Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web

Roy T. Fielding
fielding@ics.uci.edu
Department of Information and Computer Science
University of California, Irvine CA 92717-3425
April 17, 1994

Abstract

Most documents made available on the World-Wide Web can be considered part of an infostructure — an information resource database with a specifically designed structure. Infostructures often contain a wide variety of information sources, in the form of interlinked documents at distributed sites, which are maintained by a number of different document owners (usually, but not necessarily, the original document authors). Individual documents may also be shared by multiple infostructures. Since it is rarely static, the content of an infostructure is likely to change over time and may vary from the intended structure. Documents may be moved or deleted, referenced information may change, and hypertext links may be broken.

As it grows, an infostructure becomes complex and difficult to maintain. Such maintenance currently relies upon the error logs of each server (often never relayed to the document owners), the complaints of users (often not seen by the actual document maintainers), and periodic manual traversals by each owner of all the webs for which they are responsible. Since thorough manual traversal of a web can be time-consuming and boring, maintenance is rarely or inconsistently performed and the infostructure eventually becomes corrupted. What is needed is an automated means for traversing a web of documents and checking for changes which may require the attention of the human maintainers (owners) of that web.

The Multi-Owner Maintenance spider (MOMspider) has been developed to at least partially solve this maintenance problem. MOMspider can periodically traverse a list of webs (by owner, site, or document tree), check each web for any changes which may require its owner's attention, and build a special index document that lists out the attributes and connections of the web in a form that can itself be traversed as a hypertext document. This paper describes the design of MOMspider and how it was influenced by the nature of distributed hypertext maintenance and requirements for the good behavior of any web-traversing robot. It also includes discussion of the efficiency requirements for maintaining world-wide webs and proposed changes to HTML and HTTP to support distributed maintenance. The paper concludes with a short description of MOMspider's future and pointers to its freeware distribution site.

1. Introduction

The World-Wide Web (WWW) can be described in many ways. From an organizational perspective, it is an initiative aiming to give universal access to a world of documents [Hughes94]. Technically, it is a distributed object management system designed for information retrieval via textual relations. From a practical viewpoint, however, the WWW is a synergistic combination of three technologies: a means for providing potentially useful information in such a way that it can be accessed by distributed (and sometimes distant) users; a means for users to access information stored at distributed sites without requiring knowledge of the underlying access mechanism; and a means for structuring information such that it can be discovered, retrieved and viewed by those who would find it useful. These three technologies are enabled by WWW server and client programs and a set of proposed standards which allow them to communicate, to identify information objects, and to structure and view the information such that it can be traversed via hypertext links.

The Hypertext Transfer Protocol [HTTP] is used by distributed servers to communicate with each other and with a variety of client applications. Many of these clients are also capable of communicating with other information services, such as FTP, Gopher, and WAIS. A client can send commands to any server accessible via a TCP/IP connection. The command is usually a request for transfer (GET) of an information object, which is then displayed (or saved) locally by the client.

An information object is identified by a Uniform Resource Locator [URL] which, in its canonical form, can include the access scheme (e.g. http, gopher, telnet, etc.), an IP/hostname address and TCP port for the server location (e.g. www.ics.uci.edu:80), and a name recognizable by the server as representing that object (usually in the form of a relative file pathname). For example, the full URL for this document is:
<http://www.ics.uci.edu:80/WebSoft/MOMspider/WWW94/intro.html>. In most cases, the URL is embedded in other documents as a hypertext reference (link) associated with some meaningful text pointer (anchor). Viewed graphically, these links form a hypertext "web" of related information.

The Hypertext Markup Language [HTML] is used to structure information such that it can be readily displayed by viewing clients. Because these clients exist on heterogeneous platforms and may vary in their rendering abilities, HTML emphasizes the description of content and structure rather than form. Of primary importance is HTML's ability to define portions of a document as being hypertext — pieces of text or images which are linked (via anchor references) to other documents. End-users of the WWW interact with a viewing client (such as NCSA's Mosaic) by reading documents and traversing to related documents by selecting the hypertext anchors. In addition to the main body of information, HTML documents can contain special header information (metainformation) such as the document's title, expiration date, and version.

