Goals for a Configuration Management Network Protocol

E. James Whitehead, Jr.

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
Tel: +1 949 824 4121
E-mail: ejw@ics.uci.edu

Abstract. Experience from research on integrating versioning and configuration management support to the Web has shown that building such support on the basic Web infrastructure leads to undesirable architectural choices. This paper presents goals for a standardization effort called Delta-V which is extending the Web infrastructure, specifically the core network protocols HTTP and WebDAV, with capabilities for remote versioning and configuration management. Important goals include providing equal support for all content types, allowing versioning unaware applications to participate, supporting both mutable and immutable revisions, ensuring that human-readable strings are internationalizable, and provision of strong authentication and transport security. These goals are currently being used to develop the Delta-V protocol within the Internet Engineering Task Force.

1 Introduction

What if you were an employee of a small design firm called DesignHaus, working on a Web site update for a client whose pages are *very* time sensitive. To benefit from a broad pool of design talent, DesignHaus draws its designers from around the world, and you're working in a team active 24 hours a day — Kenji supplies graphics and icons from Tokyo, Denise provides page layout in Paris, while simultaneously Peter writes custom JavaScript in New York. You're in Los Angeles, in charge of content, working closely with the client to fine tune the Web site's message.

Though unaware of this fact, you are using a network protocol called Delta-V, a series of extensions to the Web's core protocol, the Hypertext Transfer Protocol (HTTP) [9] and the WebDAV protocol [13] (itself an extension of HTTP) which open the Web for collaborative authoring and versioning. From your perspective, you are simply collaborating over the Internet using your favorite tools. After finishing a phone call with the client, you start your Web page editor, and start editing a product information page by typing its URL into the "File... Open" dialog box. Behind the scenes, the tool makes a network protocol request to check out the Web page from the group's work area located in the URL space of a Web server at the DesignHaus corporate headquarters in New York. The editor learns this check out will result in parallel development, so it pops up a dialog box indicating that Denise is also working

on this Web page at the same time. Not a problem — you decide to work in parallel with Denise. In the background, the editor sends a network request to download the Web page, and you see the Web page quickly appear on screen, ready for changes.

Once you've finished altering the content based on the client's feedback, you save it using the "File... Commit" dialog in the editor. The editor saves the final version by writing over the network directly to the URL, and then sends a network request to perform a check in. The check-in succeeds, and the editor discovers that Denise has also committed her latest changes. It pops-up a dialog box asking you if you want to merge your changes with Denise's. You select yes, and the tool brings up a merge utility. The editor already has a copy of your latest changes, but it has to perform a network request to retrieve Denise's latest revision, and then check-out the Web page to make the merged revision. Since Denise was altering layout while you were altering content, there are few conflicts which are easily resolved. When done, you select "File... Commit", and the editor makes two network requests to write back the Web page to its URL, and check in the new merged revision.

Since you've been editing directly in-place on the Web, you're now ready to call back your client, tell them the URL, and get their feedback on the new revision.

Developing the network protocol which makes this scenario possible is the goal of the Delta-V working group of the Internet Engineering Task Force (IETF). While work on the Delta-V protocol is still underway, and the final design is still in flux, there are many interesting preliminary results. This paper reports on the initial experience of the Delta-V working group in developing a network protocol for remote versioning and configuration management of Web sites, software projects, and other complex information artifacts.

Adding versioning support to WebDAV will support the following capabilities:

- Development teams can collaboratively develop complex information artifacts in-place on the Web, using locking to prevent overwrite conflicts, and versioning to support parallel development. Due to the distributed nature of the Web, these workgroups can have members from within the same organization, or across organizational boundaries.
- All the types of artifacts in a typical software development lifecycle or encountered in a typical Web site can be remotely versioned, including requirements, design documents, test cases, code, GIF and JPEG images, CGI scripts, Java code, and much, much more. The protocol will support versioning of HTML and XML just as easily as it supports versioning of existing word processing, spreadsheet, text, graphics, and all other formats.
- The protocol will provide a common interface to a wide range of repositories, such as configuration management systems, file system based versioning systems (e.g. RCS [26]), databases, document management systems, etc. In essence, WebDAV makes the Web look like a versioned, large-grain network-accessible file system. But, unlike a conventional file system, a WebDAV-enabled repository provides Internet access, and allows all "files" to be viewed using a standard Web browser.

The paper is organized as follows. It begins with highlights of the Web's client-server architecture, including a description of the existing capabilities of HTTP and WebDAV. Next is a presentation of key high-level goals that have been identified by the Delta-V group, interspersed with some discussion of significant technical challenges and initial design choices. The paper ends with discussion of related work.

2 Delta-V, a Client-Server Architecture

The Delta-V protocol is a series of extensions to the WebDAV protocol, which itself extends HTTP, a client-server protocol. Figure 1 shows three different Delta-V clients interacting via the HTTP/DAV/Delta-V protocol with a hypothetical Web server that provides interfaces to several different storage repositories. The advantage of this client-server architecture is *distribution*. In the case of WebDAV and Delta-V, by allowing the client/server information flow to take place over the Internet, clients, which author and version Web resources, and servers, which provides persistent storage for these resources, can be physically separated by thousands of miles. Furthermore, the network protocol in a client/server system acts as a bridge across organizational and system boundaries, allowing remote cooperative authoring by collaborators located in different organizations (or different divisions of the same organization), using different hardware platforms.

