Java Proxy Server

School of Computer Science,
University of Birmingham

2 May 1999

Author: Robert Neild

Degree: Electronic and Software Engineering

Supervisor: Dr T.H. Axford

Table of Contents

Abstract

This project's aim was to write a program to speed up Internet web browsing for the home PC user. The solution is a Java proxy server that provides extra caching, downloads links in the background, and filters animated GIFs down to single image ones. A user will start the proxy server running on their home computer and set-up their browser to send requests through the proxy server.

Acknowledgements

I would like to thank my project supervisor Dr. Axford, and also my flat mates Kris Beaumont and Steven Beech, who's help in testing, GUI design and functionality was invaluable. I also got help with the format of this report from Mike Davis.

Chapter 1: Introduction

1.1 Project Background

More and more people are accessing the Internet from home, using a modem. However, the speed, viewing web pages can become quite slow. This degradation in service can be due to many different factors, some can be improved and some can't. When downloading a large file, the limiting factor is probably the maximum speed of the modem. However, with web pages, there are ways to improve the perceived speed of download. This project aims to implement some of these methods.

1.2 Aim

The main aim of this project is simply to *'Speed up Internet browsing'*. The program is targeted specifically at speeding up web page viewing times[1], and improving off-line cache retrieval hit rates. The target user will be accessing the Internet at home with a modem using a PC.

1.3 What this Program Does

This program is a proxy server written in Java. It runs on the same computer as the browser (which will be set up to pass all HTTP requests to the proxy server). This means the proxy server has full control over what to do with the requests.

This program can then ;-

- Provided more intelligent caching - Cache documents specifically for when we go off-line.
- Download links in the background - When the user is looking at a web page, the program downloads all the pages that are linked to the page. This makes more efficient

1 The time it takes from typing a URL into a browser, until the time the web page is completely visible in the browser.

use of bandwidth.

- Filter animated GIFs to single image ones - This reduces the amount of information that has to be downloaded.

1.4 What is Presented to the User

There is only a minimal GUI to this program so it doesn't get in the way of the browser. The window has a few labels to give the user some information on what is going on, and there are pull down menus to select cache management, configuration dialogs etc.

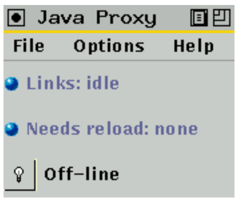


Illustration 1: Main GUI

1.5 Technical Detail

The implementation of this project required a lot of low level technical detail, such as HTTP, Java threads, and the format of GIF files. Some detail I have not included because it was not necessary, as I have tried to write at as high a level as possible throughout this report. However, I have had to include a reasonable amount of HTTP. To help the uninformed reader I have included a small HTTP digest in the appendixes.

1.5 Structure of this Dissertation

Chapter 1 - This introduction.
Chapter 2 - Some background research, investigation and ideas.
Chapter 3 - Higher level design of the program
Chapter 4 - Implementation
Chapter 5 - How I managed this project
Chapter 6 - Results from testing the proxy server
Chapter 7 - Speed test results
Chapter 8 - Analysis of program and results
Chapter 9 - Conclusions
Appendixes
Bibliography
References

Chapter 2: Background Material

2.1 What's Already on the Market to Speed up Internet Browsing

When starting any project it is helpful to take a moment to look at similar products on the market. This can ;-

- Identify gaps in the market.
- Gauge other programs performance, providing a benchmark for your own.
- Identify some popular features that your product should implement.
- Identify some desirable features that your product could implement.
- Helps clarify who your target user is.
- Become more familiar with the general field.

There are various types of program available that can help speed up web browsing, and I took a brief look at them ;-

Proxy Servers

There are a few public domain proxy servers available, although not nearly as many as actual web servers. The most popular is SQUID[2], which is developed with much the same methodology as the Apache web server[8]. It's advantages are;-

- Free to download.
- Source code is available.
- Runs on many systems.
- Extremely stable.

It can speed up browsing, just by providing an extra layer of caching on top of the browser cache. However, it has it's problems;-

- Doesn't know when off-line, so can't tailor cache hitting.
- Doesn't cache some documents at all i.e. cookied[2] requests.
- Programmed, assuming a permanent connection to Internet.

2 HTTP cookies are a mechanism for maintaining state between clients and servers. A HTTP request, that contains a cookie, is routinely not cached because the important stateless nature of HTTP requests (a URL always returns the same document) is lost.

Improved Browsers

Early browsers e.g. Netscape Navigator 3[3] (and earlier) didn't have a concept of off/on-line Internet access and assumed a permanent connection. They are improving though, with Channels being invented, where a chunk of information can be downloaded in one go for off-line browsing, and they are improving in the amount of information they cache for off-line access. Some problems with browsers though are;-

- Channels are effectively useless, with the balance between content size and download time difficult to weigh.
- Caching seems inconsistent. You can look at a site, go back to it a minute later when off-line and it is no longer available to you ?
- Limited cache Control. Especially in Netscape, the cache can't be viewed at all.

Prefetch Link Downloaders

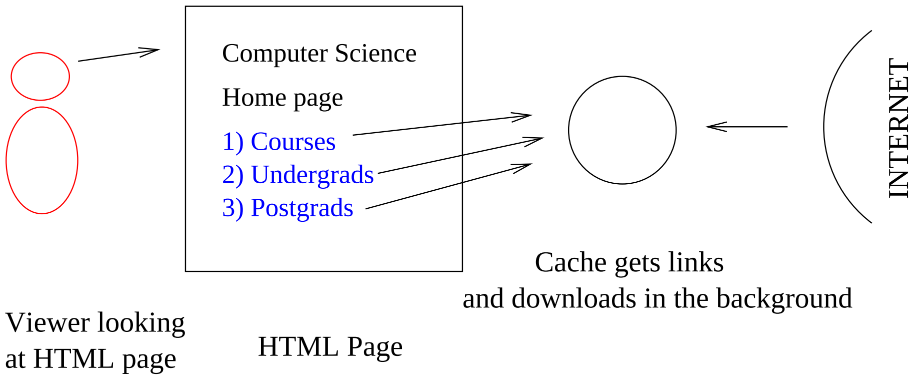


Illustration 2: How links are downloaded

These programs use a technique called 'prefetching'. While a person is on-line and reading a web page, these programs work away in the background downloading all the links from the current page they are looking at, using the spare bandwidth.

Some popular link prefetch programs I had a look at are;-

- WebEarly98 [3] - $24.95
- NetAccelerator 2.0 [1] - $29.95

3 Copyright © 1994-1997 Netscape Communications Corporation, All rights reserved.

All of them seem to operate extremely well, however I found it annoying that they didn't give very verbose messages as to what they were doing. Unfortunately, all the ones I have found cost money. They also tend to just concentrate on downloading links and offer no improvement to off-line performance or offer other 'content filtering' services.

Conclusions

There are some programs around at the moment that can increase browsing speed. However, I feel there is a need to combine the functions of a few of these program into one package and to produce a free alternative, to sometimes quite expensive, niche products. I definitely think there is a need to redefine how caches work when off-line.

2.2 Identified Possible ways to Increase Browsing Speed

There are many ways that I can think of to increase browsing speed. A lot of them depend closely on the nature of the Internet and the habits of people browsing the net. There are that;-

- The Internet changes over time. Sometimes in seconds, sometimes in months.
- People browsing the Internet tend to stop at a page for a few seconds while reading it.
- People tend to return to the same page over and over again.

These characteristics can be taken advantage off.

Extra/More Intelligent Caching

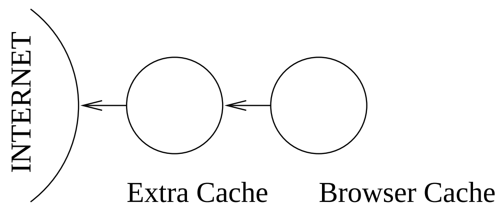


Illustration 3: An extra cache level

An extra level of caching could be used as well as the browsers cache. However, the same speed up could be done much more simply by increasing the size of the browsers cache.

Once there is an extra level of caching some extra functionality could be added to it;-

- Gui maintenance (adding/deleting/saving)
- Customised garbage collection algorithms
- Decide exactly what to cache
- Make the cache aware of off/on-line situation

Download Links in the Background

When a user is looking at a certain HTML page, the bandwidth from their computer to the Internet is idle. This can be used up by downloading at all the links from the HTML page they are currently looking at. When the user then presses one of the links on the page, hopefully the page will have already have been downloaded for them.

This obviously suffers from some problems. It is impossible to predict what link the user will click on next, so a blanket download of all links has to be done. If the number of links on a page if high then the amount of redundant data to download, just to get the right link can be huge.

Some people think this large waste of Internet bandwidth is 'anti-social' and if everyone was doing it, the Internet could be brought to it's knees.

Advert Filtering

Foreign & Commonwealth Office
dti
Department of Trade and Industry
Whether you're new to overseas trade or an established exporter...
Use our expertise for your export ease
CLICK HERE

Illustration 4: Typical advert

As the Internet becomes more and more commercialised, sites are starting to show more adverts. These can be quite large, annoying to look at, and take time to download.

It would be nice if these adverts could be blocked from downloading. This can be quite complicated because adverts have to be distinguished from normal graphics.

Adverts could be distinguished by using AI, a list of blocked adverts etc.

Background update Checks

This is quite an interesting feature that hasn't been implemented in other programs as far as I know.

A browser occasionally checks to see if a file in it's cache is up to date by sending a request to the server where the file originally came from. Unfortunately, some browsers generate these requests even when they are off-line, and when on-line, the browser has to wait for the response from the remote server before proceeding. It would be better if these checks could, somehow, be done in the background and handled better.

Chapter 3: Design

3.1 Target User and Environment

Through my research I have identified a need for a program to speed up Internet browsing, that is free, and targeted to the individual user with a temporary connection to the Internet.

