Distributed Configuration Management via Java and the World Wide Web

James J. Hunt, Frank Lamers, Jürgen Reuter, and Walter F. Tichy *

University of Karlsruhe, Karlsruhe, Germany

Abstract. The introduction of Java has been heralded as a revolution in network computing. Certainly, machine and operating system independent applets flittering through the Internet promised to jazz up web surfing; but could they be used to advantage for distributed computing? The authors had encountered substantial problems in implementing a distributed revision control system, called WWRC, based on passive Web browsers. Java seemed to offer solutions to these problems. To this end, the authors have developed WWCM, a successor to WWRC written in Java. WWCM extends the concepts of WWRC to distributed configuration management by using CME—a new configuration management API. WWCM demonstrates that most of the design difficulties encountered with WWRC can be solved with Java. Furthermore, WWCM offers a test bed for a configuration management paradigm called *template regulated alternative development.*

1 Introduction

There has been a flurry of development pertaining to the Internet in the last few years. Two innovations have led to this surge of interest in the Internet: the World Wide Web (WWW) and Java. The WWW provides a means to access information easily from anywhere on the net. Java allows programs to run anywhere on the net. It is now significantly easier to distribute information on the net than it was just two years ago.

The authors have been investigating the application of the WWW to software development for a similar period of time. Though systems exist for distributed revision control, e.g. ClearCase[3], the WWW offers a "lightweight" alternative to such systems. To this end, the authors developed a WWW based revision control system, called World Wide Revision Control (WWRC[21]).

WWRC provides revision control over the WWW. In order to access an archive on a remote machine, one need only install a Web browser such as Netscape™ and a small helper application. WWRC uses the HTTP basic authentication scheme, so the user need no system access beyond the HTTP connection with the server. WWRC demonstrates that the WWW can be used as a distribution base for software management tools.

* The authors addresses are `(jjh|lamers|reuterj|tichy)@ira.uka.de` respectively

Like all prototypes, WWRC was designed to prove a concept. It illuminates not only the strength of using the Web for such tools, but also its weaknesses. The authors will demonstrate that many of these weaknesses can now be eliminated.

Based on the experience with WWRC, the authors have built a new system, called World Wide Configuration Management (WWCM). By combining the strengths of the WWW, Java,™ and a new configuration management API, the authors will show that practical distributed software development tools can now be produced that are independent of the underlying hardware and operating system. Specifically, WWCM provides distributed configuration management over standard Java capable Web browsers.

2 New Kid on the Block: The Problems

Though the concept behind WWRC is sound, its implementation has significant drawbacks. Since WWRC was written before Java was widely available, it relied on HTML to describe the entire user interface. In addition, to test the concept quickly, the server for WWRC was written as an *inetd* client. Furthermore, the scope was limited to revision control, rather than supporting full configuration management. These decisions proved to be too restrictive.

2.1 Static Format for Dynamic Information

HTML was designed to display relatively static data. On the other hand, the WWRC server must provide dynamic data each time the user makes a request that affects his view of WWRC. The server could only do this by generating a new HTML page for each request. The constant interaction between browser and server has two negative side effects. The first is that a response from the server is required for each and every action, so the system becomes unusable for connections with long round trip times to the server. The second is that the user's browser fills up with pages, many of which have become invalid due to subsequent operations. Thus the normal forward and backward commands in the browser are virtually useless. To solve the second problem, a page time-stamp scheme needed to be implemented to prevent the user from accessing pages that have become invalid.

2.2 Passive Interface

Since the browser can only act in response to user requests by sending a command to the server, a helper application is needed to assist the browser in file transfer for check-in and check-out. Though Netscape has a facility to upload and download files to and from a server, these could not be used. They would have required the user to manually specify from whence the file should be read or to where the file should be written. No additional processing could be done. Even with a helper application, the multi-part message protocol needed to be abused in order to start the helper and still be able to give a response to the user. Since

the server can only provide one response to a request from the browser, the first sub-message was used to start the helper application and the second to notify the user of the result returned by the helper application via the server. This means that the second part could only be sent after the helper was finished. Extra delay must be inserted between parts of the message. This delay could lead to timeouts on slow connections and/or large data transfers. Even though the authors had full control of the server, they could not alleviate these problems, because Web browsers are not designed to accept unsolicited messages from Web servers.