To perform an action in HTTP, a client opens a network connection to a server, then creates a request message, which is divided into three parts, the request-line, a series of headers, and a request body. The client issues a command, called a *method*, by encoding the command in the request line. Parameters for the method are passed in two forms, as attribute-value pairs, called *headers*, and as Extensible Markup Language (XML) [7] in the message request body. Once the request is complete, it is sent to the server, which unpackages the request and executes the command. After processing the request, the server creates a response message, which contains a status line, a series of headers, and a response body. Success or failure is indicated by a status code, a three digit number located in the status line.

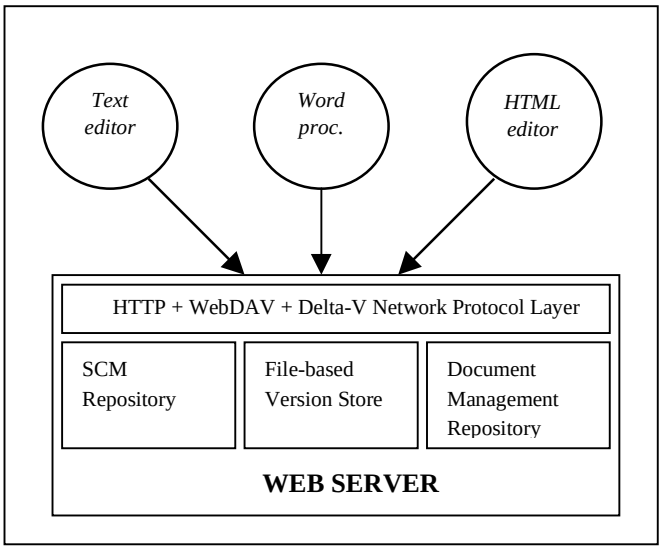

Fig. 1. This diagram shows three versioning capable remote authoring tools communicating via the HTTP protocol using WebDAV and Delta-V extensions to a Delta-V capable server. The server shows three different persistent storage interfaces, highlighting the ability to map Delta-V operations across a wide variety of repositories. Typically no one server will implement so many storage interfaces simultaneously.

The basic object within HTTP/WebDAV/Delta-V is the *resource*, which represents a network accessible data item which provides a set of operations, its methods. A resource contains two kinds of state, state associated with the *body*, and *property* state. The body is the content retrieved by a read request on a resource, and is a *representation* of the body of the resource, not necessarily the exact contents of the body state. This extra abstraction permits dynamic behavior of resources, allowing a read request response to be the output of a computational process, like a CGI script or an active server page (ASP). A resource also contains properties, name/value pairs, where the value of a property is a sequence of well-formed XML.

Table 1. Methods supported by HTTP and WebDAV

HTTP METHODS	
GET	Retrieve a resource and simple metadata, such as its length, MIME type, and cache tag.
HEAD	Retrieve just the simple metadata, but not the contents of a resource.
POST	Submit form data. Often used for tunneling other RPC schemes.
PUT	Write a resource.
DELETE	Remove a resource.
OPTIONS	Discover which methods are supported by a resource.
TRACE	Debug method.
WEBDAV METHODS	
PROPFIND	Retrieve properties (complex metadata) from a resource, or a tree of resources.
PROPPATCH	Set and remove properties on a resource.
MOVE	Move a resource, or a tree of resources.
COPY	Copy a resource, or a tree of resources.
MKCOL	Create a new collection.
LOCK	Lock a resource, or a tree of resources, preventing overwrite conflicts.
UNLOCK	Remove a lock.

As Table 1 highlights, HTTP provides operations for retrieving Web resources, the most frequently used HTTP operations. HTTP also provides basic authoring capabilities with the PUT and DELETE methods, which are only now starting to enjoy widespread client support as WebDAV is adopted. WebDAV extends base

HTTP with capabilities for overwrite prevention, properties, and namespace management, which are described briefly below, with details available in [28].

Overwrite prevention. Keeping more than one person from working on a document at the same time. This prevents the "lost update problem" in which modifications are lost as first one developer, then another writes changes without merging the other developer's work.

WebDAV provides overwrite prevention via its shared and exclusive lock capability. This dual lock support provides sufficiently flexible locks to accommodate a wide range of collaborations, with shared locks best supporting collaborators who have a lot of awareness of each other's activities, and exclusive locks providing a more stringent guarantee of conflict avoidance for less aware collaborators, or during periods of high contention for a resource. Locks may have a scope of a single resource or a hierarchy of resources, such as a source code tree. A lock discovery mechanism (a WebDAV property) allows authors to find out if any locks exist on a resource. Since the Web is designed so that no lock is required to read a Web page, there is no concept of a read lock.

Properties. Creation, removal, and querying of information about resources, such as its author, last modified date, etc. Also included is the ability to make hypertext links between resources of any content type.

WebDAV properties are name, value pairs where the name is a URI, and the value is a sequence of well-formed Extensible Markup Language (XML) elements. Using URIs as property names provides a globally unique property namespace. Since property names can be URLs, which have a domain name as a component of the URL, property names can be given uniqueness without central registration by using URL property names chosen from within a domain whose name is controlled by the party defining the property. So, for example, a company which controls a given domain name, like "widgets.com" can chose a property name from within this domain, like "widgets.com/properties/color".

Name space management. Creation, removal, and automatic consistency maintenance of the membership of collections containing sets of Web resources. Also, the ability to copy and move Web-accessible artifacts, and to receive a listing of resources in a collection (similar to a directory listing in a file system).