The user will be expected to be using a conventional home PC with a modem connection to the Internet.

I have felt it necessary to narrow down the above statement, to make development and implementation easier and more realistic, given time constraints. These specifications are;-

- Home PC with at least an Intel Pentium processor (or clone).
- At least thirty two Megabytes of RAM.
- A few hundred Megabytes of free hard disk space.
- Modem to ISP, connection speed of 33kbs to 56kbs.

3.2 Description of Features

I have chosen the following features to implement in this program;-

Caching, and a GUI maintenance window

The program will have its own cache, and a cache GUI to delete and view entries in that cache.

Prefetch links

A record of the last HTML to be downloaded will be stored, and all the links from that page downloaded in the background.

Background update checks

Filter animated GIFs to be single image GIFs

These features were chosen because they were thought to provide the best Internet speed up, to implementation difficulty, ratio.

The decision to filter GIFs was though to be a good way to try and filter adverts. It was thought to be too hard to identify with much certainly what are adverts on a page and what aren't, but I have noticed that a lot of adverts are animated GIFs. The decision to just download the first image in an animated GIF, stops the full advert downloading but also allows you to get an idea what the picture should look like.

Hopefully non-advert, animated GIFs being changed to single image ones shouldn't have a detrimental affect on a web page as the idea behind an animated GIF can still be seen.

3.3 Outline of Program Operation

The program will listen for connections from the browser on a specified port. Although it might be thought that when running on a single computer, there would only ever be one connection from the browser to the proxy server, this is not so. Most browsers will make as many as four connections to a proxy server. This is used to compensate if one of the connections is slow, and to allow for slow servers. Because of this, the proxy server must service more than one connection at once. The only way to do this efficiently is to use threads. Each thread servicing one connection.

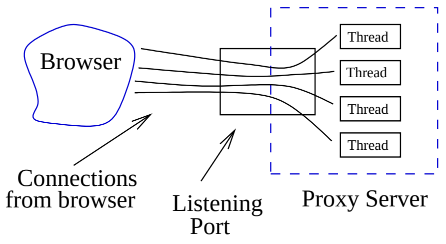


Illustration 5: Thread servicing

Every time someone connects to the port a thread is spawned off. The thread will then service that connection, accessing the cache and Internet as needed.

Web servers, that use a similar technique, have to be careful that they don't spawn off too many threads at a time. This is not deemed to be a problem with this program as only one browser should be accessing it at a time.

Web servers, also tend to create a 'pool' of threads when they start. Every time a connection comes in it is allocated a thread, and when the connection finishes the thread returns to the pool. It does this because the continued creation of threads can be a slow process.

I have decided that using a pool of threads is not necessary for this program. This is because this program will hopefully be using persistent connections to the browsers. This should mean that connections to the browser can remain connected for a long time, and the number of threads that needs to be created reduced.

3.4 Design of Cache

The program will provide an extra level of caching. This on it's own will not speed up Internet browsing much, as the same thing could be achieved by increasing the browser cache. As a convenience the cache will be able to be viewed and maintained from a GUI window.

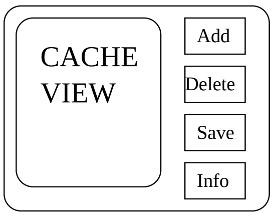


Illustration 6: Design of Cache Maintenance GUI

The main advantage of providing another level of caching is that is can be made aware of whether the user is off/on-line and tailor whether certain requests will hit the cache or not. A cache is also needed for the prefetch link downloading feature.

The following table outlines whether certain types of requests should try and check the cache before going out on to the Internet;-

HTTP Request	***Hits cache while on-line***	***Hits cache while off-line***
Contains a cookie[4]	no	yes
POST[5]	no	yes
Contains a 'Pragma: no-cache'[6]	no	yes

These types of requests aren't cached by normal proxy servers or browsers, because cookies and POSTs can result in different information being returned from the server for the same URL, and the 'no-cache' element specifically tells us not to look for the URL in the cache. However when off-line getting the response from the cache is better then nothing at all.

Where to Store the Cache

There are three main places that could be used to store the cache;-

- In memory
- On disk
- Both

The problems with storing the cache just in memory, is that is is restricted to a certain size, probably a few Megabytes. This will not be enough. The advantages with memory storage, however, are very quick access and transfer times.

Storing the cache on disk has the advantage of almost unlimited cache storage space. In practice it will probably not want to be greater than several Megabytes. The disadvantage with disk storage, is access and transfer times. However, since we are restricting the program to use Internet connections of 56kbs, the speed of most hard disks should be more than ample.

The best of both memory and disk storage can be achieved by using both. On

4 "HTTP Cookies are a mechanism for maintaining state between client and origin servers. They allow the server to issue a "token" to the client, which the client will send to the server on every subsequent request."[4].

5 POST commands, request a certain URL but also passes extra data to the server.

6 The 'Pragma: no-cache' directive specifically tells any intermediate servers not to use there caches for that particular request. Usually associated with browsers refresh buttons.

consideration the program will *not* use this because the added complexity of tying to co-ordinate, what is effectively two caches, was not deemed to have an equivalent speed benefit. This conclusion was arrived at because it was thought the transfer speeds of hard disks, for single users, are probably more than enough for browsers to handle. Also, the overhead of checking both memory and disk caches would be too high.

How to Store Cache on Disk

Storage of the cache on disk can be done in a few ways ;-

The most obvious way is to store cached files under a filing system directly relating to it's URL. So, all the hosts from a certain domain would be stored under a top level directory with the same name as the host. For example, the two URLs;-

www.cs.bham.ac.uk/manual/index.html

www.cs.bham.ac.uk/manual/pic.gif

would be stored as files called *'index.html'* and *'pic.gif'*, under a directory called *'manual'*, and a top level directory called *'www.cs.bham.ac.uk'*.

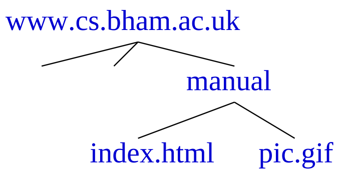


Illustration 7: Cache storage example

This method was used in the first Web server, CERN httpd[5]. The advantages of this storage scheme;-

- Simple to implement.
- Files from the same host are all stored under the same directory.

However the performance of this caching system is quite slow. The top level directory will tend to fill up with a lot of sub directories, one for every host that has cached files. This means every time a file is retrieved from the cache, because of the way filing systems work, the top level directory has to be sequentially searched for the correct directory. If this search has to go through hundreds of entries to find the correct one, it can become slow.

It also has to be checked that all the characters in the URL are supported on the underlying filing system. If not, then they will have to be substituted somehow.

A better way to store cached files is using a **hashing code**.

This method changes the URL that we want to store, into a code, using a hashing function.

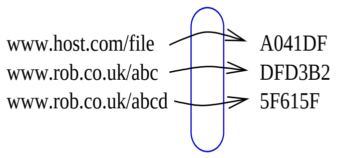


Illustration 8: Hash function

This code is then sectioned up into a few parts. The first part gives the name of the top level directory where it will be stored, the second part the name of the next level directory, and the last part becomes the name of the actual file.

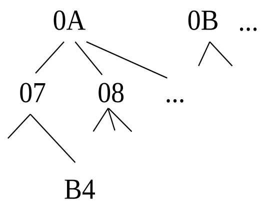


Illustration 9: Example of storing file 0A07B4

This method is used by most browsers and proxy servers. Evaluating this method;-

<u>Advantages</u>

- A fixed directory structure can be made when the program is run for the first time, to save making them later.
- Cached files are always distributed evenly through out the filing system making file access fast.

<u>Disadvantages</u>

- Files from the same host aren't stored together.
- Hashing functions are generally one way only i.e. It is impossible to go from a hashing code back to the original URL. This means identifying where a given file comes from is impossible.
- The same hashing code can be produced from two, or more, different URL. This is a clash and has to be detected and handled.

I have chose to use hashing code storage method because of the advantages listed above. It was thought, not to be that much harder to implement than the direct URL to file system mapping.

Design of Cache Table of Contents

To overcome some of the problems from using the hashing code storage method, I have decided to keep a *table of contents* for the cache. This table will be kept in memory and saved to disk when the program is stopped.

Unfortunately, at a certain cache size the table of contents will become too large to fit into memory. This imposes a maximum size on the cache, but it is not thought that this will become a problem (see appendix A for the rough calculation).

The advantages of using a table of contents are;-

- Enable the size of the cache to be known almost instantly - Just add up all the sizes in the contents.
- Keeps count of number of files in cache - Just add all up the number of records in the contents.
- Simplifies the task of garbage cleaning[7].
- Could help solve hash collision (see below).

The table of contents will need to store under a particular URL;-

1. Filename (based on hash code)
2. File size
3. Last access count

7 Garbage cleaning is the process that is done occasionally on the cache to delete out excess files, so it doesn't become too large.

A major advantage of having a table of contents is that it can help solve cache collisions, because the filename that is stored for a particular URL doesn't necessarily have to be the original filename that the hash code generator produced. If there is a clash then the filename could just be modified slightly by adding a character onto the end of it until no disk filename clash is found.

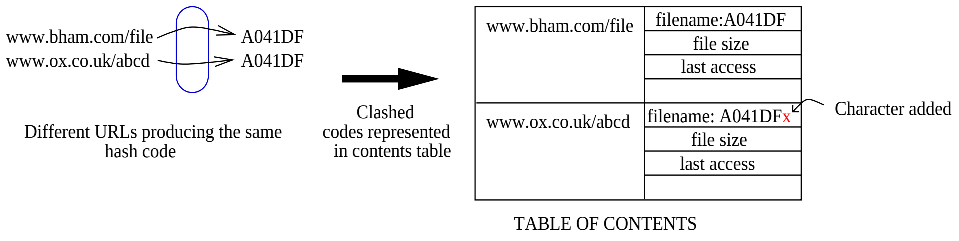


Illustration 10: Clashed URLs being stored in contents table

Cache Garbage Cleaning

The cache will obviously keep getting larger and larger as files are added to it. At some point files will have to be deleted from it. There are three issues to consider when doing this;-

1. What files do we delete ?
2. When will these files be deleted ?
3. How to safely delete files from a concurrent system. (This is dealt with in another section)

1) When there a lot of files in the cache, how do we decide what files to delete ?

