Multi-user publishing in the Web : DReSS, a Document Repository Service Station

Eindhoven University of Technology
Department of Mathematics and Computing Science

Multi-User Publishing in the Web:
DReSS, A Document Repository Service Station

by

Paul de Bra and Ad Aerts

96/02

ISSN 0926-4515

editors: prof.dr. R.C. Backhouse
prof.dr. J.C.M. Baeten

Reports are available at:
http://www.win.tue.nl/win/cs

Computing Science Report 96/02
Eindhoven, February 1996

Multi-User Publishing in the Web: DReSS, A Document Repository Service Station

Paul De Bra and Ad Aerts
Information Systems Section
Department of Computing Science
Eindhoven University of Technology
PO Box 513, 5600 MB Eindhoven
The Netherlands
E-mail {debra,wsinatma}@win.tue.nl

Abstract

Many WWW servers contain information written by several authors. These authors either need an account on the server machine, and special permissions to create information in the server space, or else the Webmaster needs to put the information in that space or allow the server to point to the author's own space.

We present DReSS, a system to enable authors to deposit (and update) documents on a WWW server, using standard WWW features only. Authors do not need login permission on the server machine, ftp upload access, or even electronic mail. As the documents live in the WWW server space there is no need for the server to be able to access documents outside its space. Thus, our system will work on even the most securely shielded servers (running in a chroot environment).

DReSS consists of a set of CGI-scripts and two small auxiliary programs running on the client machine. It can be used with any (HTML-2.0-capable) WWW browser, and with any WWW server. DReSS does not use special features of a specific browser or server, and does not require any modification to the browser or server. For example, it does not use the HTTP PUT method, mostly because not every browser and server supports it. It does not use multi-part mime documents, file inclusion in forms, or server push features. It also does not use protocols (like ftp or smtp) other than HTTP.

We indicate where the current WWW architecture makes managing WWW servers with multiple authors difficult. This leads to suggestions for new browser and server features that could improve the authoring process significantly.

Keywords:
multi-user authoring, electronic publishing, document repository

1. Introduction

The World Wide Web (WWW or Web) [Berners-Lee94] is an open, distributed information system that lets groups of people share information. Most WWW servers contain documents written by several authors. Typical examples are departmental servers in research institutes and universities. They contain personal homepages, scientific papers, descriptions of departments and research programs, and often also courseware. Recently an increasing number of Internet providers for the general public started offering the possibility to put personal home pages on their WWW server.

The focus of the Web is on providing read-only access to information. Support for authors has been limited mostly to graphical (sometimes incorrectly called WYSIWYG) HTML editors, such as SoftQuad's HoTMetaL, and to add-ons for word-processors, such as Microsoft's HTML-assistant. (See [Carl-Davis] for a nice overview.) The problem of moving the created documents to the WWW server has been largely ignored until recently. In [Lavenant-94] a distributed hypermedia authoring system is presented that uses special-purpose WWW browser/editor extensions, and an invented "CTRL" non-HTML tag. In [Pitkow-95] a document repository system is described, which is extended with a versioning mechanism, a hyperlink database, HTML-verification and other features. This system uses a modified WWW server in order to do authentication, and uses a proposed form-based file uploading [Nebel-95], not supported by most WWW browsers. In [Bentley-95] the BSCW system is described, which supports so-called *Basic Support for Collaborative Work*. BSCW provides a document repository service with concurrency control and versioning. It requires a minor modification to the WWW server, and a *helper* application on the client. In [Andrews-95] the integration between WWW and Hyper-G is described. Hyper-G has been turned into a very appealing and powerful system, which can be accessed through WWW browsers and can contain HTML documents. In doing so it provides much more than a document repository system for WWW authors.

In a departmental server environment or the WWW server of an Internet provider, the main reason for using a document repository system is to avoid the need for authors to have login permission on the server machine. Features like link databases, versioning, on-the-fly generation of indexing information and others are worthwhile, but secondary to the goal of providing a document repository system which is easy to install and to use, and which can be used with different browsers (and servers) without requiring any modifications to "standard" WWW software. In this paper we introduce **DReSS**, a tool that turns a WWW server into a **D**ocument **Re**pository **S**ervice **S**tation, enabling authors to move documents to the WWW server, and to update documents on the server, without compromising the server's or client's security. To ensure that the security of the server and author's machines is not compromised by using DReSS the set of requirements given below contains several constraints related to security:

- Authors need no login permission on the server machine.
- The server machine needs no (nfs) access to the author's home directory (or any other directory on the author's machine).
- The author's machine must not need to host an ftp or WWW server in order to upload documents to the main server.
- All communication is done by means of HTTP.
- The system must use standard software: unmodified WWW browsers and servers.
- Most of the functionality of the system must be realized on the server machine.
- The author's WWW browser is the user-interface for the whole system.
- Elaborate access control is needed. The Webmaster must be able to decide which authors may create and edit documents in which directories and the authors must be able to decide who may edit and view their documents.
- The system must prevent simultaneous editing of the same document by different authors.

