Workgroup Middleware for Distributed Projects

Gail E. Kaiser and Stephen E. Dossick
Columbia University
Department of Computer Science
1214 Amsterdam Avenue, MC 0401
New York, NY 10027 USA
212-939-7000/fax:212-939-7084
{kaiser, sdossick}@cs.columbia.edu

ABSTRACT

We have developed a middleware framework for workgroup environments that can support distributed software development and a variety of other application domains requiring document management and change management for distributed projects. The framework enables hypermedia-based integration of arbitrary legacy and new information resources available via a range of protocols, not necessarily known in advance to us as the general framework developers nor even to the environment instance designers. The repositories in which such information resides may be dispersed across the Internet and/or an organizational intranet. The framework also permits a range of client models for user and tool interaction, and applies an extensible suite of collaboration services, including but not limited to multi-participant workflow and coordination, to their information retrievals and updates. That is, the framework is interposed between clients, services and repositories — thus "middleware". We explain how our framework makes it easy to realize a comprehensive collection of workgroup and workflow features we culled from a requirements survey conducted by NASA.

KEYWORDS: Distributed software development support, distributed document management, monitoring and managing distributed development processes, distributed change management, workflow management and coordination support in distributed projects, Internet-based software process coordination

INTRODUCTION

We have developed a middleware framework for distributed workgroup environments founded on what we call a *referential hyperbase* paradigm for representing and hyper-linking the information resources of interest to a collaborating team intent towards some joint purpose or goal (as opposed to incidentally browsing the same materials for independent use, such as most World Wide Web users). A referential hyperbase differs from a conventional hyperbase in that the documents need not reside in the system's own repository. A referential hyperbase differs from a linkbase or link server in providing full object-oriented database functionality including a query language and entity typing.

The referential hyperbase system communicates with a variety of local and remote information repositories, each through its original protocol. User clients and tools interact with the hyperbase in client/server fashion, also each through their appropriate protocol. The hyperbase may contain both *native* objects, internal to the system, and *protocol* objects, representing external documents residing in one of the repositories. Native objects are retrieved and presented to clients entirely through the hyperbase's own object management facilities, but accessing the contents of protocol objects generally involves forwarding the appropriate operation(s) to their home repositories and transmitting the results through the hyperbase back to the originating client. An extensible set of collaborative work support services are performed explicitly in response to requests from clients and/or implicitly upon accesses to objects, independent of the protocol(s) used. These *groupspace*, or group workspace, services may send and receive requests to/from each other as well as through the protocols to backend repositories and frontend clients.

Our system does not hardwire any particular protocols for interacting with backend repositories or frontend clients. Instead, backend access protocols provide means to read and write the documents required by the workgroup, and additional predicates and actions may expose the capabilities of specific legacy and new information resources. Frontend presentation protocols tailor our system so that it appears to

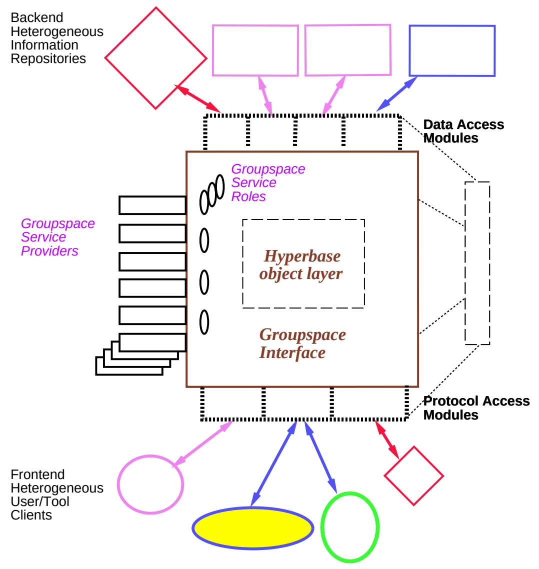
Backend Heterogeneous Information Repositories
Data Access Modules
Groupspace Service Roles
Groupspace Service Providers
Hyperbase object layer
Groupspace Interface
Protocol Access Modules
Frontend Heterogeneous User/Tool Clients

Figure 1: Framework Architecture

clients as if they are communicating with their own information server. Thus users can continue using their familiar legacy system commands, if desired, while still receiving the benefits of the multi-protocol workgroup framework.