Possible methods could be to ;-

- Delete files from the cache at random.
- Clear the cache every time the program starts.

However, these methods aren't very efficient.

What is needed is an algorithm that deletes the cached files that probably will not be used in the future. The goal being to optimise disk space usage while keeping the cache hit rate as large as possible.

There are various algorithms that could be used;-

- *LRU (Least recently used) algorithm*. When someone is browsing the Internet, is is statistically more probable that they will return to sites they have just visited, rather than sites visited days ago. Because of this, we could prefer to delete the files in the cache that have not been accessed for the longest time.

- *Weighted LRU for recent accesses.* The above algorithm may give a false sense of how popular a cached file is. It is possible that a certain file may only be looked at once and never again. Implementing this algorithms is the same as the basic LRU, but having a weighting that prefers files that have had many accessed.
- *Weighted LRU for size.* If there is a very large file in the cache, and deleting that file means that many other files wouldn't have to be deleted, then it might be prudent to use the LRU algorithm with an add in factor that prefers larger cache entries.

The algorithm I have decided to use is the basic LRU. This is because the effectiveness of the other algorithms is debatable, and the complexity of them I think is completely unjustified over the simplicity of LRU.

2) When will files be deleted. A decision needs to be made when files are deleted;-

- A file could be deleted whenever a file is added. This would means the effort at deleting files is spread out over time.
- Or, files could be deleted in a clump at certain time intervals. This means that the effort to delete files would be concentrated into a small, infrequent time periods.

I have chosen the second scheme. This is because the contents table will probably have to be processed to sort the cached files into the order of last access. This sort could take some time so it is better to do this infrequently.

Because of the bursty nature of Internet requests it is also likely that if the garbage clean is done in a burt, it is probable that is will occur at a time when the proxy server is not doing anything. Of course, it is possible that two bursts may collide. This could be alleviated by giving the garbage clean process a much lower priority,

3.5 Concurrency Problems

There are some major concurrency problems within a program of this type. The main one is with the cache control. Items can be deleted from the cache at any time by the garbage cleaning thread, deleted by the user, or added by a servicing thread. This produces obvious problems if an item in the cache is deleted while it is being read by a another thread etc.

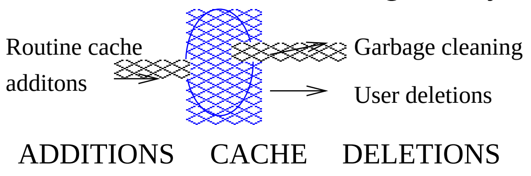


Illustration 11: What accesses the cache

Some Solutions to this problem;-

1) Lock cache - The easiest would be to allow only one thread to access the Cache at a time. This would prevent all concurrency problems but would have an obvious performance overhead as threads would spend time waiting to get access to the cache.

2) No locking - Another solution, would be to have no locking at all. Initial experimentation with this method showed that this is a surprisingly acceptable method. The chances of a clash are very small. When a clash is detected it could be handled quietly e.g. by cutting the connection to the browser in mid flow, if the cached file that it was reading from is deleted from 'under' it. This would result in the browser not showing a certain picture etc. but would only happen very occasionally.

3) Record locked URLs - However the solution I have chosen to adopt it to somehow 'lock' URLs, while they are being accessed. There are many advantages to this;-

- It allows the cache to be accessed by many threads at once.
- Prevents concurrence problems.
- Should only have a small performance overhead.

3.6 HTTP Link Parser

For the part of the program that prefetches links in advance, is some code that is able to go through a HTTP page and extract out all the links from it to other HTTP pages.

HTML links are in form <A HREF="http://www.bham.ac.uk">. The program needs to parse out the 'HREF' element of the 'A' tag. There may be other elements in the tag e.g. <A HREF="http://www" NAME="rob">. These extra elements need to be ignored.

The 'A' tag has been in HTML specification since version 2.0[8], so it has been established for a long time and in unlikely to change. It is possible extra elements will be added to it, along with the HREF element that we want, so we must be careful to ignore them properly.

8 HTML 2.0 is defined in RFC 1866

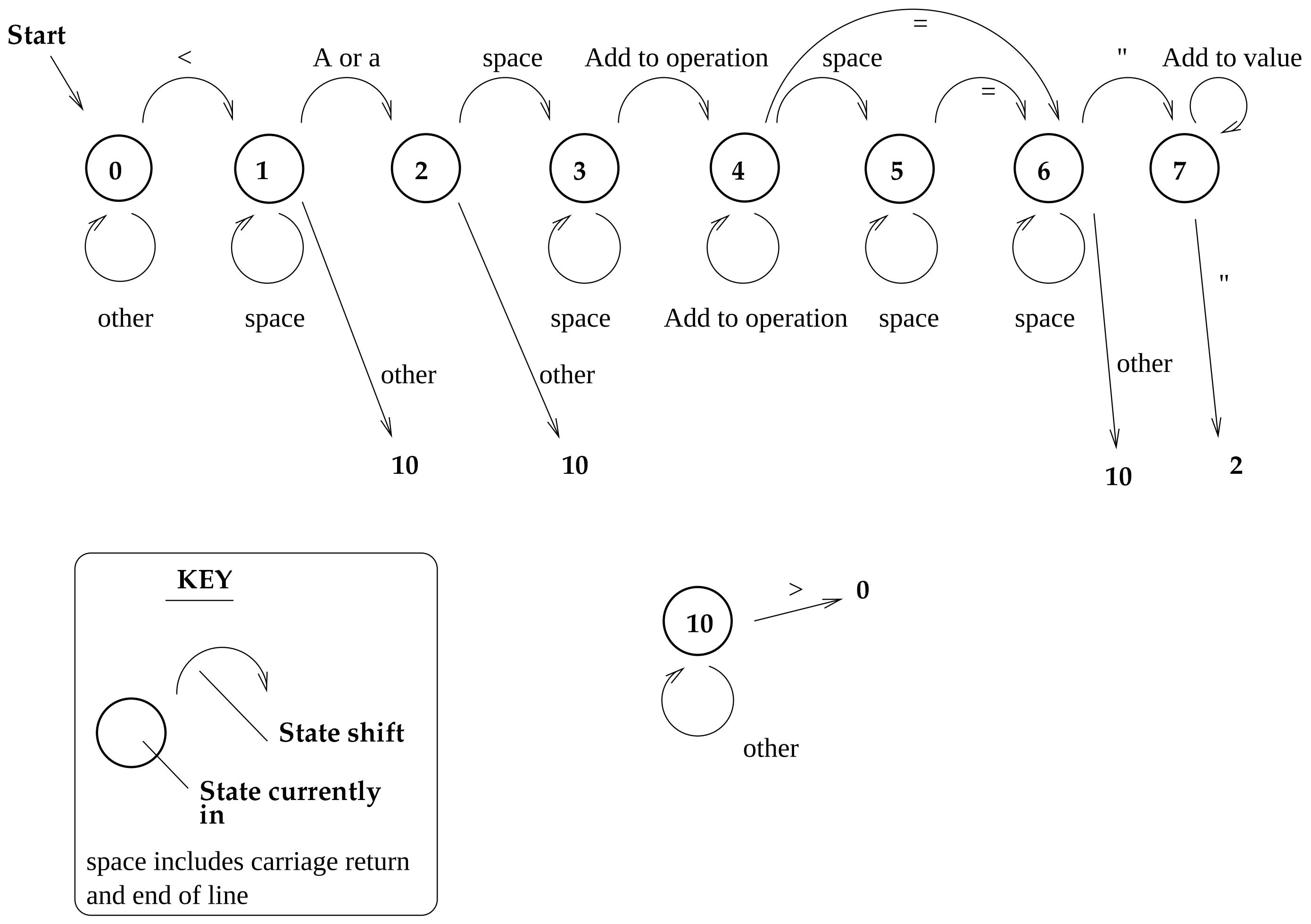


Illustration 12: Link parser

The state machine designed above has been made to recover reasonably from any errors in the HTML. If in the middle of a tag some unexpected character is found, it goes to state 10 to skip that tag. When the tags terminator, '>', is found it recovers and starts looking for another tag.

3.7 Background Update Checks Feature

This feature checks pages in the background. Usually a browser has a setting which tells it when to check its copies of cached documents to see if the master copy on the Internet web server has changed. The result of these checks is that the cached copy is up to date. The browser has to pause to make these checks, so a way to speed up browsing would be to 'take' these requests off the browser and do them in the background.

Illustration 13: Netscape 3 configuration

When the browser sends out a 'if-modified-since' request (this is what they are called in the HTTP request) we will intercept it and send back an immediate reply that the page is up to date. The requests will then be put onto a first in-first out stack. A process when takes these requests off the stack and checks them in the background. If a page is found to be up to date then no action needs to be taken. If it is found to be out of date when a dialog needs to be popped up to alert the user that the page they are currently looking at is old and they need to press reload.

