

Horses for Courses. Fusing Dynamic and Static Web Sites

James Uther

DEDE
University of Sydney
Sydney, NSW, Australia

Hemul@gmp.usyd.edu.au

Vicki Taylor

DEDE
University of Sydney
Sydney, NSW, Australia

Vickit@gmp.usyd.edu.au

Abstract

The Graduate Medical Program (GMP) operates a large (>10⁴ document) web site to support teaching and organisation. The management of such a large site has proved challenging. DBMS site management tools exist but their limitations have caused us to avoid them. We explore these limitations and some planned solutions.

Keywords Document Databases, Document Management, WWW, Internet, Multimedia.

1 The GMP Web Site

The Graduate Medical Program operates a web site to support its teaching and organisation needs. As a Problem Based Learning (PBL) course, the GMP requires the broad distribution of resources such as x-rays, patient interview videos, pathology reports, ultrasound videos etc. These must be distributed not only to workstations on the main campus, but also to machines in clinical schools across the state connected via a wide range of bandwidths, and even to students in rural placements communicating across modem links. Consequently, the resources are static, enabling efficient caching at the destination. Additionally the site supports a number of applications including an online assessment system for students to gauge their progress against a bank of questions supplied by the faculty, and an involved feedback system that captures and routes comments on the site and course. These involve heavy use of databases and dynamic web pages.

In addition to our central web site and database servers, we use six caching servers at major clinical schools to ensure reasonable quality of service even over the often slow and clogged hospital Internet connections.

2 Current Art

The most common method of web site development is to author web pages in some form of editor, and place the pages under the document root of a web server, which then reads the pages from the filesystem. By not parsing pages, and by fully supporting the caching

Proceedings of the 3rd Australian Document Computing Symposium,
Sydney, Australia,
August 21 1998.

capabilities of HTTP/1.1[1], such sites operate efficiently. They are also suited to ad-hoc web sites, or sites with little structural similarity between pages.

An alternative to static web sites is to place the web site in a document database, and use explicit structure and relationship information to solve many of the management issues. This solution has been popularised by many (often DBMS) vendors. A prime example is Lotus Domino [2], which exports a notes database to the web, translating documents to HTML on demand. In these systems a database stores the raw document content, and generates HTML pages from this content on request. Templates are used to define page types, and so to author a page simply requires choosing a template, and then filling in the fields required by that page type. Flat identification schemes or queries name the pages, rather than a filename, which avoids any renaming issues. Similarly, the use of referential integrity within the database can prevent the removal of a page that would cause a broken link. These systems excel in installations where the content of the pages already exists in something like a product database. Linking the database directly into the web site ensures the site is always current requiring no page maintenance.

In our experience with and evaluation of both methods of web site management and creation, we have observed problems and limitations with both.

3 Limitations of Current Methods

3.1 Static Sites

Allowing the system to disregard page structure and relationship information for the sake of efficiency can lead to difficult site management issues. For instance, changing the name of a page on the web site also involves changing all pages with links to the changed page. Similarly, deleting a page can produce a broken reference within the site. Although site management tools that attempt to mitigate these problems exist, such as SiteMill [3] and FrontPage [4], we have found that they have trouble with large sites. Additionally, changing the style of the site involves changing all documents. New standards such as Cascading Style

Sheets (CSS) [5] address this problem, but still lack support in browsers. Proprietary solutions exist, but often fall into the problems inherent in dynamic sites.

3.2 Dynamic Sites

These systems require pages to follow an explicit structure (page type). Multiple page types may be defined, however in diverse and developing web sites such as ours the number of page types can start to become unmanageable in itself. Web sites undergoing a rapid revision cycle or with a large set of page types are difficult to model in these systems.

The web is slow. High speed Internet service providers such as @HOME and the Big Pond cable modem service rely on large caches of web content to ensure the download speeds their customers are paying for. We use the same system in our remote sites. In essence, as our users retrieve documents, a copy is stored on a server on the local high-speed local network. When the next user requests the document, our cache server sends an 'If-Modified-Since' header to the main web server to see if the document has changed. If not, the local copy is sent at LAN speed. Only if the document has changed does the cache server request the new version from the main web server. If that document is a five-megabyte video, the cache server allows a download time of thirty seconds instead of five minutes.

In many cases, systems that offer dynamic sites use weak web servers that break the caching functions of HTTP/1.1, or are not programmed to reply sensibly to an 'If-Modified-Since' request. We require good quality of service at the end of slow links, and achieve this with cache servers. Any web server we use must reply sensibly to an 'If-Modified-Since' request.

We also make copies of the web site on CD-ROM and require the site to look like a filesystem, and many of these systems use URLs that are illegal in some filesystems. Others use URLs in a flat naming scheme, which in cases such as ours with many documents could expose limitations in the filesystem's handling of large directories.