2.3 Platform Dependent

The WWRC server was implemented as an *inetd* client. Though an *inetd* implementation was quick to implement, it had three main deficiencies: portability, efficiency, and coordination. Since *inetd* is Unix specific, the server would need to be modified to run under other operating systems. In addition, a new server process is spawned at each request, so its response time is relatively long. Separate processes for each request also make it more difficult to communicate between different instances of the server. This communication is essential for the coordination of the helper application and the browser. Each request interacts with the server over its own HTTP connection, hence through a different process. However, the information generated from the browser/server pair is needed by the helper/server pair and vice versa. Therefore, a simple but slow file based interprocess communication scheme was implemented. Though this server succeeded in demonstrating the feasibility of the general approach, it is unsatisfactory for more than just a prototype.

2.4 Revision Control vs. Configuration Management

The last limitation of WWRC had more to do with the tools at hand than with WWW. WWRC supports just distributed revision control and not configuration management. This decision arose from the fact that the authors had developed an appropriate API for revision control in the form of the Revision Control Engine (RCE)[13], but had not yet extended it to configuration management.

3 A Browser's View

Most of the shortcomings of WWRC have been solved with the help of Java and an extension to RCE called the Configuration Management Engine (CME). Combining CME with a new Java based server and browser side applet goes a long way towards solving the shortcomings of WWRC. These changes are not just internal. Rather, they are immediately apparent to the user as well. Before diving into the details of the implementation, a quick overview of the user interface is in order.

As in WWRC, the first page one sees is the login page (figure 1). WWCM relies on the basic authentication scheme provided in HTTP for validating users.

