



Versioning and relation management in a Zope ECM

Florent Guillaume
fg@nuxeo.com

EuroPython
27 June 2005



Abstract: versioning

Use cases

Some Zope versioning systems

- ◆ CPS 3
- ◆ *zope.app.versioncontrol*
- ◆ Occams
- ◆ CMFEditions

The DeltaV model



Abstract: relations

RDF

Relations in Zope

- ◆ Zsemantic
- ◆ Plone Relations
- ◆ Schoolbell relations



Abstract: future

Toward a versioning model for Zope 3

- ◆ proxies or working copies ?
- ◆ relations



Goals

Design a flexible solution for a Zope 3 ECM

Identify standard problems and solutions

Find what's best



Use cases: user

Make sure nobody changes my doc behind my back when I'm working on it

Allow me to work on a new version while the published one stays available

Revert to earlier version

- ◆ Advanced: diff

Provide staging of sets of documents

- ◆ Snapshots of site structure
- ◆ Publishing of whole hierarchies



Use cases: framework

Simple implementation if all ZODB

Use an existing repository system (SVN, DeltaV, JSR170)

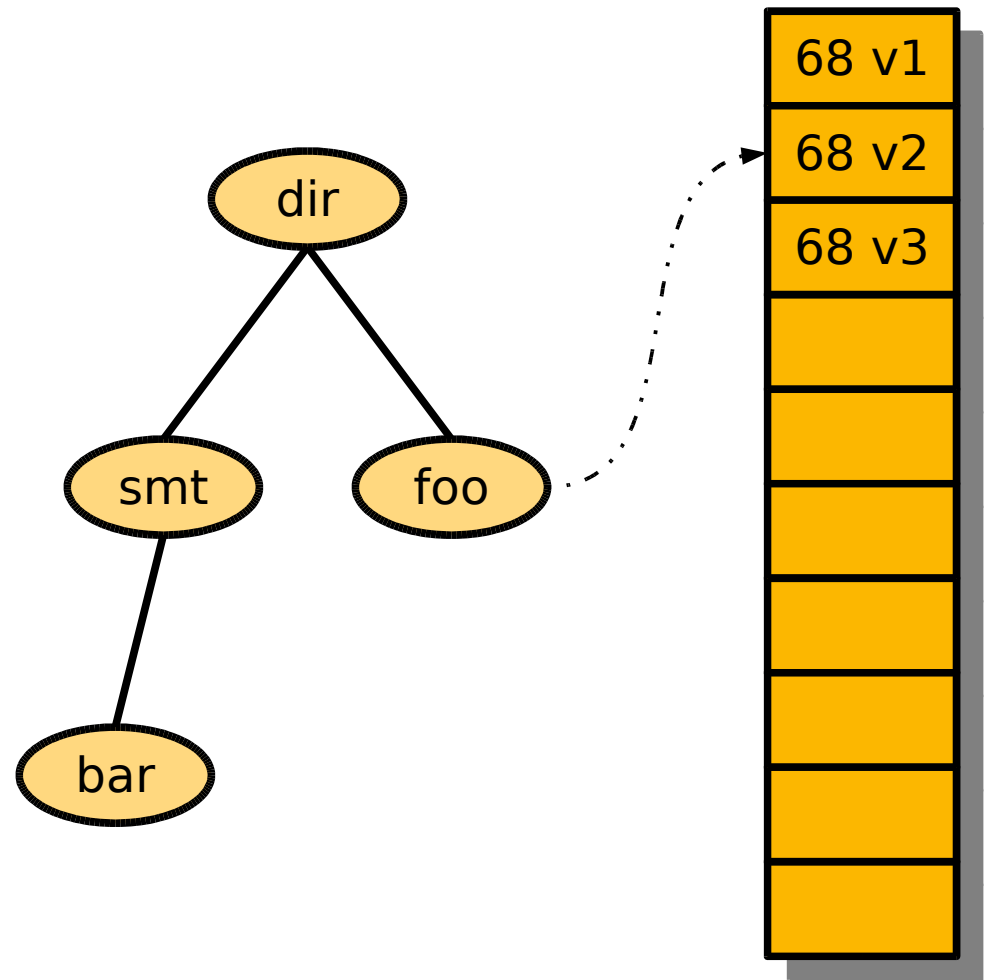
Use an external repository that doesn't know about versioning (filesystem, SQL)

CPS 3

Repository holding
all objects

Content space
holding “proxies”

