# Versioning at Last!

## An overview of CMFEditions and what it provides for Plone 3.0

by
Alec Mitchell
(alecm -- apm13@columbia.edu)
with material by
Gregoire Weber

# Talk Overview

- Part 1
  - Versioning and CMFEditions overview
- Part 2
  - Some technical details
- Part 3
  - Demo!

# Versioning, what is it?

- Versioning means different things to different people
  - Viewable history of content revisions
  - Reverting to older revisions (reliable undo)
  - Check-out – edit – check-in
    - This is "Staging" and CMFEditions doesn't do it (sorry)

# CMFEditions Overview

- An extensible set of CMF tools, templates, and scripts which provide versioning for Plone

  - Supports stored revisions of the stock CMF and Archetypes based types for Plone, and custom content types (really any python objects)

  - Efficiently saves revisions of folderish types, including the state of child objects if desired

  - Modular (though not yet "componentized"™)

# Using CMFEditions

- Versioning use cases are common and varied

- CMFEditions provides the basics

  - Save the current state of an object

  - Show revision history

  - Revert to old versions of ZODB content

  - Have a look at an old version (without reverting)

  - Look at the differences between revisions

  - Revert folders without reverting the objects inside the folder (e.g., only the properties of the folder)

  - Revert folders while also reverting any subobjects (useful for RichDocument, Smart Folder, ...)

# Project Goals

- It can't cover every use case

- However we can make the product modular and extensible

  - Pluggable modifiers allow customization of the stored object during storage and retrieval

  - Configurable (per type) versioning policies

  - Replaceable version storage back-end (currently uses ZopeVersionControl and stores revisions in the ZODB)

  - More can be done to make it even more modular and extensible using the Zope 3 Component Architecture

# Part 2: The Details

- What gets stored with a version and why
- Customizing what gets stored and retrieved
- Customizing when we store revisions
- A little about the architecture

# What We Save

- All python attributes are saved (the entire __dict__) with a deep-copy (mostly)

- Save modifiers can alter, remove or add object attributes before storage

  - Used to explicitly store workflow state on revisions

  - Also can be used for use case specific needs, but generally it's best to store everything and worry about use cases on retrieval.

# What We Save Part 2

- Saving children in a folder (ObjectManager) would be very very bad! We might save the whole site as a single revision.

- Instead we replace children with references to particular versions of those children.

  - How? Using a modifier.

- Some folderish objects may want to enforce saving revisions of children.

  - For this we have a specialized modifier.

# What We Retrieve

- Version retrieval can be very use case dependent

- Certain things probably should not be retrieved with the saved object.  In particular changing the following on revert could cause confusion

  - The object id

  - Workflow state, including effective and expiry info

  - Permissions, local roles

  - Child objects (usually)

# Modifiers

- Why use a modifier?

  - e.g., workflow state may be considered intrinsic

  - You may want to avoid retrieving some attributes (e.g. ratings or user tags stored in annotations)

- Modifiers are classes implementing an interface

  - ISaveRetrieveModifier: beforeSaveModifier, afterRetrieveModifier

- Registered with a tool and, usually, use TAL expressions to determine when they apply.

# Versioning Policies

- Default: versions need to be saved manually

- Policies may be registered for specific types to allow alternative means of saving versions