DReSS meets all the above requirements. It uses CGI-scripts on the server side, and two small auxiliary programs on the client side that are completely invisible to the author. These programs are based on the common www library to ensure easy portability. Although DReSS could use the HTTP PUT request we decided not to, because there currently is little support for it in browsers and servers. Other rarely used requests such as

CHECKOUT, CHECKIN and UNLINK are also avoided.

This paper is organized as follows: in Section 2 we introduce the problem of employing multi-author WWW servers and show some limitations of the current Web standards that make the authoring process and repository service difficult. In Section 3 we describe the authoring process and repository service using DReSS. We show how DReSS circumvents the difficulties mentioned in Section 2. Section 4 discusses issues related to collaboration, concurrency, versioning, authorization and network security.

2. Web Authoring and Publishing Limitations

Authoring a document typically goes as follows: the author generates a document using a word-processor (or other kind of editor), and decides where to store the document. Later revisions are done by retrieving the document using the word-processor and saving it once the editing is done. Pitkow and Jones [Pitkow-95] distinguish between:

1. *direct publishing*, which means that the author edits the documents on the server's file system. (This may generate difficulties in preventing users from reading temporary, unfinished versions of documents.)
2. *centralized publishing*, which means that authors can easily copy their documents from their home directory to the server area through a secure LAN or WAN.
3. *distributed publishing*, where authors can only upload documents to the server using e-mail or similar methods of file transfer.

The World-Wide Web belongs to the third category. In general the author has no login permission on or nfs access to the server area, and the server has no access to the author's machine either. Distributed publishing on the Web is made difficult because the Web has a stateless client-server architecture. A client (WWW browser) opens a connection to a server, requests a document (using the HTTP protocol) and receives a reply, hopefully containing the requested document, after which the connection is closed. One would like to be able to start an editor from within the browser, change the document and then send it back to the server. While most browsers support the start of an editing session, they do not (yet) support sending the document back to the server. Even if they were, there would still be no mechanism to tell the server that an editing session is started and then to prevent other readers from also editing this document. The HTTP protocol does not (or no longer) supply a "LOCK" request, which could be used to notify the start of a session. It has a largely obsolete CHECKOUT request, which locks the document and then sends it to the browser. CHECKOUT is not helpful if we wish to avoid retrieving the document a second time. Also, CHECKOUT is not supported by the current generation of servers. Using the inverse of CHECKOUT or LOCK (CHECKIN and UNLOCK) should be restricted to the session in which the CHECKOUT or LOCK was initiated. Keeping track of the session should be done by the server, because it is possible that the browser has exited before requesting a CHECKIN or UNLOCK. Leaving "dangling locks" on documents is not acceptable.

The creation of new documents is even more problematic. After loading the document into the WWW browser as a local file one would like to assign a valid URL to it and then send it to the server. Changing the URL of a loaded document is not possible in the current generation of browsers. Changing the URL could be done using a form, but including the document in the same form is also not yet possible in most browsers.

In order to implement editing sessions in the stateless WWW we need extra information on both the client and the server side to remember which client is engaged in which session.

On the server side files generated by CGI-scripts can be used for this purpose. On the client side the only way to make the browser remember something (without showing it to the reader) is to put it in a hidden field in a form. This hidden information is automatically sent back to the server with the next request. By sending this information back and forth the server can associate each request to the appropriate session.

When the editing is done the problem arises of how to send the document back to the WWW-server. A common approach is to connect the client and server machines in such a way that (a CGI-script on) the WWW-server can simply read or even serve the document from the author's home directory (or a subdirectory thereof). To enable this features, servers translate the "~user" directory name to a "public" subdirectory of the user. Another way to generate a similar effect is to create (symbolic) links from the server space to other (nfs-mounted) directories. Serving document from user directories has a number of serious drawbacks:

- When a WWW server has the ability to read files outside the server space (directory tree) the information outside that space might be attacked by cracking the server.
- The documents used by the WWW server are no longer nicely grouped together and owned by a single account, making it more difficult for the Webmaster to control which documents (and possibly also CGI scripts) are available through the WWW server.
- Because the documents are owned by the author and stored in the author's directory this addressing scheme is less suitable for collaborative authoring.
- When the authors are located remotely, and maybe even on machines that are not necessarily always powered on or reachable, it becomes difficult for the server to maintain nfs or other access to the author's machine.