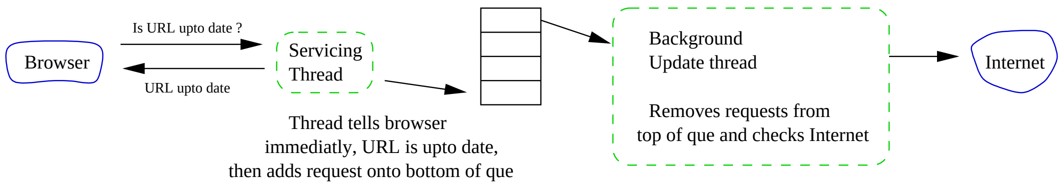

Browser
Is URL upto date ?
URL upto date
Servicing Thread
Thread tells browser immediatly, URL is upto date, then adds request onto bottom of que
Background Update thread
Removes requests from top of que and checks Internet
Internet

Illustration 14: How background update checks are handled

Chapter 4: Implementation

4.1 Language

Java was decided as a language to implement this project. The main reason for this was that a Java program would be able to run on all the operating systems that run on peoples home PCs (Linux, Windows 95/98/NT). Java has certain other advantages;-

- Built in thread, and thread synchronisation support
- Easier to debug that C++
- A very stable, common, GUI

Java does have some disadvantages though;-

- The speed of Java may be a problem. Since this program doesn't do any heavy raw processing, I predicted that it would probably not be a major problem.
- Another problem is Java's rather small set of features in its natively implemented GUI (awt). The way to get around this is to use Suns package of lightweight[9] GUI components, called Swing.

4.2 HTML Link Parser

The implementation is obviously best done using a parser, and the obvious way to do this is to use a parser generator such as Lex/Flex[10]. Although Flex is designed to generate a C parser, there is a port of Flex to Java called JLex[6].

A initial generator was written using JLex and then tested. Some of the points for and against using JLex are;-

Advantages -

- Very easy to change and modify quickly

Disadvantages -

- Perhaps slightly slower than a hand made parser

9 Lightweight Java GUI components don't rely on the native system because Java draws the controls itself.

10 Flex is a GNU implementation of the more restrictive Lex.

- Source code it produces is hard to understand

It was decided, after an initial prototype had been done in JLex, to write a parser by hand. This was because it could be made to run faster than the JLex generated one, and have cleaner, commented source code (see appendix D for speed testing results).

The implementation was then a simple task of writing a state machine parser that was written according to the designed state machine.

4.3 Gif Filtering

The filtering of GIFs is mainly a technical exercise. It involves reading in a GIF file until the end of the first image, and at that point the connection from the browser to the remote server can be terminated.

This process is complicated by a GIF file having lots of different types of blocks in it. The different blocks have to be partially understood by my program, as I will have to pass them through properly. The types of blocks differ from fixed and variable width lengths. The GIF specification documents the different types of blocks that are possible.

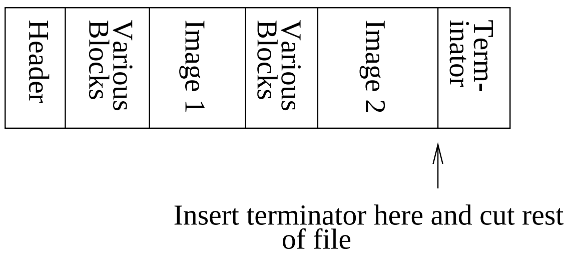


Illustration 15: Filtering a GIF

4.4 Adapting the Standard Date class for Rfc 822

I have included the section below mainly because I think it is interesting, but it only an sample of many other small, lower level implementation problems.

The HTTP/1.0 standard in RFC 1945 defines the date format HTTP sources should generate, and the larger set of date formats that should be understood.

The three formats that should be understood are ;-

Sun, 06 Nov 1994 08:49:37 GMT	RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT	RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994	ANSI C's asctime() format

The only format that should be generated is the first one in the above table. After some testing of the Java util.Date class (Appendix ?) it can be seen that Date is able to parse all the above date formats but the format is creates is not compatable with Rfc 822.

The solution to this problem is quite simple and just requires the use of the SimpleDateFormat class to explicitly specify the format of the date output. The result of this is a class derives from Date but with to 'toString' methods overwritten, called Rfc822Date.

The testing of the Date class can be found in appendix B.

4.5 Persistent HTTP Connections

With version 0.9 and the older implementation of 1.0, HTTP has been a single request-response connection. After the requested file has been retrieved, the server cuts the connection. The graceful[11] cutting of the connection was used to signify the end of the file. With a HTTP/1.0 attribute 'Connection-Length', this is no longer needed as the browser end can be told the length of the file explicitly, before transmission.

The three way handshake, used to establish a TCP/IP connection, is quite costly so a way was developed to 'keep alive' HTTP connections. This allowed multiple HTTP requests to be sent, and multiple files to be retrieved using one persistent connection.

So, how do we use persistent connection ? We have to explicitly ask for persistent

11 A TCP/IP connection can be cut gracefully, and the receiving end will know that it was done deliberately. There are other types of connection cutting which are done if there is a time-out or error.

connections using a HTTP element in the request 'Connection: Keep-Alive'. This tells the remote server to *try* and keep the connection alive after the requested file has been downloaded. This is only a suggestion to the server, so the program has to be prepared for the connection to be terminated. The server will do this if it is unable to send us the file length[12], therefor cutting the connection is necessary to tell us when transmission is over. An important point with persistent connections is that the remote server has restrictions on how long a persistent connection will remain idle for, before being terminated.

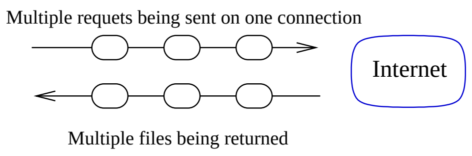

Multiple requets being sent on one connection
Internet
Multiple files being returned

Illustration 16: Sending multiple requests on a persistent connection

I have implemented persistent connections in a class that tries to hide some of the complexity. The *'PersistentHTTPConnection'* class is constructed around a certain server. It is then given a HTTP request and it tries to deliver it to the server it is associated with. If it has already send a request to the server on a persistent connection, then it tries to send the request on the already existing connection. If that fails then it ties to open another, and if that fails then it throws an error up to the class that called it.

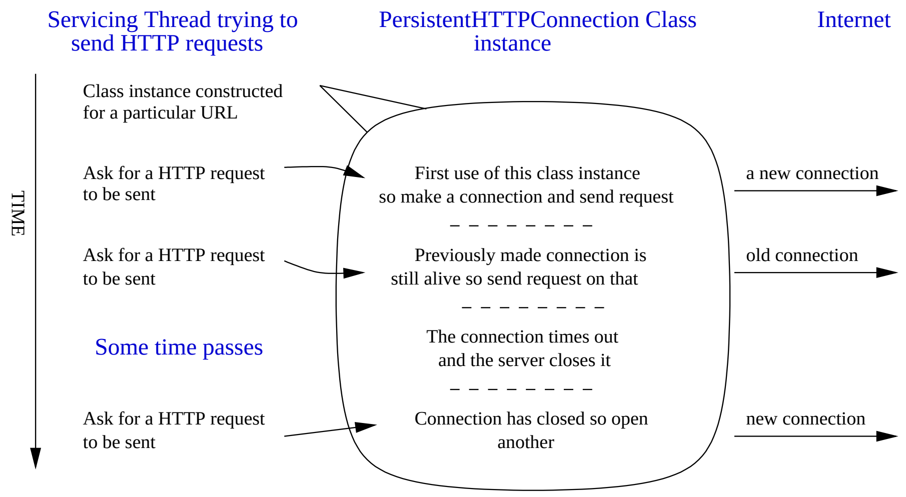

Servicing Thread trying to send HTTP requests
PersistentHTTPConnection Class instance
Internet
TIME
Class instance constructed for a particular URL
Ask for a HTTP request to be sent
First use of this class instance so make a connection and send request
a new connection
Ask for a HTTP request to be sent
Previously made connection is still alive so send request on that
old connection
Some time passes
The connection times out and the server closes it
Ask for a HTTP request to be sent
Connection has closed so open another
new connection

Illustration 17: An example of how the PersistentHTTPConnection class works

12 The server may not know the length of a file if it is being created dynamically using a CGI program.

4.6 Cache Hashing Function

As stated in the main body of this document, a hashing algorithm is integral to the caching of URLs. The algorithm will need to take a URL and output a hashing string that can be used to store, and access it from disk.

Format of hashing code

The format of the outputted hashing string is important, as this will be the filename of the cached document on disk. Thus, it can only contain the set of characters that are allowed in filenames. However, there is no definitive set, as the program can be run on many operating systems. The safest set of characters to use is a-z and 0-9, as most file systems should support filenames containing those characters.

Possible hashing algorithms

There are many hashing codes available. Lots of research and effort is being made to make algorithms that are robust i.e. it is difficult to give the algorithm two different inputs that would produce the same hash code. This is because these algorithms are used in encryption codes. This level of security is not required in this program where speed is the main consideration.

The most obvious algorithm is the one contained within the URL class itself. I chose to verify this was the fastest algorithm by comparing it with another popular algorithm MD5. This is described in RFC 1321. An implementation of MD5 in Java was found on the Internet [7]. It keeps very closely to the RFC 1321 description.

Comparison

String Input to Hash function	***MD5 (ms)***	***Java Native Hash (ms)***
http://www.teststring.co.uk/robertneild/	23	13
one two three four five six seven eight nine test testing hash coding algorithms. the cat sat one the mat. the cow jumped over a moon. is the moon made of cheese ?	46	23

The testing results show that MD5 on average is about half the speed of the Java URL hash method, with the speed being largely linear with input string length.

Because of the speed results it was decided to use the java native hash method. However, taking a look into the Java class internals, the algorithm they have used to generate the code is very primitive. Because of their implementation, it makes the likelihood of hash collisions more probable. This is not going to be a problem because the handling of clashes is handled well, using the cache contents table I implemented.

4.7 Cache Concurrency

Java does have some inherent methods to lock objects but this doesn't particularly translate to cached objects on disk well, so it will probably have to be implemented just using a list of locked URLs. However the implementation of locking and unlocking URLs in the cache doesn't stop there.

What is needed is two methods that both take a URL;-

LockUrl - Checks if the URL is already in the list of locked URLs. If it is then wait until it is removed. Then add the URL to the list of locked URLs.