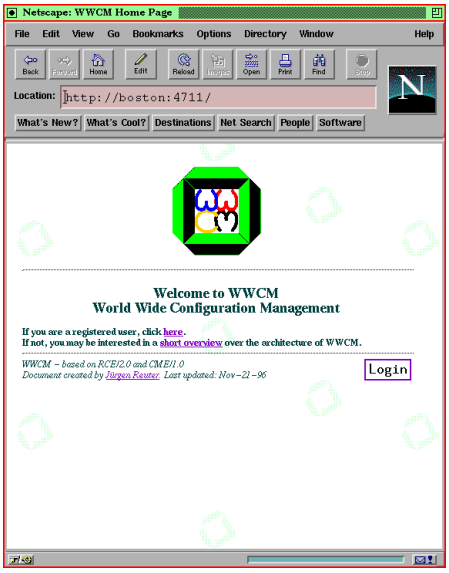

Fig. 1. Log into WWCM

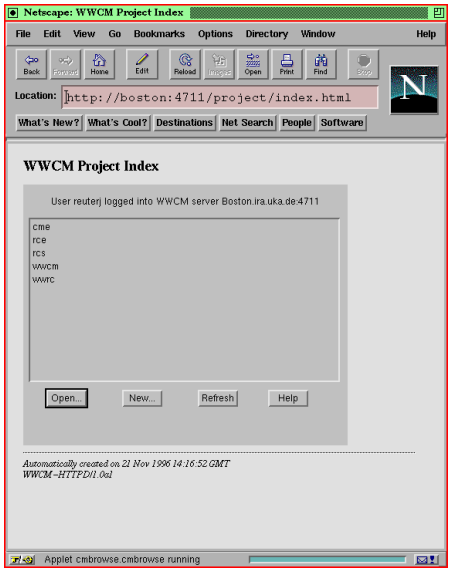

Fig. 2. Selecting a Project

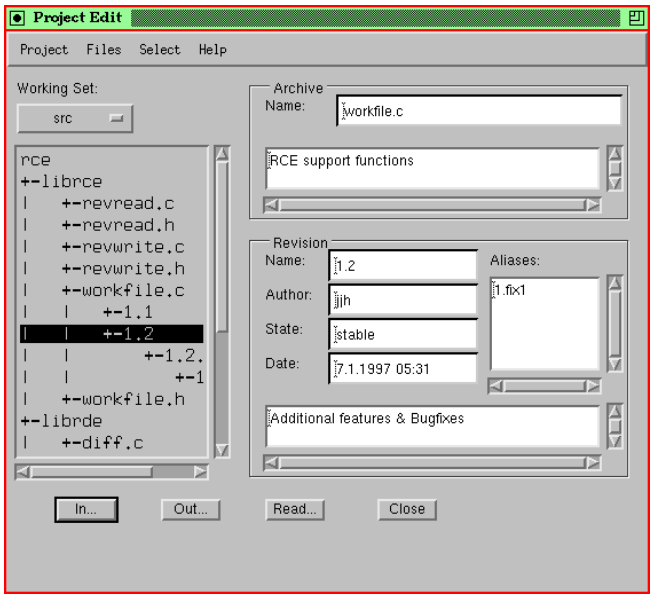

Fig. 3. Manipulating a Project

But the similarity ends there. The entire paradigm of WWCM is different than WWRC.

From the user's perspective, there are two main concepts that extend CME beyond RCE, and by extension WWCM beyond WWRC. CME defines projects and working sets to organize files into a larger context. These two concepts provide a structure for organizing the archives of RCE into a single repository.

After password verification, the user encounters the project selection page depicted in figure 2. This is the last page displayed in the browser. All the rest of the interaction occurs over dialog boxes generated by the applet that accompanies the project selection page.

Here the user is confronted with choosing a project. In WWCM, a project is a collection of files that belong together for implementing and documenting a particular end product or system. CME uses a tree structure to help organize the various parts of the project, just as a user might organize her personal workspace.

Once the user has selected a project (rce in the example), the first dialog appears. In the upper left corner of the dialog is a field for selecting a working set. An example is depicted in figure 3. Working sets are defined by users or administrators to provide two functions. They are used to define subsets of the project and to set check-in and check-out policies. Filtering is done by listing files or subdirectories of interest. Along with each entry, there is a policy selection to go with it.

The user can select from the list of working sets or from the entire project. The contents of the project or selected working set is listed in the large selection box on the left hand side of the dialog. The right hand side gives summary information for the last file or revision selected on the left. The user can browse through the project, look at individual revisions, select a working set, and mark elements of interest. The working set operations are available through the pull-down menu at the top of the dialog. Working set operations include check-in, check-out, and get-read-only. One can also edit the contents of the project and the working sets with the project operations.

In the case of check-out, one first selects the working set or a subset thereof in the dialog. Then one simply selects the "check out" option in the file pull-down menu. A new dialog then appears so that one can specify information about the versions being checked out such as their state, a new alias, and commentary describing the intent of the check-out. The comment can be amended or extended at check-in time. It is important to keep in mind that the main operations are performed on groups of files at once, whether they be the entire project, a working set, or a subset thereof.

Other operations are implemented analogously. Together, the dialogs act like a local application that simply accesses the remote store. The project manipulation dialog remains active until the user presses the close button or the browser is exited. This means the user is free to use the browser for other things.

4 Behind the Scenes

Behind the user interface, the situation is a bit more complicated. Though the combination of CME and Java makes WWCM much more powerful and cleaner than WWRC, there are still some stumbling blocks. The most difficult problem that still remains is the inability to communicate directly from the applet to local store. All this interaction must be diverted over the server to a helper application on the client machine. Despite this, CME and Java have brought some important capabilities to WWCM.

4.1 Beyond RCE: Distributed Configuration Management

The check-in and check-out interface of RCE alone are not sufficient for a configuration management system. For this reason, the authors developed the CME API to complement RCE. RCE supports file based archiving with the ability to manage alternate lines of development via branching. In addition, RCE has a full API and templates for tracking work in progress. CME extends RCE by providing a means of maintaining a collection of files—a project. CME also leverages the template concept of RCE to offer a means of managing alternative lines of development—Template Regulated Alternative Development (TRAD).