Moving the documents to the server, instead of serving from the author's directory, can be done by means of a CGI-script that either mounts the author's directory and then copies the files or that uses a file transfer protocol (like ftp or http) to do the copying. Mounting the author's directory has similar drawbacks as serving from that directory. Using ftp or http requires that an ftp or WWW server is installed and running on the author's machine. Again, just like when serving from the author's directory, running such a server makes the author's machine (and files) prone to outside attacks.

A final range of problems with Web authoring is authorization. When we get the server to actually receive the documents from the authors it will install them in whichever directory the author requests (and is authorized to). Once this is done the documents become files owned by the server. Thus, a separate database is needed to remember the names of the authors and the access rights selected by the authors for their documents. With HTML documents it would be possible to store that information as meta information in the header, but with other files, like images and sound fragments, this is not possible. Structured files can be used to implement this *session database.* Using a real database system has its advantages, but makes porting the repository system more difficult.

When an author wishes to edit (or create) a document the server has to determine the identity of that author. Currently the Web provides no secure way to verify an identity. Like for read access, username/password combinations can be used, but WWW browsers do not send passwords over the Internet in a secure way. Passwords are encoded but not encrypted. They may be intercepted. (For telnet and rlogin connections the situation is even worse, but largely ignored by most Internet users.) An additional identity verification is possible using the RFC931 protocol. Some WWW servers offer the possibility to verify the identity of the sender of an http request by contacting the RFC931 server on the sender's machine. However, that client machine may have a bogus RFC931 server that lies about

the identity, or even no such server at all. The http protocol makes it also possible for the WWW browser to send the requester's identity along with the request. Most browsers do not support this feature, but even if they did this identity is easily faked by an intruder. Making password communication secure in the WWW goes beyond the scope of this paper. It is needed for read access to restricted WWW documents as well as for authoring and publishing. While we cannot (yet) make the communication between the authors and the WWW server secure, we must do our best to restrict the impact of a successful attack to only the documents on the WWW server.

3. DReSSing the Web: A Simple Document Repository System

The DReSS system supports creating new documents and updating or deleting existing ones. It can transparently restart a session after (accidentally or deliberately) exiting the WWW browser. We will concentrate on the creation and update problem. Deletion is fairly straightforward. The creation of a new document and the modification of an existing one go as follows:

1. First the **startup form** is requested.
2. In this form one enters the author's name, password, a (partial) URL for the document on the server, pathname of the document on the local machine, and optionally some access control information.
3. When this form is submitted an **action form** is generated which has an EDIT, COMMIT, VIEW and ABORT button. The ABORT button can be used to cancel a session when one decides not to create, change or delete the document after all.
4. When the EDIT button is pressed the document is retrieved from the WWW server, and passed on to the appropriate editor or word processor (which is started automatically). When creating a new document the editor is started without attempting to retrieve the document first.
5. When the COMMIT button is pressed the (local) document is transmitted to the WWW server. The actual transmission takes place invisibly in the background.
6. The VIEW button is used to view the final result after updating the document on the server. After pressing this button one only gets the document if the commit is complete. Otherwise a message is displayed that the commit process is still in progress.

We now look at some of these steps in detail.

Starting a DReSS session

The **DReSS startup form**, with example input is shown in figure 1. (It is normally preceded by a glossy banner identifying the WWW-server for which DReSS offers its repository service.) It lets you select whether you want to create a new document or modify or delete an existing one.

DReSS will allow only one user to create, edit or delete a document at the same time. Therefore, the URL of the document (as entered on the startup form) can be used as a session identifier. In the example the URL of the document to be created or updated is only partial: it is not necessary to specify the protocol and hostname, since only the http protocol is supported and the current (initial) version of DReSS can service only one WWW-server, which must be the one containing the startup form. If a full URL is given, only the "path" part is used for the session id.

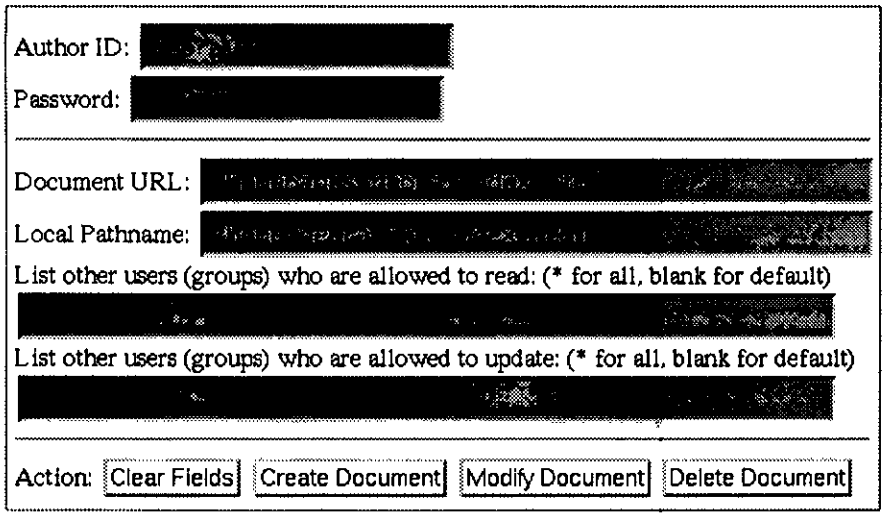

Figure 1: DReSS startup form.