We are currently rethinking and reimplementing workflow and transaction-based coordination components previously integrated with our earlier, single-protocol (HTTP) referential hyperbase system targeted specifically to distributed software development projects [11]. We have also ported and continue to develop that system's distributed tool launching component [12]. [9] describes our experience using the earlier HTTP-only system for our own continuing software development efforts. Our new multi-protocol framework is introduced in [10]. Here we briefly discuss the new framework and our initial implementation, and then focus on how it enabled us to rapidly fulfill most of the generic workgroup requirements adapted from the publicly available results of a survey conducted within NASA in 1995 [8].

REFERENTIAL HYPERBASE

Figure 1 illustrates the middleware architecture. Each frontend client communicates with the referential hyperbase server in the client's own native protocol, which is understood by the corresponding *Protocol Access Module* (PAM). The referential hyperbase server retrieves and updates documents from/to each backend repository using that repository's native protocol, through the corresponding *Data Access Module* (DAM). The PAMs and DAMs are written specially for each client and repository, respectively, as "plugins" with respect to the hyperbase system, and are dynamically loaded as needed. Groupspace services are also interfaced to the hyperbase server via plugins. The hyperbase system itself consists of four main components: an object layer; an underlying object management system (OMS), not shown, that realizes the object layer; a data schema that may be extended by DAMs, PAMs and groupspace services; and an API for writing DAMs, PAMs and the service interfaces.

The data schema predefines a `DAM` base class, which is then extended by each new DAM. In general, a DAM treats its protocol object instances as "stubs" for *external* documents — thus the referential nature of the hyperbase. These documents might reside anywhere on the Internet and/or organizational intranet. A protocol object might reflect any granularity, from an individual tuple in a relational database, to a GIF image, to an MPEG clip, to an entire website hierarchy or database management system, to another hypermedia server, or even to another workgroup environment. Different protocol objects might represent the same document in different ways or at enclosing granularities, e.g., a URL pointing to a named fragment of an HTML file and a URL referring to the whole file can co-exist in the same hyperbase. Arbitrary application-specific *meta-data* and *links* can be represented by the attributes of protocol objects, independent of any links embedded in their contents and any meta-data maintained by their home repositories. Such attributes can be used, e.g., to represent dependencies utilized during change propagation.

Every DAM necessarily publishes a `read` method and usually also a `write` method; e.g., in the case of HTTP, these would perform GET and PUT functions, respectively. Reading a protocol object from the hyperbase generally results in retrieving the document content from its repository and sending that content, along with application-specific meta-data and links from the hyperbase, to the appropriate client. A client may modify the content and write the protocol object, which typically results in updating the corresponding repository. Optional methods may provide type-specific or repository-specific functionality for particular protocol object subclasses. For example, printable materials might be associated with a `print` method, and a protocol object corresponding to an Oracle or Sybase database might provide an SQL query interface. Any of these methods may potentially employ the object layer as a persistent object store, e.g., to construct auxiliary objects as children of some protocol object.

One or more attributes of a protocol object designate the location of the corresponding external document. For example, this might be a string attribute indicating the URL for a World Wide Web entity. Other attributes, also interpreted by the class's DAM, locally cache the document content; what constitutes content of course depends on the type. For a WWW entity this might be an HTML file or a GIF image, and for another collaboration environment the content might be anything from a list of active users to a "to-do" list of pending tasks to its events registrar.

GROUPSPACE SERVICES

A groupspace operates on top of a referential hyperbase to actively support workgroup tasks, most significantly but not solely through workflow, rather than simply providing passive access to documents. Groupspace functionality is not specific to any particular protocol(s), but inserted independently of the mix of repositories and clients. However, a groupspace service might internally perform distinct actions for different document classes; for instance, in a software development environment, an editor reading, and possibly later writing, a prose functional specification would probably be treated as a completely different task, with different workflow constraints, than a compiler reading source code in order to generate and write executables.

Groupspace services are divided into two aspects: providers and roles. *Roles* are standard interfaces for those services "expected" to be provided by a particular workgroup environment. Note different environment instances may define different roles, or define what are in essence the same roles differently. These interfaces may be employed by the referential hyperbase system itself, as well as by other groupspace services and by PAMs and DAMs. For example, conventional concurrency control and failure recovery is required by all multi-user hypermedia systems [17], and thus must be realized by some role if not built into the underlying OMS.

