WebDAV:
Distributed Authoring and Versioning

Greg Stein

http://www.lyra.org/

gstein@lyra.org

Adobe Technical Seminar Series, May, 1999

Agenda

- Overview
- Technical Discussion
- Closing
- Q&A

WebDAV Overview

Adobe Technical Seminar Series, May, 1999

What is WebDAV?
(1 of 4)

Web-based Distributed Authoring and Versioning

- "DAV" is the usual short form

Goal: enable interoperability of tools for distributed web authoring

What is WebDAV?
(2 of 4)

Turns the Web into a *writeable* medium

Applies to all kinds of content - not just HTML and images

Based on extensions to HTTP

What is WebDAV?
(3 of 4)

Properties ("metadata")
- Documents can have associated metadata
- Browsing/searching metadata

Overwrite protection

Namespace management

More in progress (described later)

What is WebDAV?
(4 of 4)

Receives benefits of HTTP infrastructure

- Strong authentication
- Encryption
- Proxy/firewall navigation
- Worldwide deployment
- Huge talent pool; numerous tools, apps, etc

Provides more infrastructure (continued)

DAV Infrastructure

Overview

DAV can provide infrastructure for:

– Collaboration

– Metadata

– Namespace management

– Ordered collections

– Versioning

– Access control

– Searching

DAV Infrastructure
Collaboration

Provides:

- Whole-resource locking enables remote, collaborative authoring of any media type (HTML, images, presentations, etc)

Infrastructure for development of remote, collaborative authoring and editing tools

DAV Infrastructure

Metadata

Provides:

- Property (name/value) pairs can be created, modified, deleted on any Web resource
- Values can be managed by server or client
- Values are well-formed XML

Infrastructure for recording information *about* Web content

DAV Infrastructure
Namespace Management

Provides:

– Copy, move, add, delete individual resources and collections/hierarchies of resources
– Manage ordering of collections’ resources
– Create references (links) on server

Infrastructure for remote management of the organization and viewing of resources

DAV Infrastructure

Versioning

Provides key features of WebDAV:

- Check in/out with comments and metadata
- Version graphs and histories
- Browse/retrieve old versions
- Automatic versioning for “down-level” clients
- High-level configuration management

Infrastructure for versioned resources

DAV Infrastructure

Access Control

Provides:

- Remotely manage *who* can read/write *which* resources

Infrastructure for remote management of groups of collaborators

DAV Infrastructure
Searching

Provides:

- Search for property existence or a value
- Search for a substring in a resource body
- Scoping of searches
- Extensions for more powerful searching

Infrastructure for remote searching

DAV Infrastructure

Summary

DAV can provide a “substrate” for building complex applications, tools, and systems

A Few Scenarios

Collaborative authoring

Document management system

Network file system

Unified repository access protocol

Remote software engineering

History

(1 of 2)

1996: Jim Whitehead (UC Irvine) looked at using the Web for software development

Jim, Dan Connolly (W3C), Larry Masinter (Xerox), and others began discussions on remote authoring

- Jim tossed versioning in the ring

By late ‘96, Microsoft, Novell, Netscape were involved

History
(2 of 2)

- 1997: continued spec development
  - – Effort is redefined as a “core” plus extensions
- Nov 1998: spec accepted by IETF
- Feb 1999: RFC 2518 is issued
- 1999: continuing development on DAV extension specifications

Present State

RFC 2518 defines “core” features: properties, namespace management, lockin

Notably: no versioning... “WebDA”? :-)

Future Directions

This year:

- Advanced Collections (mid 1999)
  - Ordering, references

Next year:

- Versioning
  - Workspaces, Configuration Management, etc.
- Access Control (ACLs)
- DAV Searching and Locating (DASL)

Tools, Servers, Apps
(1 of 2)

Commercial products

- Microsoft: IIS, IE5, Office 2000
- CyberTeams WebSite Director
- Glyphica PortalWare
- Xerox Docushare
- DataChannel RIO

Tools, Servers, Apps
(2 of 2)

Open-source efforts

- mod_dav
- sitecopy
- Zope
- Client APIs for Python and Java

How Will it be Used?
(1 of 2)

Not restricted to the Internet

LAN environments
- Departmental workgroups
- Software development teams

WAN/VPN environments
- Remote workgroups, development

How Will it be Used?

(2 of 2)

Mostly for authoring tools

Base protocol for client/server interactions

DAV manipulates data but does not provide an RPC mechanism

Provides a data model

Taking Advantage of DAV

Use DAV as the "wire protocol"

Tools that layer onto DAV can operate against any DAV-enabled server

- Great flexibility, customer choice
- Mix/match to build tuned systems

Leverage! (clients, servers, talent, ...)

