Distributed Revision Control Via the World Wide Web

Jürgen Reuter, Stefan U. Hänßgen, James J. Hunt, and Walter F. Tichy *

IPD, University of Karlsruhe, Karlsruhe, Germany

Summary. Revision control has long been a standard part of software development. With the enormous expansion of the Internet and its increasing use as a means of communicating among geographically dispersed software developers, the need for distributed version control over the Internet has become acute. In order to address this need, the authors have developed a revision control server based on the World Wide Web (WWW) and RCE (an outgrowth of RCS). This proves to be possible and it also highlights the strengths and weaknesses of using the Hyper Text Mark up Language and standard WWW browsers such as NetScape$^{\mathrm{TM}}$and Mosaic to accomplish this goal.

1. Introduction

Programs for revision control have been available to software engineers for over a decade. The best known examples are RCS[12] and SCCS[11]. Up until relatively recently, revision control was only available locally. Though the network could be used to mount a file system containing revision control archives and archives could be accessed using remote login and ftp, one had to first obtain system access to the appropriate system before one could access an archive. Recently ClearCase[10] has introduced a distributed revision control system that can be used between a number of geographically separated groups. But here again, accessing the archive requires access to the host on which the archive (or versioned object base) resides.

There are many situations where this type of access is not desirable. For example, in a scenario where two companies collaborate on a short term project, they may not wish to grant one another full access to each other's machines. Still, relatively unrestricted access is needed to the the shared source code. Another example is supporting working from home or free-lance software development. Here again, a firm may want to maintain control of the source code without the risk of unrestricted dial-in access.

An obvious solution for such situations is to develop a revision control system based on a client/server architecture. This type of architecture has been quite successfully used for the X Window System and the World Wide Web. It gives all users on the net (including those using dial-in network protocols like PPP and SLIP) potential access to the archives without the need to login to the server machine. Access control to the archives can be

* The authors can be reached at `(reuterj|haenssgen|jjh|tichy)@ira.uka.de` respectively

managed totally independently of any host login process. Thus the desired level of access and protection can be tailored to the individual needs of each development team.

2. Building Blocks

At first glance, this solution seems to require a tremendous amount of effort to implement. A server needs to be developed that supports all aspects of revision control. An interface format must also be developed for communicating between the clients and the server. A client program must be written to present the user interface. Finally, this client must be ported to each machine architecture and operating system where it is to be used. The only way to minimize this development work is to take advantage of already existing tools.

2.1 RCE

At the server end, much work can be spared by using an existing revision control system. There are several revision control systems available, but only one — the Revision Control Engine (RCE) — offers a full fledged API that does not rely on external program invocation. RCE is a descendant of RCS with a number of enhancements, the most important one being that it uses a much better differencing algorithm than RCS[9]. This means that RCE archives are much more compact than RCS archives and binary data can be efficiently stored along side text data. By taking advantage of the RCE API[3], much work can be spared in building a revision control server.

2.2 Using Existing World Wide Browsers

This still leaves the interface format and the client side of the system. As mentioned above, there are two very successful client/server systems that are available on almost every platform: The X Window System and the World Wide Web[5] (WWW). The latter is designed to support the distribution of documents. This is a goal that covers at least half of the client side needs of a distributed revision control system — the selective transmission of data from the server to the client and managing user interaction.

WWW is based on a client/server model where clients request documents from any number of servers across the network. The interface format for WWW is Hyper Text Mark up Language. HTML[2] offers a means of describing documents and user interfaces in a machine independent form. It also contains a mechanism for uniquely locating other documents (servers) throughout the Internet. The client program or WWW Browser is responsible for interpreting HTML, then presenting the result to the user and channeling user responses to the appropriate server.

By combining components from both RCE and WWW one could build a full distributed revision control system. The server can be build on top of the RCE API and must generate HTML and respond to WWW Browser requests. Standard WWW Browsers such as NetScape™and Mosaic can then be used to access the WWW base revision control server. This means the system can take advantage of the enormous base of existing WWW clients. Only small additions on the client side are needed to transfer revisions of data between the client and server.

3. World Wide Revision Control Prototype

The authors have succeeded in building a prototype system named World Wide Revision Control. Before diving into the architecture, here is the user's perspective of the WWRC system in action. This should provide some insight into how it all works. Screen dumps 1 to 4 show different stages of the user dialog involved in checking out a revision. The WWRC Server generates all the HTML pages automatically from the RCE archives. As a popular example of a WWW browser we here use NetScape™. All screen shots are taken from our WWRC prototype implementation written in Modula-3[8] and running on a Sun SPARC 5 under SunOS 4.1.3.