2. The Maintenance Problem

All documents in the World-Wide Web can be considered part of an *infostructure* — an information resource collection with a specifically designed structure [Tilton93]. An infostructure is created any time an amount of information is organized in a useful manner, such as a dictionary or this paper (Figure 1). In the WWW, infostructures can include a wide variety of information sources, in the form of interlinked documents, and each document may be shared by multiple infostructures. In turn, an infostructure can be contained within higher-level infostructures, just as the table of contents is contained within the overall structure of this paper. The World-Wide Web as a whole can be considered the ultimate online infostructure.

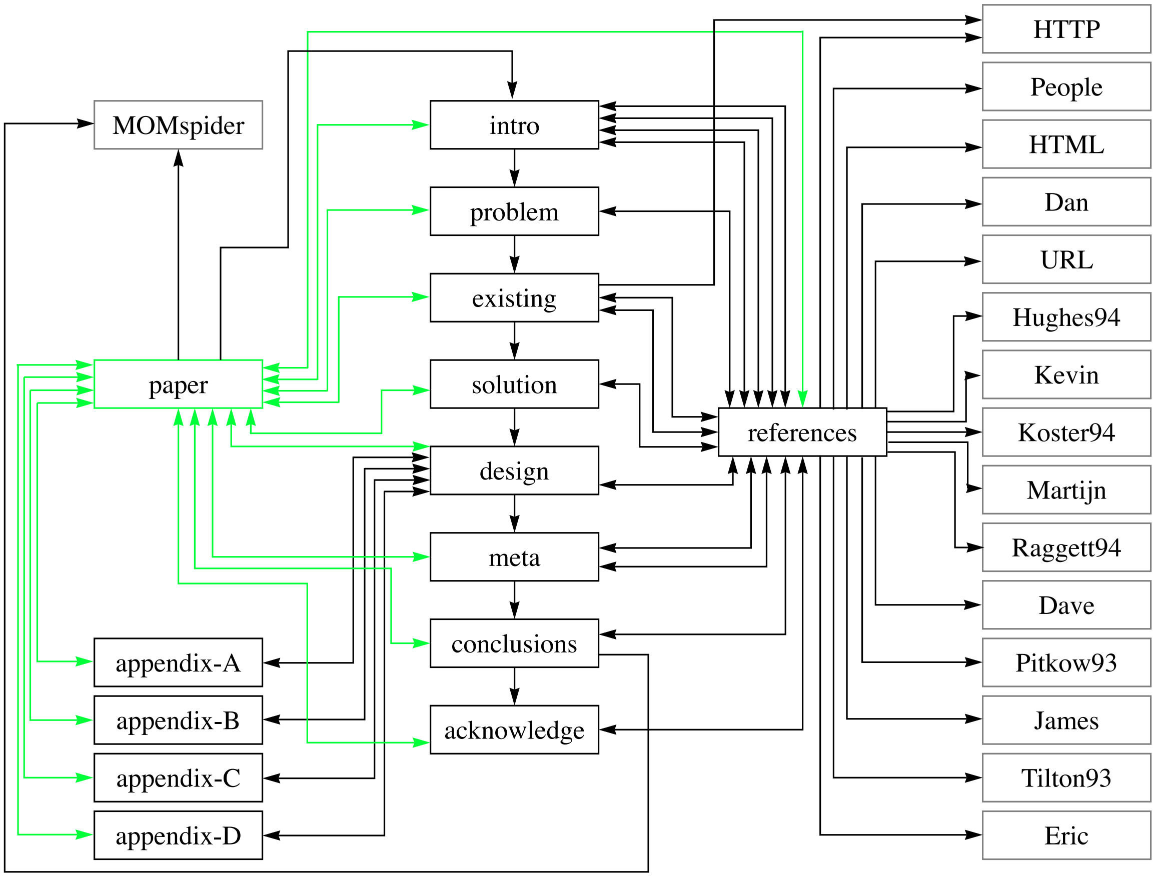

Figure 1: The infostructure for this paper. The table-of-contents substructure is in green (gray).

A good infostructure doesn't just exist; it exists by **design**. The documents are linked specifically to allow readers to navigate through the information presented. If that information is presented as hypertext, the reader should not be constrained by the linear nature of traditional documents. Therefore, hypertext authors attempt to provide as many routes of navigation as are conceived to be useful for potential readers. Each hypertext link creates a dependency between the source and destination documents.