UnlockUrl - Removes the given URL from the list of locked URLs.

Only one thread should be allowed access to the **lockURL** and **UnlockURL** functions at a time. Fortunately Java has built in methods that can handle this quite simply.

Java has a 'synchronized' key word what can be used to get an exclusive lock on the list of locked URLs. Once in the lock it is also possible to call a 'wait' methods that gives up the lock and waits for some other thread to call the 'notify' method.

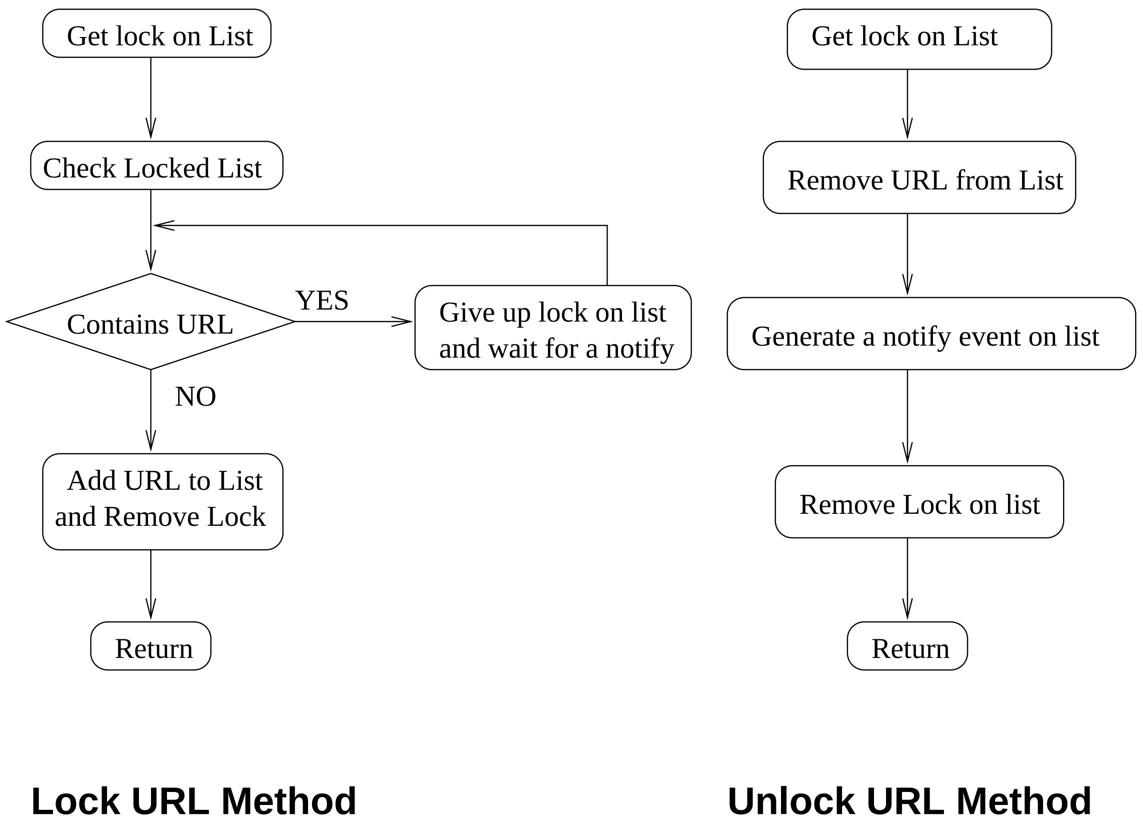


Illustration 18: Flow chart of how URL locking methods work

So before any thread tries to access the cache it calls the LockURL method to lock the particular URL it wants to access, and calls UnlockURL after it has finished.

There methods are implemented in the Cache class.

4.8 Stream Copying

After some initial testing of the program, it was found that the maximum throughput speed of the program was very low. This was traced to the routines that copy data between streams. They were doing this byte by byte. Some more efficient functions were needed.

These method were put into a global class, Util.

The stream copying methods are complicated because it is sometimes needed to copy data from an input stream, coming from the Internet, to two output streams (one going to the cache, and one to the browser). It was decided to copy to both streams concurrently, as this allows the method to return quickly, give a quick response to the browser, and be more efficient.

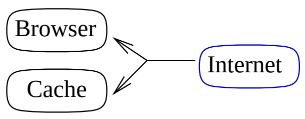


Illustration 19: One stream in, two streams out

To find an efficient method of copying, it was needed to look at the Java class documentation. It was found that there are methods that allow the reading, and writing, of arrays of bytes. The methods were implemented using this way.

It was needed to decide on the size of the copying array. Too large and the response to the browser may be too slow. It was decided on a value of one thousand bytes. This was chosen because of the speed of the connection to the Internet.

4.9 Some Implementation problems

There were many problems that were found and corrected during the implementation of this project, and I am obviously not going to list them all here. However there were a few that I think are both interesting and affected coding the most, that I will document.

Java Sockets

This problem was detected when upgrading the program to use persistent connections. An important part of doing this was to find out if a certain socket was no longer valid i.e. It had idled out and the remote server had closed the connection. There is no explicit command to test a socket, to see if it is valid. Having some experience with sockets programming on Unix and PC machines, using C++, I approached the problem the same way.

When I sent the the request to the server, if the request was seemingly sent without an error being returned, then I assumed the connection was valid and I could listen for the response. However, I found quite often when I started listening for the response, it returned an error saying the socket had closed. My program dealt with this assuming the remote server hadn't liked the request and had closed the socket on me. A HTML page was sent back to the user to signal this to him.

After some testing I found that it was possible for Java to seemingly send data over a closed socket without signalling an error. I am uncertain as to where the data was being sent.

The problems was corrected by using read to test whether a socket was valid because this seemed to return an error if the socket was invalid. To stop the read command blocking until it received some data, before the test was done, the blocking time-out on the socket was set very low.

Incompatibles between Java Runtime Environments

Some of my development was done on different systems, and thus on different JREs. At one point during development the program depended on the finalise methods of my classes being called, so that I could save the table of cache contents out to disk. This worked reasonably well on the Solaris operating system but would only work occasionally on the Linux operating system.

I could increase the probability of the finalise methods being run by explicitly telling the runtime systems to do a garbage collection, and then sleeping for a while, but this still wasn't one hundred percent certain.

The solution to this problem was to explicitly do the cleaning up I wanted when the close methods of the windows were run.

4.10 Program Flow / Class Structure

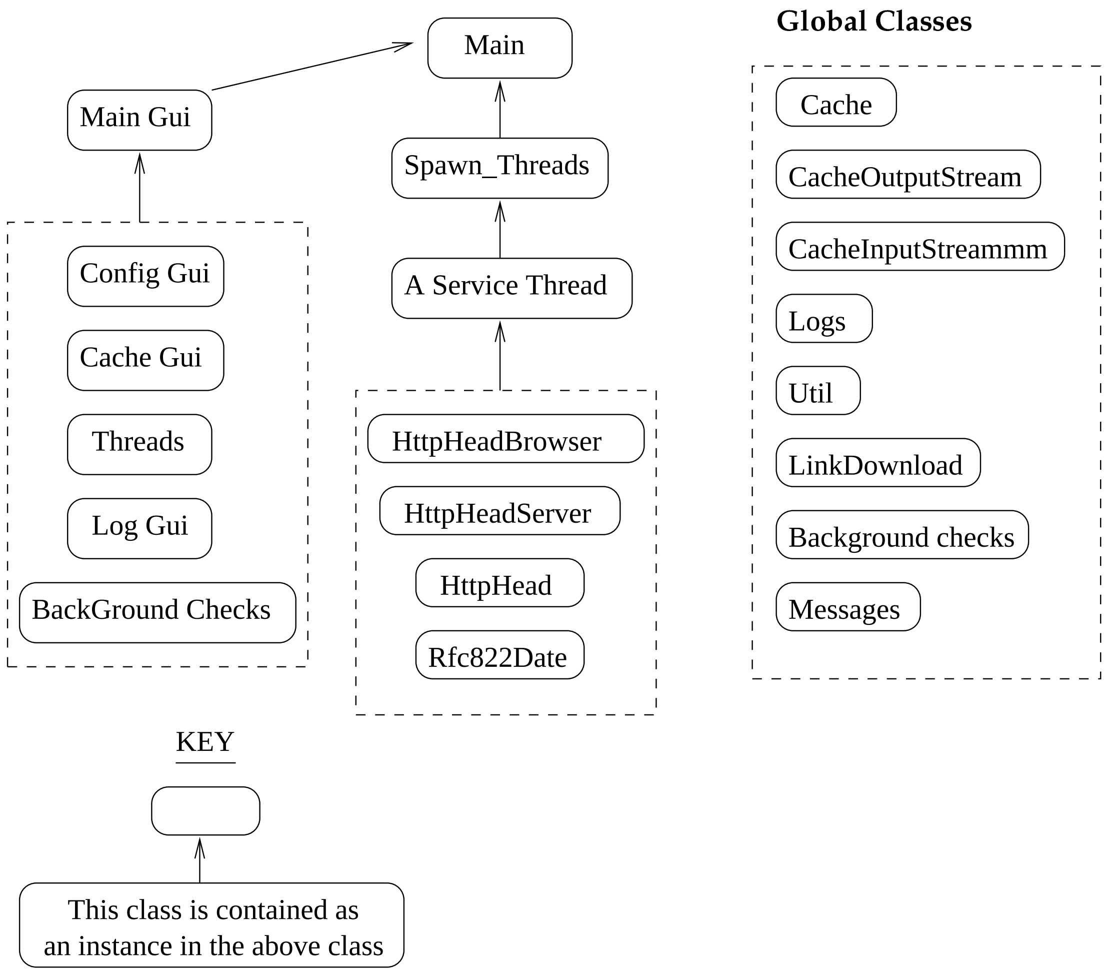


Illustration 20: A simple class structure of the program

The class structure of this program is quite complex and I don't intend to write about it in detail. In the diagram I have only included the main classes, half of the more minor classes have been excluded.

