Using the Web for Document Versioning: An Implementation Report for DeltaV

James J. Hunt
Forschungszentrum Informatik
Haid- und Neu-Straße 10-14
76131 Karlsruhe, BRD
+49 721 608 6320
jjh@fzi.de

Jürgen Reuter
Institut für Programmstrukturen
und Datenorganisation (IPD)
Universität Karlsruhe
Am Fasanengarten 4, 76128 Karlsruhe, BRD
+49 721 608 3934
reuter@ira.uka.de

Abstract
The current suite of systems that offer client/server capabilities for document versioning relies on proprietary protocols for communicating between a central versioning repository and a remote client. In order to support better document authoring via the Web, the DeltaV working group of the WebDAV (WWW Distributed Authoring and Versioning) project of the Internet Engineering Task Force is working on a standard protocol for versioning over HTTP. The authors present a prototype of DeltaV based on the 04.5 draft. This system demonstrates that, though important aspects of the protocol need to be revised, versioning via the Web can be a practical means of supporting remote access to a central versioning repository.

Keywords
DeltaV, Versioning, WWW, WebDAV, RCE

1 Introduction

During the last decade, the World Wide Web (WWW) has evolved to become the most popular user interface for accessing remote resources via the Internet. The development of the Internet also exerts a great influence on structures in business, science, and at home. Globalization describes this process of structures growing to span the entire world with the Internet and the WWW as key technologies. Groups of people working together can use the Internet for distributed authoring and development. Distributed authoring and development itself requires a refined organization and management of working processes. Versioning is a technology that addresses these concerns, but, so far, has not been available in standardized fashion over the net.

The designers of HTTP, the communication protocol of the Web, had the idea of supporting versioning over the WWW in mind from the early days of the Web; but while HTTP has offered various facilities for remote accessing of a server's resources from the beginning, it still lacks adequate support for distributed authoring and versioning. There are no mechanisms in HTTP, such as locking, to coordinate concurrent accesses to a server for modifying resources on that server. A general framework to store and manage meta data about resources is also missing. WebDAV (*Distributed Authoring and Versioning over the Web*) addresses these problems by extending HTTP with a couple of new message headers, HTTP methods and status codes. Though its name already implies support for versioning, WebDAV just provides the basic infrastructure for authoring. It leaves the realization of versioning DeltaV—another protocol which itself is an extension of WebDAV. DeltaV is still being developed, but is likely to become a proposed Internet standard within the next few months.

The authors have built a prototype client/server implementation of DeltaV based on the 04.5 draft specification. Though the prototype is in an early, experimental stage and still far from fully compliant with the specification, it already sheds light onto the workability of the specification. Though the authors encountered difficulties while implementing DeltaV, DeltaV represents a large step towards better Web based distributed collaboration.

2 The foundation: HTTP, XML, and WebDAV

DeltaV is an extension of WebDAV, which itself extends HTTP. A short look at the design of HTTP, WebDAV, and the concepts of XML is needed to understand the foundation upon which DeltaV builds. Of these, HTTP is the most fundamental.

> The *Hypertext Transfer Protocol* (*HTTP*) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods[9].

The overall operation of HTTP follows a request/response paradigm. An HTTP client sends a request message to an HTTP server which handles the message, creates a response message and returns it. An HTTP message consists of a message header with MIME-like message header lines and, optionally, a message body that may contain data of any type. A request message header also contains a request line with the request method, and a response message header contains a status line with a status code.

XML, the *Extended Markup Language*[10], is a subset of SGML[1]. XML documents contain character data and markup elements, giving the document a logical structure.

XML provides a mechanism to impose constraints on the storage layout and logical structure of XML documents. Unlike HTML, which is an application of SGML with a fixed markup grammar, XML is generic. The XML specification defines some rules that a document must satisfy to be a *well-formed* XML document. This includes the syntax of the entities of an XML document and a proper nesting of the markup, i.e. criteria that can be checked with a push down automaton. A document is a *valid* XML document, if it is a well-formed XML document and satisfies some additional rules. In particular, to be valid, an XML document must have an associated *document type declaration* (*DTD*), and the document must comply with the constraints expressed in that definition. In general, testing validity requires symbol table based analysis. The document type definition contains or points to markup declarations that provide a grammar for the logical structure of the document.

