GT2003GianugoMatthewDavid

Cocoon, Webdav and more...

with Matthew Langham, Gianugo Rebellino and David Nuescheler (from Day Software - the company Roy Fielding is working for)

Raw notes taken collaboratively during the talk by Stefano Mazzocchi, Bertrand Delacrétaz, Rogier Peters, Steven Noels, Jeremy Quinn.

Matthew starts.

WebDAV history

How did WebDAV start?

- Tim Berners-Lee meant www to be a writeable medium.
- NCSA Mosaic pushed publish/browse to the foreground, writing to the background
- Navigator Gold had an editor

WebDAV mission

- The World is a Folder
- Edit and save metadata

Multiple people can work together, on different platforms Simple, extendible, uses HTTP infrastructure

Can be based on Databases, File System, CVS, etc. Many tools support WebDAV. Moving from a WebDAV server to another is fairly easy.

WebDAV functionality

- Basic functions
  - Locking, metadata, namespace
- DeltaV
  - auto-versioning, checkout/checkin
- ACL
  - Access control
- DASL
  - Searching

Diagram: HTTP at the base, WebDAV on top, DeltaV, ACL and DASL on top of WebDAV.

WebDAV servers

- Tamino
- Apache 2.0
- Catacomb
- Slide
- Others damn he's fast

WebDAV clients

- XmlSpy
- Xmetal
- Microsoft
  - Office
  - Explorer
- Adobe
  - Photoshop
  - GoLive
- MacOSX Finder

Can access WebDAV servers as file folders, file browser integration.

Cocoon and Webdav (Gianugo kicks in)

(yet another) dynamic duo? Cocoon and WebDAV!

WebDAV is an HTTP protocol with XML payload. Cocoon is a perfect candidate for a WebDAV interoperation.

Technical aspects

Good points

- WebDAV is a networked file system with strong point on metadata
- Extensible metadata exposed as Namespaced XML
- Revamped http + xml protocol, WebDAV is just an extension, you can still use HTTP only.
- Your WebDAV server is still an HTTP server
- Rich semantics for metadata. Until now we stored documents in RDBMs with metadata
- Easy to use for easy tasks
- Ubiquitous and cross platform (vendor independent)

Weak points

- http is not used as a pure transport layer.
- No clear encapsulation.
  - Some information is in payload, some information is in request headers.
- no real support for "real" XML metadata
  - it's an implementation issue
  - if you put an XML snippet as a metadata, it's just stored as a blob
  - you can't do xpath queries on it
- difficult to use for difficult task
  - Cocoon might shield some of this complexity, but might not go too far and you have to do the rest
  - Like moving directories around
- specs are somehow unclear, too many extensions in draft phase (DeltaV, DASL, DAV ACT).

So basically WebDAV is good for simple stuff but can get in the way for more complex requirements, specially if you need the advanced features (versioning, search, access control).

What can we do with WebDAV in Cocoon

- Support in the block
- you can use it as a client (stable, used in production with no issues)
- you can make a WebDAV server with Cocoon (needs work), this is probably the most promising way.
  - client is stable
  - server still needs work
- Proxy for a back-end WebDAV server, forward requests with optional processing

Let's go through a propfind request

A propfind on a collection returns an XML multistatus response with properties for every document in the collection - can be processed by an XSLT transform easily.

WebDAV as a client

- WebDAV Source:
  - Avalon source implementing all subinterfaces (Traversable, Modifiable, Inspectable)
  - You can store a bunch of stuff (stylesheet, sitemap, xsl, xml, assets) inside your WebDAV server.

SourcePropsWritingTransformer: enables write access to resource properties.

DASLTransformer: performs DASL queries (sort of a SQLTransformer for a WebDAV DASL-enabled repository)

WebDAV as a server

If we want to build a server with WebDAV we don't need any particular component (well, almost... because generating all the specific WebDAV stuff is not so easy, so having special component might be helpful).

The WebDAV block contains a dir2propfind.xsl to ease property handling. It converts a DirectoryGenerator output into a PROPFIND response.

We need user feedback to understand what you want to do with a Cocoon enabled WebDAV server.

WebDAV as a proxy

- Why use a proxy?
- usecase is to implement missing features on backend WebDAV server (for example DASL)
- Problem is, proxying is done at the Servlet API level, inefficient, lots of cloning of objects.
- Add virtual resources (a PDF view or resized images)
- Send email or events when something happens (storage, attribute change etc)
- Provide easy and effective authentication
- Mangle properties using simple Explorer-like file managers (a simple way to connect and edit the repository and its properties ... no special needs for clients)
  - Dragging a file on the client could change a property of the WebDAV file instead of actually moving it (IIUC?)

WebDAV architectures that could be used

Traditional

- dav server
- admin app
- web interface to the admin app
- delivery app -> client

Problem: no actual use of WebDAV protocol if WebDAV is used only internally, doesn't bring any benefits?

Gianugo's proposal

- Put the dav server in the Cocoon core
- All the different clients are accessing the webdav server directly
- Then you build the delivery app
- Problem: do you really want your client to connect directly to your DAV server
- Solution: cocoon between WebDAV server and client
- Use WebDAV, DBs, LDAP as backend, accessed over WebDAV

Using cocoon as a virtual WebDAV server brings many benefits: aggregation, "interception" of WebDAV operations, etc.

Dream list

- having much easier access to the request body
- Is a different Environment enough? Need new sitemap semantics?
- WebDAV proxying needs transation supports (for example, in order to implement DASL) . Failure needs to be reported, because inconsistencies might pop in.

Transactions is a hard problem to solve - hard for Cocoon to find out exaclty what happened on the back-end and manage failures correctly.

Catacomb is an apache httpd (mod_dav) extension which does DASL but can only store data in MySQL.

Pro-netics (Ricardo Rocha) is working to enhance Catacomb, supporting ODBC as the backend.

JSR 170

http://www.jcp.org/jsr/detail/170.jsp

The specification lead of JSR 170, working at Day

Use case: the MyBank website. The bank uses a CMS for the website, another CMS for their intranet and they have to duplicate a bunch of stuff.

The problem with proprietary CMSes is that your stuff gets shielded behind one out of 200+ APIs related to CMS.

David shows some code snippets of different API that are used to connect to the various proprietary repositories.

Aim: One API for all content repositories - architecture, data source and protocol neutral.

Content is Data ++
Content is managed Data/Information

JSR has two levels of compliancy. This is similar to yesterday's talk, see GT2003HackathonJsr170

Level 1 is simple to implement for vendors. Scope is:

- Access
- Read/Write
- Support hierarchy operations (copy/move/link)
- XML serialization
- Search
- Content typing

Level 2 (extensive functionality)

- Versioning
- Locking
- Observation
- Transations
- Access control

Lots of overlap between webdav and JSR 170, a good parallel is HTTP and Servlet API: one is the protocol, another thing is the API.

He shows one of Gianugo's slides and adds JSR 170 (also based on yesterday's talk about how to build a CMS on top of cocoon)

```
backend <--> repository <--> frontend
               ^
               |
               V
             sideend
```

Current Situation

- going thru "community review"
- the open source reference implamentation is inside Jakarta Slide (inside the CVS as a proposal)
- the current RI is webdav-enabled already

final thoughts

- webdav is an excellent transport layer for JSR 170 -> RI
- JSR 170 can be used as a generic client API to talk to a granular (property) level to a WEbdav server (and friends)