.
- You **rename** foo.c to baz.c in your working copy.
- Note that the target locations of both moves differ.
- Sally commits.
- You update.

# Another example

- Sally **renames** foo.c to bar.c in her working copy.
- You **rename** foo.c to baz.c in your working copy.
- Note that the target locations of both moves differ.
- Sally commits.
- You update.
- What do you think should happen in your working copy now?

# Another example

How does Subversion 1.5 handle this case?

# Another example

```
$ svn mv foo.c baz.c
A         baz.c
D         foo.c
$ svn update
D    foo.c
A    bar.c
Updated to revision 2.
$ svn status
A  +   baz.c
$ ls
bar.c  baz.c
$ svn ci -m "log message"
Adding         baz.c

Committed revision 3.
```

# Another example

The same file content now lives at two different locations.
Clearly, there is room for improvement.

# Another example

How does Mercurial 1.0 handle this case?

# Another example

```
$ hg rename foo.c baz.c
$ hg pull /home/sally/repo
pulling from /home/sally/repo
added 1 changesets with 1 changes to 1 files
(run 'hg update' to get a working copy)
$ hg update
warning: detected divergent renames of foo.c to:
 baz.c
 bar.c
$ hg status
A baz.c
$ ls
bar.c  baz.c
```

# Another example

It prints a warning.
What do you guess does Mercurial do when we commit now?

```
$ hg commit -m "commit message"
```

???

# Another example

## Did you guess right?

```
** unknown exception encountered, details follow
** report bug details to http://www.selenic.com/mercurial/bts
** or mercurial@selenic.com
** Mercurial Distributed SCM (version 1.0.1)
Traceback (most recent call last):
  File "/usr/local/bin/hg", line 20, in <module>
    mercurial.dispatch.run()
  File "/usr/local/lib/python2.5/site-packages/mercurial/dispatch.py", line 20, in run
    sys.exit(dispatch(sys.argv[1:]))
  File "/usr/local/lib/python2.5/site-packages/mercurial/dispatch.py", line 29, in dispatch
    return _runcatch(u, args)
  File "/usr/local/lib/python2.5/site-packages/mercurial/dispatch.py", line 45, in _runcatch
    return _dispatch(ui, args)
  File "/usr/local/lib/python2.5/site-packages/mercurial/dispatch.py", line 364, in _dispatch
    ret = _runcommand(ui, options, cmd, d)
  File "/usr/local/lib/python2.5/site-packages/mercurial/dispatch.py", line 417, in _runcommand
    return checkargs()
  File "/usr/local/lib/python2.5/site-packages/mercurial/dispatch.py", line 373, in checkargs
    return cmdfunc()
  File "/usr/local/lib/python2.5/site-packages/mercurial/dispatch.py", line 356, in <lambda>
    d = lambda: func(ui, repo, *args, **cmdoptions)
  File "/usr/local/lib/python2.5/site-packages/mercurial/commands.py", line 557, in commit
    node = cmdutil.commit(ui, repo, commitfunc, pats, opts)
  File "/usr/local/lib/python2.5/site-packages/mercurial/cmdutil.py", line 1179, in commit
    return commitfunc(ui, repo, files, message, match, opts)
```

# Another example

```
  File "/usr/local/lib/python2.5/site-packages/mercurial/commands.py", line 555, in commitfunc
    force_editor=opts.get('force_editor'))
  File "/usr/local/lib/python2.5/site-packages/mercurial/localrepo.py", line 832, in commit
    new[f] = self.filecommit(f, m1, m2, linkrev, trp, changed)
  File "/usr/local/lib/python2.5/site-packages/mercurial/localrepo.py", line 712, in filecommit
    meta["copyrev"] = hex(manifest1[cp])
KeyError: 'foo.c'
```

Clearly, there is room for improvement :)

# An example for merge

Example for merge (which can't happen with update):

- You have **renamed** a file in your branch.

# An example for merge

Example for merge (which can't happen with update):

- You have **renamed** a file in your branch.
- Sally has **added** a new file in trunk at the location you have renamed your file to.

# An example for merge

Example for merge (which can't happen with update):

- You have **renamed** a file in your branch.
- Sally has **added** a new file in trunk at the location you have renamed your file to.
- You want to merge your branch back into trunk.

# An example for merge

Example for merge (which can't happen with update):

- You have **renamed** a file in your branch.
- Sally has **added** a new file in trunk at the location you have renamed your file to.
- You want to merge your branch back into trunk.
- What do you think should happen to the files during merge?

# Tree conflicts are not trivial to handle

- Large problem space.
- Some tools just ignore the problem (e.g. Subversion 1.5).
- It is likely that most tools only handle a subset of the problem space.
- Is there any tool that can handle them all?
- How can we know whether a tool covers the whole problem space?

# Tree conflicts are not trivial to define

Can we make a list of all tree conflicts?