WebDAV (*Distributed Authoring and Versioning over the Web*)[12] extends the HTTP protocol by introducing the concept of properties, collections, name space operations, and a resource locking mechanism. For this purpose, it defines a set of new HTTP methods, headers, and status codes. As an alternative to using message headers for transmitting parameters, WebDAV uses XML to transmit structured data in the message body of HTTP messages. Aside from providing well-defined structuring, XML has an additional advantage over message headers in its suitability for describing an unlimited quantity of data. The protocol specifies mechanisms for storing and managing meta data about resources. This meta data is stored in the form of properties associated with each resource. WebDAV distinguishes further between live properties and dead properties. Live properties are managed by the server, whereas dead properties are only stored on the server, but are managed by the client. Properties are expressed using XML. Though most core HTTP servers already order their resources hierarchically in a file system like manner, WebDAV explicitly introduces *collections* as a special resource type for representing and managing groups of resources. This is directly analogous to directories in a file system. The new methods `PROPFIND` and `PROPPATCH` enable a client to query and modify properties of resources in the server's name space. Name space operations such as `MKCOL`, `MOVE`, and `COPY` enable a client to create new collections and to move and copy resources, respectively. WebDAV also addresses the "lost update problem" by introducing the framework of an extensible locking mechanism with the methods `LOCK` and `UNLOCK`.

3 DeltaV: the V In WebDAV

DeltaV[6] is intended to complete the effort of integrating versioning into the Web. DeltaV builds on the infrastructure of HTTP and WebDAV. Its main goal[5] is the support of versioning data of all formats (including binary formats), without overly restricting versioning-unaware clients.

DeltaV uses a resource-based check-in/check-out model to create new versions of a resource from existing ones. It introduces the concept of a *versioned resource* to track all revisions in a history. To create a versioned resource, a client applies the `VERSION` method (in draft 12, `VERSION-CONTROL`) to a non versioned resource that is applicable to be put under version control. The server then replaces the non versioned resource with a newly created versioned resource and creates an initial revision that represents the formerly non versioned resource. Whenever a new revision is created, the server creates a new, unique URL called *stable URL*. A client can access the revision by simply accessing the resource via its stable URL. The URL is called stable because it will not change, even if the associated versioned resource is moved to a different location in the server's name space. Checking out a revision with method `CHECKOUT` creates a server-side *working resource*. A working resource may be manipulated by applying core HTTP methods such as `GET` or `PUT` to retrieve and overwrite the working resource. A `CHECKIN` method applied to a versioned resource with the working resource as argument creates a new revision and usually deletes the working resource. `UNCHECKOUT` cancels a previous `CHECKOUT`.

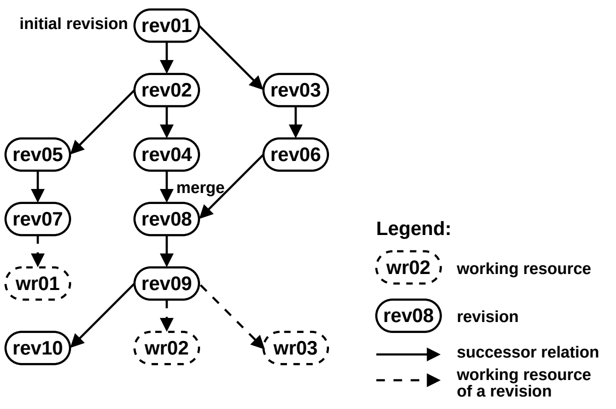

Figure 1: Typical DeltaV revision graph

DeltaV introduces a powerful method called `REPORT` which provides an extensible mechanism for obtaining information about resources. In particular, a *property report* can be used to retrieve properties from a set of resources. While WebDAV's `PROPFIND` operates on resources via the server's name space hierarchy, a property report operates on a set of resources listed in a property of another resource. This enables a client to retrieve properties from a set of resources, such as from all revisions of a versioned resource, with a single request. This is true even if the resources are scattered over the server's name space.