Functional goals for the capabilities which will be provided by the Delta-V protocol are described in the next section.

3 Functional Goals for a Web Versioning and Configuration Management Protocol

This section describes the primary functional goals for the Delta-V protocol. Due to space limitations, it is not possible to list all goals. A complete listing is given in the Delta-V working group's consensus goals document [24]. Where appropriate, text in this section is quoted from the working group's goals document.

Throughout this paper, the terms *revision* and *versioned resource* will be used. Informally, a versioned resource is an abstraction representing all of the revisions of a particular resource over time, such as all of the revisions of the HTML page "index.html" over its lifetime. A revision is a persistently stored specific instance of a

resource, for example, the contents of "index.html" as checked-in at a specific moment in time. Formally, a versioned resource is an abstraction for a resource which is subject to version control, a resource having a set of revisions, relationships between those revisions, revision identifiers and labels, and named branches that track the evolution of the resource. A revision is a particular version of a versioned resource

3.1 Equal support for all content types

The Web is composed of documents, images, and objects of many content, or Internet media types [11]. The Web is composed of more than just text. *It must be possible to version resources of any media or content type, that is, a Web versioning and CM protocol must treat all content types equally.*

A protocol which only provides operations for resources of one preferred content type, such as HTML, would have limited applicability due to its lack of support for the wide variety of other content types. Furthermore, since many common content types are in constant evolution, in order to ensure stability of the protocol a strict separation between the protocol, and the format of the objects operated upon by the protocol, must be maintained. For example, during the development of the WebDAV protocol itself, new standards for HTML 3.2, 4.0, and XML were issued, highlighting how quickly these document formats can develop and evolve. Tailoring an authoring protocol too closely to any one content type would rapidly make the protocol obsolete.

3.2 Versioning aware and non-versioning aware clients must be able to interoperate

Many WebDAV authoring clients are expected to be in use by the time the Delta-V protocol is supported in shipping products. There are also some authoring tools which just use HTTP, and are not even WebDAV aware. To aid adoption of the Delta-V protocol, it is important to provide a minimal level of versioning support to these clients, so they can interact with a versioning server without having any knowledge of the versioning protocol. This allows people to gain versioning capability without having to change their tools, and provides a ready-made base of clients when versioning servers first become available. It does, however, raise the design challenge of ensuring that versioning-aware and versioning-unaware clients can smoothly interoperate, preventing unforeseen negative interactions.

Non-versioning aware clients should be able to request the contents of a versioned resource without specifying a revision and receive a well-defined default revision. A non-versioning aware client should be able to write to a versioned resource and have a new revision automatically be created. It is expected that a write operation will perform an implicit check-out, write, and check-in to maintain versioning semantics and avoid lost updates. A subsequent read on the same versioned resource by this client will return the new revision. However, the versioning operations available to non-versioning clients will be very limited. For example, such clients will not be capable of submitting comments or properties on check-in or check-out, and will be limited to either linear versioning, or working on only a single branch.

3.3 Configuration management capability is an optional extension to versioning capability

Two kinds of clients are expected to support the Delta-V protocol. The first is an authoring client, such as a text editor or a word processor, which will only provide versioning capabilities, allowing a check-out, edit, check-in style of interaction. The second type of client is a configuration management "control panel" application, which understands all of the configuration management capabilities of Delta-V. For Delta-V, a *configuration* is a versioned resource that contains a set of specific revisions of versioned resources, thus making a each versioned resource a configuration item. One example of a configuration is a snapshot of a Web site, where the configuration records the revision of every Web page and graphic image in the site. A Delta-V configuration management control panel application can perform actions like creating configurations, check-out and check-in of configurations, and reverting to a previous configuration. Though these control panel operations are not expected to appear in authoring tools, it should be noted there is nothing to prevent an authoring tool from supporting these configuration management capabilities.

The major benefit of this two-tier approach is that it provides a simple initial set of functionality which, if supported, provides useful versioning features. By limiting the scope of functionality initially required to be Delta-V compliant, it is much easier to convince tool makers to add Delta-V support, thus boosting adoption of the protocol. The same logic works on the server side too. Since some use situations do not require configuration management support, such as a small set of artifacts being developed by a small, closely knit team, a versioning-only server might provide all the support they need. Existing versioning systems can provide Delta-V support without having to add configuration management capabilities to their system, thus creating a versioning-only server which can be used by these smaller projects.

However, as sometimes happens when projects grow, if a versioning-only team discovers they need configuration management capabilities, using their versioning-only authoring tools, they can trade up to a server which supports configuration management capabilities in addition to versioning operations. The only additional tool required is a configuration management control panel. Thus, supporting this requirement creates two stable plateaus of functionality, a versioning level, and a configuration management level, which give tool and server makers an easy, full-featured entry level for small to medium scale projects, and a clear upgrade path to configuration management capability for large-scale projects.

3.4 Revisions may be mutable or immutable

Versioning support for software development has traditionally emphasized the creation of revisions which, once checked-in to the control of the versioning system, can never be modified again, making them immutable. This makes sense for software development due to the machine-readable nature of program code, where every change needs to be tracked. However, versioning support for documents within document management systems is often more relaxed about tracking every possible change. In these systems, it is more important to maintain the logical name of a revision (e.g., "June Sales Brochure") than to track every small change that has been

made. As a result, these systems do allow changes to a revision, *even after it has been checked-in*. Users of these systems appreciate the ability to make minor changes to these mutable revisions, like fixing a spelling error, without having to create an entire new revision. For these use cases, the immutable style of versioning is just not appropriate.