Providers are independent components, connected to the hyperbase system via plugins, that each realize some service such as workflow automation, concurrency control and recovery, tool management, intelligent search, guided tours, and so on. Providers register callbacks implicitly invoked by accesses to instances of particular classes (and their subclasses), or by accesses to any object. Providers may also publish menus whose entries may be explicitly invoked. Exactly one provider may register as the default implementation of a role. It is not necessary to know anything about the provider that fulfills a role in order to use the standard facilities for that role, since the referential hyperbase's API is extended with these operations (for that environment instance).

The referential hyperbase system triggers groupspace service roles, and corresponding providers, in two basic modes, which we call *wrap-around* (or implicit) and *direct* (or explicit). In either case, the services are independent of the particular communication protocols used. The wrap-around mode involves activating the particular service both *before* and *after* each request from the hyperbase to a DAM and hence a backend information repository.

For example, a workflow service might treat reading a protocol object as a primitive task, and check preconditions before the operation's execution and assert postconditions afterwards. Plausible preconditions might include the user's access rights to the document as well as his/her authorization and responsibilities with respect to the enclosing composite task, the status of all inputs (have prerequisites been fulfilled?), and so on. Possible postconditions might include checking deadlines for production of outputs (have implications been fulfilled?), notification of supervisors, routing to another user for postprocessing as another task, etc. The workflow engine itself does not need to know whether the actual task reflected by the protocol object read involves invoking an X.509 authentication service, retrieving a Web page, or submitting an SQL query.

Wrap-around services are *stacked*, meaning a specific order among services is configured for the before callbacks, and the reverse order is employed for the after callbacks. Multiple providers for the same role may participate in the stack without awareness on the part of clients. In contrast, clients cannot take advantage of direct services except through the standard interface of a role, or by requesting certain methods of a named provider — requiring some knowledge of that provider's functionality.

Direct mode assumes that each service exposes a *menu* of activities (no visual interpretation is implied) whose entries then can be explicitly selected. The simplest example is a list of tools handled by a tool launching service. Note that while a given tool probably needs to know the format of its input and output data, the launching service does not; they can be passed as BLOBS (binary large objects). Another possibility is a workflow service's set of tasks; some of those tasks may correspond exactly to reads and writes of individual protocol object instances, as above, but others may be significantly more complex and involve sophisticated computations on perhaps hundreds of objects, e.g., marketing and requirements analysis for revamping the "look and feel" of an existing product line.

User clients might be developed specifically as an adjunct to the middleware framework, with their PAMs speaking the "language" consisting of its published terms — from the standard API plus the roles specific to a given environment instance. In contrast, the PAMs for legacy presentation protocols convert or translate to the features provided by the workgroup environment, and vice versa, in a manner meaningful for the relevant legacy system. Which services and which menu entries are made available to a particular client depends on the PAM. A sophisticated PAM might filter a menu to make available only some of its entries only under certain circumstances, supply details about the expected parameters, and express the menu entries in a way meaningful to its users.

IMPLEMENTATION

Our referential hyperbase server, called **Xanth**, is implemented as a Java application and runs on any platform that supports Java 1.1. Its Java graphical user interface client runs as an applet in Java-enabled browsers, which must be configured to use our special HTTP proxy (which may be cascaded with conventional caching or firewall proxies and/or with other special-purpose proxies).

We use ObjectStore's Persistent Storage Engine (PSE) as the OMS. Xanth's object layer supports the usual hierarchical navigation and addition/deletion of objects, and provides an OQL subset as its query language (PSE does not supply its own query language).

PAMs are explicitly registered; DAMs are implicitly registered via inclusion of their protocol subclass in the object layer's data schema. Every PAM runs in its own Java thread and is responsible for handling any network communications required by the protocol. DAMs are woken up only when needed, though they may spawn their own background threads during initialization. Xanth's API thus includes atomic operations where warranted, to protect shared data from concurrent access, since PAMs, DAMs and groupspace service providers can access the OMS through the object layer. All our DAMs currently follow the JavaBeans conventions, and we plan to soon convert all the PAMs and service plugins. Groupspace service roles are implemented as Java interfaces.

Xanth does not hardwire any particular groupspace services, but a particular Xanth workgroup environment called **OzWeb 1.0** defines three roles, WorkflowManager, TransactionManager and ToolMediator, with one service fulfilling each role. Details are outside the scope of this paper; see [10].

EVALUATION

The italicized capabilities listed below are paraphrased from the NASA workgroup/workflow requirements survey. We adopt these as generic goals for our middleware framework, independent of any particular application, but most of them are obviously useful for distributed software development. We respond to each requirement with a brief description of how Xanth makes it relatively easy to realize.