In a world of information, most of the best sources are maintained at sites other than the author of a particular infostructure. Furthermore, such information is rarely static, consisting of living documents maintained by the owner (usually, but not necessarily, the original document author) at those distributed sites. Rather than copy the contents, the WWW enables infostructures to be composed simply by referencing the desired object's URL within the guiding text of any HTML document. In addition to providing document reuse, this allows new routes of navigation to be developed by the consumers of that information, independent of the design considerations of existing infostructures.

Unfortunately, the flexible and dynamic nature of the World-Wide Web leads to a glaring problem — one that should already be familiar to many information providers. Most infostructures are time dependent. They reflect the navigational desires and information contents that were available at the time of their design. However, as living documents change and new documents are added, the resulting structure may vary from that intended. Documents may be moved or deleted, referenced information may change, and hypertext links may become broken. A living infostructure must therefore be actively maintained in order to prevent structural collapse.

The maintenance problem is further exacerbated by that of scale. As it grows, an infostructure becomes complex and difficult to maintain. Each new document may result in an exponential increase in document links. A large infostructure will therefore require many human maintainers, particularly if it is spread over multiple distributed sites. However, this also compounds the problem — as individual maintainers make changes to support their own part of the infostructure, those same changes may break the structures maintained by others.

3. Limitations of Existing Maintenance Methods

Maintenance of World-Wide Web infostructures currently relies upon the diligence of each document owner and a small stream of maintenance information. The primary source of that information is from users, commonly in the form of complaints from those who encountered broken links or malformed documents. However, information providers cannot rely on user complaints. Most users of the Web are very tolerant of the errors encountered — after all, since the information is often provided by volunteers, many users feel that it would be impolite or disrespectful to point out any problems. Furthermore, many documents do not explicitly indicate to whom a complaint should be sent. Finally, even when a complaint is received, it is often directed to the wrong person (i.e. the one maintaining the source of a link instead of its destination).

A second source of maintenance information is provided by the logfiles of each server. The server records (or attempts to record) each document request and, if an error occurred, the nature of that error. Such information can be extremely useful for identifying requests for documents that have moved and those that have misspelled URLs. However, only the server managers have access to that information. The error is often never relayed to the document maintainer, either because it is not recognized as a document error (users frequently mistype document URLs when accessing them via an "Open..." dialog) or because the origin of the error is not apparent from the error message. Although this situation will improve as WWW clients begin using the Referer header in requests [HTTP], log information will never be sufficient to cover maintenance needs. Logs cannot reveal failed requests that never made it to the server, nor can they support preventive maintenance and problems of changed document content.

Although rarely applied, static analyzers of document infostructures can be a third source for maintenance information. One such tool is the html_analyzer [Pitkow93]. It can examine a local infostructure and validate the document links for accessibility, completeness, and consistency. This type of information could be very useful for infostructure maintenance, but tends to be applied more as a means for one-time verification than as a regular maintenance process. Also, it fails to provide adequate support across distributed infostructures and for situations in which the document contents are outside the control of the program user.

Given these limitations, the only existing method for performing adequate maintenance of WWW infostructures is the brute-force one of periodic manual traversals, by each owner, of all the webs for which they are responsible. Such traversals are repetitive, time-consuming, and boring — a guaranteed recipe for human inattentiveness. They also require a great deal of duplication of effort for overlapping infostructures, as each owner retests the same link destinations. The result is that maintenance is rarely or inconsistently performed and the infostructure eventually becomes corrupted. What is needed is a means for automating this traversal process such that a human maintainer (owner) of an infostructure need only investigate documents which are likely to require maintenance effort — those that are known to have changed, expired, or which contain broken links.

4. Automated Traversal as a Maintenance Solution

Given that a means for automating the traversal process is desired, we need to define the requirements and limitations of such a solution. The primary requirement is that it improve the existing maintenance process by reducing the detrimental effects of human inattentiveness, duplication of effort, and distributed document ownership.

Manual traversal is both time-consuming and boring. Current WWW browsers are designed for the normal viewing process — they make no distinction between old documents and those that have recently changed, nor do they show the user a document's last-modification and expiration dates. In addition, their only method for testing a link is to actually request and transfer the document contents. This is so inefficient (particularly for sites with slow network connections) that many document owners avoid testing those links at all. Even when applied repetitively (as is required for consistent maintenance), manual traversal fails because no human being can remain consistently attentive during a repetitive, time-consuming, and boring process.