Beyond core versioning, DeltaV also supports advanced versioning features including the management of activities and workspaces, versioned collections and merging of revisions; however this article only studies the core versioning part of the protocol.

4 Putting RCE and VRCE to the task

The prototype client/server implementation described herein uses *RCE*[7], the *Revision Control Engine* as revision control back end on the server side, and VRCE, the *Visual RCE*, as the graphical front end on the client side.

RCE uses a check-in/check-out paradigm to create revisions just like DeltaV. RCE stores and manages an arbitrary number of revisions per file. It organizes the revisions into an ancestral graph, called a *revision graph*, and stores them in a file called an *archive*. It automates the storing, retrieval, logging, and identification of revisions. A revision can be labeled to give it additional names called *aliases*. Thereby, RCE provides selection mechanisms for composing configurations. When a revision is checked out of an archive into a working file for modification, RCE creates a *template*, i.e. an empty revision, as a placeholder for the new revision. This revision will be filled when the work file is checked into the archive. This enables the user to store data about the new revision before the work file is checked in. Figure 2 shows a typical revision graph. RCE works with data files of any format, including, but not limited to binary data. It provides an application programming interface (API) that enables it to be integrated into, or called from, any application.

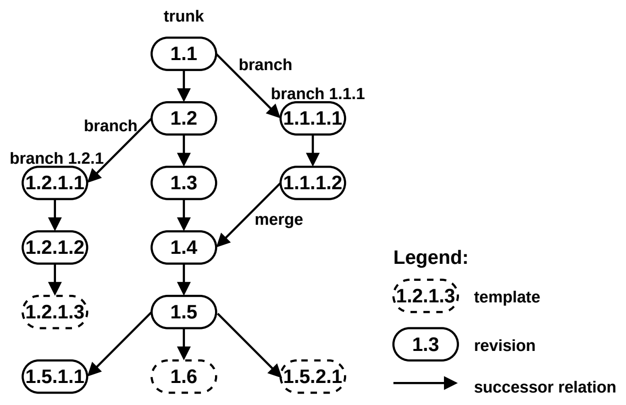

Figure 2: Typical RCE revision graph

VRCE is a graphical front end for using RCE. Rather than calling RCE directly, it defines a *repository interface* that provides a high-level abstraction of the RCE functionality needed by VRCE. The standard distribution of VRCE uses a repository that maps into RCE function either directly or via Java RMI. To create a DeltaV client, it was sufficient to implement a repository that maps VRCE functionality to the DeltaV protocol.

5 Some assembly required

For DeltaV to become widely accepted, the protocol must be adaptable to a wide range of existing versioning control systems and document management clients. It must be simple enough to allow easy implementation of a DeltaV server, even on the basis of a revision control system with limited functionality. On the other hand, it must be complex enough to enable configuration management systems with advanced features such as versioned directories or special handling of workspaces to offer these features to clients in a consistent manner without undue effort. DeltaV addresses this requirement by defining core versioning as a set of minimally required functionality and advanced versioning as a set of optional extensions to the base protocol.

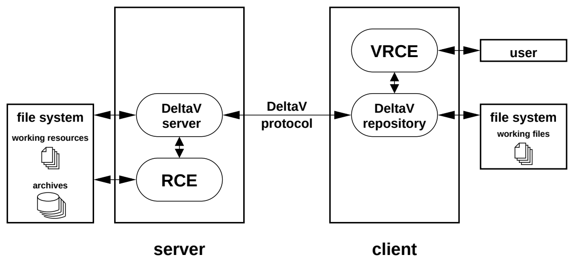

Figure 3: Architectural overview of VRCE via DeltaV

The authors' implementation serves as an example of what difficulties may arise when mapping the functionality of an existing revision control system to the model of DeltaV, and how to solve such difficulties. It turns out, that the functionality of RCE, which served as back end revision control system, roughly matches that of core versioning. While both, DeltaV and RCE, use a check-in/check-out model, there are still some significant differences between their versioning models. Only RCE provides the concept of templates. RCE's archive and revision attributes differ in use from DeltaV's resource properties. Furthermore, while in DeltaV all branches of a revision graph are handled equivalently, RCE's branching concept distinguishes between a main branch and side branches, and it provides special handling for so-called default branches. The branching issue is of special interest, since it will also apply to many other revision control systems such as RCS[11] or ClearCase[2] when they are mapped to DeltaV's versioning model.