Cross-platform support, covering Unix workstations, PC Windows, Macintoshes, and pen-based PDAs: Both Xanth clients and servers run on any platform supporting Java 1.1. We've tested Windows, Solaris, Linux and Irix. MacOS should work, too, but Apple had just released their Java 1.1 support as of this writing. We've developed partial client support for the Newton MessagePad 2000 that exploits its native "to-do" list protocol: A PAM listens to the machine's serial port for the Newton "hotsync", accepts updates indicating completed assignments, and sends a zip-like file with new work assignments and their `ASCII` parameters. A corresponding DAM defines Newton-specific "to-do" lists as a subclass of the `todoList` base class, in turn a subclass of the `DAM` class. A shell script invokes a utility to convert the `todoItem` children to/from "newton" and stores/retrieves them in the proper synchronization subdirectory.

Access via email, HTTP and dialup modem: A mail filter program captures email to a Unix userid and resends the contents of the message to the Xanth server in HTTP format; responses are similarly converted back from HTTP to email. Another HTTP PAM handles conventional WWW-style requests and responses, and the `WebObject` DAM communicates with backend websites via HTTP. Whether email or HTTP access is appropriate for dialup depends on the modem speed.

Reliable, secure and authenticated data transmission and storage: This could be handled separately by every concerned PAM and DAM, but it would be better to provide X.509 authentication, public-key encryption and so forth as a wraparound groupspace service automatically applied to all data accesses and updates as well as to all requests for direct services. This feature has not yet been implemented.

Support for off-line work and merging: `Checkout` and `Checkin` operations are transmitted by clients via HTTP to the component fulfilling the TransactionManager role in OzWeb 1.0, to place and remove persistent locks permitting off-line work in the meantime without need for merging — provided that all other users of the relevant backend repository(ies) also update it only through the same environment instance. However, if the repository itself supports the checkout model or long-duration locks, e.g., a WEBDAV-compliant website [15], then its corresponding methods would be exposed through the DAM for use by the TransactionManager provider in enforcing the concurrency control policy. If a non-exclusive form of `Checkout` where used instead, then either the DAM or the backend repository itself would need to provide the merging utility, which is inherently specific to the data format and semantics. Our transaction manager component is presented in [18].

Version Control: The TransactionManager or some other groupspace service(s) could implement versions, but again those versions would only be accessible through Xanth and the backend repository would only contain the latest variant. The `CVSPackage` DAM, and its auxiliary `CVSFile` DAM for mirroring the files from a CVS directory within the hyperbase, support an example of a backend repository that itself provides versioning [13]. We're working on another DAM that monitors (polls) the CVS global log for changes and queues up these events, which could trigger workflow and/or provide input to a notification service.

Addressing, routing and tracking. Directory services: It is not clear to us what NASA had in mind here. One form of routing and tracking is provided by the WorkflowManager component. A general tracking utility could be provided by another groupspace service for logging whatever is appropriate on behalf of all the other services (and DAMs and PAMs). LDAP [19] and other directory services could be searched via corresponding DAMs.

Personal and shared calendars with meeting and resource scheduling: Our `calendar` base class, a PAM, defines common functionality for scheduling systems. As an example, the `icalSchedule` PAM represents the individual and group calendars implemented by the public domain *ical* scheduling tool [7].

To-do items that are prioritizable, assignable to others, and trackable for status: The generic `todoList` PAM class defines a common representation for the "to-do" lists of various tools. For instance, the "to-do" items stored in *ical* calendars are automatically forwarded to the next day until deleted or marked as "done".

Integrated project files for email, calendar, to-do, etc., with data enclosures including BLOBS: The integration is provided through the Xanth hyperbase. Arbitrary protocol objects can be attached via links to any other protocol objects. One protocol object can be enclosed with another when transmitted back to clients either as just the OID for later access, or as contents and attributes.

Threaded dialogues: Our NNTP PAM provides access to Internet and intranet news servers, and represents articles accessible via these servers. Threaded articles may be linked to each other through the hyperbase. Since the cross-article links are implemented by the hyperbase layer, it is possible to make links between articles in different discussion groups or on different news servers independent of the underlying usenet threads.

Chat/conference: We're implementing the proposed Rendezvous Protocol (RVP), an IETF Internet Draft [5], to support scalable instant messaging (one-line messages) and chat (real-time text conversations) among users of the same or different Xanth environments. Each RVP server will operate as a Xanth module (either groupspace plugin or PAM), and the clients as JavaBeans, which can be incorporated into JDK1.1-compliant Web browsers or other tools.