- Included policies
  - Version on every save (for AT objects only)
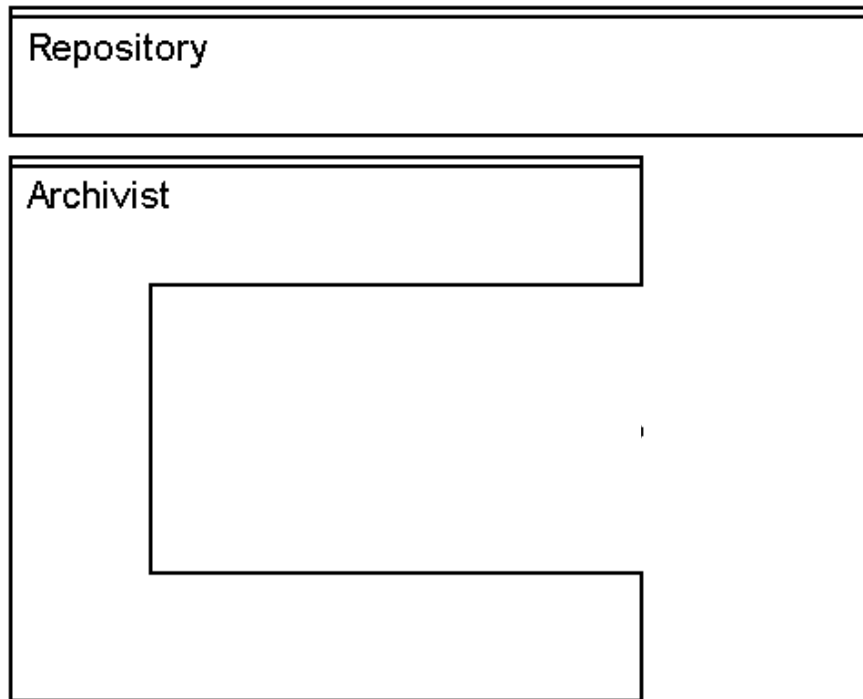  - Save a new version after reverting
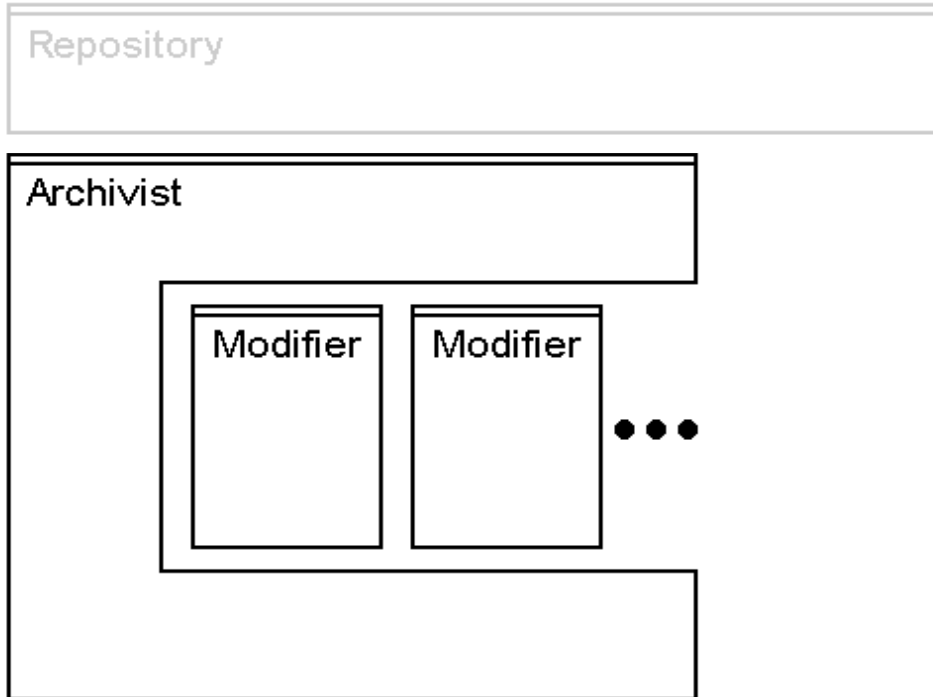
# Architecture