RCE's concept of reserving a template on check-out is useful for storing attributes of a revision that has been declared by a check-out operation, but will not be available until the corresponding check-in. A check-out under DeltaV creates a server-side DAV-enabled working resource, whose properties will be saved along with the revision when it is checked in. Working resource properties map naturally to RCE template properties. Thus each working resource can be represented by a file and a template. The only difficulty is that care must to be taken when traversing a revision graph. RCE treats templates just like ordinary revisions during traversal. The server must skip all templates when representing a revision history. On the other hand, the VRCE client can assume the existence of a template for each working resource. This can be done even if another server is used, whose underlying revision model does not include templates. DeltaV enables the checking in of a revision while preserving the working resource. Effectively, this is a check in followed by a check out. Thus, beyond filling the template to become a new revision, the server must create a new template to represent the properties of the new working resource. Similarly, deleting a working resource means deleting a template by applying an uncheck-out. Of course, DeltaV provides a special method called `UNCHECKOUT` as the proper way to delete a working resource, rather than using the core HTTP method `DELETE`.

As mentioned above, branches in RCE are not all equivalent. However, the special handling for the trunk or default branch may be turned off; hence from this point of view, it is not a problem to implement a DeltaV server on the base of RCE. The other way around is a little bit more complicated. To implement a client that uses a DeltaV server as revision control system and mimics RCE's functionality, it must provide additional handling to distinguish between special branches. WebDAV servers should support handling of arbitrary user-defined dead properties. Assuming that the DeltaV server supports this feature on revisions and working resources, a client can mark resources with additional properties. For example, whenever creating a new working resource, the client may add a property to represent its own revision numbering scheme and thus provide a means to distinguish, for example, a trunk from other branches. Note, however, that using non-standard properties can lead to interoperability between different clients that share access to the same resources of the same DeltaV server. Alternatively, a client may evaluate properties, such as those holding the creation date and time of a revision, to decide which branch is the oldest one. This assumes that at each junction work proceeded on the main branch first. This will work unless one starts branching off *before* continuing the current branch. To summarize, a DeltaV client may–within limits–distinguish between branches the way RCE does by providing additional handling. The only clean way to solve this problem is to introduce optional properties into the advanced versioning part of the DeltaV protocol to mark branches.

What remains to be examined is to what extent RCE attributes match DeltaV resource properties. Some RCE attributes, such as those containing the author of, or a comment about a revision, correspond directly to resource properties for the same purpose. Other RCE attributes, such as those describing the predecessor/successor relations between revisions, need a more refined handling to map to the corresponding resource properties. RCE's branching model assumes that at most a *single* successor of a given revision on the same branch; any further successor is marked as a side branch and stored in an *ordered list*. Thus RCE distinguishes between main branches and side branches. Since DeltaV does not distinguish between branches, it models the successor relation as an *unordered set* of successors. Mapping RCE's model to DeltaV's can easily be done by just uniting all successors to a single set. The opposite mapping requires once again either additional handling, such as either marking branches with client-defined properties or evaluating additional properties such as the creation date and time of revisions. As mentioned above, the mapping of the successor relation must also consider that RCE templates match DeltaV working resources. Similarly, RCE provides an attribute to distinguish between a merge predecessor and a predecessor of the same branch while DeltaV handles them in a single set. Once again, additional handling similar to that of the successor handling can be used to uniquely map both models.