Almost all user actions consist of selecting an item or button by clicking on it. Only adding items to lists or entering text fields (e.g. comments) involves the keyboard at all. The WWRC server handles the actions accordingly and produces a new HTML page as a result[1].

3.1 Login

Before the user can access any archives, he has to give his name and password. The server checks them against its own database and uses them later to tell different connections apart, enforce access rights, etc. This database might be quite different from the system login database. Only once the user is verified, can he make use of the archives belonging to that server.

3.2 Revision archive handling

After login, the user can select an archive and perform various operations on it, e.g. show its log file or its revision graph. To allow easy access to frequently used archives, the user can also define a pickup list from where he can directly choose archives later. Administration functions such as changing the WWRC password, as well as online help are offered here, too.

[1] with the exception of the check in/out process which necessitates transfer of other data than HTML files, as described in the Architecture section.

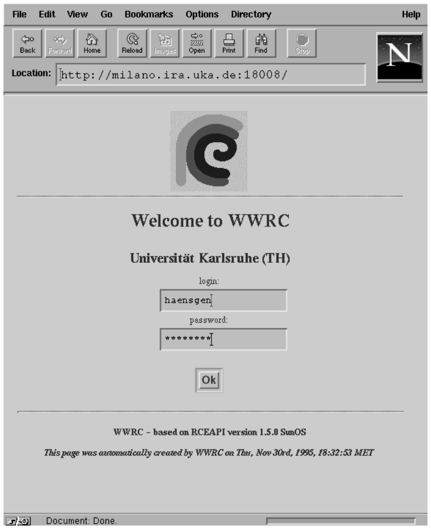

Fig. 1. The WWRC login screen

Fig. 2. Archive options

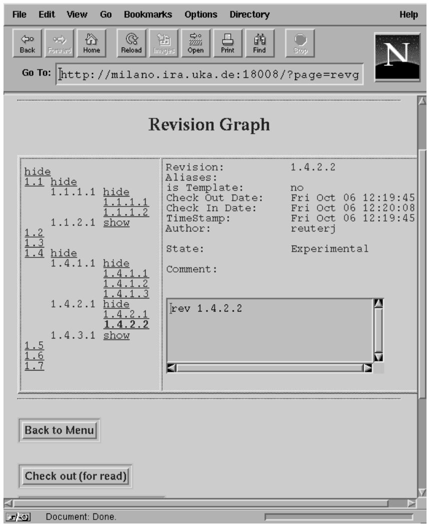

Fig. 3. Revision graph of one archive

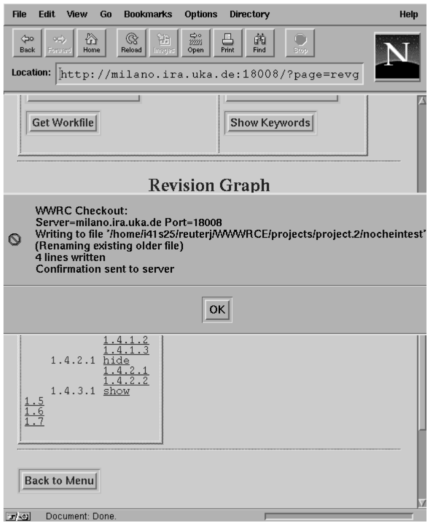

Fig. 4. Successful check out

3.3 Revision graphs

The revision graph view shows the selected archive's revisions. Subgraphs can be hidden or shown again to give a clearer overview. Selecting one revision displays more information on it, such as its time stamps, the author, its state, and commentary about the revision. The graph is displayed as formatted text as shown in figure 3. This saves network bandwidth compared to dynamically generating pictures.

3.4 Check Out

Once a revision is selected, it can be checked out and transferred to the user's machine. How this is done internally using just the WWW browser and a small helper application is the subject of section 5.. The user just clicks on the check out button and gets a confirmation of the check out's success.

3.5 Other functions

The functionality described above is just a representative selection. WWRC makes all of RCE's functions accessible over the Internet. Checking in files, adding comments, selecting work files, creating new revisions, etc. are all supported.

4. Challenges

In order to develop this prototype, it was necessary to overcome three problems. How can large amounts of data be transferred automatically, particularly back to the server? How can consistency be maintained with a stateless protocol? How can actions that are inherently separate in HTML be combined to produce a comfortable user interface?

Getting information from a server is the very purpose of WWW, however automatically storing the information in some defined place on the client machine without asking the user for directions is not supported. Also, transferring larger amounts of data from the WWW Browser to the Server is far from trivial. HTML now offers forms for user input, but they are not at all adequate for automatic data transfer in the megabyte range. Therefore, some suitable mechanism for the check in and check out file transfer is necessary.