You can see in the figure that the program starts in a class called Main. This class contains an instance of the GUI class, that contains some other GUI classes.

The program works by going in a spawn_threads class. It then spawns off instances of the service_thread class for each connection. That class then manages the connection in it's own thread.

You can see from the diagram above that there are a lot of global classes. This is because it is necessary to keep a lot of global information that all the threads can access. The obvious example of this is the cache class, but there are a lot of other classes, like the background check class, that can only have one instance, but all the threads need to access it.

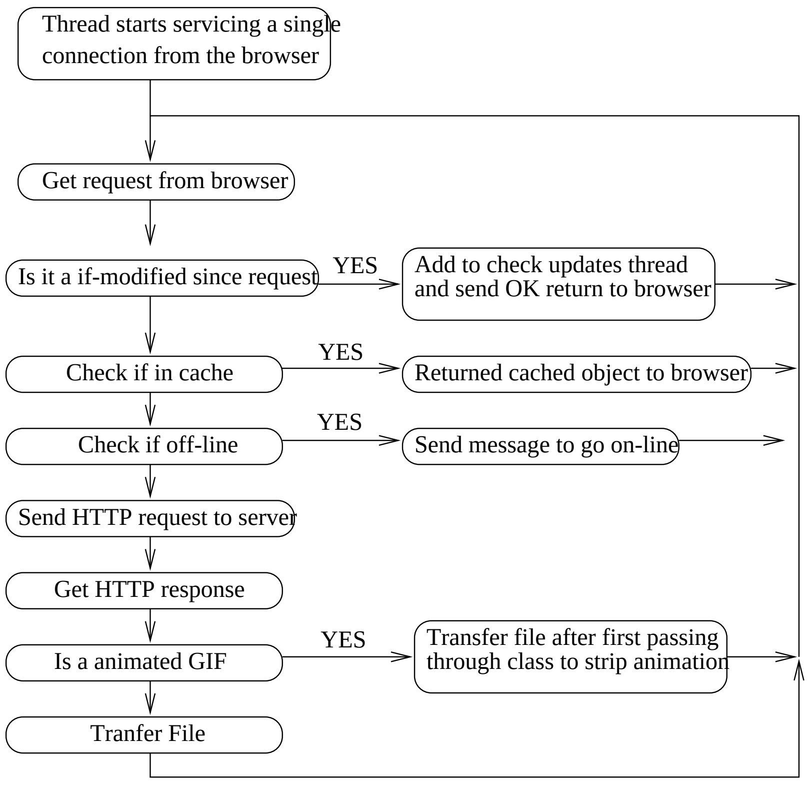

Thread starts servicing a single connection from the browser
Get request from browser
Is it a if-modified since request
YES
Add to check updates thread and send OK return to browser
Check if in cache
YES
Returned cached object to browser
Check if off-line
YES
Send message to go on-line
Send HTTP request to server
Get HTTP response
Is a animated GIF
YES
Transfer file after first passing through class to strip animation
Tranfer File

Illustration 21: Simplified program flow while servicing a thread

Chapter 5: Project Management

Early on in the project a specification was written. The main function of this was to make a final decision on what features the program would implement. I found this decision very important as it stopped me developing peripheral functions, focusing development on the features I had initially decided upon. I think this resulted in the program having a set of well written, reliable features rather than a lot of unreliable ones.

5.1 Top-Down / Bottom-Up

In an effort to develop the program using a top-down methodology a basic GUI was developed early in the project implementation. It was designed to reflect the program features that had already been decided upon in the specification.

Once that has done, I decided to do a lot of the development using bottom-up methods. This was mainly due to me being very unfamiliar with Java and the features that I was implementing.

For example a standalone program was written that could parse a HTML file, before the class was actually inserted into the main program.

5.2 Time Management

Surprisingly I didn't stray too far from the time plan I submitted at Christmas. I think this was due to some past experience I have had with projects and making sure the timeplan was reasonably flexible.

I ended up spending about twice as much time on implementing a basic proxy server than I has first imagined. This was due to me not fully realising the complexity of proxy servers and how they have to handle errors. Fortunately I was able to catch up with the time plan by working a bit at Christmas.

The project was finished a few weeks before the inspection. This time was meant just for testing but I found with extensive testing, some errors. I had to be careful at this point not to produce a cascade of errors by changing something small.

5.3 Conclusion

The design approach I used was more bottom-up than other peoples approach. Comparing this approach with other peoples is very interesting. I found people who had used the top-down approach to excess, tended to have bugs in the program which were hard to track down.

Chapter 6: Testing Proxy Sever Errors

This project, while mainly being concerned with speeding up web Internet access, is also a full proxy server. The proxy server should handle all the normal errors that a regular proxy server would. The results of the test are on the next page (page 39).

How are the Errors Handled

During the standard operation of a proxy server, due to malformed user input, or requests that can't be serviced, the proxy server needs to indicate back to the user why it can't proceed. All the basic errors, like *'host unknown'*, return a suitable, simple, HTML page to the browser. This page shows the error name, a brief description, and also a clear indication that is was generated from the proxy server.

Server Errors

Some of the more interesting errors can happen while connecting to the server. It is possible that ;-

- The host for a request exists, but no connection could be established to the given port.
- The host in a request exists, and a connection could be made to the given port, but the given port isn't a web server and no response comes back when we send the HTTP request.

The testing confirms my program handles these conditions properly with a relevant HTML page being returning.

Major IO Error

Very occasionally when connected to a server the connection can be prematurely terminated. This can be due to a slow server. There is not much that can be done in response to this error as no HTML page can be send to the browser. Testing confirms that my program behaves correctly and shuts down the relevant thread and the connection to the browser.

Chapter 7: Test Results

The speed testing of a program of this type can be extremely difficult due to the changeable nature of the Internet and the dependence on a users habits. The changeability of the Internet can be compensated for by running many tests over time and taking an average. Below follows the results for the individual tests that were run, and what we can conclude from each test ;-

Maximum Throughput Speed Test

This test is to check how much my proxy server, being in the way of a fast connection, would slow the retrieval of date down. It is inevitable that downloading a large file from a fast site though a proxy server will be slower that a direct connection.

Test description: Download of a 3 Meg text file from my Computer Science home page.
Direct browser connection: Average transfer rate = 320k/s
Through proxy connection: Average transfer rate = 43.2k/s

We can conclude from this test result that my program will not transfer data above a maximum rate of 43k/s. Although this is disappointingly slower that the direct speed, it is not thought to be a problem because it is a lot greater than the maximum throughput of a modem (7k/s).

This test was run after I had performed some low level optimisation of the transfer algorithms. This leads to me to believe the limiting factor may be Java's speed.

Download of a large file from a slow server

This is to test that when speed is not the limiting factor, can my java proxy server keep up with transferring a file.

Test description: Download http://www.netcolony.com/members/honeymonster/bigfile.txt - 130830 bytes
Direct download speeds (s) : 126, 165, 57. Average time = 116 seconds.
Through proxy server download speeds (s) : 72, 67, 117. Average time = 85.3 seconds.

We can conclude from this test that the proxy server, when presented with a transfer of a file that is at a rate below it's maximum rate of about 45k/s, it is able to keep up. I think we can discount the speed of the proxy server being faster then the direct connection, in this test, given the widely recorded transfer speeds.

Download of a web page with animated GIFs

A web page with six animated GIFs was put on a reasonable slow server to Test if the animated GIF filter feature works and if it has any speedup.

Illustration 22: Animated GIF Page

Web page URL: http://www.netcolony.com/members/honeymonster/index.html
Direct download speed(s) : 84, 109, 131. Average = 108 seconds.
Proxy (nocache) download speeds (s) : 104, 70, 90. Average = 88 seconds.

From the results we can see there is a small speed up. There figures aren't more impressive because most of the time seems to be taken with looking up the IP address of URLs and making TCP/IP connections.

Garbage Clean Speed

This test isn't essential because hopefully the time to do a garbage clean won't affect the speed of the rest of the program. However, it is interesting to know as it could be a limiting factor on the size of cache.

Cache Size	*Cache Max Size*	*Number of bytes garbage clean deleted*	*Garbage cleaning time (secs)*
635,445	200,000	439,245	1.08
1,026,774	1,000,000	29,776	0.558
1,006,951	1,000,000	7,536	0.088
1,027,286	1,000,000	34,944	0.550

These results are quite impressive and a lot better then expected. The speed of a clean seems to depend more on the amount of data that has to be deleted from the cache, as then the size of cache to start with.

Background Update Check Test

Testing this feature needs to be done on a web site that I can manage, so I used my computer science one.

A web page was loaded through the proxy server, then the *'touch'* command was run on the files in my web page to update the file access times. When the reload button is pressed on the browser, the browser returns immediately and after a short time a dialog pops up saying the current page is old and that reload needs to be pressed again.

This is exactly as it should operate.

Prefetch Link Test

Testing of this feature was done on the Computer Science home pages as they are simple and stable. Tests verify that when a HTML page is looked at, the links from it are downloaded in the background. However, testing revealed a problem with this feature in that if the user changes the page they are looking at, while links are being downloaded, the links continue to be downloaded from the old page. Other than that problem this feature works correctly.

Chapter 8: Analysis

8.1 Performance

The performance of the program in my eyes is 'reasonable'.

The general speed of it when working as a general proxy server is good, as long as the maximum transfer speeds don't reach it's limits. The speed of accessing the cache, even with large cache sizes of two Megabytes is very fast.

The link download, background checks and GIF filtering all work at acceptable.

I have found that as the program get larger, the speed seemed to slightly decrease. I think this is down to the Java interpreter. It would be interesting to compare the performance of to a similar program written in C++.