Resource type	DeltaV property	RCE attribute
Versioned Resource	DAV:author	ARCH_AUTHOR
	DAV:comment	ARCH_DESCRIPTION
	DAV:revision-set	REV_NEXT, REV_BRANCHES
	N.N.	ARCH_WORK_PATH
	N.N.	ARCH_RCA_PATH
	DAV:creationdate	ARCH_DATE
	N.N.	ARCH_COMMENT_LEADER
	N.N.	ARCH_DEFAULT_BRANCH
	N.N.	ARCH_USERS
	N.N.	ARCH_KEYS
	N.N.	ARCH_MODES
	N.N.	ARCH_LOCK_LEVEL
Revision	DAV:author	REV_AUTHOR
	DAV:comment	REV_DESCRIPTION
	DAV:predecessor-set	REV_PREV, REV_MERGE_FROM
	DAV:successor-set	REV_NEXT, REV_BRANCHES
	DAV:creationdate	REV_DATE_OUT
	DAV:checkin-date	REV_DATE_IN
	DAV:getlastmodified	REV_TIMESTAMP
	DAV:revision-id	REV_NAME
	DAV:revision	N.N.
	DAV:working-resource-id-set	REV_NEXT, REV_BRANCHES, REV_WORK_PATH
	DAV:label-set	REV_ALIASES
	N.N.	REV_STATE

Table 1: Relation between DeltaV resource properties and RCE attributes

RCE supports some attributes for which there is no corresponding DeltaV resource property. As long as the server supports user-defined dead properties, these attributes can be mapped to such dead properties, provided that the attribute is not expected to affect the server's versioning behavior. And, once again, using user-defined dead properties may impact interoperability between different clients. An example of such a property is RCE's state attribute. This stores the state of a revision and may have values like experimental, stable, or released.

DeltaV also does not support keyword substitution. If required, a client will have to do the substitution by itself, for example whenever retrieving a working resource or revision from the server. It may once again use dead properties to store parameters for the substitution process.

The issue of access control is still open. RCE supports access control attributes, but DeltaV mechanisms for resource locking and an access control list extension to WebDAV are still under development. It is not yet clear how this will be resolved in DeltaV. Thus no reasonable mapping can be made at present.

Table 1 summarizes some of the relations between RCE attributes and DeltaV resource properties.

A moving target

Part of the difficulty of prototyping a new protocol is the unavoidable fact that one is shooting at a moving target. Since the acceptance of this article for publication, the DeltaV protocol has undergone a major restructuring. The current draft 12 as of January 20, 2001, differs noticeably from the early 04.5 draft that was used to build the prototype.

A prime goal of DeltaV is to build a protocol that is applicable to a wide variety of revision control and document management applications. There are three models of use that the restructuring addresses: version unaware clients, thin clients with server managed work spaces, and thick clients with client managed workspaces. This was not very clear in earlier versions of the specification. To improve this situation and to make simple implementation possible for document management systems with versioning unaware clients, core versioning has been drastically reduced to the bare minimum of features and instead defines several protocol options that clients and servers may, but need not implement. A client can ask the server for the options that it supports to know what it can use in subsequent requests. There is no longer any notion of an advanced versioning server that supports all options, because not all options combinations are necessarily desirable. For instance, a server that supports baselines has no need for labels.

As of draft 12, core versioning now requires some extensions of WebDAV that do not depend on versioning. These are included in DeltaV as appendix, but will likely become part of a future version of WebDAV. Among these, there is an extension to the HTTP `OPTIONS` method that allows clients to explore the server's capabilities in a more refined way. The generic method `REPORT` enables clients to retrieve information about resources via parameterized requests.

Core versioning essentially introduces the resource types *version-controlled resource* and *version*, the HTTP method `VERSION-CONTROL` and additional semantics for existing HTTP methods. `VERSION-CONTROL` is used to turn any versionable resource into a version-controlled resource. A version-controlled resource acts as a proxy to either the last checked-in version or checked-out modifiable copy of the last version. This and all other versions are accessible via stable URLs that are guaranteed to never change. New versions are created by implicit checkin/checkout via methods like `PUT` and `PROPPATCH` that modify resources or resource properties. There are resource properties defined on versions and version-controlled resources that enable a client to implicitly navigate through the associated version history; clients can also retrieve selected properties from all versions of an implicit version history via a special `REPORT` method.