Technical Discussion

Coverage

- We’ll concentrate on the “core” of DAV:
  - Properties
  - Namespace management
  - Locking
- Skipping Advanced Collections, DASL, Versioning, ACLs
  - Time limitations
  - Hard to do more since these are in flux

Assumptions

- Some familiarity with URIs/URLs, HTTP, XML, and XML Namespaces
  - If not, then don’t worry... there’s enough context to float by on
- Links to these specs are available via the webdav.org site

Properties
(1 of 5)

Properties are name/value pairs

All resources have properties

“Live” properties are known by the server

– May be read-only

– May be validated (structure, value ranges, etc)

“ Dead” properties are client-defined

Properties
(2 of 5)

Property names are URIs

Benefits:

- By definition, they are *unique*
- Domain name owners can quickly deploy properties using a URL and the domain name
- Stable, long-term properties can use IANA-registered URI schemes

Properties
(3 of 5)

Values are well-formed XML fragments

Benefits:

- Extensible, structured definitions of values
- Internationalization support
- Lots of tools, talent, knowledge (leverage!)
- Availability of XML-based, value-added systems (i.e. RDF, DOM)

Properties
(4 of 5)

- Property name defines syntax and semantics
- One instance of a property per resource
  - The property may be multi-valued, however
- Client-defined (dead) properties
  - Allows for properties unknown to the server
  - Server cannot help with consistency, though
- Live/dead properties provides flexibility

Properties
(5 of 5)

What to do with them?

Record metadata:

- Author
- Abstract
- References
- Timestamps
- Use your imagination!

Namespace Management
(1 of 2)

- A server’s URL hierarchy defines a “namespace”
- Collections of resources
  – Collections are resources, too!
  – “Collections” and “resources” can be seen as a fancy way to say “directories” and “files”

Namespace Management
(2 of 2)

“ Namespace management” is about managing your server’s URL hierarchy

Creating new collections

Moving and copying resources

Deleting resources

Digression...

"Collections", "resources", and "namespaces" … what the heck?

Not my fault :-)

Fancy terms are used because the server might not be using directories and files!

- Databases
- Document management systems
- Lots of repositories out there...

Locking
(1 of 9)

Write locks only, in core spec

Shared and exclusive are defined

- Shared, in this sense, means among a group

Affects modification of the resource (body and properties)

- Server can still change live properties, though

Locking
(2 of 9)

Methods affected:
- PUT, PROPPATCH, MOVE, COPY, DELETE, MKCOL
- Also affects LOCK, UNLOCK, POST

Locks apply to a whole resource
- Cannot be applied to portions of a resource

Locks can be acquired on non-existent resources (name reservation)

Locking
(3 of 9)

Locks have a *depth*

- "0" means the lock applies to just the resource
- "infinity" means the lock applies to the resource and all members (recursively)

A 0-depth lock on a collection will prevent the addition of members

Infinite-depth locks are all-or-none (any internal locks must be compatible)

Locking
(4 of 9)

Infinite-depth locks provide:

- All members in the hierarchy are locked
- Removing a resource from the hierarchy removes the (implied) lock on it

Moving/copying a hierarchy never moves/copies the lock to the destination

- Destination may already be locked, however

Locking
(5 of 9)

- Exclusive locks can be too rigid
  - People forget to release the lock
  - May require an administrator to release it
- Shared locks allow for out-of-band access negotiation
  - Won’t hold up the group

Locking
(6 of 9)

- Locks are identified by a unique *lock token*
  - Token is issued when the lock is requested
- Lock tokens are discoverable
- Each principal acquiring a shared lock will receive a new, unique lock token

Locking
(7 of 9)

Locks have two owners
- – Identified by the Authorization: header
- – Human-usable identification

Client must submit the lock token *and* the appropriate authorization

The identification is simply recorded by the server and made available to clients

Locking

(8 of 9)

Locks also have timeouts

Client may request a specific timeout

Server is free to supply its own timeout

Clients must assume locks can disappear at any time

Locks may be *refreshed* to reset the timeout

Locking
(9 of 9)

Locking is optional

A server may support any mix of shared or exclusive locks across its namespace

– Portions of the namespace may correspond to different repositories with different capabilities

Implementation

Great, you described the core features in more detail... now what?

The features are implemented as *extensions* to HTTP/1.1

Detailed DAV feature review after a sidetrack to HTTP extensions and XML

Extending HTTP

Don’t “tunnel” using POST

HTTP/1.1 is *designed* to allow new method

When semantics are visible (the HTTP method), then systems can be smarter

- (caching) proxies, firewalls
- authorization systems on the server

Use new HTTP headers when appropriate

Header or Body?

Parameters for a request can appear in the headers or the body. Which is “right?”

DAV Working Group said:

- constrained-length values go in the headers
- complex structures and binary/arbitrary data goes into the body

New HTTP Methods

PROPPATCH, PROPFIND

COPY, MOVE

MKCOL

LOCK, UNLOCK

Updated semantics for HTTP/1.1 methods:

- GET, PUT, DELETE, OPTIONS

New HTTP Headers

DAV:

If:

Depth:

Overwrite:

Destination:

Lock-Token:

Timeout:

Status-URI:

Request Bodies
(1 of 2)

All parameter bodies are XML (params, not content such as that of a GET or PUT!)