Fortunately, these are the characteristics for which automation is most effective. An automated traversal program can test a link without transferring the document contents by using the HEAD request method rather than the GET used by browsers [HTTP]. Provided that document metainformation is available in the response headers, such a program can also check for special conditions that would interest the infostructure owner, such as a recent Last-modified date or an approaching Expires date. Furthermore, the program can restrict its focus to the web's structure and not be distracted by the contents of each document.

With manual traversal, duplication of effort occurs because different infostructure owners don't see the results of others' traversals. World-Wide Web infostructures are encouraged to overlap (i.e. to reuse documents created for other infostructures). For example, just think of how many documents at your site reference the What's New With NCSA Mosaic document. If the owner of each infostructure independently checks each link with a HEAD request, the result would be a great deal of duplication, wasted network bandwidth, and an unnecessary load on the document servers. An automated traversal program should therefore be required to handle multiple infostructures, possibly maintained by different owners, and share its testing information across them.

Sharing maintenance information can also be beneficial in reducing the problem of distributed document ownership. Since the program is performing traversals for multiple owners, it needs to place the results where all can gain access. Surely the best place for such information is on the Web itself, in the form of HTML index documents generated for each infostructure. In this way, the document owners can make use of shared maintenance information even when they are not located at the site where the program is executed. It also allows a single site to perform the maintenance traversals for many others.

Unfortunately, no automated traversal program can completely solve the maintenance problem. A program cannot tell when a document's contents are changed such that they no longer represent the intentions of a given infostructure. Nor can a program, once it has discovered a broken link, determine why that link is broken or how to fix it. These tasks must still be performed by human maintainers. However, a traversal program can greatly ease the process by alerting the human maintainer and explicitly pointing to those documents that have changed and links that are broken.

Clearly, an automated traversal program would be useful for easing the maintenance of hypertext infostructures. We have developed the Multi-Owner Maintenance spider (MOMspider) for this purpose. MOMspider is a web-wandering robot that, given a list of instructions that details what infostructures to traverse, whom to notify for problems, and where to put the resulting maintenance information, will traverse each infostructure and fulfill all of the requirements listed above. The remainder of this paper will focus on the specific design of MOMspider, its capabilities and limitations, and proposed enhancements to HTML and HTTP which would further increase its usefulness.

5. MOMspider Design

The design of MOMspider focuses on fulfilling the requirements of multi-owner maintenance while at the same time minimizing its effect on World-Wide Web servers and network bandwidth. Because the MOMspider client is oriented toward maintenance issues in general, it also attempts to maximize the benefit to information providers while respecting any limits they may place on wandering robots.

5.1 Functionality

MOMspider gets its instructions by reading a text file that contains a list of options and tasks to be performed (an example instruction file is provided in Appendix A). Each task is intended to describe a specific infostructure so that it can be encompassed by the traversal process. A task instruction includes the traversal type, an infostructure name (for later reference), the "Top" URL at which to start traversing, the location for placing the indexed output, an e-mail address that corresponds to the owner of that infostructure, and a set of options that determine what identified maintenance issues justify sending an e-mail message.

For each task, MOMspider traverses the web from the specified top document down to each leaf node, where a leaf node is defined to be any information object which is not of document-type HTML (and thus cannot contain any further links) or which is outside the given infostructure. MOMspider determines the boundaries of an infostructure according to the task's traversal type: **Site**, **Tree**, or **Owner**. Site traversal specifies that any URL which points to a site (the pairing of hostname/IP address and port) other than that of the top document is considered a leaf node. Tree traversal specifies that any document not at or below the "level" of the top document is considered a leaf node, where level is determined by the pathname in the URL. Owner traversal specifies that any document beyond the top which does not contain an "Owner:" metainformation header equal to the infostructure name is considered a leaf node.

The maintenance information produced by each task is formatted as an HTML index and output to the file specified in the task instructions (an example of which is provided in Appendix B). The index contains the following maintenance information:

- Information regarding how and when the index was generated (i.e. program options and execution time);
- A hypertext link to the one prior version of the index document;
- The following for each non-leaf document accessible via the "top":
  - An anchor which links to the actual document;
  - Document header info (Title, Modification Date, Expires Date, etc.);
  - A list of all unique hypertext references made by the document, with each reference including:
    - The type of reference made (i.e. get, query, img, ftp, etc.);
    - An anchor which duplicates the reference;
    - Document header info if available (Title and Modification Date);
    - If the referenced object is within the current infostructure (i.e. not a leaf), then an additional anchor is provided to cross-reference jump to its own entry in the index document.
- A list of cross-reference anchors which point to interesting changes as reflected in the index entries.