4.1.1 Structure CME provides basic structures for configuration management. Project elements are stored in a hierarchic list. Working sets are provided to manage interesting subsets of this list. Access control lists are available to support user roles. Finally, element names are managed with an internal numeric registry for managing renaming.

It is often convenient for the developers in a project to arrange the elements of the project in a hierarchic structure. CME maintains the elements of a project as a list with hierarchic names for each element. This structure was seen in the last section. When checking a project out of its archives, CME fills out a directory subtree in the user's work area which reflects the hierarchy of the project name space.[2]

One seldom needs the entire project at one time. As mentioned above, CME provides working sets to simplify the manipulation of just the elements in the project that are pertinent. Any operation that one could perform on the entire project can also be performed on a working set.

Access can be set at any levels of a configuration from the entire project down to individual archives. There are three level of access: read, write, and administrative. This is based on the access rules of RCE. CME extends these rules to the configuration as a whole.

To help organize these structures, CME assigns a unique number to each element in the project list. This number is an internal unique identifier for the project element. The user should never see this number. Each number stays

[2] CME also supports user mapping for when a particular user or application requires some other view.

with its node or archive regardless of how often the node or archive is moved or renamed. With this simple mechanism, the project itself can be versioned. Files that appear in a previous version of the project can be found no matter what they are called in the current version of the project. By maintaining a "graveyard" directory for old files, the system can even locate files that have become obsolete for the current version of the project.

4.1.2 Templates CME's main task is to coordinate the states of all the archives for the files in a project. This task becomes interesting in the face of multiple developers working on the same data. There are two main strategies for handling this problem. The first is reserved check-out. Locks prevent more than one person from working on the same file simultaneously. The second is unreserved check-out. Free access is allowed to all revisions with the proviso that the first one finished wins the slot on the main branch. By making use of RCE templates, CME can go beyond these models to provide a flexible policy for coordinating check-out and check-in.

4.1.3 Basic Templates When a user checks a file out of an RCE archive, an empty revision is added to the archive. This revision holds information about who checked the file out, when this happened, and to where in the file system it was deposited. A second user can check the same revision out, but his template will start a branch. Enough information is contained in the template to ensure that when a user checks a file in, it goes into the correct template. The system always has a record of who has what checked out.

4.1.4 TRAD—Template Regulated Alternative Development The authors wished to provide a mechanism for private check-pointing in CME. This is for maintaining version control on code that is not ready to be shared with other users. The result is a system that distinguishes between public revisions and private revisions. Public revisions are those that have been released to other developers. Private revisions are those that should only be accessible to a single user. A private revision forms an alternate to the main line of development and to that of other users. One would like to ensure that all released revisions in a project are consistent with one another. One would also like to prevent one user alternate from superseding that of another accidentally and to support reconverging policies for promoting revisions back onto the main line of development. TRAD was designed to do just that.

TRAD takes the RCE template mechanism a step further. Each user receives his own branch when he checks a file out. If the revision being checked out is the last revision on a public branch, a place holder template will be created on the public branch before the user's branch is created with his private template. The user's login name will then be used to tag the template. Users continue to work on their private branch until they are satisfied that their code is ready to be released to other developers. At that point, the system can use the template

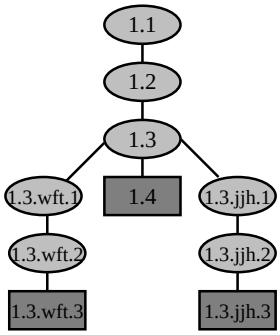

Fig. 4. Parallel Development

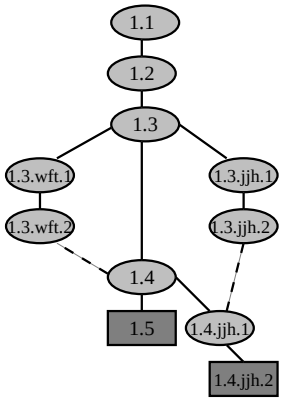

Fig. 5. Merge Resolution

information to determine if the user can take over the place holder template on the public branch. If not, the system can generate the necessary merge.