- Well, sort of...
- Most are very obvious (like modify vs. delete)
- Some are open to interpretation, e.g. because they could be handled as text conflicts (like modify vs. replace)

# Basic thesis idea

- Define what tree conflicts are and how they can be described. (done)
- Investigate how other projects deal with them. (started)
  - FOSS: Mercurial, git, CVSNT, darcs, ...
  - Proprietary: ClearCase, AccuRev, Perforce, ...
- Identify good strategies, if any.
- Come up with better or alternative strategies if possible.
- Suggest (at a high level) how these strategies could be employed in Subversion's new working copy library "wc-ng" (which is still in design stage).

So first, we need a list of tree conflicts.

# Analysing the tree conflict problem space

We can try to make a list of tree conflicts by:

- Looking at fundamental events during update/merge.
- Checking how they interact with local changes.
- Identifying cases that are problematic at the directory-tree level...
- ... and calling those “tree conflicts”.

# Analysing the tree conflict problem space

Problems with this approach:

- Version control systems differ widely.
  - Fundamentally different concepts of revision control (e.g. centralised vs. distributed).
  - Different tools offer different feature and command sets.
- Cannot guarantee “completeness”

# Analysing the tree conflict problem space

Migitating the problems:

- Stick to most common features.
- Try to model tree conflicts in a simple and flexible way, so more cases can be added later.
- Don't try to be theoretically perfect, but good enough in practice.

# Items

**Items** are files or directories under version control.
Items can change over time.
We can identify items either indirectly by the path they're occupying at a given revision of the tree, or some other unique identifier.

- Subversion: Path + revision number
- Many distributed systems: Hash value. Path is item attribute.
- Database-backed systems such as ClearCase: Object ID in database. Path and revision are object attributes.

# Actions

**Actions** create, change, or destroy an item at a given path.
Their operands are paths.

# Actions

**Actions** create, change, or destroy an item at a given path. Their operands are paths.

- Modify
- Add
- Delete
- Copy
- Move
- Replace

# Operations

**Operations** carry out actions on multiple items.
Their operands are trees.

# Operations

**Operations** carry out actions on multiple items.
Their operands are trees.

- Update, two arguments:
  - source tree
  - target tree
- Merge, three arguments:
  - source tree at merge-left revision
  - source tree at merge-right revision
  - target tree

# Flags

**Flags** describe what happened at a path in the target tree (local changes).

- For **update**: Relative to the last time a commit was made from the target tree (working copy) to the source tree (repository).
- For **merge**: The differences between the source tree at merge-left revision and the target tree.