A common, but misguided, approach to mutable revisions is to simulate mutable revisions using immutable revisions. One technique is to provide a mechanism that performs a check-out, modification, and check-in in one convenient operation, thus preserving the immutability of revisions, while still allowing easy changes. However, the ease of making changes isn't the main point. The intent of mutable revisions is to avoid creating a new revision for a small change, and to have each revision maintain a stable logical value. For example, with mutable revisions, a version history for a monthly sales brochure need only have 12 revisions, one for each month. In many use environments where there are minimal archival storage requirements, there is little need for storing intermediate revisions as the next month's brochure is developed. Similarly, if minor spelling or grammatical errors are fixed in a brochure before a second printing is made, storing the original, error-containing document has little value.

Since configuration management systems have a fundamental constraint that revisions must be immutable, it is expected that a mutable revision will not be a permissible member of a configuration. In fact, it is likely that servers which support mutable revisions will only support versioning, and will not provide any configuration management support. This is due to the semantics of configurations that require the ability to revert to a previous revision of the configuration and get exactly the previous contents. For many use environments, especially software and system configuration control, the ability to revert and get exactly the previous contents is critical for providing control over large systems. However, there are many use environments where such stringent control is not needed, and is in fact burdensome. Since the Delta-V protocol is intended to be used in environments that vary in their configuration control requirements, both immutable and mutable styles of versioning should be supported.

3.5 Revision history support

Storing previous revisions of resources, along with descriptive comments for each revision, is a major benefit for versioning Web resources. Therefore, retrieving a revision history which lists all of the revisions of a versioned resource, details predecessor relationships among the revisions, lists comments submitted on each revision, and gives the URL for each revision, is an important feature. This information can be used by versioning-aware clients to display a graphical representation of the version history, and allow direct navigation to an individual member of the revision history.

A significant design choice in Delta-V is to make the Web server the control point for consistency maintenance of the predecessor and successor relationships in the revision history of a versioned resource. Some systems, such as NUCM [14] and the NTT web versioning system [19] place the client in control of creating and updating these relationships, providing the significant benefit of allowing version histories to span multiple servers. However, an implicit design goal of Delta-V is to provide a

Web gateway to existing configuration management and document management systems, where the expected use environment will have some people using these systems via the Delta-V protocol interface while others simultaneously use the system via its local access interface. Since these existing systems maintain the consistency of predecessor and successor relationships that are stored within their repository, they were unwilling to cede this consistency control to clients. Furthermore, if consistency control were the purview of clients only, the possibility exists that a single poorly implemented, or poorly behaving client could disrupt version histories created by other, well-behaved clients. This was considered unacceptable.

One implication of having the versioning relationships controlled by the server is the difficulty of providing version histories which span servers. Indeed, in the server-controlled scheme, server-spanning version histories require a server-to-server protocol for transmitting messages to coordinate these histories. Since no such protocol exists or is planned, server-spanning version histories are a feature awaiting future work. However, it is a goal to ensure that nothing in the current design of Delta-V makes it impossible to add cross-repository version histories in the future, primarily requiring Delta-V to ensure identifiers used in version histories are unique across all servers, not just a single server.

3.6 A revision is a resource, with its own URL

The HTTP protocol operates on resources, hence this goal has the result of making each revision an object which can be operated on by the HTTP/WebDAV/Delta-V protocol. This provides two important benefits. The biggest benefit is that a revision can be the target of an HTML link, allowing hypertext browsing of previous revisions of a resource. Close behind is the ability to perform other protocol operations on revisions, such as retrieving properties.

3.7 A mechanism must exist for giving a human readable name to a single revision

Version control systems often reserve the right to give individual revisions an identifier (e.g., “1.5”), and this identifier may not be intelligible to a human user. As a result, it is useful to be able to associate a human-readable *label* to a revision. This label is unique within a revision history, and can be used to identify a specific revision.

The introduction of labels may seem confusing at first, since, due to the previous requirement, a revision must have its own URL, and one of the design requirements for a URL is human readability. Hence it might seem that the per-revision URL meets this requirement. However, labels and URLs differ in their uniqueness requirements. The URL for a revision must be globally unique, since it identifies just a single resource. In contrast, a label need only be unique within a versioned resource, identifying a single revision among the set of revisions in a versioned resource. The same label is expected to be used across multiple versioned resources on the same server, and can be used in revision selection rules by workspaces and configurations to select, for example, all the revisions that make up a particular release. Furthermore,

while revision URLs are expected to refer to the same revision over time, a label may be moved across revisions of a versioned resource.

3.8 Versioning should not disrupt relative URLs

Documents on the Web in HTML format often employ hypertext links or image tags whose target location is expressed relative to the URL of the current resource. One typical use of this feature is to create a directory of images and icons containing a common appearance for a site. A link to one such image might look like "../images/logo.gif". To ensure that versioning support will not break links in HTML documents containing relative URLs, versioning support should not disrupt relative URLs. A corollary of this requirement is, if a set of resources is arranged in a URL hierarchy before they are placed under version control, then this hierarchy should be the same after the resources are version controlled.