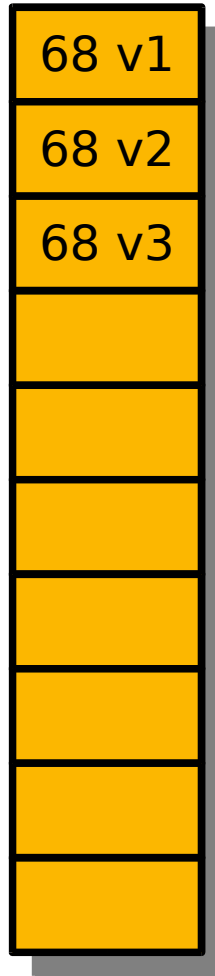
Proxies refer to
objects in the
repository



CPS 3: Repository

Stores all versions of all documents

- ◆ including those being edited
- ◆ different revisions of the same document share the same *docid* (an integer)
 - *docid* = 68
- ◆ the revision (*rev*) is a small integer
 - *revision* = 1, 2, 3



CPS 3: Proxies

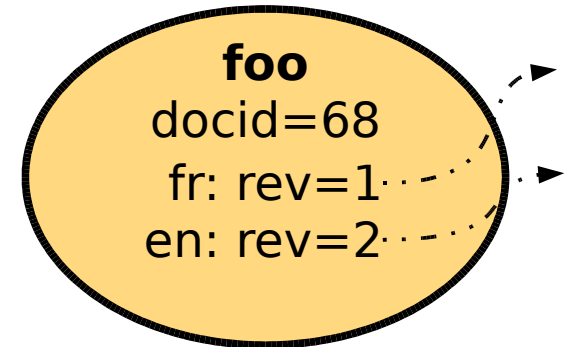
Proxies contain information about

- ◆ the docid referenced
- ◆ one or several revisions

Proxies are multilingual, each language has its revision

Proxies are explicitly dereferenced by skin methods

- ◆ getContent()
- ◆ getEditableContent()





CPS 3: Creating new revisions

All content is stored in the repository

Some revisions are *frozen*, immutable

- ◆ `getEditableContent()` returns a new non-frozen revision
- ◆ provides lazy copying on edit
- ◆ usually done as soon as submitted or published

Non-*frozen* revisions correspond to those being worked on (working copy)

Freezing decided by workflow transitions





CPS 3: Proxy Tool

Holds information about which proxy objects point to which repository objects

Is a form of relation tool

Provides fast indexes of these relations



zope.app.versioncontrol (1)

The following also applies mostly to
Zope 2's ZopeVersionControl

Objects in content-space are checkouts
(copies) of some object of the repository

Repository

- ◆ storage policy
- ◆ versioning operations
- ◆ objects boundaries

Objects manipulated have versioned and
non-versioned data





zope.app.versioncontrol (2)

CVS-like operations (working copy)

- ◆ put under version control
- ◆ checkin
- ◆ checkout, update
- ◆ uncheckout (revert)
- ◆ get log entries, version info
- ◆ add labels
- ◆ create branches





Occams

Central repository of objects

Copy on write, auto versioning

Stagemaps: snapshots of the versions of a full tree

Gateways: objects loaded from repository according to stagemap during traversal
⇒ virtualized hierarchy

Interceptors: know the boundaries of objects, and adapt protocols





CMFEditions

Repository

- ◆ main access point for versioning, policy

Archivist

- ◆ does object copies to/from storage

Modifiers

- ◆ use by archivist
- ◆ know object boundaries

History storage

- ◆ physical storage



DeltaV

Extension that provides the V (versioning) of WebDAV

Part of it is used as the underlying model of Subversion

Richest of all models

- ◆ has versioning of full hierarchies
- ◆ clean concepts
- ◆ separation of concerns
- ◆ vocabulary is hard to understand