DReSS allows only the creator of a document to change the access rights. The default action for existing documents is not to change the access rights. For a new document the default is to let everyone read the document and allow only the creator to update and delete the document.

After pressing the "Create Document" or "Modify Document" button a second form is displayed, called the **action form**, containing the EDIT, COMMIT, VIEW and ABORT buttons. A CGI-script on the WWW server creates an object (a file) which is associated with the session (i.e. with the document's name), and which contains all the information about the document, as given in the startup form. It also *checks out* the document, disabling other users from editing the same document until a commit or abort is performed. The same user is still allowed to create a new session for the same document. This provides a "silent" way to resume a session that was interrupted because the WWW browser was exited. The generated action form contains the session id (document name) in a hidden field. It also contains the author's identity and encoded (but not encrypted) password, both also in hidden fields.

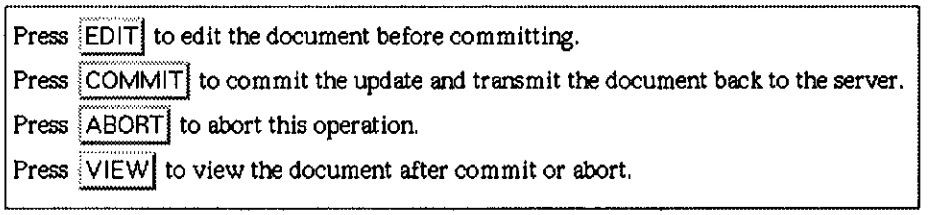

Figure 2: DReSS action form.

The EDIT procedure

Getting an existing document into the appropriate editor on the author's machine is done by means of a special mime type. The author has to bind this mime type to a so called *external viewer* which in this case is a small wrapper program that moves the document to the desired place (the local pathname given in the startup form) and starts the appropriate editor. In case the document is to be created, not updated, the editor is started with the local pathname, which can be an existing file or a name of a document which is still to be written. Note that binding mime types to external viewers is usually done in the user's mailcap file, but the Webmaster may decide to do the binding in a system-wide mailcap file. Different mime-types can be used (invented) for calling appropriate editors for different document formats.

Because the EDIT procedure is implemented using a special mime type the WWW-browser does not alter its display after invoking the external viewer. It still displays the EDIT, COMMIT, VIEW and ABORT buttons.

The EDIT button is only present because most WWW browsers do not understand multipart mime-files. After completing the startup form, a reply containing the action form as one part, and the document as another part would eliminate the need for an EDIT button. The document would be immediately passed on to the appropriate editor.

Committing updates on the WWW server

When the author is satisfied with the contents of the document (and has saved it on the local machine, in the specified local pathname) the document needs to be transmitted to the WWW server. The browser cannot perform this task itself, since it has passed on the document to the editor, and has subsequently forgotten about it. In the future some browsers may have the ability to load a local file, assign a URL to it and then send it to the WWW server (possibly using the HTTP PUT method). For now we will assume that an auxiliary program is needed to perform this transfer. In any case the initiative for the transfer must come from the author's machine, because the server has no rights to retrieve any information from the client's machine.

Activating the **transmission program** cannot be done directly by the browser. Most WWW browsers do not have the ability to start an external program on the client machine by pressing some button. We circumvent this shortcoming by binding the COMMIT button to a (link to a) CGI-script on the WWW server that generates a tiny document of yet another new mime type, which the author has to bind to the auxiliary transmission program (again, in the author's mailcap file, unless the Webmaster has added the mime type to the system-wide mailcap file). This CGI-script gets the session id, author id and password from the browser, because they are contained in hidden fields of the form containing the COMMIT button. Hence the script can associate the COMMIT request to the correct session, and verify that the session indeed belongs to this author, and also that the correct password was supplied.

The transmission program on the author's machine gets the session id, local pathname, author id and password from the CGI-script and constructs an HTTP POST request, containing the session id, author id and password for verification, followed by the document itself. This POST request activates the **commit CGI-script** which first verifies the session id, author id and password, and then *checks in* the document on the server (i.e., moves it in place and enables writing by others again).