The primary effect of this requirement is to constrain the types of changes that can be made to the URL namespace to support versioning. URLs do not have any support for adding revision identifier information, and several schemes have been proposed to address this deficiency. One commonly suggested scheme is to place all the revisions of a versioned resource in a collection that has the original name of the resource. In this scheme the URL for revision 1.1 of "index.html" would be "index.html/1.1". However, such a scheme would disrupt relative URLs, since it introduces an extra hierarchy layer. This particular scheme also has the drawback of not handling versioned collections well.

In the design of the protocol, there is a tension between giving every revision its own URL, and not disrupting relative URLs. Since URLs do not have provisions for expressing revision identifier information, once each revision is given its own URL, every resource which used to have a single URL before it was versioned now has multiple URLs, one for each revision. These URLs need to go someplace in the URL namespace, and it is likely they will not end up at the original URL of the resource. Current designs address this tension by creating separate spaces for the revision URLs and the "edit" URL, the location where the resource is actually edited. At the edit URL location, the behavior of relative URLs in links will be preserved, while relative URL links may not work at the revision URL location.

Providing configuration management support for Web sites containing relative URL links has the implication that these links depend on a specific hierarchical structure of the site. Since the site hierarchy is expected to change over time, it does raise the possibility that a configuration of web pages could be created where the relative URL links are broken. This could occur in a snapshot of a Web site that is in the middle of modifications during a site redesign. As a result, the requirement that versioning should not disrupt relative URLs is interpreted as meaning if the relative URLs worked when the resources were not versioned, then once the same set of resources is placed under version control, relative URLs in the same configuration should still work. Or, put another way, if the relative URL links do not work, this should be the result of an operator action, and not the result of URL namespace restrictions imposed by the Web versioning system.

Once a separation is made between the edit URL space, and the revision URL space, the edit URL becomes a window, or a view, onto a versioned resource,

associated with one of the revisions in a versioned resource. This leads to the following requirements on the capabilities supported by the edit URLs.

3.9 Read requests on a URL to a versioned resource should return a default revision

If a non-versioning aware client, such as a current generation Web browser, makes a read request at the URL for a versioned resource (a.k.a., an edit URL), it is desirable for the server to reply with some default revision. Likewise, it is desirable to be able to set this default revision. This goal is a subset of the following, more general requirement.

3.10 A mechanism must exist for associating a particular revision to a URL for a versioned resource

When a request is made against a URL for a versioned resource (a.k.a., an edit URL), which specific revision is returned? The answer to this question depends on the current work being performed. When making a change, the answer will be the latest stable configuration, except for the changes just made.

To address this goal, the Delta-V design employs the notion of a *workspace*, which acts as a mediator between the edit URL space and the revision URL space by using a set of revision selection rules to choose a specific revision for each edit URL. The expectation is that each user will have their own workspace, and as a result, workspaces support parallel development since each user can have the semblance of their own private working space. The server performs the action of evaluating the revision selection rules of the workspace, associating a revision URL with each edit URL. To perform an action inside a workspace, a client passes both an identifier for the workspace (as the value of an HTTP header) and a specific edit URL to the server. The server evaluates the revision selection rule, and applies the operation to the selected revision.

3.11 Some properties on revisions may be changed without creating a new revision

This goal is motivated by the recognition that there are two types of properties. The first type contains metadata about the resource, and this metadata is directly dependent on the content of the resource. The second type of property is used for protocol operations, typically to expose some system-maintained information about the resource. Access control permissions are a commonly discussed example of this kind of property, motivated by the desire to modify the access permissions of a revision even after it has been checked-in.

3.12 A mechanism must exist for logically grouping sets of changes to one or more versioned resources

Frequently a single logical change, such as updating a link whose destination has changed, requires modifications to multiple resources. A mechanism for mapping a logical change to actual changes provides several benefits. It allows these mappings to be preserved; without it, this information would need to be stored in comments, and would be difficult to reconstruct. This mechanism can also be used to merge together parallel work.

The Delta-V design makes use of the *activity* concept to support this goal. An activity contains a set of versioned resources, and for each of these, one or more revisions. An activity is not versioned. The Delta-V protocol allows an activity to be merged into a workspace, supporting parallel work. If the act of merging an activity into a workspace causes change conflicts, a *conflict report* is generated. A client is expected to resolve these conflicts.

3.13 A mechanism must exist for creating versioned sets of specific revisions of versioned resources

A primary feature of configuration management system is the ability to freeze important configurations of the system for later retrieval and manipulation. While similar to activities, configurations do have some differences. A configuration is versioned, an activity is not. A configuration is used to create and store persistent views of the state of an entire system, while an activity is used to represent a single logical change to a part of the system.

3.14 Revision Operations

Goals for operations that should be supported by revisions are:

- Create a versioned resource from an unversioned resource and set its initial revision to the contents of the unversioned resource.
- Check-out a revision in an activity
- Check-in a resource and either create a new revision (immutable revisions) or update the existing revision in place (mutable revisions)
- Cancel a check-out
- Describe a revision with human-readable comments

These goals provide the operations needed to place an unversioned resource under version control. A Delta-V enabled Web server will typically have part of its URL namespace that is not versioned, part that is versioned, and part that is a mix of versioned and unversioned resources.

Delta-V will use the library model for versioning, using a check-out operation to make a versioned resource suitable for editing, and check-in operation to create a new revision. It will be possible to cancel a check-out, if the entire edit operation needs to be cancelled. Since it is a typical versioning operation to associate human-readable comments with a particular revision, Delta-V will also provide this capability.