8.2 Reliability

The current reliability of this program is of some concern to me. Browsing on 'safe' standard site, such as the Birmingham University web pages, is found to be completely reliable after testing. When going around different sites on the Internet the program does seem again to perform with out problem.

However, after half an hour of use, it is possible to find sites where minor 'strange' things happen (connection going down unexpectedly etc.). Although these problems are not serious I am not satisfied that the program is completely full proof.

Through the experience of writing this program, I have realised that there is a need for extensive beta testing on any program developed in this area. I believe this is due to the various, slightly different, implementations of HTTP on different servers, and the concurrent, multi-threaded aspects of this program.

I believe this is confirmed by the steps the people who make the Apache web server and SQUID proxy server take to test their products.

8.3 Un-implemented / Non working Features

I have not finished the storage of cached files in a hierarchical directly structure, however, I am storing them using a hashing code.

I also think there is a bug in the link downloader that keeps downloading links from a page, even when the user has gone to another page.

Both these problems are only thought to be reasonably minor.

8.4. Class structure

The class structure of this program is a little strange, mainly because of the need for threads to access common data. I still think that there is room for it to be improved. This could have been done by doing a fuller, more detailed design.

Chapter 9: Conclusions

9.1 Possible Extensions

There is certainly some space to extend the configuration dialogs e.g. To tailor the link download options some more. I would also like to improve the reliability of the program. I would guess this would best be done by some beta testing.

As for major extensions to the program, it would be possible to make the program filter out adverts more intelligently. Whether this would be done by an AI, a list, or a online database I don't know.

I think it would be interesting to slightly change the orientation of this program to be a large, multi-user cache, that knows when it is off/on-line. This could be then be used by companies that have only temporary Internet connection e.g. A LAN connected to a modem that dials out when ever someone makes an Internet request.

9.2 What would I have done differently

One of the main problems with this project was my tendency to go for a bottom-up implementation. Following that approach meant the upper levels of the program had to be re-written a lot, as lower level, and other levels that depended on it, were added to it. This, I think at the time was reasonably justifiable, because of my lack of knowledge of Java and of the area. I found as I progressed in the project, I came across many problems I had not foreseen. Certainly now I would be able to produce a more detailed specification and design.

9.3 Achievements

A list of the more tangible skills I think I have gained a lot of knowledge in is ;-

- Java
- HTTP
- HTML

- Threads/Concurrency/Locking
- Proxy servers
- Web servers
- Hashing codes
- Garbage cleaning

As well as these specific skills, I think the main skill I have gained experience in, out of all the less tangible ones, is self motivation. I think for me, this was the key skill that the whole project depended on.

When the project was started, and for the first few months, the project is exciting and interesting to work on. However, after that period it becomes very difficult to motivate yourself to get the project finished. I have found, with this program, that that completing the last ten percent of the program, seemed to take at least thirty to forty percent of the time.

I also leant ;-

- How to design a large program.
- Testing methods.
- Customer research.
- Time management.
- Writing a large report.

9.4 Overall Conclusion

Writing this project has been hard going at times, but enjoyable. I have been surprised at all the areas I've had to get involved in. The project turned out to be a lot larger than I had imaged but I have learnt a lot or skills and methods along the way.

Whether the project turns out to be useful to someone is yet to be seem, but I sincerely hope it will be.

Appendix A: Some Calculations

Calculate the approximate size of cache table of contents for different sizes of disk cache

Disk cache size = *x*

Average size of files in cache = 5000 bytes

Each line in contents table contains the;-

URL Name	40 characters	
Hashing code	10 characters	
Size of file on disk	6 characters	
Last access time	6 characters	
================================		
Total	62 * 2 bytes =	124 bytes

Number of files in cache = *x*/5000

Size of table of contents = (*x*/5000) * 124

Cache Size (x)	1,000,000 (1M)	10,000,000 (10M)	100,000,000 (100M)
Size of table of contents	24,800	248,000	2,480,000

Appendix B: Testing the java.util.Date[13] Class

The following code was used to test the *java.util.Date* class parsing, and the format of it's standard output to see if it was acceptable for this program.

```
import java.util.*;
import java.text.*;

class DateTest
{
   public static void main(String args[]) {
               System.out.println("Started....");
               System.out.println(new Date("Thu, 18 Mar 1999 11:13:17 GMT")));
               System.out.println(new Date("Sunday, 06-Nov-94 08:49:37 GMT")));
               System.out.println(new Date("Sun Nov  6 08:49:37 1994")));
               System.out.println(new Date().toString());
      }
}
```

The Output of the above source code

```
Started....
Thu Mar 18 11:13:17 GMT 1999
Sun Nov 06 08:49:37 GMT 1994
Sun Nov 06 08:49:37 GMT 1994
Thu Mar 18 20:53:08 GMT 1999
```

Conclusions

The output proves that the Date class if able to parse the three formats correctly but its output format is unacceptable as it doesn't conform to RFC 822 which has a form;-

Sun, 06 Nov 1994 08:49:37 GMT

13 Copyright (c) 1995, 1996 Sun Microsystems, Inc. All Rights Reserved.

Appendix C: HTTP Digest

HTTP is a request/response protocol. A client sends a HTTP request to a server with the server then responding with a HTTP response header, followed by the requested data body. Although usually, the requested item is a HTML page, HTTP is not tied to this protocol and so any type of data can be returned.

HTTP Versions

There are three HTTP versions at the moment;-

- 0.9 - Now very old and not used.
- 1.0 - The currently most popular version. Described in RFC 1945.
- 1.1 - A reasonably new version, not fully defined and just being implemented. Described in RFC 2068.

Version 1.1 is just appearing in newer browser and servers. Unfortunately, being a new protocol, it isn't as stable as 1.0. Because of this, and because 1.0 offers all the functionality I need, I have chosen only to support 1.0 in my proxy server. Version 1.1 servers will be able to understand 1.0 requests anyway.

Unfortunately, unlike some other Internet application level protocols, like FTP, HTTP is not implemented with the same strictness. There are many different servers and browsers in use, and they can implement very slightly different flavours of HTTP.

In particular there is some blurring between 1.0 and 1.1. The main one being persistent connections that are widely used in 1.0.

HTTP Request Format

HTTP requests are in ASCII text form and a typical one looks like;

```
GET http://www.cs.bham.ac.uk/ HTTP/1.0
User-Agent: Mozilla/4.0
Accept: text/html
```

The first line in the request is the most important one, as it specifies the action we want to the server, the URL that we want, and the HTTP version we are using.

HTTP Response Format

A typical HTTP Response looks like;-

```
HTTP/1.0 200 OK
Server: Apache
Date: Sun, 11 May 1997 09:30:37 GMT
Content-Size: 1000

<HTML> ................... etc.
```

Again, the important line is the first one. In this case it tells us that the request was successful and that the requested date will follow after the HTTP response header. The end of the header is indicated by a blank line. After the blank line, follows the data.

Appendix D: Comparing the Performance of JLex and a hand written parser

JLex Parser

The main body of the trial parser, that was written using JLex is listed below. The full listing for the JLex test is in jlex/, and the manual parser test is in link_extract_test/.

```
<YYINITIAL> "<"            { yybegin(STATE1);  }
<YYINITIAL> {WHITE_SPACE}+ { }
<YYINITIAL> .              { }

<STATE1> {WHITE_SPACE}+    { }
<STATE1> [Aa]              { yybegin(STATE2); }
<STATE1> .                 { yybegin(STATE6); }

<STATE2> {WHITE_SPACE}+    { }
<STATE2> ({H}{R}{E}{F})    { yybegin(STATE3); }
<STATE2> .                 { yybegin(STATE6); }

<STATE3> {WHITE_SPACE}+    { }
<STATE3> "="               { yybegin(STATE4); }

<STATE4> {WHITE_SPACE}+    { }
<STATE4> "\""              { yybegin(STATE5); }

<STATE5> ({ALPHA}|{DIGIT}|{WHITE_SPACE}|{PUCTUATION})+  { return new Yytoken(yytext());
}
<STATE5> "\""              { yybegin(STATE6); }

<STATE6> ">" { yybegin(YYINITIAL); }
<STATE6> {WHITE_SPACE}+    { }
<STATE6> .                 { }
```

Performance Comparison

A test was done by parsing a reasonably simple HTML file, with 22 links in it, as well as some links that should be parsed out. An average was taken of eight runs.

JLex: 328, 330, 330, 323, 328, 328, 324, 324	*Average = 326.8 ms*
Hand written: 266, 265, 251, 334, 264, 284, 300, 269	*Average = 279.1 ms*

Conclusion

The hand written parser is marginally faster.

Appendix E: Users Guide

Environment

This program is written in Java and so requires a JRE (Java Runtime Environment) to run[14]. The specific requirements of this are;-

- Swing 1.0.3 (or greater)
- JDK 1.1.6 or 1.1.7
- Can use native or green threads (Better performance with native)
- Not JDK 1.2

You can download JREs for Microsoft's and Sun's operating systems from www.java.sun.com, and the JRE for Linux can be found at www.blackdown.org.

Swing can be downloaded from Sun's Java web site as well. Once downloaded it needs to be set-up so that the JRE can find it. This means that the 'CLASSPATH' environment variable includes the swing package.

The Java JDK 1.2 came out while I was doing this project. It includes Swing as default. However, there are some minor changes between using 1.2 and 1.1.6 so the program will not run without modification on 1.2.

Compiling

The program can be compiled by Suns javac when in the directory containing the source Java files, using the command ;-

```
javac  -depend Main.java
```

This will create a directory called java_proxy that will contain all the byte compiled Class files. This directory can be zipped up to create a compressed, self-contained package.