The complicated procedure invoked by pressing the COMMIT button can be easily avoided by invoking the transmission program from the wrapper that activates the editor. However, this would violate the requirement that each action must be initiated from the browser. Also, the author may wish to abort the session after editing, i.e. to not update the document on the server after all. Sending the document back to the server unmodified may not be acceptable as a no-operation, in case the server is running an automatic versioning system that records every update request. Also, the changed modification date on the server would suggest that the document was altered, while in fact it was not.

In case the session is aborted instead of committed, an **abort CGI-script** is called which cancels the checkout (i.e. which simply removes the write-lock). Here an HTTP UNLOCK

request would have been useful if available. In the implementation of DReSS the commit and abort CGI-scripts are actually the same.

The VIEW process

The transmission program can operate silently and invisibly in the background. As a consequence the author does not know when the commit is actually completed. This is important in a PC and modem environment where the author may wish to power down the PC or to disconnect the modem after the commit. Pressing the VIEW button activates a CGI-script on the server that returns the document if the commit is completed, and a warning message (and another VIEW button) if the transmission is still going on.

It would be nice if the document were simply shown by the browser as a result of pressing the COMMIT button. After transmitting the document the commit CGI-script could trigger the server to send the document to the browser without the browser actually asking for it. Such a process is known as the experimental *server push* procedure. DReSS doesn't use this because it is not standard.

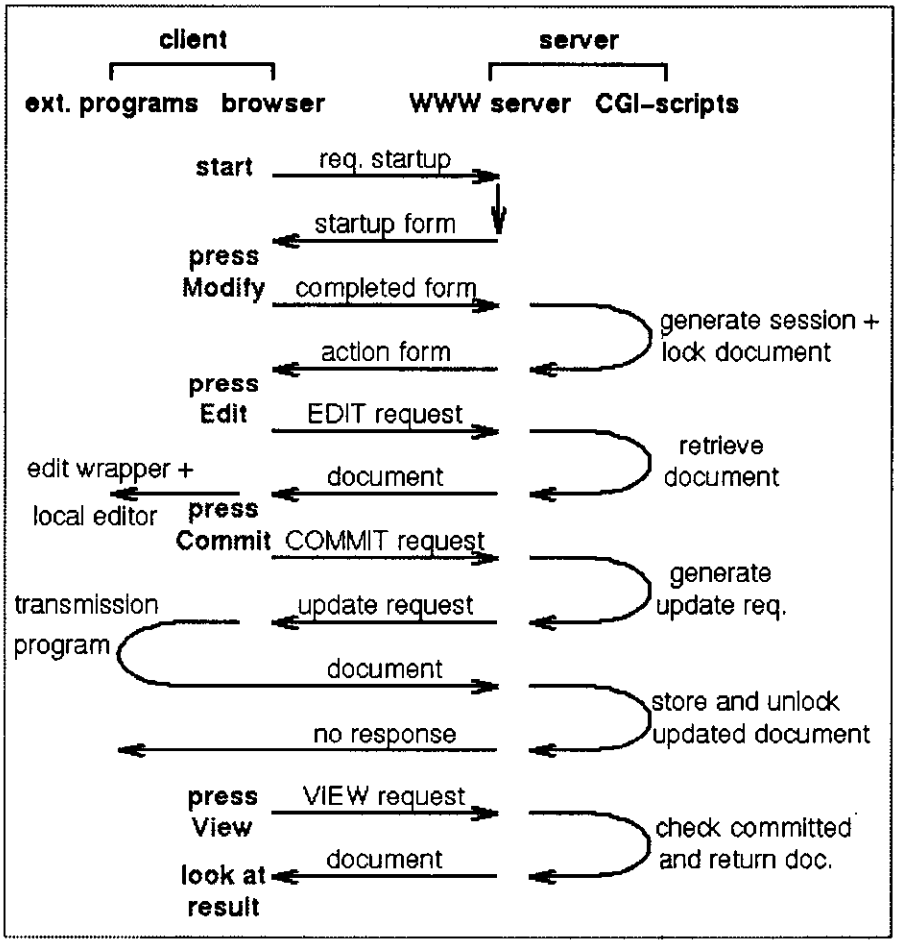

Figure 3: Communication for updating a document.

Figure 3 shows the entire communication between the author's machine and the WWW server, for the case where an existing document needs to be updated. While this communication looks (and is) complicated, the dialog between the author and the system is really very simple. (For color readers: the author's actions are written in blue.)

4. Concurrency, Collaboration and Security

Concurrency

When DReSS is used to generate documents on a WWW server all documents are owned by the same account (determined by the server configuration file). DReSS maintains a database to remember the owner and access rights for every document. In the initial implementation the database contains a separate file for each document. By inspecting that file a CGI-script can find out whether a document is checked out (locked) or not. Because near-simultaneous attempts to start more than one session on the same document will probably not occur frequently simple file locking on the session files is used. Updates to the session files only take a fraction of a second, causing very little delay when a CGI-script has to wait in order to find out whether a document is checked out (locked) or not. If the document is locked DReSS will issue a message telling the user which author has checked out that document.