The WWRC server itself also does not 100% fit the WWW paradigm at first glance. It has to keep some state information for each user, but HTTP[1] is inherently stateless. Also, the server has to perform the RCE functions using the API and generate WWW pages showing archive and control information on the fly.

Furthermore, as a response to a user action such as pressing a form button, the server is limited to just generating one kind of reply: it can either create a new WWW page containing updated information or it can transfer data. This is just like on an ordinary WWW server: one can either follow some link or down load a file, e.g, some PostScript document. In some cases it would be preferable to do both at once.

5. Architecture

A brief look beneath the hood of WWRC will help clarify how WWRC manages to use standard WWW browsers to present the afore described user interface.

To tackle the bidirectional file transfer problem, WWRC uses the concept of helper applications. These are small programs that the WWW Browser calls in order to display those kinds of data it cannot visualize itself. An example is the Browser receiving a PostScript file. After down loading the file, it may start the `ghostscript` application to display it. Adding a new helper application just involves adding the its file type and its calling procedure to the Browser's configuration files. This can easily be done by the user.

The idea of how to perform check in and check out is based on this concept: Upon a check out request, the Server first does a local check out of the file on its machine into a special directory that is unique to each WWRC user. To transfer this checked out file, it then transmits a header stating the file's name and directory position, as well as some administrative information, and finally includes the file contents themselves. The type of this generated transfer file is set to "`rceco`" which causes the Browser to call the new helper application "`checkout`" with the transfer file. The application reads the header, determines where in the user's directory tree the resulting file should be created, and writes the file's contents into there[2]. After that, it sends back a confirmation directly to the server stating the file has been received successfully. The server then marks the locally checked-out file as expired to remove it automatically on the next request for an HTML page. In case the transmission failed, this concept still allows the user to request the file once again. Figure 5 illustrates this process.

Similarly, to perform a check in from some user file to the server, the server transmits a dummy file stating which user file to transfer. The corresponding helper application reads the file on the client machine. It then connects itself directly to the server, which puts the file into a special directory there where it is then checked into the server's archive. The server knows into which archive to put the file and to which user it belongs because administrative information from the dummy file is sent back to it together with the file itself. All file paths are taken relative to some root directory on the server and some point in the user's directory tree, so the file structures below that point are the same on client and server.

[2] any already existing file is uniquely renamed to avoid overwriting it.

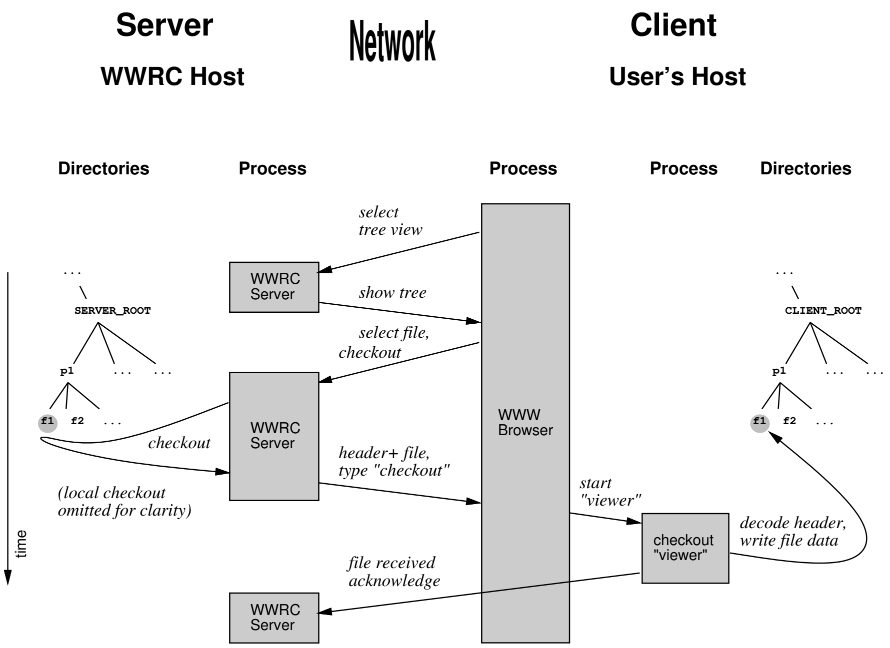

Fig. 5. Internal processes during check out

The server and the helper application have complete control over the connection and the original data on the server is kept until its reception has been confirmed by the client. This means that any protocol can be implemented to handle possible losses of connection during large file transfers or to encrypt data for transfer. These concepts are well known in the Internet community and could be adapted to WWRC as well; the current implementation prototype does not yet support that functionality.