Video conferencing: This could potentially be realized either through a stream-oriented DAM/PAM pair or by a groupspace service. It is not clear, however, what it would mean to apply wrap-around services to a video conference, or whether the real-time requirements could be satisfied if the video stream passed through the hyperbase system. It might be better for the ToolMediator to fork the video application separately.

Compatibility with current desktop software, and access to desktop and legacy data. A DAM could be constructed to extract/insert data from/to ActiveX applications. A PAM implementing the ActiveX Container standard could pop up the appropriate application when such data is accessed by clients. This has not yet been implemented. However, the ToolMediator service already runs arbitrary Windows applications on the user's PC, or on a server machine and export the display to the user's Unix workstation via WinDD.

Group whiteboards, editors and drawing tools: There's nothing particularly special in Xanth about multi-user versus single-user tools. We have previously developed session management and workflow support for synchronous what-you-see-is-what-I-see groupware, see [14, 3], but have not yet incorporated those particular functions into the new groupspace components used with Xanth.

User-friendly. Sophisticated data entry, access and warehousing facilities: Existing legacy clients, whether deemed to be particularly "user friendly" and sophisticated, or not, can continue to be used through construction of appropriate PAMs. And of course new clients can be developed, either for the workgroup framework in general, or for some legacy information resource, and also connected to Xanth via a PAM. And analogously for DAMs with respect to legacy and new repositories. The generic approach to groupspace services permits, e.g., introduction of data mining independent of the information source.

Ability to integrate additional facilities using standard protocols. Enhanceable with respect to interfaces and third party products. These are precisely the main contributions of our multi-protocol architecture. It took the second author four hours to complete the CVS DAM and six hours for the NNTP PAM. Another member of our lab, who was not directly involved in the development of Xanth except to interface the TransactionManager component, was able to add HTTP `checkin` and `checkout` operations in about an hour. Not counting Xanth and the Oz 1.0 roles and providers, development of all the workgroup facilities indicated as completed above was done in two weeks.

Scalable to thousands of users while remaining inexpensive and supportable with a small staff: Because Xanth and its ToolMediator provide a range of mechanisms for running remote tools and otherwise operate as middleware for network-based computing, fewer applications will need to be maintained on personal computers, hence administration costs should be reduced. We have some preliminary ideas about how to organize "alliances" of multiple Xanth servers to scale up to teams of teams, based on our previous work on Oz [2], but this requirement will have to wait for future work.

We also developed a Tcl scripting console as a PAM, so one can telnet to a port on the machine running the Xanth server and debug by directly calling service plugin, DAM and PAM methods. The facilities above were tested in this way, rather than constructing full-blown graphical user interfaces.

CONCLUSIONS

The Open Hypermedia Protocol (OHP) [6] is being negotiated in the hypertext community so that viewers for one hypermedia system can be used with any other compliant system. Writing a PAM for a given protocol seems substantially simpler and quicker than writing or adapting a viewer to speak OHP, and analogously for DAMs and repositories. Obviously, a PAM can be developed for OHP so that OHP-compliant clients can be utilized. We do not intend to imply that Xanth qualifies as an open hypermedia system, since we do **not** intend to build our own viewers, but instead that mixing and matching of legacy hypermedia systems can also be achieved by integrating them with Xanth via PAMs and DAMs — at the same time automatically enhancing them with the available groupspace services.

A variety of collaboration environments have been developed on top of the World Wide Web infrastructure, each providing some of the workgroup facilities presented above, e.g., the Virtual Collaboratorium [1] and BSCW [4]. But these are generally specific to HTTP, and do not support other protocols, while most hypermedia systems support their own "proprietary" protocol, possibly augmented with OHP.

Our approach is closer to that of the Hyperform framework for hypermedia system development [16]: We provide a framework around which sophisticated, protocol-independent hypermedia collaboration environments may be designed, much as Hyperform provides a rich set of hypermedia abstractions in its framework. We do not focus, however, on development tools like those provided in Hyperform. Instead, we concentrate on providing support for development of multiple protocol interfaces, giving workgroup environments built with our toolkit access to, as well as allowing access from, many different legacy information systems.

ACKNOWLEDGEMENTS: We would like to thank Jingshuang Jack Yang, Wenyu Jiang, and Xiaoning Zhang for their work with the authors on OzWeb. Discussions with members of Dick Taylor's group at the University of California at Irvine have also contributed substantially to this research.