Collaboration

DReSS is intended for multi-author WWW servers. Authors may not be working on the same document most of the time, and when they are, they will probably not find documents locked often, because of the hypertext nature of the Web. The basic hypertext principle is that authors write their own small documents and create links to each other's documents. The large number of small documents together form a *hyperdocument*. Thus, the problem of coping with multiple authors is less a problem of dealing with concurrency and locking than a problem of ensuring that the documents and their links together form a sensible hyperdocument. In order to avoid dangling links (links pointing to documents that do not yet, or no longer exist) DReSS needs to disallow the deletion of a document as long as at least one other document (on the same server) points to it. Also, it needs to give warnings when a new document is created containing dangling links, or when new dangling links are introduced by modifying a document.

The "Intelligent Publishing Environment" of Pitkow and Jones [Pitkow-95] allows the deletion of a document when there are still links to it (within the same WWW server). It automatically removes the links from these other documents. We consider this behavior unacceptable in general. The user removing a document may not have permission to alter these other documents. Also, the modification that is needed in these other documents for them to still make sense may be much more difficult than the simple removal of a link. This modification has to be done by the authors of these documents.

Collaboration between authors working on the same document generally leads to more modifications, sometimes cancellations, than documents written by a single author. In order to be able to quickly undo changes that were committed, a versioning or source-code control system can be used. The initial implementation of DReSS does not yet support versioning. This feature will be added by employing the RCS system.

Security

DReSS follows an unusual approach towards security. The Web only provides *basic* security, which means that user identities and passwords can be used, but passwords are not transmitted in encrypted form over the Internet. As a result, it is currently not possible to make DReSS secure in the sense that the documents on the WWW server are well protected and that sessions cannot be broken into by intruders. For this reason the

passwords used by DReSS should never be the same as the passwords that author's use to login onto any computer.

The main security issue for DReSS is making sure that the computing environment of the author is not affected by the vulnerability of the WWW server. When a server is configured carefully, its only risk is that an intruder might alter the documents on the server. By shielding a WWW server in a *chroot* environment it becomes absolutely impossible for the WWW server to access, let alone alter any file outside the WWW server space. Server protection can be further enhanced by not offering any inherently dangerous programs in that shielded environment, such as a shell and the popular Perl interpreter. The WWW server at our department (`www.win.tue.nl`) runs in such a shielded environment. Both the NCSA and CERN servers run on the same document tree. Even though our site is a popular target for would-be crackers, none of the security problems discovered in WWW software have been successfully exploited by crackers attacking our server. But even if they would have been, only the files visible to the WWW server could have been altered. In order to make DReSS usable in such an environment, all CGI-scripts are written in C.

The protection of the author's machine is first guarded by delegating all initiative in DReSS to the author's machine. The WWW server never tries to contact the author's machine unless the author specifically asks it to do so. This is another good reason for not using the experimental server-push feature. Apart from sending information to the author's browser, there are only two other operations that might threaten the author's machine: the two auxiliary programs that are activated upon request by the author, but through the server. A bogus, cracked server, could try to trick the auxiliary programs into performing unauthorized actions. Since these programs reside on the author's machine, not on the server, the programs themselves cannot be modified by an intruder who has cracked the server. The binding between mime-types and "external viewers" is also done on the author's machine, so the server cannot trick the browser into executing different programs. The auxiliary programs may be tricked into undesirable behavior however:

- The wrapper program for the editor may overwrite the document whose local filename it receives from the server. In order to protect documents from being overwritten by a malignant server the wrapper program should be run as a user with little or no permissions (e.g. it could be run setuid nobody). As such it can only write into publicly writable directories such as /tmp. Also, the wrapper program can be run in a shielded chroot environment, where it can do no harm.
- The transmission program (sending the updated document to the server) can be tricked into sending a different document. Again, by running the wrapper program with minimal permissions (as nobody) it can only be tricked into sending documents that can be accessed by every user on the author's system. By running the transmission program in a shielded chroot environment (preferably the same as for the wrapper) the program can be further restricted to only being able to transmit documents that appear in that shielded environment. By doing so one can for instance prevent the program from transmitting the system's password file.

On single-user operating systems (for instance DOS and Windows) the vulnerability of the author's machine can be reduced by restricting the filenames that can be used by the auxiliary programs. Both the wrapper and the transmission program could be restricted to only work with files in `C:\TMP` for instance.