The advanced versioning part of the protocol now covers many options that formerly were obligatory parts of core versioning. In particular, even *checkout* is explicitly an advanced versioning option, which defines the `CHECKIN`/`CHECKOUT` requests. A version selection mechanism is provided via the `UPDATE` method of the *update* option. The *version-history* option makes the revision graph of a versioned resource available via a URL. Finally, the *label* option lets a user give a revision aliases.

Draft 12 now makes the distinction between client managed workspaces and server managed workspaces clearer. Client managed workspaces can be implemented with the *working-resource* options. Server managed workspaces are specified by the *workspace* option.

There are several options of the old advanced versioning that complete the set of options. The *merge* option provides a means of recombining development paths. The *baseline* and *version-controlled-collection* options add configuration management capabilities to DeltaV. The *activity* option is used to denote a set of versions on a single line of descent, possibly on multiple version histories; activities may be used to manage change sets and branches. The *fork-control* option enables a client to control when and where branches in a version history may occur. Finally, the *variant* option is provided to help manage alternative versions of a document.

Although these options are not fully independent from each other in the sense that some options affect the behavior of others, a server may support almost any subset of these options. The only exception is that the workspace option requires the version history option. In general, the new organization is easier to understand.

6 Cracks in the foundation

While implementing the DeltaV server and the client-side repository, several difficulties showed up. Most of the difficulties pertain to WebDAV. Resolving some of these appropriately would require changes not only to DeltaV, but to WebDAV as well.

In the Internet community, a protocol must go through a series of states or maturity levels before it becomes a standard, namely *Proposed Standard*, *Draft Standard* and, finally, *Standard Protocol*. The process is described in detail in the *Internet Official Protocol Standards* document which is frequently re-issued as an RFC, e.g. as RFC 2400[8]. The standardization process can last for a long time. As the history of the HTTP protocol shows, it may span several years. Although first practical experiences with the WWW date back to 1990, HTTP became a Proposed Standard, the lowest maturity level in the standardization process, not earlier than in 1997, when HTTP/1.1 was issued as RFC 2068.

The first work on WebDAV is documented as RFC 2291, which was issued in early 1998 as informational RFC. In early 1999, less than a year later, WebDAV already became a Proposed Standard (RFC 2518). Though WebDAV concentrates on only a few aspects of client/server interaction, such as resource properties, collections, locking, and name space operations, the time frame seems quite short. Indeed, the list of open issues on WebDAV[4] has grown dramatically since its publication as Proposed Standard. The quickly growing number of implementations that claim to support WebDAV will make it difficult to incorporate major changes in future versions of the protocol.

What the authors regret most about WebDAV is that the protocol does not use valid XML, but only well-formed XML. At the same time, WebDAV still defines a document type declaration, whose primary use, a reader of the protocol could expect, is to validate XML documents. Discussions on the WebDAV working group mailing list[3] came to the conclusion that the use of validating XML seems to be possible with only minor modifications of the protocol. However, most participants did not agree with the authors on the need of using validating XML. Indeed, an error in the document type definition, as well as in an XML example of the protocol, has been detected by partial validation. This clearly demonstrates a benefit of validation. Another benefit is improved interoperability, because faulty implementations of servers or clients will be detected sooner. This does not mean that all WebDAV implementations *must* validate XML; but the protocol should be designed in such a way that an implementation *may* validate XML. However, the current version of the WebDAV protocol uses invalid XML and thus makes compliant validation of XML impossible. The authors' prototype implementation uses an XML parser that checks for well-formedness only; approximate but not fully compliant validation is performed manually on a "best effort" strategy within the limitations of the protocol.

7 Conclusion

The authors succeeded in implementing a core versioning DeltaV server and client prototype as of DeltaV draft 04.5 with the discussed limitations, thus proving the practicability of the protocol. This process has been useful both for providing input to the development of the protocol and outlining problems with the base protocol WebDAV.