MOMspider looks for four types of document change which may be of interest to the owner:

1. referenced objects which have redirected URLs (moved documents);
2. referenced objects which cannot be accessed (broken links);
3. referenced objects with recent modification dates; and,
4. owned objects with expiration dates near to the current date.

Each interesting item is placed in the closing cross-reference table and, if the corresponding option is requested, enclosed in a single e-mail message and posted to the owner at the task's completion.

5.2 Efficient Use of Network Resources

A key design constraint of MOMspider is that of efficiency — particularly in regards to network bandwidth usage. It would be irresponsible to develop a maintenance robot which overly taxed the limited resources of networks like the Internet. Therefore, MOMspider minimizes the load on network bandwidth by using the HEAD request wherever possible, keeping track of nodes that have already been tested, grouping multiple tasks within a single execution, and allowing the user to restrict the traversal of certain URLs.

Aside from the restrictions described above regarding the task's traversal type, MOMspider also enables the user to specify any URL prefixes which must always be avoided or leafed. These URL prefixes are listed in the system-wide or user avoid files (an example of which is provided in Appendix C). Each entry in the file includes the action (Avoid or Leaf), the URL prefix on which to apply that action, and an optional expiration date for the entry. This allows the user to completely avoid documents for which maintenance is not a concern or which could trap an unsuspecting spider (some forms of computational hypertext can have that effect).

5.3 Being Friendly to Service Providers

The second key design constraint for MOMspider is that it minimize its impact on information providers (destination servers) while at the same time maximizing the indirect benefits they receive from the traversal process. All HTTP requests are similar to:

```
HEAD /path HTTP/1.0
User-Agent: MOMspider/0.1
From: user@machine.sub.dom.ain
Referer: http://www.site.edu/current/document.html
```

This allows server maintainers to properly recognize the source of the request and, if necessary, place restrictions upon a particular spider. It also provides them useful information, including how to contact the person running the spider and what document contains the reference being tested.

As an additional precaution, MOMspider periodically looks for and obeys any restrictions found in a site's /RobotsNotWanted.txt document as per the standard proposed by Martijn Koster [Koster94]. Before any link is tested, the destination site is looked-up in a table of recently accessed sites (the definition of "recently" can be set by the user). If it is not found, that site's /RobotsNotWanted.txt document is requested and, if found, parsed for any restrictions to be placed on MOMspider robots. Any such restrictions are added to the user's avoid list and the site is added to the site table, both with expiration dates indicating when the site must be checked again. Although this opens the possibility for a discrepancy to exist between the restrictions applied and the contents of a recently changed /RobotsNotWanted.txt document, it is necessary to avoid a condition where the site checks cause a greater load on the server than would the maintenance requests alone. An example sites file is provided in Appendix D.

6. The Need for Visible Metainformation

MOMspider needs some method for obtaining the Owner, Last-modified date, and Expires date of any maintained documents. For efficiency reasons, this metainformation must be returned as HTTP headers in response to a HEAD request on that document [HTTP]. This means that existing HTTP servers must be changed to send those headers and some means must be provided for the server to obtain the relevant metainformation for each document it serves. There are three possible means of providing that information to the server:

1. A special server configuration table (or tables) which lists each served document and its corresponding metainformation.
2. Special server directories which mirror the names of served documents and contain the metainformation corresponding to those names.
3. Optional HTML META elements which can be embedded in the document and parsed by the server.

The first and second solutions are more efficient (for the server) and are applicable to both HTML and non-HTML documents. However, it adds an additional maintenance problem of keeping the metainformation table or files consistent with the corresponding documents. This may be alleviated by distributing the table or directories amongst the infostructure, as in special .metainfo documents in each server directory.

The third solution is less efficient for the server (because it must parse the HTML headers each time the document is requested) but is much more flexible and easier for distributed authors to maintain. Also, since MOMspider does not care about getting Owner information from non-HTML objects, this is the preferred solution for now.

For this purpose, the META element has been added to the HTML+ specification [Raggett94]. Each maintained HTML file would include optional META elements within the HEAD part of the document like the following:

```
<META header name="Owner" value="AnyOwnerAlias">
<META header name="Expires" value="Fri, 01 Apr 1994 00:00:00 GMT">
```

Unfortunately, this still does not solve the problem of getting HTTP servers to provide the parsing necessary to produce the actual headers. It is likely that this will only occur once it becomes clear how useful that information can be. For the meantime, MOMspider has been designed so as not to be dependent on that information and yet be able to make full use of it whenever it does become available.