Finally, such sites only accept documents that adhere to the set page structures. This usually requires a proprietary authoring tool, and these tend to be restrictive.

4 Solutions

The solutions are largely a matter of choosing the right tool for the job. This section will detail some of the parts of our web site, and how we have overcome the problems described above, or how we plan to overcome them in the future.

Our online assessment system is a classic example of a dynamic web site. It's just a database application. We have made no effort to enable our cache servers to store the basic pages, as they are by

definition different for each request. On the other hand, we have made sure that the dynamic data is small (a few lines of text), and that any large bodies of data, like images for navigation buttons, are taken from a static part of the web site and may be cached. In this way, we have ensured that the application is useable even over slow links.

Our test result pages present many possible resources, and each page has been hand authored to best display the content. For instance, although a pathology slide and an x-ray are both images, clinicians require different displays of each. X-rays are particularly difficult to display well, requiring many edit-review cycles with radiologists before the staff are satisfied with the presentation. These pages are also large, and so very cache-dependent. These pages exist on a static web site and we have no immediate plans to move them.

We have a middle ground as well. The course consists of 540 'Learning Topics'[6], each of which consists of a title, author, keywords, 1-2 pages of topic summary, and references. The format is standard. Although these pages do not change often, we can see the advantages in storing the information in a database and producing the pages when required. The problem has been to ensure these documents are copied on our caching servers, and that the generated URLs look like they come from a reasonable filesystem. Our solution (currently in design) will use message digests and PathInfo information.

PathInfo is information passed to a web executable (servlet, CGI, etc) that describes path information in a URL beyond the name of that executable. For instance, if we have a servlet called `http://webserver.au/servlets/LearningTopics` but use a URL `http://webserver.au/servlets/LearningTopics/more/path` then the servlet `LearningTopic.class` is executed and given `/more/path` as it's PathInfo parameter. This simple mechanism allows us to use an executable to generate pages, but still access the site through a meaningful and useful URL namespace.

A more difficult problem is ensuring the servlet passes back a reasonable answer to an 'If-Modified-Since' request header. In this case, the client request headers contain an If-Modified-Since line and a date. It is requesting the document only if the document is newer than the date specified. The servlet must decide whether this is the case. In many simple cases where the page data is contained in a single row in a database, we obtain this information from the timestamp of that row. In complex cases involving multiple queries, this becomes prohibitively difficult and so we intend to implement a solution using message digests.

Message digest algorithms such as SHA-1 [7] or MD5 [8] process a body of data and produce a short string of bits that is probably unique to that data. For

example, the SHA-1 algorithm takes a file smaller than 2^{64} bits, and produces a 160-bit signature string from that file. To a very high probability, any changes to the file will change the signature. Similarly, there is a very low probability of ever producing a file with the same signature.

By storing a digest of generated web resources and the date of digest generation we can tell the date last changed of a resource in all cases. For instance, when resource X is first requested, we generate the page and send it, but also generate a SHA-1 digest of the page and store that along with a timestamp. When a request for resource X arrives with an 'If-Modified-Since' header, we regenerate the resource and generate a digest. If the digest and the previous digest don't match we update the digest and timestamp and send the page. If the digests match, we can then compare the digest timestamp and If-Modified header date and only send the page if the page is newer than the header. One further optimisation would be only to do the calculation every given timeout period, and cache the page within the generating servlet in the interim. This would be useful for high load pages with a low change rate.

5 Conclusion

Many solutions exist to the many problems of building and maintaining a large web site. Each solves a problem, but not all problems. The GMP web site thus uses a combination of solutions, each solving the most pressing problem of that area of the site, but with modifications to ensure that the solution to one problem does not cause more. We initially found that our needs for flexibility, development and caching precluded the use of database-based site management tools for the bulk of the site. As our site structure and requirements have firmed we have developed methods to move some of these pages to a database without compromising our performance.

Acknowledgements

The whole of DEDE, and in particular Stewart Barnet for providing requirements to keep us honest. Judy Kay for encouragement and bottom kicking.

6 References

- [1] R.Fieding et. Al. Hypertext Transfer Protocol -HTTP/1.1
<http://www.w3c.org/Protocols/rfc2068/rfc2068>.
- [2] Lotus Domino.
<http://www.lotus.com/products/domino.nsf>
- [3] Adobe Pagemill.
<http://www.adobe.com/prodindex/pagemill/pagemill20/siteben.html>

- [4] Microsoft Frontpage
<http://www.microsoft.com/frontpage/>
- [5] Cascading Style Sheets.
<http://www.w3.org/Style/>
- [6] The GMP Visitors Site.
<http://www.gmp.usyd.edu.au/visitors/>
- [7] The SHA-1 Algorithm.
<http://www.itl.nist.gov/div897/pubs/fip180-1.htm>
- [8] The MD5 Message Digest Algorithm.
<http://theory.lcs.mit.edu/~rivest/Rivest-MD5.txt>