DeltaV: Resources

Resource

- ◆ object, document
- ◆ lives in content space

Version

- ◆ one version of a versionable resource
- ◆ lives in history space

Version history

- ◆ the set of versions for a resource
- ◆ tree of successors, predecessors

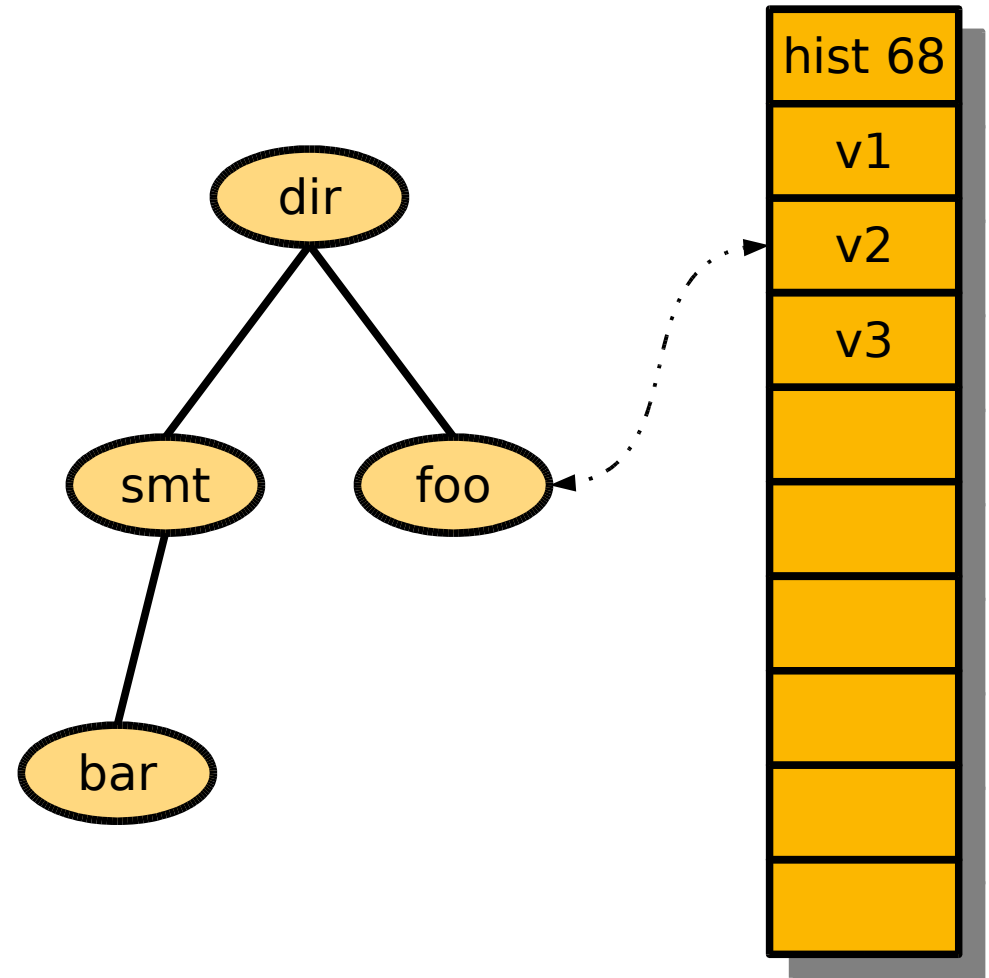


DeltaV: Resources

The resource *foo* is part of history 68 and is either

- ◆ checked-in from v2
 - not modified
- ◆ a checkout of v2
 - may have been modified