Repository

- **Content repository**
  - Provides the public API
  - save, revert, retrieve, restore, getHistory, isUpToDate, ...
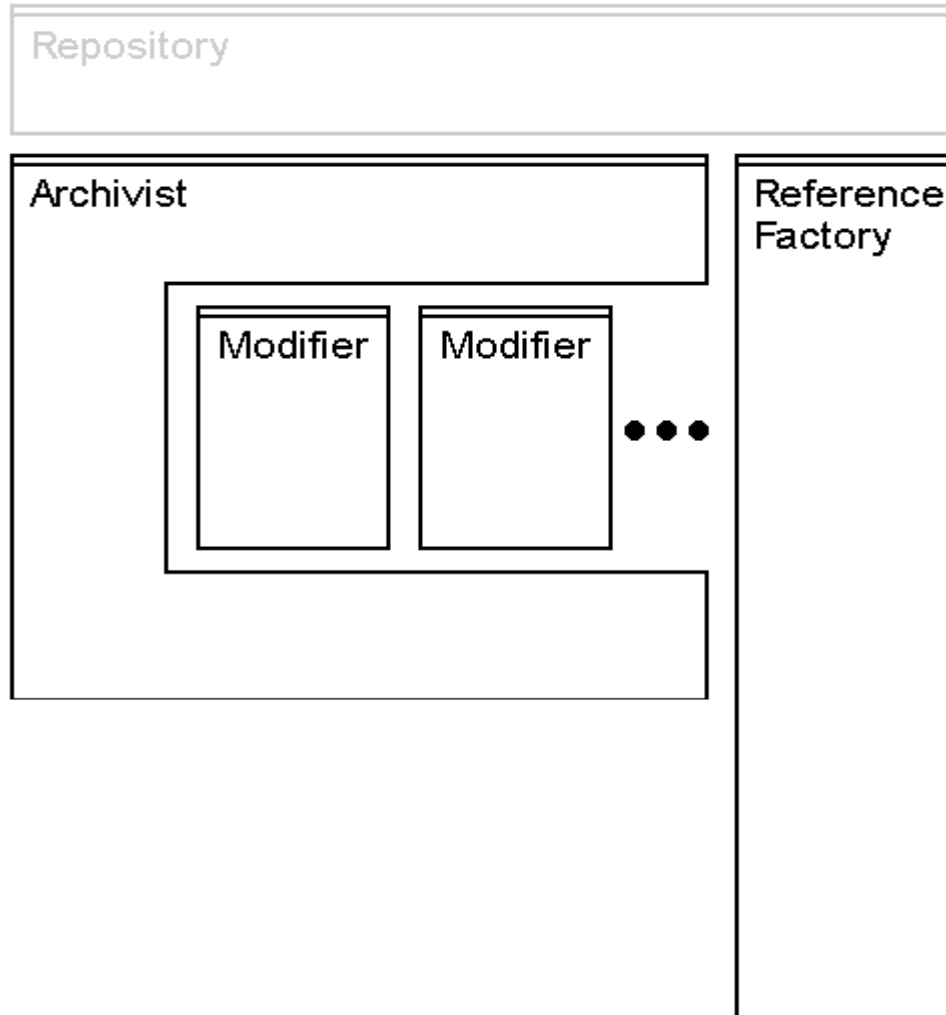
# Architecture