3.15 Label Operations

Goals for operations that should be supported by labels are:

- Apply a label to a particular revision
- Change the revision to which a label refers
- Retrieve all labels on a particular revision

An important use of labels is to provide a consistent, human-readable name that can be used to decorate specific revisions in multiple versioned resources. These labels can then be used within a revision selection rule by a workspace to create a consistent set of resources that can be edited.

3.16 Activity Operations

Goals for operations that should be supported by activities are:

- Create and name an activity
- Check-out a revision in an activity
- Merge an activity into a workspace, possibly creating a conflict report
- Get a list of the resources modified in an activity
- Apply a label operation to all resources in an activity

Within Delta-V, activities are created by authors to organize related changes to resources, and to provide a basis for parallel development and merging concurrent changes to the same resource. An activity can contain revisions of multiple versioned resources, and/or multiple revisions of the same versioned resource along a single line-of-descent. The activity operations listed above give the critical functions a protocol must support to provide activity functionality.

3.17 Configuration Operations

Goals for operations that should supported by configurations are:

- Create/delete a configuration
- Add/remove revisions from a configuration
- Use a configuration in a workspace’s revision selection rule to choose revisions in that configuration.
- Determine the differences between two configurations by listing the activities in one and not the other.

These operations provide the base set of capabilities needed to support configurations. They allow a configuration to be created and populated with revisions. Once created, a configuration can be used by a workspace's revision selection rule so to only select revisions within the configuration, thus providing the ability to revert to a previous configuration.

3.18 Internationalization

Since the Internet is in use around the world, it would be arrogant and inappropriate to hard-code dependencies on any human language into the protocol. For the Delta-V protocol, internationalization support means that any string used within the protocol that would typically be displayed to a human operator must store sufficient information such that it can display that string in most known human character sets. Recent protocol specifications have met this requirement by supporting one of the encodings of the ISO 10646 [16] standard, which encodes most known human character sets. The contents of the string also needs to be augmented with the human language of the string, so that it can be properly displayed, an issue with ideographic languages where the display of the same character may vary across languages.

One aspect of the protocol affected by this requirement is revision labels, which are definitely exposed to a human operator. Instead of using just an ASCII string, internationalization requirements require a label to be a tuple of string, encoding, and language. Storing multiple variants of each string (e.g., a Japanese and an English representation of the same label) is not a requirement.

3.19 Security

Typically, configuration management systems operate within a single organization where most users are known, and have often met other users in person. There is inherently more trust in this situation than will exist on the Internet, where collaboration may be taking place across organizations, or within a virtual organization comprised of people from diverse geographic locations. These collaborators may never have met each other in person. As a result, *a protocol for versioning and configuration management over the Internet needs to support strong authentication* so users can correctly be identified. This needs to be backed-up by operational procedures that ensure authentication credentials are given to appropriate people.

Furthermore, on the Internet the possibility exists that the traffic across a connection may be spied upon. Since source code is often proprietary, and may contain trade secrets, it is important that source code contents not be exposed during transmission across the Internet.

The Delta-V protocol intends to leverage existing HTTP technologies to address the dual problems of authentication and transport security. Digest authentication [10], developed as an alternative to HTTP Basic authentication and mandated by WebDAV, allows a client to send a multiply one-way hashed username/password pair to the server to authenticate the client to the server. Since Delta-V uses HTTP, the Transport Layer Security (TLS, more widely known as SSL) [8] standard can be used to encrypt a connection between client and server to prevent eavesdropping.

4 Related Systems and Protocols

Systems which are similar to those which would employ the Delta-V protocol can roughly be categorized as either Web-based or distributed versioning and

configuration management systems. Though the Web-based systems can rightly be viewed as a subset of the class of distributed systems, they are examined here separately due to the focus on Web support in this paper. Selected systems from each category are described below to highlight their differences and similarities to the Delta-V work.

4.1 Web-based Systems

There has been interest in providing versioning support for the Web from its very inception. Tim Berners-Lee, in his original design notes on the Web describes as important the issue, "keeping track of previous versions of nodes and their relationships." [6].

Initial work on Web versioning concentrated on browsing the contents of version repositories. An early paper describing this capability is [20], where the authors highlight the need for version support in digital libraries, and provide this support via a CGI script which interprets ",{version identifier}" at the end of a URL as a request to retrieve that revision from an RCS repository. This idea has recurred in the literature, most recently in [25], which presents a module for the popular Apache Web server [2] that interprets URLs appended with a ":{version identifier}" or a ":{date}" as a request to retrieve either the specific revision, or the revision that was current as of the date. This approach suffers from two drawbacks which preclude their use in a versioning protocol standard. First, URLs are intended to be opaque, with no meaning encoded into their syntax. Since there are relatively few reserved characters in URLs, if meaning is encoded into URLs, this extra meaning will likely use these reserved characters, and over time lead to semantic collisions between extensions. While an individual server may choose to constrain its URL namespace and use one of these extensions, it is improper for a standard to constrain *all* server namespaces in this way. The second drawback is that the interior path elements of a URL cannot be easily extended, and hence the technique falls short when identifying a specific revision of a versioned collection.

Another common architecture for adding versioning services to the Web is the "form fill-in" style. Examples of this type of system are BSCW [4] and WWRC [23]. These systems share the approach of using HTML pages to create the user interface to a revision control system. Commands are either appended to URLs, or sent to the system using the HTTP POST method, which has a sufficiently broad definition that different remote procedure call mechanisms can tunnel through it. Internet content types attuned to helper applications on the client side help the transfer of information from server to client for editing.