5. Discussion and Conclusions

DReSS is a tool that makes multi-user publishing on the Web easier. The four main goals it

achieves are:

- Ease of use, through forms and CGI-scripts.
- Simple server maintenance.
- It works with standard WWW browsers and servers, and thus shows that a document repository system can be built without adding features to the browser, server, HTML or HTTP protocol.
- Minimal risks for the authors by limiting the security problem area to the server's directory tree.

The use of standard browsers and servers and the ease of use through forms cause a lot of communication between the author's machine and the server. Suggestions for future improvements to the WWW architecture are:

- The ability to send a LOCK request for the url of the currently displayed document by pressing a button on the browser (outside the document window). A second retrieval of the document in order to start a registered editing session could then be avoided. When receiving the LOCK request the server should ask for the author's identification and password.
- The ability to change the url of a loaded document in the browser, or the ability to include a file in a form. (The former is preferred, but the latter is probably going to become popular first.)
- Support for the CHECKIN request and a button on the browser for sending the current document back to the server. The browser must send the author's identification and password (obtained when requesting the LOCK) along with the CHECKIN (or PUT) request. Otherwise the server would generate a "not authorized" reply and ask for retransmission of the request with authorization. Since the CHECKIN or PUT request contains the document this would then cause the document to be transmitted a second time.

DReSS is publicly available through our anonymous ftp server. It resides in the directory at the url `ftp://ftp.win.tue.nl/pub/infosystems/www/dress/`. The initial implementation concentrates only on the document repository aspect. Future extensions will include link verification (possibly done by means of MOMspider [Fielding-94] or by EIT's tool [McGuire-95]) and version control (probably using RCS [Bentley-95] best resembles DReSS. BSCW offers version control and a graphical interface to the shared workspaces, which DReSS doesn't. BSCW modifies the URL's provided by authors, making it difficult to predict what the URL of a document (still to be created) will be. DReSS preserves the URL's given by the authors. This property, and the possibility to upload an entire directory at once, make DReSS better suited for publishing hypertext documents consisting of many small text fragments that are linked together.

References

[Andrews-95]
Andrews, K., Kappe, F. and Maurer, H., Serving Information to the Web with Hyper-G, *Third International WWW-Conference*, Darmstadt, 1995,
URL: http://www.igd.fhg.de/www/www95/papers/105/hgw3.html.

[Bentley-95]
Bentley, R., Horstmann, T., Sikkel, K., Trevor, J., Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system, *Fourth International WWW-Conference*, Boston, 1995.

[Berners-Lee94]

Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H., and Secret, A., The World-Wide Web, *Communications of the ACM*, 37(8): 76-82, 1994.

[Carl-Davis]
Davis, C., HTML Editor Reviews,
URL: http://www.interaccess.com/users/cdavis/edit_rev.html

[Fielding-94]
Fielding, R., Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web., *First International WWW Conference*, Geneva, 1994,
URL: http://www.cern.ch/PapersWWW94/fielding.ps.

[Lavenant-94]
Lavenant, M.G. and Kruper, J.A., The Phoenix Project: Distributed Hypermedia Authoring, *First International WWW Conference*, Geneva, 1994,
URL: http://www.cern.ch/PapersWWW94/j-kruper.ps.

[McGuire-95]
McGuire, J., Verify Web Links,
URL: http://wsk.eit.com/wsk/dist/doc/admin/webtest/verify_links.html.

[Nebel-95]
Nebel, E., Masinter, L., Form-based File Upload in HTML, *Internet-Draft,*
Current URL: ftp://ds.internic.net/internet-drafts/draft-ietf-html-fileupload-02.txt.

[Pitkow-95]
Pitkow, J.E. and Jones, R.K., Towards an Intelligent Publishing Environment, *Third International WWW-Conference*, Darmstadt, 1995,
URL: http://www.igd.fhg.de/www/www95/papers/72/publish/publishing.html.

[Tichy-85]
Tichy, W.F., RCS-A System for Version Control, *Software-Practice & Experience*, 15(7), pp. 637-654, 1985.

Computing Science Reports

Department of Mathematics and Computing Science
Eindhoven University of Technology

In this series appeared:

93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.
93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness, p. 24
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.

93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.
93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.

94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doombos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.
94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.