Repository

Archivist

- Archivist
  - Understands how to copy python objects
  - This is where the scary stuff is
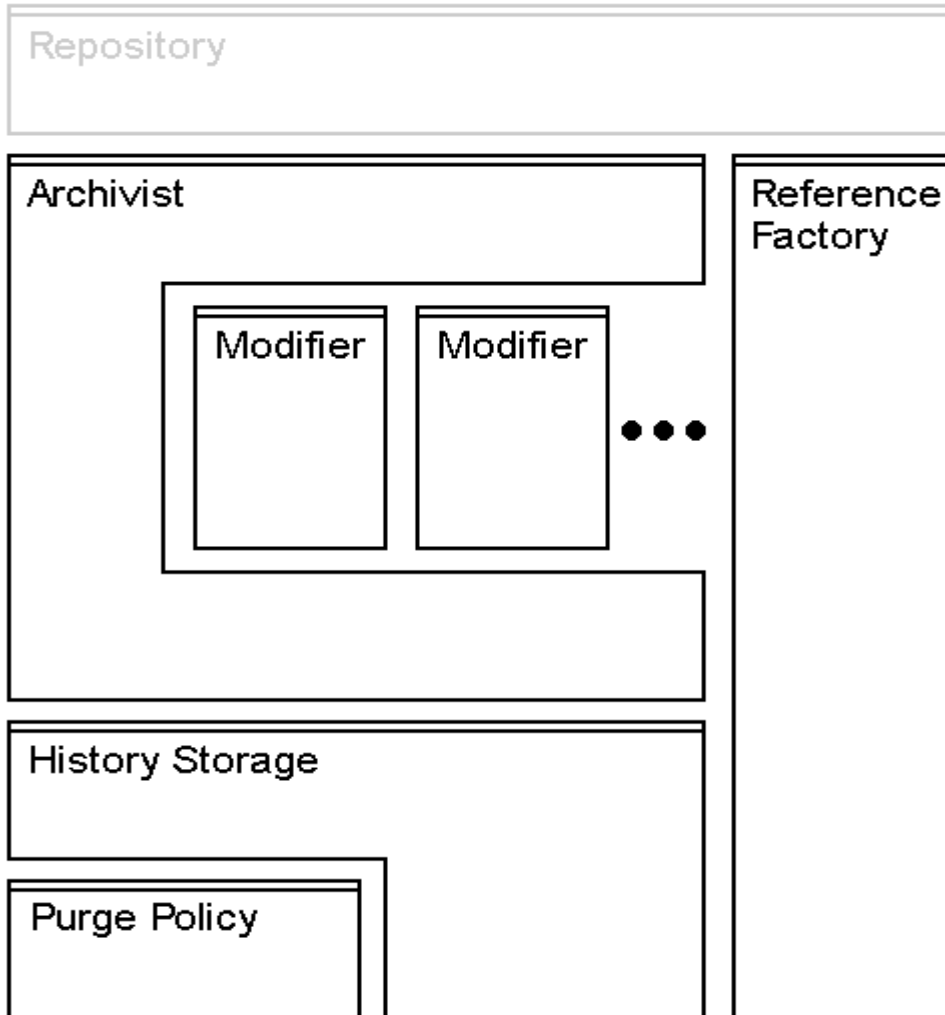
# Architecture



- **Modifiers**
  - Applied before archival or after retrieval
  - We'll see these in action shortly

# Architecture



- ## Reference factory

  - Knows how to construct new instances of missing objects

# Architecture



- Storage
  - Simple persistence of the Archivist output
  - Currently ZVC, could easily use something simpler
  - Even something outside the ZODB
  - Includes purging of undesirable history

# Demo!!

Let's see it in action (under Plone 2.5)

# Credits

## Contributors

- Alberto Berti (azazel)
- Alec Mitchell (alecm)
- Brent Hendrix (brentmh)
- Duncan Booth (duncan)
- Francesco Ciriaci (ilbestio)
- Gregoire Weber (gregweb)
- Riccardo Lemmi (rlemmi)
- Rob Miller (rafrombrc)
- Sune Broendum Woeller (sunew)
- Sylvain Thenault (syt)
- Tomek Meka (tomek)
- Varun Rastogi (varun)
- Vincenzo Di Somma (vds)

### Translations

- Danish: Anton Stonor
- French: Godefroid Chapelle (godchap)
- German: Gregoire Weber (gregweb)
- Polish: Piotr Furman
(by svn blame and cvs anno statistics)

## Sponsors

- Oxfam GB: www.oxfam.org.uk

- ZEA Partners: www.zeapartners.org

- Musee de l'Afrique Centrale:
    www.africamuseum.be

- RedCOR: www.redcor.ch

- Zehnder Group: www.zehndergroup.com

- The Open Planning Project:
    www.openplans.org

- Reflab: www.reflab.com

- Metapensiero: www.metapensiero.it

- Incept: www.incept.ch

# Questions?

- Ask now

- Mailing list: **collective-versioning@lists.sourceforge.net**

- Product Page: **http://plone.org/products/cmfeditions**

- IRC: **#cmfeditions**