A more sophisticated architecture for adding versioning to the Web is the "Java helper app." approach. In this technique, a Java application is downloaded into the browser, and acts as an intermediary between a version control repository and the user's local environment. This technique is employed in the WWCM [15] and MKS WebIntegrity [18] systems. Similar to these is the WebRC system [12], which uses the CORBA RMI protocol instead of HTTP. WWCM will be discussed as an exemplar of this approach. In WWCM, a Java application running in the browser initiates most versioning and CM operations, but due to the security model of Java which prevents a browser-based application from writing to the local filesystem, it also requires the use of a second helper application which is free from this restriction.

Since direct communication between the browser application and the helper application is not permitted, WWCM must employ a circuitous sequence of three to four network round trips to perform check-out and check-in operations. While WWCM is an exemplar of the kind of CM functionality which can be integrated with the Web using only existing standards, the awkwardness of the two applications and the multiple network round trips highlights the need for developing extensions to the core standards to better support this capability.

Web-based versioning and CM systems assume that the Web server is responsible for maintaining the predecessor and successor relationships between revisions. However, one proposal [19], based on research at NTT Labs., suggested using the (now deprecated) LINK method in HTTP to have client-maintained relationships between resources in a version history. Though not adopted by the Delta-V effort, the idea has the advantage that version graphs can span multiple servers without requiring cooperation between these servers. However, it has the drawback of making operations that span the version history expensive. For example, in such a system it would be difficult to use labels, since ensuring the uniqueness of a label requires a traversal of the version history, an action which may span many servers, some of which may be unavailable due to network outages at any given moment. It also has the drawback that clients must be well-behaved — a single misbehaving client could corrupt a revision history.

A different take on client-side versioning is VTML [27], which augments HTML to store all modifications to an HTML file internal to the HTML file, essentially making each HTML file its own version store. Clients which support authoring of VTML documents are responsible for maintaining the internal revision structure. One major benefit of this approach is that it easily supports simultaneous collaborative authoring of the same HTML file. Since VTML stores all changes to a file, no locking is necessary as the overwrite problem cannot occur because no data is ever overwritten.

4.2 Distributed Configuration Management

A common problem faced by Delta-V, ClearCase [22], and by *n*-DFS [3], is the desire to improve a pre-existing system by adding CM support. Both ClearCase and *n*-DFS add this support to the filesystem, and hence the low-level interface to data stored in the system is via operating system library calls. In contrast, since Delta-V is building upon the Web, access to data is via HTTP, not operating system calls, and this introduces new design flexibility — for example, the semantics of namespace bindings (akin to Unix hard links) can be tailored to the needs of versioning. However, unlike ClearCase multisite [1], and *n*-DFS, Delta-V is not initially planning on providing facilities for repository to repository replication, although one design goal for Delta-V is to ensure adding such services in the future is not prohibitively difficult.

NUCM [14] is a client-server CM system in which a NUCM client interacts with a remote NUCM repository server using primitive operations, upon which are implemented higher-level CM styles. In this respect, NUCM is similar to the NTT Labs. work [19] in that the client is responsible for maintaining the consistency of relationships in the remote repository.

Remote CVS [5] is a client-server CM tool in wide use on the Internet today, with a proven track record of supporting open source development projects. It uses its own human-readable non-HTTP protocol [17], which consists of a stateful connection where the client issues one-line commands that may elicit a reply, depending on the command. The cvsweb [29] utility provides a Web forms-based interface to a CVS repository. Another recent non-HTTP protocol for remote CM is [21] which presents a client-server protocol which uses ASN.1 as its marshalling syntax.

Acknowledgements

The discussion of goals in this document was based on the consensus goals document developed by the Delta-V design team, Jim Amsden, Alan Babich, Geoff Clemm, Bruce Cragun, Chris Kaler, Jeff McAffer, Bradley Sergeant, John Stracke, and the author. The author has contributed to, but is not solely responsible for the content of this consensus goals document. The current goals document is itself based on an earlier goals document by Judith Slein, Fabio Vitali, and David Durand. Discussions on versioning with the WebDAV design team, Yaron Goland, Asad Faizi, Steve Carter, and Del Jensen also helped crystallize my understanding of these issues.

References

1. L. Allen, G. Fernandex, K. Kane, D. Leblang, D. Minard, J. Posner, “ClearCase MultiSite: Supporting Geographically-Distributed Software Development.” In J. Estublier (ed.) *Proc. SCM-4 and SCM-5, Software Configuration Management: Selected Papers*, LNCS 1005, Springer-Verlag, SCM-4 and SCM-5, 1995, pages 194-214.
2. Apache Server Project, “Apache Project” Web site. http://www.apache.org/, April, 1999.
3. D. Belanger, D. Korn, H. Rao, “Infrastructure for Wide-Area Software Development” In I. Sommerville (ed.), *Proc. SCM-6, Software Configuration Management: Selected Papers*, LNCS 1167, Springer-Verlag, ICSE’96, SCM-6, Berlin, Germany, March 25-26, 1996, pages 154-165.
4. R. Bentley, T. Horstmann, J. Trevor, “The World Wide Web as enabling technology for CSCW: The case of BSCW” In *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, vol. 6, nos. 2-3, 1997, pp. 111-134.
5. B. Berliner, “CVS II: Parallelizing software development” In *Proc. Winter 1990 USENIX Conference*, January 22-26, 1990, Washington, DC, pages 341-352.
6. T. Berners-Lee, “Versioning”, A Web page that is part of the original design notes for WWW. http://web1.w3.org/DesignIssues/Versioning.html
7. T. Bray, J. Paoli, C. M. Sperberg-McQueen, “Extensible Markup Language (XML) 1.0” World Wide Web Consortium Recommendation REC-xml, February, 1998.
8. T. Dierks, C. Allen, “The TLS Protocol Version 1.0” Certicom. Internet Proposed Standard Request for Comments (RFC) 2246, January, 1999.
9. R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1” U.C. Irvine, DEC, MIT/LCS. Internet Request for Comments (RFC) 2068, January 1997.
10. J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart, “An Extension to HTTP: Digest Access Authentication” Northwestern University, CERN, Spyglass, Microsoft, Netscape, Spyglass, Open Market. Internet Request for Comments (RFC) 2069, January, 1997.