94/39	A. Blokhuis T. Kloks		On the equivalence covering number of splitgraphs, p. 4.
94/40	D. Alstein		Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks D. Kratsch		Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet J.J. Vereijken		Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse M. Bijsterveld		Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
94/44	E. Brinksma R. Gerth W. Janssen S. Katz M. Poel C. Rump	J. Davies S. Graf B. Jonsson G. Lowe A. Pnueli J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben		Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46	R. Bloo F. Kamareddine R. Nederpelt		The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo F. Kamareddine R. Nederpelt		On Π-conversion in Type Theory, p. 12.
94/48	Mathematics of Program Construction Group		Fixed-Point Calculus, p. 11.
94/49	J.C.M. Baeten J.A. Bergstra		Process Algebra with Propositional Signals, p. 25.
94/50	H. Geuvers		A short and flexible proof of Strong Normalazation for the Calculus of Constructions, p. 27.
94/51	T. Kloks D. Kratsch H. Müller		Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
94/52	W. Penczek R. Kuiper		Traces and Logic, p. 81
94/53	R. Gerth R. Kuiper D. Peled W. Penczek		A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
95/01	J.J. Lukkien		The Construction of a small CommunicationLibrary, p.16.
95/02	M. Bezem R. Bol J.F. Groote		Formalizing Process Algebraic Verifications in the Calculus of Constructions, p.49.
95/03	J.C.M. Baeten C. Verhoef		Concrete process algebra, p. 134.
95/04	J. Hidders		An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.
95/05	P. Severi		A Type Inference Algorithm for Pure Type Systems, p.20.
95/06	T.W.M. Vossen M.G.A. Verhoeven H.M.M. ten Eikelder E.H.L. Aarts		A Quantitative Analysis of Iterated Local Search, p.23.
95/07	G.A.M. de Bruyn O.S. van Roosmalen		Drawing Execution Graphs by Parsing, p. 10.
95/08	R. Bloo		Preservation of Strong Normalisation for Explicit Substitution, p. 12.
95/09	J.C.M. Baeten J.A. Bergstra		Discrete Time Process Algebra, p. 20
95/10	R.C. Backhouse R. Verhoeven O. Weber		Math∫pad: A System for On-Line Prepararation of Mathematical Documents, p. 15

95/11	R. Seljée	Deductive Database Systems and integrity constraint checking, p. 36.
95/12	S. Mauw and M. Reniers	Empty Interworkings and Refinement Semantics of Interworkings Revised, p. 19.
95/13	B.W. Watson and G. Zwaan	A taxonomy of sublinear multiple keyword pattern matching algorithms, p. 26.
95/14	A. Ponse, C. Verhoef, S.F.M. Vlijmen (eds.)	De proceedings: ACP'95, p.
95/15	P. Niebert and W. Penczek	On the Connection of Partial Order Logics and Partial Order Reduction Methods, p. 12.
95/16	D. Dams, O. Grumberg, R. Gerth	Abstract Interpretation of Reactive Systems: Preservation of CTL*, p. 27.
95/17	S. Mauw and E.A. van der Meulen	Specification of tools for Message Sequence Charts, p. 36.
95/18	F. Kamareddine and T. Laan	A Reflection on Russell's Ramified Types and Kripke's Hierarchy of Truths, p. 14.
95/19	J.C.M. Baeten and J.A. Bergstra	Discrete Time Process Algebra with Abstraction, p. 15.
95/20	F. van Raamsdonk and P. Severi	On Normalisation, p. 33.
95/21	A. van Deursen	Axiomatizing Early and Late Input by Variable Elimination, p. 44.
95/22	B. Arnold, A. v. Deursen, M. Res	An Algebraic Specification of a Language for Describing Financial Products, p. 11.
95/23	W.M.P. van der Aalst	Petri net based scheduling, p. 20.
95/24	F.P.M. Dignum, W.P.M. Nuijten, L.M.A. Janssen	Solving a Time Tabling Problem by Constraint Satisfaction, p. 14.
95/25	L. Feijs	Synchronous Sequence Charts In Action, p. 36.
95/26	W.M.P. van der Aalst	A Class of Petri nets for modeling and analyzing business processes, p. 24.
95/27	P.D.V. van der Stok, J. van der Wal	Proceedings of the Real-Time Database Workshop, p. 106.
95/28	W. Fokkink, C. Verhoef	A Conservative Look at term Deduction Systems with Variable Binding, p. 29.
95/29	H. Jurjus	On Nesting of a Nonmonotonic Conditional, p. 14
95/30	J. Hidders, C. Hoskens, J. Paredaens	The Formal Model of a Pattern Browsing Technique, p.24.
95/31	P. Kelb, D. Dams and R. Gerth	Practical Symbolic Model Checking of the full μ-calculus using Compositional Abstractions, p. 17.
95/32	W.M.P. van der Aalst	Handboek simulatie, p. 51.
95/33	J. Engelfriet and JJ. Vereijken	Context-Free Graph Grammars and Concatenation of Graphs, p. 35.
95/34	J. Zwanenburg	Record concatenation with intersection types, p. 46.
95/35	T. Basten and M. Voorhoeve	An algebraic semantics for hierarchical P/T Nets, p. 32.
96/01	M. Voorhoeve and T. Basten	Process Algebra with Autonomous Actions, p. 12.