WebDAV provides a great amount of the infrastructure that DeltaV is based upon. Still, it suffers from a tremendous (and yet growing) amount of known and documented issues. Some of these issues can be addressed by extending the protocol. Other issues will be difficult to change because they would introduce incompatible changes to the protocol. What makes this particularly difficult is the quickly growing number of commercial WebDAV implementations. This is unfortunate because the changes needed to make the XML used by the protocol valid will probably not be accepted by the WebDAV community. Though validation need not be done on every transaction, it provides a convenient method of testing both implementations and the protocol definition. Some thought should be given to methods for upgrading protocols, like requiring some sort of version header, and to how XML should be used in IETF protocols.

The task of cleanly mapping the functionality of RCE as an example of a branching model version control system turned out to be reasonable. The DeltaV property report turned out to be a key feature for efficient property handling on versioned resources, when operating on many revisions or working resources. This capability has become a DeltaV suggested extension to WebDAV. Problems such as the ordering of branches, can be solved by introducing proprietary resource properties, but this is undesirable because it makes interoperability with other clients that share the same DeltaV server name space difficult. The authors are active participants in the DeltaV working group and are working to resolve these issues. Future extensions to WebDAV, like the ACL specification, may eliminate the need for user-defined properties and thus enhance interoperability between different clients that share the same versioned data on the same DeltaV server.

DeltaV is a sensible and practical extension to WebDAV. The main difficulty in designing and testing the protocol is the wide variety of version systems that need to be supported. It remains to be seen, whether or not the protocol is significantly well described, so that a separately developed clients and server will work together.

8 Future work

The current prototype is limited to core versioning of draft 04.5. DeltaV provides a number of interesting features for advanced versioning which this study does not consider. The latest draft has also changed the conceptual basis for core versioning to a minimal system suitable for document management systems. Everything else has been moved to options. More investigation is needed to examine the practicability of DeltaV's options and their interaction. Efficiency problems of the very early prototype currently reveal weaknesses in its design and implementation rather than weaknesses of the protocol. More work on the prototype is needed to provide a better insight into the efficiency and effectiveness of DeltaV. The ultimate test will be to test this client and server against someone else's server and client respectively.

REFERENCES

[1] *ISO 8879:1986(E): Information processing—Text and Office Systems—Standard Generalized Markup Language (SGML)*. First edition – 1986-10-15 edition, 1986.

[2] Rational ClearCase – Featuring Unified Change Management D-707a. WWW, Jan. 2000. http://-www.rational.com/products/clearcase/prodinfo/-media/ucm_ds_feb8.pdf.

[3] E. Whitehead et. al. IETF WEBDAV Working Group Home Page. WWW. http://www.ics.uci.edu/pub/ietf/-webdav.

[4] E. Whitehead et. al. WebDAV Distributed Authoring Protocol Issues List. WWW. http://www.ics.uci.edu/-pub/ietf/webdav/protocol/issues.html.

[5] E.J. Whitehead, Jr. Goals for a Configuration Management Network Protocol. volume 1675: SCM-9 Workshop, pages 186–203. Springer, Sept. 1999.

[6] G. Clemm and C. Kaler. Versioning Extensions to WebDAV. WWW, Feb. 2000. http://-www.ics.uci.edu/pub/ietf/webdav/versioning/draft-ietf-deltav-versioning-04.5.txt.

[7] J. J. Hunt and W. F. Tichy. *RCE API Intro. and Ref. Manual*. DuraSoft, 2000.

[8] J. Postel and J. Reynolds. RFC 2400: Internet Official Protocol Standards. WWW, Sept. 1998. http://-www.ietf.org/rfc/rfc2400.txt.

[9] R. Fielding, et al. RFC 2068: Hypertext Transfer Protocol – HTTP/1.1. WWW, Jan. 1997. http://-www.ietf.org/rfc/rfc2068.txt.

[10] T. Bray and J. Paoli and C.M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. WWW, Feb. 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[11] W. F. Tichy. RCS — a system for version control. *Software—Practice and Experience*, 15(7):637–654, July 1985.

[12] Y. Goland and E. Whitehead and A. Faizi and S.R. Carter and D. Jensen. RFC 2518: HTTP Extensions for Distributed Authoring – WEBDAV. WWW, Feb. 1999. http://www.ietf.org/rfc/rfc2518.txt.