14 The full JDK will be needed if you want to compile the program

Running

As long as the java_proxy package is in the CLASSPATH the program can be run by;-

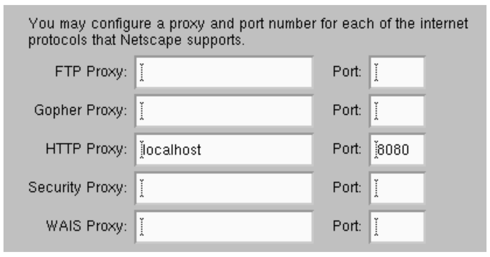
java -native java_proxy.Main

The -native switch makes threads use the native system implementation. This improves the performance of the program.

Setting up you browser

In order for this program to work, all Internet HTTP requests must pass through it. To do this you must manually setup your browser to use it as a proxy server.

Find the relevant dialog from your browser preferences and enter into the HTTP proxy server field 'localhost' (your computer) and then put in what ever port you have configured the proxy server to use.



You may configure a proxy and port number for each of the internet protocols that Netscape supports.

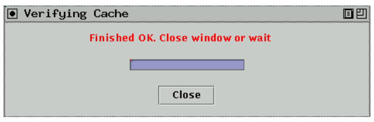
FTP Proxy:		Port:	
Gopher Proxy:		Port:	
HTTP Proxy:	localhost	Port:	8080
Security Proxy:		Port:	
WAIS Proxy:		Port:	

Illustration 23: Screen shot of configuring Netscape Navigator 4

GUI Description

Straight after the program is run, it puts up a dialog while it is checking the cache. It checks to see if its table of cache contents, which it loads from disk, reflects the cached files on disk.

If no problems are found with the cache then the dialog automatically closes.



Verifying Cache

Finished OK. Close window or wait

Illustration 24: Dialog that shows, while cache is being checked

The program opens a small window with a small footprint, so as not to intrude on the browser you are using. It can now be used as a proxy server.

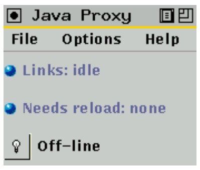


Illustration 25: Main GUI window

Appendix F: Running the Program from Birmingham University’s Computer Science Department

Setting up Swing and Java

On the Birmingham University, Computer Science computers, the set-up can be done for you by running *’setup Java’* and *’setup Swing’*.

Compiling

First change to the directory that contains the source code;-

cd ~rxn/java_proxy/

Then the program can be compiled using;

make javac[15]

This puts the java_proxy package in a directory *compiled*.

Running

The program needs to be run from the *compiled* directory. This contains all the image files that are used for buttons, the preferences files, and the directory *cache* that contains all the cached files.

To go to the compiled directory from the directory containing the Java files type;-

cd compiled

Once there program can be run by ;-

15 Compilation takes about 90 seconds using javac. An alternative that I used during development is a Java compiler called Jikes. This is available for free from www.ibm.com.

java -native java_proxy.Main

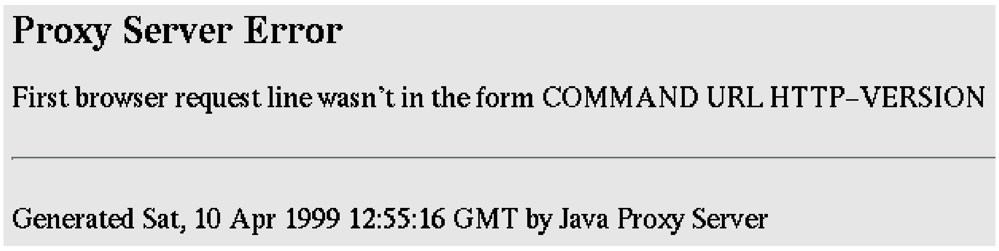
(The -native switch makes threads use the native system implementation. This improves the performance of the program.)

Here is how that looks when I go through the above;-

```
rxn ~ $ cd java_proxy/

rxn ~/java_proxy $ make javac

javac -d compiled -depend Main.java

Note: 4 files use deprecated APIs.  Recompile with "-deprecation" for details.

1 warning

rxn ~/java_proxy $ cd compiled/

rxn ~/java_proxy/compiled $ java -native java_proxy.Main
```

Appendix G: Screen Shots

Screen Shots of HTML Page Responses from Proxy Server to HTTP Request Errors

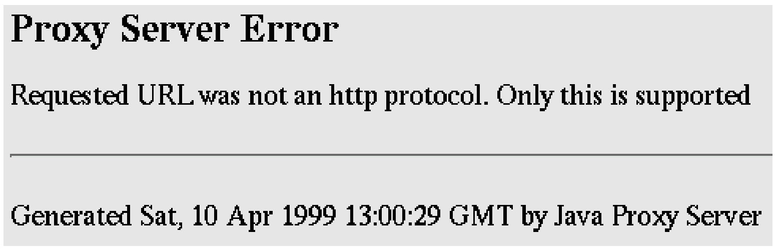


Proxy Server Error

First browser request line wasn't in the form COMMAND URL HTTP-VERSION

Generated Sat, 10 Apr 1999 12:55:16 GMT by Java Proxy Server

Illustration 26

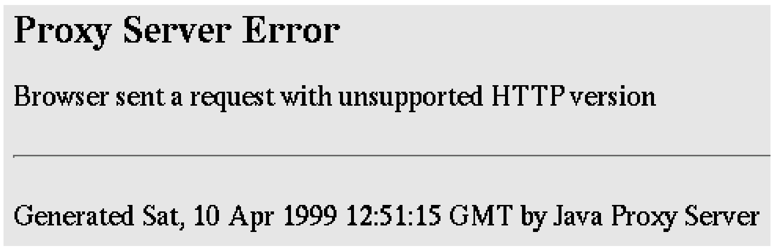


Proxy Server Error

Requested URL was not an http protocol. Only this is supported

Generated Sat, 10 Apr 1999 13:00:29 GMT by Java Proxy Server

Illustration 27

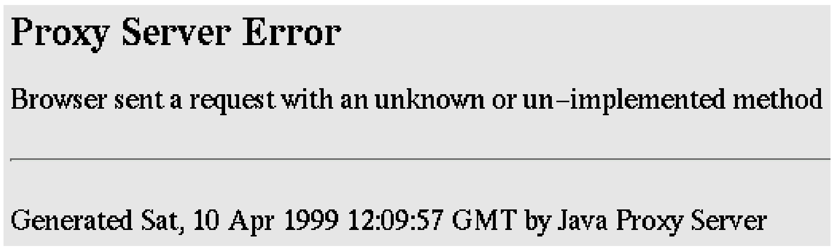


Proxy Server Error

Browser sent a request with unsupported HTTP version

Generated Sat, 10 Apr 1999 12:51:15 GMT by Java Proxy Server

Illustration 28



Proxy Server Error

Browser sent a request with an unknown or un-implemented method

Generated Sat, 10 Apr 1999 12:09:57 GMT by Java Proxy Server

Illustration 29

Appendix H: Brief Class Descriptions

Global Classes

Log This class is accessed by most of the other classes and provides logging functions to them. It is a debugging class that controls various files on disk. The various methods it contains, write out text to the corresponding file.

Util This class is a utility class that contains a few commonly used, simple functions. It has functions to exit the program on an error, copy data efficiently between streams etc.

CacheReadInputStream Allow the reading of a cached object through a conventional input stream. When it is constructed, it locks the URL while it is reading from it. Because it locks the URL when it is constructed, it’s close method must be called by the owner when it has finished with it, so the URL can be unlocked.

CacheOutputStream (+ some private classes) This class allows the additon of a file to the cache. It is constructed around a certain URL, locks the URL, and then it can be used as a conventional output stream to add data to it. The close method is called to close it. The internals of this class are complicated because it allows the addition to the cache of a file of unknown length. In this case the file is buffered into an array. Only when close is called is the file actually written to disk, because the file size is now know.

Cache Contains lots of methods for manipulating the table of contents, the cache, and locking URLs.

Bibliography

RFCs

BERNERS-LEE, T. "**RFC 1866:Hypertext Markup Language-2.0**".

BERNERS-LEE, T., FIELDING, R. and FRYSTYK, H. "**RFC 1945:Hypertext Transfer Protocol-HTTP/1.0**".

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H. and BERNERS-LEE, T. "**RFC 2068:Hypertext Transfer Protocol-HTTP/1.1**".

RIVERST, R. "**RFC 1321:The MD5 Message-Digest Algorithm**".

Books

LUOTONEN, ARI. "**Web Proxy Servers**". *Prentice Hall, New York*. (1998)

EDDY, S.E. "**HTML in Plain English**". *MIS:Press, New York*. (1998)

APPEL, A.W. "**Modern Compiler Implementation in Java**". *Cambridge University Press, Cambridge*. (1998)

Papers

WESSELS, D. and CLAFFY, K. "**ICP and the Squid Web Cache**". *National Laboratory for Applied Network Research*, http://www.nlanr.net/wessels/Papers/icp-squid.ps.gz.

Programmers Reference

UNKNOWN AUTHOR. "**GIF98a Specification**". *Compuserve Incorporated Columbus, Ohio*, http://www.phys.s.u.-tokyo.ac.jp/local/other-faq/gif89a-e.html.

References

[1] NetAccelerator2.0 - IMIS, http://www.imisoft.com/

[2] SQUID, Squid Project group, http://squid.nlanr.net/Squid/

[3] WebEarly98 - Goto Software, http://www.webearly.com/

[4] Web Proxy Servers, by Ari Luotonen. Page 107.

[5] CERN httpd is also refered to as W3C httpd as the development moved from CERN to the W3 Consortium.

[6] JLex: A Lexical Analyzer Generator for Java(TM), by Elliot Berk. http://www.cs.princeton.edu/~appel/modern/java/JLex/

[7] Java MD5, Santeri Paavolainen, http://www.cs.hut.fi/~santtu/java/

[8] Apache web server, Apache project, http://www.apache.org/.