A resource can also be non-versioned





DeltaV: Collections

(Version-Controlled) Collection

- ◆ folder
- ◆ holds names of “internal members”
- ◆ lives in content space

Collection version

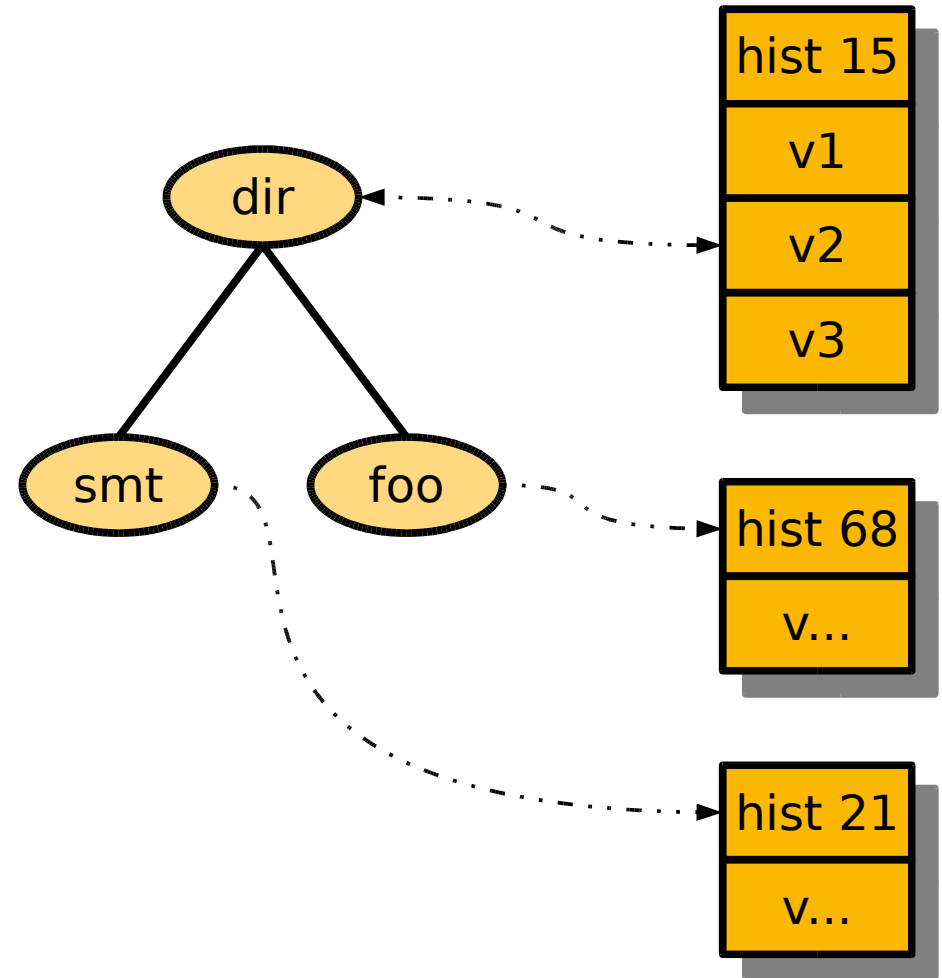
- ◆ one version of a versionable collection
- ◆ holds names and version histories of internal members, but not the specific version of each
- ◆ lives in history space, along with other versions



DeltaV: Collections

The collection *dir* has two internal members, *smt* and *foo*

The versioned collection doesn't know about the members' versions





DeltaV: Configurations

(Version-Controlled) Configuration

- ◆ a collection and the recursive resources in it
- ◆ lives in configuration space

Baseline

- ◆ a configuration and the set of version of each resource in it
- ◆ lives in version configuration space