Content-type should be "`text/xml`" or "`application/xml`"

Should specify a character set:

- `Content-Type: text/xml; charset="utf-8"`

Request Bodies
(2 of 2)

- DAV requires the use of XML Namespaces
  - DAV elements
  - Clients’ property names and values
- Recall that namespaces are defined as a UR
- DAV elements use “`DAV:`”
  - Example: `<D:propfind xmlns:D=“DAV:”>`
- Most (deployed) namespaces will use an HTTP URL rather than custom URIs

Reponse Bodies

DAV uses many existing HTTP response codes; bodies are already defined

207 (Multi-Status) defines its response body to be XML

- Used to provide info for one or more resources

Quick Sample

```
ROPFIND /sample.html HTTP/1.1
ost: www.example.com
ontent-Type: text/xml; charset="utf-8"

?xml version="1.0" encoding="utf-8" ?>
D:propfind xmlns:D="DAV:">
<D:prop>
  <D:getcontenttype/>
</D:prop>
/D:propfind>
```

```
HTTP/1.1 207 Multi-Status
Server: Apache/1.3.4 (Unix) DAV/0.9.6
Date: Tue, 09 Feb 1999 00:52:55 GMT
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
<D:response>
  <D:href>/sample.html</D:href>
  <D:propstat>
    <D:status>HTTP/1.1 200 OK</D:status>
    <D:prop>
      <D:getcontenttype>text/html</D:getcontenttype>
    </D:prop>
  </D:propstat>
</D:response>
</D:multistatus>
```

Depth: Header

Three values:

- "0" refers to only the resource specified by the Request-URI
- "1" refers to the resource and its members if the resource is a collection
- "infinity" refers to the resource and all members (recursively)

PROPFIND
(1 of 2)

Retrieves resource(s) properties
- All name/value pairs
- Specified name/value pairs
- Just the names (discovery)

Depth: header may be used

PROPFIND
(2 of 2)

Request body is optional
- Not provided: fetch all name/value pairs
- Otherwise, the body specifies the semantics
  - Which values to fetch
  - Just fetch the names

PROPPATCH

- Add, change, remove properties
- Request body required
  - Specifies the properties to affect
  - Specifies the operations to perform
- Operations are performed in sequence
- The set of changes are atomic

COPY and MOVE

COPY/MOVE resources around

Best effort rather than all-or-none

Copies are by-value

Resources include their properties!

Request body may contain instructions on (live) property handling

Uses Destination:, Depth:, Overwrite:

Overwrite: Header

Two values:

- "T" will overwrite the destination
- "F" will cause an error if the destination exists

Overwrite is defined as a DELETE followe
by the MOVE/COPY

- Not a merge!
- Not necessarily atomic!

MKCOL

- Creates a new collection on the server
- No request body defined in core spec
- Explicit collection creation avoids overloading PUT semantics

LOCK and UNLOCK

- LOCK creates a new lock
  - Lock-Token: header returns the token
  - Request body specifies lock characteristics
- UNLOCK removes the lock
  - Identified by the Lock-Token: header

Feature Discovery

- HTTP/1.1 OPTIONS method
  - Allow: header specifies available methods
  - DAV: header specifies conformance classes
    - "1" meets all MUST requirements of the spec
    - "2" also supports locking semantics
    - Example: `DAV: 1,2`
    - Classes, not a conformance level!

Protocol Design Points

- Methods are designed to avoid round-trips
  - – PROPFIND/PROPPATCH in particular
  - – MOVE/COPY keeps the resource off the wire
- With HTTP pipelining, one TCP connectio may be used for multiple requests

Some Thoughts
(1 of 3)

Uber protocol?
- Easily replaces FTP, FrontPage, Fusion
- CVS, SourceSafe, etc
- POP, IMAP?
- NNTP?
- SMTP?

Requires mapping protocol actions onto DAV operations and its data model

Some Thoughts
(2 of 3)

Okay, it could be “the” protocol. Why?

Leverage

– Worldwide infrastructure, talent, tools, etc
– Clients can build upon a single library

“Just because you can, doesn’t mean you should”

– True... time will tell whether this direction actually provides benefit

Some Thoughts
(3 of 3)

- DAV servers are relatively passive (content is viewed as opaque entities)
  - Provides a data model
  - Provides ways to manage that data
- An RPC mechanism would be used to perform active processing on the content
- Or better yet, use new HTTP methods for the desired operations

Closing

What to do with DAV?
(1 of 2)

Perfect for remote data repositories

Good for distributed systems where clients contain logic, servers are relatively passive

Good template for building HTTP-based distributed systems

What to do with DAV?
(2 of 2)

Adobe is defined by its publishing and authoring tools

DAV provides an excellent infrastructure for remote publishing and authoring

Use DAV to add remote capabilities

Resources

- WebDAV Resources site
  - http://www.webdav.org/
- Working Group site
  - http://www.ics.uci.edu/pub/ietf/webdav/

Attributions

Thanks to Jim Whitehead, Chair of the WebDAV Working Group, for providing a slide deck; portions of some slides were based on his deck.

Most of the information is derived from the specifications, discussions, and experience.

Q&A