This effort is sponsored in part by the Defense Advanced Research Projects Agency, and Rome Laboratory, Air Force Materiel Command, USAF, under cooperative agreement F30602-97-2-0022.

REFERENCES

1. Pete Beckman, Todd Green, and Juan Villacis. The virtual collaboratorium. http://www.extreme.indiana.edu:80/pseware/vc/overview.html.

2. Israel Ben-Shaul and Gail E. Kaiser. *A Paradigm for Decentralized Process Modeling.* Kluwer, 1995. http://www.wkap.nl/kapis/CGI-BIN/WORLD/book.htm?0-7923-9631-6.

3. Israel Z. Ben-Shaul and Gail E. Kaiser. Integrating groupware activities into workflow management systems. In *7th Israeli Conference on Computer Systems and Software Engineering*, pages 140–149. IEEE Computer Society Press, June 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-002-95.ps.Z.

4. R. Bentley, W. Appelt, U. Busbach, E. Hinrichs, D. Kerr, K. Sikkel, J. Trevor, and G. Woetzel. Basic support for cooperative work on the World Wide Web. *International Journal of Human Computer Studies*, Spring 1997. http://bscw.gmd.de/Papers/IJHCS/IJHCS.html.

5. Martin Calysn. Rendezvous protocol, November 1997. Internet Draft. http://ds.internic.net/internet-drafts/draft-calsyn-rvp-00.txt.

6. Hugh Davis, Sigi Reich, and Antoine Rizk. OHP – Open Hypermedia Protocol working draft 2.0, June 1997. http://diana.ecs.soton.ac.uk/ hcd/ohp/ohp.htm.

7. Sanjay Ghemawat. Ical. http://www.research.digital.com/SRC/personal/Sanjay-_Ghemawat/ical/home.html.

8. Marion Hansen. Data from Workgroup/Workflow requirements survey for NASA, October 1995. http://server-mpo.arc.nasa.gov/work/intro.html.

9. Wenyu Jiang, Gail E. Kaiser, Jack Jingshuang Yang, and Stephen E. Dossick. WebCity: A WWW-based hypermedia environment for software development. In *7th Workshop on Information Technologies and Systems*, pages 241–245, December 1997. Poster paper. ftp://ftp.psl.cs.columbia.edu/pub/psl/wits97.ps.gz.

10. Gail E. Kaiser and Stephen E. Dossick. Distributed hypermedia collaboration environments supporting legacy protocols. Technical Report CUCS-003-98, Columbia University Department of Computer Science, January 1998. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-003-98.ps.gz.

11. Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, and Jack Jingshuang Yang. An architecture for WWW-based hypercode environments. In *1997 International Conference on Software Engineering: Pulling Together*, pages 3–13, May 1997. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-037-96.ps.gz.

12. Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, Jack Jingshuang Yang, and Sonny Xi Ye. WWW-based collaboration environments with distributed tool services. *World Wide Web Journal*, 1998. In press. Available as Columbia University Department of Computer Science, CUCS-003-97, February 1997, ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-003-97.ps.gz.

13. Pascal Molli. CVS bubbles. http://www.loria.fr/ molli/cvs-index.html.

14. Giuseppe Valetto and Gail E. Kaiser. Enveloping sophisticated tools into process-centered environments. *Journal of Automated Software Engineering*, 3:309–345, 1996. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-022-95.ps.gz.

15. Jim Whitehead. IETF WEBDAV working group: World Wide Web Distributed Authoring and Versioning, May 1997. http://www.ics.uci.edu/ ejw/authoring/.

16. Uffe K. Wiil and John J. Leggett. Hyperform: A hypermedia system development environment. *ACM Transactions on Information Systems*, 15(1):1–31, January 1997. http://www.daimi.aau.dk/ kock/Publications/Hyperform/-TOIS97.ps.gz.

17. U.K. Wiil and J.J. Leggett. Concurrency Control in collaborative hypertext systems. In *5th ACM Conference on Hypertext*, pages 14–24, November 1993. http://www.daimi.aau.dk/ kock/Publications/pubs/Hypertext93.ps.gz.

18. Jack J. Yang and Gail E. Kaiser. WebPern: An extensible transaction server for the World Wide Web. Technical Report CUCS-004-98, Columbia University Department of Computer Science, January 1998. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-004-98.ps.gz.

19. W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol, March 1995. Network Working Group Request For Comments: 1777, http://andrew2.andrew.cmu.edu/rfc/rfc1777.html.