7. Conclusions and Future Research

At the present time, the World-Wide Web is experiencing phenomenal growth. Our ability to sustain that growth will depend a great deal upon the manageability, and thus the maintainability, of the infostructures which make up the web. Existing maintenance methods are inadequate to support large infostructures, particularly when they span distributed sites or multiple owners. Failure to address this maintenance problem could result in the collapse of large portions of the Web and considerably reduce its usefulness for serious applications.

The Multi-Owner Maintenance spider (MOMspider) alleviates this maintenance problem by automating those tasks which are most tiresome for human maintainers. MOMspider can periodically traverse a list of webs (by owner, site, or document tree), check each web for any changes which may require its owner's attention, and build a special index document that lists out the attributes and connections of the web in a form that can itself be traversed as a hypertext document. Moreover, it does so in an efficient manner by sharing information across maintenance tasks, minimizing the use of network bandwidth, and allowing complex restrictions to be placed on its operation.

This paper has described the requirements for automated support of distributed hypertext maintenance and how those requirements have influenced MOMspider's design. The MOMspider program, including source code, is freely available and can be found at the following distribution sites:

```
http://www.ics.uci.edu/WebSoft/MOMspider/
ftp://liege.ics.uci.edu/pub/arcadia/MOMspider/
```

The future applications for maintenance tools like MOMspider will depend on the availabilty of document metainformation. For instance, the index that MOMspider generates is not significantly different than that used to index sites for search engines like ALIWEB and Archie [ED92] — all that need be added is the specific metainformation that is useful to those engines. Similarly, since MOMspider already does the work of traversing an infostructure, it would be possible to have it generate a graphical description of the webspace usable by browsing clients.

As the Web continues to grow, more applications will be found for programs that can properly traverse infostructures without becoming a burden on limited network and server resources. It is hoped that MOMspider will serve as an example of how such a web-roaming robot should be designed.

Acknowledgements

I am indebted to Mark Ackerman for his suggestion that finding a way to automate or assist hypertext maintenance would be an excellent project for the World-Wide Web. I would also like to thank Tim Berners-Lee, Dave Raggett, and Lou Montulli for their comments on the original MOMspider proposal, Martijn Koster for taking the time and effort to set up the robot exclusion proposal, and James "Eric" Tilton for coining the word *infostructure* (and telling us all about it).

This material is based upon work sponsored by the Advanced Research Projects Agency under Grant Number MDA972-91-J-1010. The content of the information does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

References

[HTML] Tim Berners-Lee and Daniel Connolly. *Hypertext Markup Language*.
Internet working draft, 13 Jul 1993 (now expired). Published on the WWW at
http://info.cern.ch/hypertext/WWW/MarkUp/HTML.html

[HTTP] Tim Berners-Lee. *Hypertext Transfer Protocol*.
Internet working draft, 5 Nov 1993. Published on the WWW at
http://info.cern.ch/hypertext/WWW/Protocols/HTTP/HTTP2.html

[URL] Tim Berners-Lee. *Universal Resource Locators*.
Internet working draft, 1 Jan 1994. Published on the WWW at
http://info.cern.ch/hypertext/WWW/Addressing/URL/Overview.html

[ED92] A. Emtage and P. Deutsch. *Archie: An Electronic Directory Service for the Internet*.
Proceedings Winter 1992 Usenix Conf., Usenix, Sunset Beach, Calif., 1992, pp. 93-110.

[Hughes94] Kevin Hughes. *Entering the World-Wide Web: A Guide to Cyberspace*.
Version 6.0, 18 Mar 1994. Published on the WWW at ftp://ftp.eit.com/pub/web.guide/

[Koster94] Martijn Koster. *A proposed standard for Robot exclusion*. Published on the WWW at
http://web.nexor.co.uk/mak/doc/robots/norobots.html

[Raggett94] Dave Raggett. *Document Type Definition for the HyperText Markup Language Plus (HTML+ DTD)*. Published on the WWW at ftp://15.254.100.100/pub/htmlplus.dtd.txt

[Pitkow93] James Pitkow. *HTML_ANALYZER-1.00 README*. Published on the WWW at
http://www.gatech.edu/pitkow/html_analyzer/README.html