The second problem, namely the statelessness of the HTTP protocol, can be solved by adding all necessary information to the URLs themselves. In addition to that, the Server keeps an internal database for each user recording information such as the current work file and the user's pickup list. To keep the user from going back in his browser's history of recently accessed pages and perform actions on these pages whose content is out of date, WWRC keeps a time stamp on each page and warns the user when he tries to use outdated forms. The state information is also necessary because the WWRC server process is started anew for each client request. This allows multiple servers to be active at once, rising the throughput of the system, but the server has to know where to resume. This is achieved by combining the user information given in the URLs with the state information database.

For the *either page or data* question we have not yet found an elegant solution that is both platform and browser independent. The problem is that pressing a button, for example to check in a file transmits the data as described above, can cause a file to be transmitted, but the WWW page shown in the browser is not updated. This means that it no longer reflects the state of the archive after the check out. The obvious cure is to add text to the page making the user aware of this behavior and urging him to press a continue button directly after check in. However, this still leaves too many possibilities for errors.

6. Related Work

There are two main aspects to this work: distributed revision control and the World Wide Web as a transport medium for computer supported cooperative work (CSCW). As mentioned above, ClearCase[10] provides for distributed revision control; however, it requires users to have login accounts on the machine where the archives are stored. BSCW[4] is an example of a system that uses the World Wide Web as a basis for CSCW; however, it does not support deltas and revision trees. WWRC is the only system the authors know of that supports both.

7. Conclusion

This paper demonstrates the possibility of using the World Wide Web for distributed Revision Control. The approach taken at the client side is machine-independent and portable — adding a new client architecture just means porting the small checkin and checkout helper applications and making cosmetic changes to the existing Browser's configuration file. Furthermore, the techniques used here may also be used to convert other applications to client-server architectures based on WWW and the Internet.

8. Future Work

While the current system is in a good working condition, as shown above, there are plans to improve usability, efficiency, security, and functionality.

Usability can be enhanced by cleaning up the user interface and integrating user side utility programs. Aside from improving the layout of some of the HTML pages that are generated, provisions need to be added to handle several files at the same time. Also, the integration with the ability to step backwards to previous sides and then forward again needs a bit more support to maintain consistency. Java[7] can be used to replace the helper application and to support other advanced functions in an even more portable manner.

The efficiency can be improved by two means. The server can be made faster by using a daemon to interact with the Web server instead of starting a new large server process for each transaction. Also, data could be automatically compressed to improve the transfer time for check in and check out.

There is a general need to limit access to any revision control system. Though password verification is already present, the data sent over the net is not protected from snooping. This needs to be addressed through the use of data encryption as mentioned above.

Currently, a WWRC version supporting multi-checkouts, i.e. allowing the user to check out a complete set of files, is in development. While concurrent accesses and possible conflicts are handled by the underlying RCE locking concept, the server has to be extended to avoid deadlocks and increase efficiency. For example, the system needs to support the case where two or more simultaneous user requests intersect one another, i.e. they both contain some of the same files. Eventually this list based approach needs to be extended with check out sets and other facilities for managing large projects. Using the system as a communication forum for the developers could help coordinate development in a large project. Configuration management can also be used to make WWRC an effective distributed project management tool.

References

1. HTTP: A protocol for networked information. `http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html`, 1993.
2. The HTML 3.0 HyperText Document Format. `http://www.w3.org/hypertext/WWW/Arena/tour/start.html`, 1994.
3. *RCE - Introduction and Reference Manual*, API Revision 1.4.5 edition, 1995.
4. R. Bentley, T. Horstmann, K. Sikkel, and J. Trevor. Supporting collaborative information sharing with the World-Wide Web: The BSCW shared workspace system. In *Proceedings of the 4th International WWW Conference*, Boston, MS, December 1995.
5. Tim Berners-Lee. World Wide Web Initiative. `http://info.cern.ch/hypertext/WWW/TheProject.html`, 1994.
6. Jacky Estublier and Rubby Casallas. The adele configuration manager. In Walter F. Tichy, editor, *Configuration Management*, pages 99–133. John Wiley & Sons, 1994.
7. James Gosling and Henry McGilton. The Java Language Environment. `http://www.javasoft.com/documentation.html`, 1995.
8. Samuel P. Harbison. *Modula-3*. Prentice Hall, 1992.
9. James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy. An empirical study of delta algorithms. *Sofware Configuration Management Workshop*, 1996.
10. Atria Software Inc. ClearCase concepts. Technical report, ., Natick, Mass., 1993.
11. M. Rockhind. The source code control system. *IEEE Trans. on Soft. Eng.*, SE-1(4):364–370, Dec 1975.
12. Walter F. Tichy. RCS: A revision control system. In *Integrated Interactive Computing Systems*. North-Holland Publishing Co, 1983.