4.1.5 Example In figure 4, two users—wft and jjh—have been working on their private branches. The template 1.4 is the place holder. Each user has his own respective template 1.3.wft.3 and 1.3.jjh.3. In this state, wft decides to check in his changes back onto the main branch. Since the parent of the place holder template (1.3) is an ancestor of his template (1.3.wft.3), the system knows that his template can be checked into the place holder template (1.4) safely. When jjh tries to do the same thing, the system notices that it is not safe to check his revision into the place holder template (now 1.5). This is because the parent of the place holder (1.4) is not an ancestor of his template (1.3.jjh.3). In order to remedy this situation, the system spawns a new branch for jjh from 1.4 (1.4.jjh.1) and fills it with the merge between 1.4 with 1.3.jjh.2. When that is reviewed and tested, he is in the position to check his version into a public branch. It is important that the parent of this new branch is 1.4 and not 1.3, so that the conflict testing works properly. Merge links are set from 1.4 back to 1.3.wft.2 and from 1.4.jjh.1 back to 1.3.jjh.2 to capture the merge relationships.

By looking across the entire project, CME can prevent mismatched alternates from appearing on the main branch without first being examined together on a private branch. The criteria for promoting an alternative line of development must be met by all elements of the project before the promotion can take place. Piecewise promotion is not used, because changes in one file can require changes in another file. It is important that the versions the user tested are all put onto the main branch together, not just some of them. Of course, files that where not changed have no bearing on the promotion. The point is to insure that the collection of current revisions on the public branch forms a consistent, tested

set. Using templates, the system can always place new revisions back into the proper place in the tree. Furthermore, the system can always determine what merges must be made to bring a developer up to date.

It is important to note that TRAD allows different promotion and merging policies to be implemented. Thus, not only can one insure that a proposed promotion onto the main branch is allowed for all files involved before the check-in onto the main branch can occur, one can also prioritize the promotion of private branches back onto the main branch. Each user may have his own set of check-in conditions with regards to other users. For example, a user of low priority would not be allowed to promote his variant to the main branch if any user had the default revision checked out. This is useful for difficult changes, that should not be checked in without merging all other development streams first. The opposite condition, where one has the highest priority for check-in, is useful for expediting a particular change. How exactly this will be handled is a topic of ongoing research.

4.2 A Stitch in Time: Threads in a Java Based Server

In order to provide a more portable and standard basis for WWCM, the server of WWRC needed to be rewritten. Before embarking on writing a totally new server, the authors examined what was available. Though there were some servers available that were customizable, none appeared sufficiently flexible to support WWCM. In the end, a different approach was taken in writing the WWCM server. It is now a multi-threaded server written in Java.[3]

The new server solves all the problems of the WWRC server. It is much more responsive than the WWRC server. Threads can communicate with one another through a blackboard. The server is also, thanks to Java, fully portable.

Two things differentiate the WWCM server from other Web servers. It has built-in classes for configuration management and it has a hierarchic configuration mechanism. The RCE and CME APIs are made available to the server through Java native methods. These are internally wrapped in classes such as Archive, Revision, Project, Working Set, etc. Both APIs support object linking to provide fast reference between API handles and wrapper classes. The hierarchic configuration mechanism of the server enables one to control directory access on a case by case basis. Attribute inheritance keeps simple cases simple. This structure allows fine control of access to the server's disk store. Thus the same server can act as Web Server and as WWCM server without sacrificing security.

4.3 Three is a Crowd: Browser, Applet, and Helper

The use of Java applets in the browser has made a world of difference on the client side of WWCM. The entire interaction with the server takes place on two

[3] There is now another similar server called Jigsaw[29] that could have been the basis for WWCM had it been released somewhat earlier.