11. N. Freed, N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies” Innosoft, First Virtual. Internet Request for Comments (RFC) 2045, November, 1996.
12. P. Fröhlich, W. Nejdl, “WebRC: Configuration Management for a Cooperation Tool” In R. Conradi (ed.), *Proc. SCM-7, Software Configuration Management*, LNCS 1235, ICSE’97, SCM-7, Boston, MA, May 18-19, 1997, pages 175-185.
13. Y. Goland, E. Whitehead, A. Faizi, S. Carter, D. Jensen, “HTTP Extensions for Distributed Authoring -- WEBDAV” Microsoft, U.C. Irvine, Netscape, Novell. Internet Proposed Standard Request for Comments (RFC) 2518, February, 1999.
14. A. van der Hoek, “A Generic Peer-to-Peer Repository for Distributed Configuration Management” In *Proc. 18th International Conference on Software Engineering (ICSE 18)*, Berlin, Germany, March, 1996, pages 308-317.
15. J. J. Hunt, F. Lamers, J. Reuter, W. F. Tichy, “Distributed Configuration Management via Java and the World Wide Web” In R. Conradi (ed.), *Proc. SCM-7, Software Configuration Management*, LNCS 1235, ICSE’97, SCM-7, Boston, MA, May 18-19, 1997, pages 161-174.
16. ISO/IEC, “Information Technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane”, May, 1993, with amendments.
17. J. Kingdon (and others at Cygnus Support), “CVS Client/Server”, a description of the CVS client/server protocol distributed in the CVS source distribution in file “cvsclient.ps”, initially written 1994, with ongoing revision.
18. Mortice Kern Systems, “Web Integrity” Web site. http://www.mks.com/solution/wi/, April, 1999.
19. K. Ota, K. Takahashi, K. Sekiya, “Version management with meta-level links via HTTP/1.1” Internet-Draft (expired), draft-ota-http-version-00, November, 1996. http://www.ics.uci.edu/pub/ietf/webdav/draft-ota-http-version-00.txt
20. R. Pettengill, G. Arango, “Four lessons learned from managing World Wide Web digital libraries” In *Proc. of the Second Annual Conference on the Theory and Practice of Digital Libraries*, Austin, TX, June 11-13, 1995.
21. S. Ramaswamy, “Version Control Protocol” Internet-Draft, work-in-progress, draft-ramaswamy-version-control-00, February, 1999. http://www.ics.uci.edu/pub/ietf/webdav/versioning/draft-ramaswamy-version-control-00.txt
22. Rational Software, “ClearCase: Configuration Management, Software Development Teams” Web page. http://www.rational.com/products/clearcase/, April, 1999.
23. J. Reuter, S. Hänßgen, J. J. Hunt, W. F. Tichy, “Distributed Revision Control Via the World Wide Web” In I. Sommerville (ed.), *Proc. SCM-6, Software Configuration Management: Selected Papers*, LNCS 1167, Springer-Verlag, ICSE’96, SCM-6, Berlin, Germany, March 25-26, 1996, pages 166-174.
24. J. Stracke, J. Amsden, “Goals for Web Versioning” Internet-Draft, work-in-progress, draft-ietf-webdav-version-goals-00, February, 1999. http://www.ics.uci.edu/pub/ietf/webdav/versioning/draft-ietf-webdav-version-goals-00
25. J. Simonson, D. Berleant, X. Zhang, M. Xie, and H. Vo, “Version augmented URIs for reference permanence via an Apache module design” In *Proc. WWW7, Computer Networks and ISDN Systems*, vol. 30, nos. 1-7, Brisbane, Australia, April 14-18, 1998, pages 337-345.
26. W. Tichy, “RCS - A System for Version Control” *Software - Practice and Experience*, vol. 15, no. 7, July 1985, pages 637-654.
27. F. Vitali, D. Durand, “Using Versioning to Provide Collaboration on the WWW” In *Proc. WWW4, Fourth Int’l World Wide Web Conference Proceedings*, World Wide Web Journal, Vol. 1, No. 1, Boston, MA, USA, 1995, pages 37-50.
28. E. J. Whitehead, Jr., Y. Y. Goland, “WebDAV: A network protocol for remote collaborative authoring on the Web” In *Proc. of the Sixth European Conf. on Computer Supported Cooperative Work (ECSCW’99)*, Copenhagen, Denmark, September 12-16, 1999.
29. H. Zeller, B. Fenner, and H. Nordström, “Hen’s cvsweb CVS Repository” Web page, http://linux.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/, April, 1999.