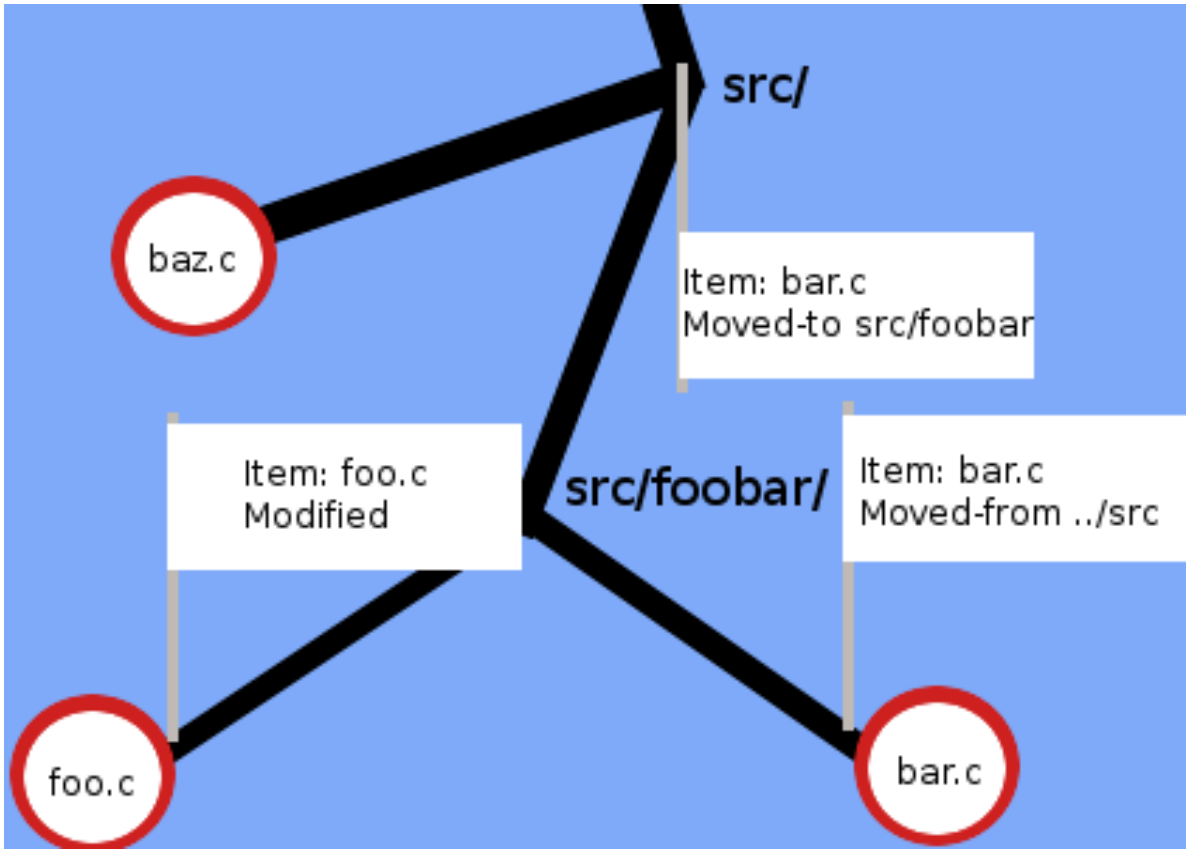
# Flags

- Multiple flags can be present at a path.
- Each flag carries information about one item only.
- No flag at a path means nothing has happened there.

# Flags

- Multiple flags can be present at a path.
- Each flag carries information about one item only.
- No flag at a path means nothing has happened there.
  - Modified
  - Added
  - Deleted
  - Copied-from
  - Copied-to
  - Moved-from
  - Moved-to
  - Replaced

# Flags (a picture)



# Applying the model

The example:

- You **modify** a file in your working copy.
- Sally **deletes** the same file in her working copy.
- Sally commits.
- You update.

Can be expressed in the model as:

# Applying the model

The example:

- You **modify** a file in your working copy.
- Sally **deletes** the same file in her working copy.
- Sally commits.
- You update.

Can be expressed in the model as:

- Operation: Update.
- Action carried out by update: Delete.
- Flags for file: Modified

# The table for update

| | | Actions | | | | | |
|---|---|---|---|---|---|---|---|
| | **Flags** | 1) Modify | 2) Add | 3) Delete | 4) Copy | 5) Move | 6) Replace |
| a) | No item at path | x | | x | x | x | x |
| b) | No flags | | i | | | | |
| c) | Modified | | x | c | | | c |
| d) | Added | x | i | | | x | x |
| e) | Deleted | c | x | | c | c | |
| f) | Copied-from | | | | | | |
| g) | Copied-to | | x | | | | i |
| h) | Moved-from | | | | | | |
| i) | Moved-to | c | x | c | c | p | c |
| j) | Replaced | c | x | c | c | c | i |

Table: Update scenarios. c: Always a tree conflict – i: Tree conflict if items differ – x: Cannot happen – p: Tree conflict if destination paths differ

See thesis paper for discussion of each cell in table.

# The table for merge

| | | Actions | | | | | |
|---|---|---|---|---|---|---|---|
| | **Flags** | 1) Modify | 2) Add | 3) Delete | 4) Copy | 5) Move | 6) Replace |
| a) | No item at path | c | | | c | c | c |
| b) | No flags | | i | | | | |
| c) | Modified | | c | c | | | c |
| d) | Added | | i | c | i | i | i |
| e) | Deleted | c | c | | c | c | c |
| f) | Copied-from | | | | | | |
| g) | Copied-to | | c | | | | |
| h) | Moved-from | | | c | | c | c |
| i) | Moved-to | c | | c | c | p | |
| j) | Replaced | | c | c | c | c | c |

Table: Merge scenarios. c: Always a tree conflict – i: Tree conflict if items differ – p: Tree conflict if destination paths differ

See thesis paper for discussion of each cell in table.

# Outlook

Problem space is relatively large.

- Tables apply to both files and directories, with a few exceptions (see thesis paper).
- Testing all 49 tree conflict cases takes a lot of time.
  Roughly 90 cases if counting files and directories separately.

# Outlook

Time for Bachelor thesis is only 12 weeks.

- No idea how many tools I will be able to investigate in the remaining time (8 weeks left).
- Might have to narrow down investigation to interesting corner cases to produce interesting results.
- If time runs out while investigating tools, don't fret. Other people can pick up the bits and continue.

# Outlook

I hope that Subversion will benefit from the results of my thesis.

- Subversion 1.5 offers merge-tracking.
- Huge improvement over 1.4.
- But it cannot deal with tree conflicts.
- So merging can still be a nightmare in some cases.
- Current Subversion tree-conflicts branch handles only 6 cases, but is already quite complex.
- It cannot easily scale to 49 cases.

# Outlook

- The tree-conflicts branch's complexity stems mostly from limitations in Subversion's current design.
- Use planned working copy rewrite as chance to reduce implementation complexity of tree conflict handling.
- Make update/merge in Subversion safe!
  Our users should not run be running into update/merge problems because of tree conflicts.

# That's it for now.

Thank you!
Questions?