Web pages: a login page and a project selection page. All other information is obtained through Java dialogs. Though a helper application[4] is still needed to support check-in, check-out, and the selection of files to be added to the archive, the protocol between the three is cleaner.

The client interface for WWCM behaves as one would expect any GUI to behave. Since there are only two independent Web pages for the interface, no page time stamping is necessary. Browser forward and back commands can no longer cause inconsistency. In fact, once a project has been selected, one could even page through other Web sites while using WWCM.

The applet controls most of the interaction with the user. The server need only be contacted when more information is needed or a revision transfer is to take place. The applet caches as much state information as possible.

The applet is a win in another way as well. It can initiate the start of the helper application and contact the server at the same time. This means that the server need not abuse the semantics of multi-part messages in order to start the helper application and also refresh the user's view on completion. Unlike the multi-part message hack, the applet will not automatically time out waiting for the completion update.

The helper application is still needed because, as of this writing, there is no secure way to give an applet access to the local file system. The helper application is the single biggest impediment left in WWCM. When secure local access becomes available, the protocol can be made much simpler. As mentioned above, there are three cases where the helper application is needed: by check-in, by check-out, and by adding a new file to the project. The first two cases are depicted in figures 6 and 7, respectively. The protocol for the last case is essentially the same as for the first. It varies only in the data that is transmitted and what the helper application actually does.

4.3.1 Check-In From the point of view of the protocol, check-in is the simpler case. The helper is needed to send the files to the server for check-in and to delete the files from local store when the check-in is successful, but it need not process additional information from the server. It just has to wait for completion.

The most difficult aspect of this protocol is getting the helper application started with the proper information. The difficulty lies in the fact that the helper application interface was designed to let a browser call another program to display information from a server that it can not interpret. This means that only a message from the server can start a helper application. The problem is, in WWCM, the applet needs to drive the request based on the user input it receives. The path to do that is quite circuitous, because it must involve the server.

After the user has selected the working set to be checked in from the project list, the applet needs to start the helper application to collect the files and send them to the server. However, to do this, the applet must first send a "show document" request to the browser with a URL requesting a check-in. This message

[4] The helper application is also written in Java for portability.

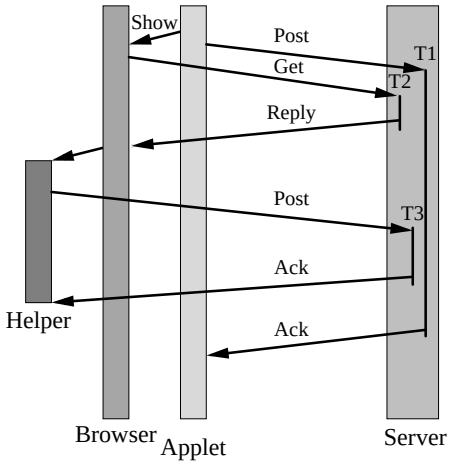

Fig. 6. Check-in Protocol

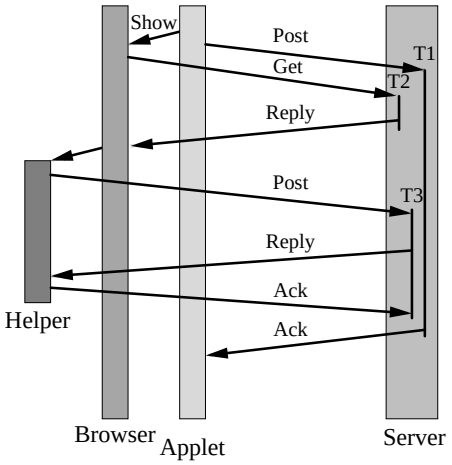

Fig. 7. Check-out Protocol

is used to instruct the browser to request the server to initiate the helper application. This is the only way to get the server to send a packet of information to the browser causing it to start the helper application. The message contains a unique token that is used to identify the request for coordinating the various connections needed to carry it out.