Baseline History

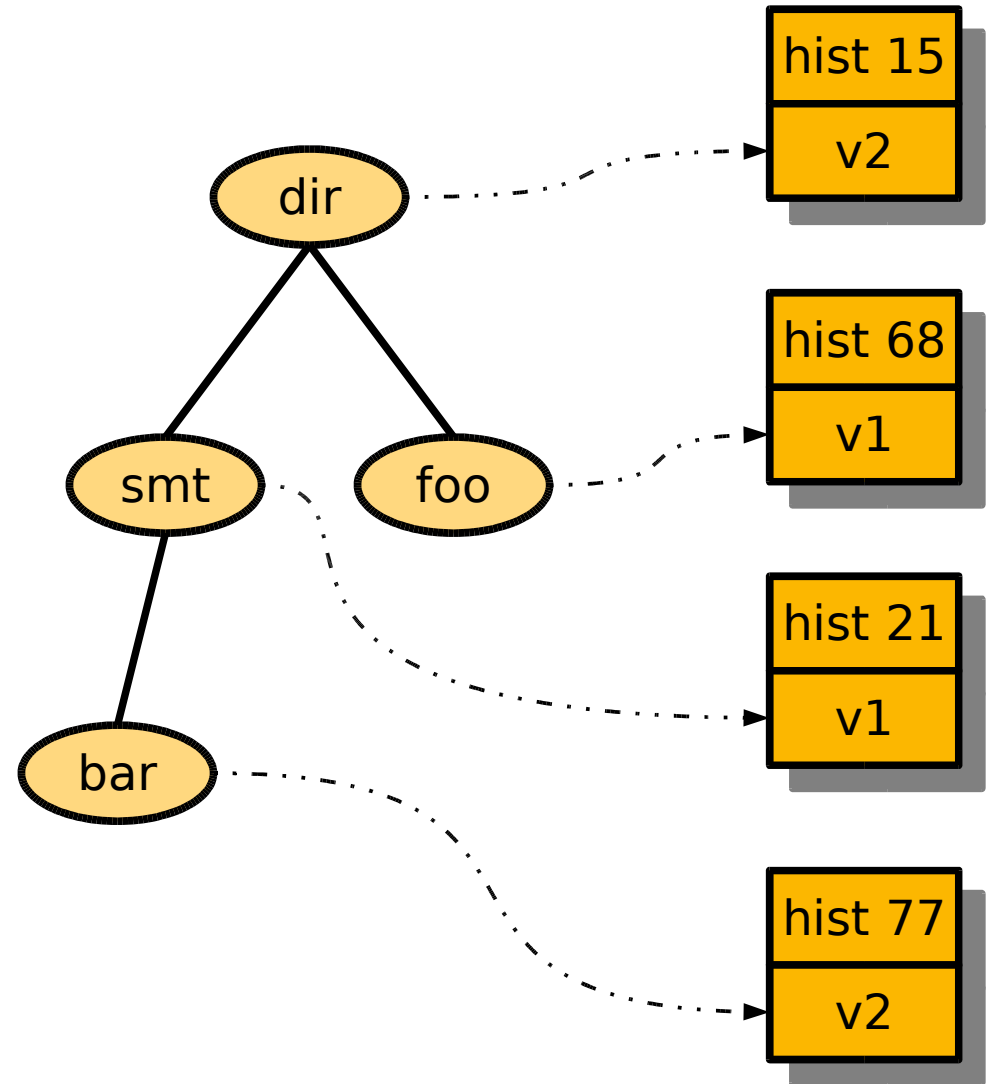
- ◆ a set of baselines



DeltaV: Baselines

Baselines provide snapshots of entire sets of versions

- ◆ versioning of hierarchies
- ◆ staging



Relations

In the context of RDF, triples

- ◆ Subject, Predicate, Object

Subjects, Objects

- ◆ documents (unique id)
- ◆ literals (text, date, integer...)

Predicates

- ◆ Title, hasParent, isInVersion

Non-binary relations more difficult



Zemantic

Zope 3 interface to RDFLib

- ◆ RDFLib has different backends for its graphs
 - memory, SQL, ...
- ◆ Zemantic adds Zope-specific backends
 - ZODB BTrees

Can import and export in RDF format (XML)

Provides storing and indexing of arbitrary relations

Includes a query language (SPARQL)

Evolves quickly





Plone relations

Manages Archetypes references

Relations between documents or document subcomponents (images, files)

Rules and rule sets

- ◆ constraints on relation creation or destruction (one-to-many, object types, object choice)
- ◆ side effects of relation changes (cascade)





Schooltool Relationship

Model comparable to RDF

Relations between annotatable components

Definition of relations through URIs

Simple queries (s, p, *), (*, p, o)

Facilities to have relation-aware components

- ◆ attribute access does a query
- ◆ ob.children ⇒ list of childrens

Also tied to the event system





Zope 3 ECM design

Explicitly use relations as much as possible

Version storage has to be flexible

- ◆ various physical storages
- ◆ various versioning policies

Separate storage from policy

Separate relations from storage





Proxies or working copies ?

Working copies

- ◆ Normal objects in content space
- ◆ But lots of content copy, duplication

Proxies

- ◆ Provide much flexibility
- ◆ Policy when dereferencing
- ◆ But are sometimes too magic
 - Hard to debug
- ◆ Slower because object boundaries are crossed for every use





Relations for documents

Standard relations

- ◆ document relations explicitly created
- ◆ inclusion (images, files, subdocuments)
- ◆ linked to, related to

Version-related relations

- ◆ part of same histories
- ◆ history tree
- ◆ baselines



Subjects/objects, predicates (1)

Document, collection (d)

- ◆ d fromVersion v

Version (v)

- ◆ v inHistory h
- ◆ v hasChildInHistory h
- ◆ v predecessorOf/successorOf v

History (h)



Subjects/objects, predicates (2)

Configuration (c)

- ◆ c hasRootCollection v
- ◆ c hasVersion v
- ◆ c fromBaseline b

Baseline (b)

- ◆ b inBaselineHistory bh
- ◆ b predecessorOf/successorOf b

Baseline history (bh)





Optimizations

These were fundamental relations

- ◆ others can be derived, to speed things up
- ◆ precompute chains of relations

The relation tool can choose to store some relations directly with the objects

- ◆ annotations

Key point: fast queries

- ◆ depends on use cases
- ◆ depends on the backend





Policies

Resource boundaries

- ◆ when is something another resource ?

Configuration boundaries

- ◆ where to stop when defining configurations ?



Thank you! Questions?

Florent Guillaume, *Nuxeo*

- ◆ fg@nuxeo.com
- ◆ <http://svn.nuxeo.org>