[Tilton93] James "Eric" Tilton. *What is an Infostructure?* Published on the WWW at
http://www.willamette.edu/~jtilton/info-p.html

Appendix A

```
# MOMspider-0.1a Instruction File

SystemAvoid /usr/local/httpd/admin/avoid.mom
SystemSites /usr/local/httpd/admin/sites.mom
AvoidFile   /usr/grads/fielding/test/.momspider-avoid
SitesFile   /usr/grads/fielding/test/.momspider-sites
SitesCheck  7
<Site
    Name          ICS
    TopURL        http://www.ics.uci.edu/ICShome.html
    IndexURL      http://www.ics.uci.edu/Admin/ICS.html
    IndexFile     /usr/local/httpd/documentroot/MOM/ICS.html
    IndexTitle    MOMspider Index for All of ICS
    EmailAddress  www@ics.uci.edu
    EmailBroken
    EmailExpired  2
>
<Tree
    Name          MOMspider-WWW94
    TopURL        http://www.ics.uci.edu/WebSoft/MOMspider/WWW94/paper.html
    IndexURL      http://www.ics.uci.edu/Admin/MOMspider-WWW94.html
    IndexFile     /usr/local/httpd/documentroot/Admin/MOMspider-WWW94.html
    IndexTitle    MOMspider Index for Roy’s WWW94 Paper
    EmailAddress  fielding@ics.uci.edu
    EmailBroken
>
<Owner
    Name          RTF
    TopURL        http://www.ics.uci.edu/~fielding/hotlist.html
    IndexURL      http://www.ics.uci.edu/~fielding/MOM/RTF.html
    IndexFile     /usr/grads/fielding/public_html/MOM/RTF.html
    EmailAddress  fielding@ics.uci.edu
    EmailBroken
    EmailChanged  3
    EmailExpired  7
>
```

Appendix B

See `http://www.ics.uci.edu/WebSoft/MOMspider/WWW94/appendix-B.html`

Appendix C

```
# MOMspider-0.1a Avoid File:  Lists URL prefixes to avoid or leaf.
# New URL prefixes can be added when the program is not running.
# The file format is:  EntryType  URLprefix  [ExpireDate]
# where EntryType  = “Avoid” or “Leaf”
#       URLprefix  = the full URL prefix for which this entry applies
#       ExpireDate =  [*] for never expire
#                  or [date] (see wwwdates.pl for valid date formats)
# This file is automatically generated, so don’t bother changing the format.

Avoid http://www.ncsa.uiuc.edu:8001/ [*]
Avoid http://info.cern.ch:8001/ [*]
Avoid http://www.contrib.andrew.cmu.edu:8001/sokoban/ [*]
Leaf  http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/whats-new.html [*]
Leaf  http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/MetaIndex.html [*]
Leaf  http://info.cern.ch/hypertext/DataSources/ByAccess.html [*]
Leaf  http://info.cern.ch/hypertext/DataSources/bySubject/Overview.html [*]
Avoid http://www.whitehouse.gov/ [*]
Leaf  http://www.ics.uci.edu/Admin/ [*]
Avoid http://www.ics.uci.edu/Test/momtest.html [Sat, 02 Apr 1994 00:00:00 GMT]
Avoid http://betelgeuse.com/name.html [Fri, 31 Dec 1999 09:30:00 GMT]
Avoid http://simplon.ics.uci.edu:8001/ [Thu, 29 Sep 1994 23:59:59 GMT]
```

Appendix D

```
# MOMspider-0.1a Sites File:  Lists IPaddress:port locations we’ve checked
# for a /RobotsNotWanted.txt file and followed its directions.
# New sites can be added when the program is not running.
# The file format is:  EntryType  IPaddress:Port  [ExpireDate]
# where EntryType   = “Site”
#       IPaddress   = the full hostname or IP address for the site
#      Port       = the numeric TCP port for the site (write 80 for default)
#       ExpireDate  =  [*] for never expire (i.e. never check this site)
#                   or [date] (see wwwdates.pl for valid date formats)
#                      Entries are automatically cleared after 7 days
# This file is automatically generated, so don’t bother changing the format.

Site www.ics.uci.edu:80 [*]
Site betelgeuse.com:80 [Fri, 31 Dec 1999 09:30:00 GMT]
Site simplon.ics.uci.edu:8001 [Thu, 29 Sep 1994 23:59:59 GMT]
```