In response to the "show document" from the applet, the browser performs a "get" from the server over a special URL. This starts a server thread (T2), whose sole task it is to send the helper application start request back to the browser, after which the thread exits. The content type is set to a special value that causes the browser to finally start the helper application.

The helper application collects the files to be checked into the server and posts them to the server. This starts another server thread (T3). When the server is finished processing the files, it sends an acknowledgment to the helper application. The helper application then does any necessary clean up and exits.

As soon as the applet finished sending the "show document" to the browser, it posts a completion notification request with the list of files to be checked in and the afore mentioned token to the server. This also starts a new thread in the server (T1), which actually starts before the two described above. This thread has the sole task of collecting information from the other threads and notify the applet of completion status when all processing is finished. Before the third thread (T3) exits, it sends a message to this thread (T1) indicating that it is finished. The first thread updates the applet status and exits.

4.3.2 Check-Out Check-out is similar except for two points. Again, the helper application is needed for file operations. The complication comes from

the fact that the main information flow is in the other direction.

Theoretically thread T3 could do the check-out and send the files to the helper via a reply to the helpers "post" message. It then would have to wait for an "acknowledge" message from the helper application before sending the confirmation to thread T1. However, this would be slow.

Instead, thread T1 started by the applet actually checks out the requested material. It then passes this information to thread T3 when it is started by the second check-out request from the helper application. One can think of this as a prefetch for the third thread (T3).

5 Related Work

There are several other approaches to distributing configuration management. They range from the simple use of a distributed file system such as NFS, AFS, or SMB, to the distributed repository approach of NUCM[2] and the distributed workspace model of the MultiSite tool[1] from ClearCase. In between there are the change propagation paradigm of the MISTRAL tool[7] in the ADELE system and the central archive access over remote procedure calls as in DRCS[18] and DCVD[12]. WWCM shares the central archive approach in so far as the user communicates with a central store over the network. However, it differs from all other methods above in level of support required. The user need have no access to the server machine to access the archives and the client side is lightweight.

6 Conclusion

The authors presented a system that uses Java and the World Wide Web to provide a portable distributed configuration management system—WWCM. This system is based on their experience with WWRC and improves on that system significantly. The system now fully conforms to the HTTP specification. This ensures better interoperability and enables the system to take advantage of more existing facilities invocation.

The inclusion of CME makes WWCM a full configuration management system. Though CME is still in the prototype stage, it provides all the basic requirements of configuration management over an API. CME uses TRAD, a paradigm for resolving conflicts between developers working on the same project, to support parallel development. It provides a means of experimenting with reconverging policies for personal variants.

The use of Java applets enables the system to be more robust and responsive. It allows the WWCM server to provide better feedback without breaking the semantics of HTTP. One would like to avoid having a helper application altogether. Unfortunately, this is not possible at present. Most browsers forbid applets to access local disk store. The result is that all communication between local store and the applet must be carried out over the server. One would expect that some future browser policy would allow access to local store for authorized

applets (or applets from authorized URLs). Perhaps signed applets will provide a means of solving this shortcoming.

The roundabout communication path for accessing local store is the biggest impediment to a truly efficient implementation. For this reason, though performance is acceptable, it can not compete with other distribution techniques on a performance basis. The setup time for operations such as check-in and check-out is high. Still, for applications that require lightweight access to a central archive, WWCM provides a viable alternative to traditional approaches.

7 Future Work

There are three main directions for future work. The first is to develop the system further to take better advantage of the network, e.g. design a proxy server that can share archives with the main server. The second is to expand on the concepts of CME to provide process control, perhaps including support for build management and external tools. Finally, one would like to have some real performance analysis for WWCM.

References

1. Larry Allen, Gary Fernandez, Kenneth Kane, David Leblang, Debra Minard, and John Posner. Clearcase multisite: Supporting geographically-distributed software development. volume 1005: ICSE SCM-4 and SCM-5 Workshop Selected Papers, pages 194–214. Springer Verlag, 1995.
2. Dennis Heimbigner André van der Hoek and ALexander L. Wolf. A generic peer-to-peer repository for distributed configuration management. pages 308–317. IEEE Computer Society Press, March 1996.
3. Atria. *ClearCase Concepts Manual*, 1992.
4. David H. Crocker. Standard for the format of arpa internet text messages, august 1982.
   WWW. http://www.cis.ohio-state.edu/htbin/rfc/rfc822.html.
5. Jacky Estublier and Rubby Casallas. The adele configuration manager. In Walter F. Tichy, editor, *Configuration Management*, pages 99–133. John Wiley & Sons, 1994.
6. R. Fielding. Relative uniform resource locators.
   WWW, June 1995. http://www.cis.ohio-state.edu/htbin/rfc/rfc1808.html.
7. Christophe Gadonna. *MISTRAL User Manual.* de Génie Informatique, Genoble, May 1995.
8. Gamelan. Earthweb's java directory.
   WWW. http://www.gamelan.com/index.shtm.
9. James Gosling and Henry McGilton. The java language environment.
   WWW, 1995. http://www.javasoft.com/documentation.html.
10. N. Haller. The s/key one-time password system.
    WWW, February 1995. http://www.cis.ohio-state.edu/htbin/rfc/rfc1760.html.
11. Samuel P. Harbison. *Modula-3.* Prentice Hall, 1992.
12. T. Hung and P. F. Kunz. Unix code management and distribution. Technical report, Stanford Linear Accelerator Center, Stanford, California, September 1992.

13. James J. Hunt and Walter F. Tichy. *RCE API Introduction and Reference Manual.* Xcc Software, 1997.
14. James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy. An empirical study of delta algorithms. *Lecture Notes in Computer Science*, 1167:49–66, 1996.
15. Javasoft: For developers.
WWW. http://www.javasoft.com/nav/developer/index.html.
16. J. Linn. Privacy enhancement for internet electronic mail. Part I: Message encryption and authentication procedures.
WWW, February 1993. http://www.cis.ohio-state.edu/htbin/rfc/rfc1421.html.
17. N. Freed. N. Borenstein. Mime (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies.
WWW, September 1993. http://www.cis.ohio-state.edu/htbin/rfc/rfc1521.html/.
18. B. O'Donovan and J. B. Grimson. A distributed version control system for wide area networks. September 1990.
19. J. Poste. Media type registration procedure.
WWW, March 1994. http://www.cis.ohio-state.edu/htbin/rfc/rfc1590.html.
20. K. Sikkel R. Bentley, T. Horstmann and J. Trevor. Supporting collaborative information sharing with the World-Wide Web: The BSCW shared workspace system. In *Proc. of the 4th International WWW Conference, Boston, MS*, December 1995.
21. Jürgen Reuter, Stefan U. Hänßgen, James J. Hunt, and Walter F. Tichy. Distributed revision control via the world wide web. volume 1167, pages 166–174. Springer Verlag, 1996.
22. R. Rivest. The md5 message-digest algorithm, april 1992.
WWW, April 1992. http://www.cis.ohio-state.edu/htbin/rfc/rfc1321.html.
23. M. Rockhind. The source code control system. Number SE-1(4), pages 364–370, December 1975.
24. M. McCahill T. Berners-Lee, L. Masinter. Uniform resource locators (url), december 1994.
WWW. http://www.cis.ohio-state.edu/htbin/rfc/rfc1738.html.
25. Walter F. Tichy. RCS: A revision control system. In *Integrated Interactive Computing Systems.* North-Holland Publishing Co, 1983.
26. Cern httpd.
WWW. http://www.w3.org/pub/WWW/Daemon/.
27. Http - hypertext transfer protocol.
WWW. http://www.w3.org/pub/WWW/Protocols/.
28. Hypertext markup language (html).
WWW. http://www.w3.org/pub/WWW/MarkUp/.
29. Jigsaw overview.
WWW. http://www.w3.org/pub/WWW/Jigsaw/.

This article was processed using the LaTeX macro package with LLNCS style