Client-side proxies

- a better way to individualise the Internet?

Master's thesis[1]

Tomas Viberg

Department of Computer and Systems Sciences
Stockholm University / Royal Institute of Technology

May 2000

Abstract

With the growth of the Internet, information overload has become a problem. Because of the sheer amount of available data, there is a need for tools that can find and transform data into useful information. Existing tools, such as online search engines, directories and more or less specialised portals are popular, but they do not adjust very well to individual needs. This thesis examines an alternative approach - client-side proxies, running on the end-user's local machine. More versatile than the original proxy servers, they have the ability to intercept communication to support information retrieval and adaptation of content.

To establish the benefits and drawbacks of the approach, a number of existing proxies has been compared with each other and with applications that use different techniques to perform similar tasks, such as integrated clients (browsers, newsreaders, etc), client plug-ins or Web services. The results of this two-phased evaluation show that client-side proxies have merits that distinguish them from other content processing applications. The combination of direct and exhaustive access to the content, client independence, support for aggregation of functionality and complete access to the power of the local computer is a strong argument to use client-side proxies for content processing. However, when usability or performance is crucial, other approaches could be better. Client-side proxies introduce greater overhead than other approaches do and they are generally harder to install and configure. Consequently, even if client-side proxies are better, there is the risk that they will only be embraced by more advanced users.

Provided as part of this thesis, Blueberry is a framework for content processing. Building on the evaluation results, this Muffin extension exemplifies how to integrate a consistent user interface with the client application to increase usability while maintaining the independence of the proxy approach. Through high-level data abstraction, it also shows a way to help developers of third-party extensions increase their productivity.

1. This thesis corresponds to 20 weeks of full-time work.

Table of contents

1 Introduction

The Internet and the World Wide Web provides vast and ever growing amounts of data with possibly great value. However, it is also a very messy place, and there clearly is demand for tools that can help clear away the rubble and transform the data into useful information for a particular user. One clear indication is that online services as search engines, directories and more or less specialised portals rate among the most visited sites on the Web [Waxman 00]. Albeit highly visible and advertised, these online services are only one way to get the work done: finding useful information and adapt it to the needs of individual users. This work will examine an alternative approach.

1.1 Aim and scope of this thesis

The aim of this thesis is to evaluate the use of client-side proxy servers for finding and adapting information according to user preferences, and to present Blueberry, a partial proxy prototype that highlights some ideas for development of the approach. Traditionally, proxy servers are specialised large-scale Web servers aimed at improving use of network bandwidth through document caching and improving network security by, for example, restricting access to certain content. In contrast, a client-side proxy is a small-scale server running on the user's local computer. It could also be used for improving performance and security, but what is of interest in this context is a more versatile type of proxy server, a proxy with the ability to support information retrieval and adaptation.

To estimate the value of such proxies as opposed to the value of using other methods, some more specific questions need to be answered: Could applications using the client-side proxy approach be better at providing information that fits the needs of an individual user? If so, under which circumstances and for whom? Are there situations when some other technique is preferable, such as browser plug-ins, online services or stand-alone applications? Are the potential benefits of this approach realised in existing systems? If not, in what ways could they be improved? These are the questions that will be examined throughout this work and some of the answers will be visualised in the proxy prototype implementation that is also a part of this thesis.

The scope is limited to the merits of using client-side proxies as an architecture in which functionality for information retrieval and adaptation can be implemented. The functions themselves, such as filtering, collaborative rating, privacy enhancement, etc, will certainly not be overlooked, since there is often a close relationship between the architecture of a system and its functionality. However, they will not receive the full scientific treatment because each of these fields could easily qualify as a separate thesis topic.

1.2 Purpose of examining client-side proxies

The common denominator of many of the popular online services mentioned earlier is that they act as a middleman between the user and massive amounts of information. Just like real-world travel agents, newspapers and libraries, the middlemen on the Web use their domain-specific knowledge and analytical skills to make it easier for the individual to find what he is looking for. This function is clearly in demand despite the prediction that the Web would replace the human middlemen with "Cool Software" that would analyse the user and efficiently provide relevant information [Ganesan 99].

Client-side proxies clearly fall under the "Cool Software" category. So why bother examine the merits of this approach, if the advantage lies with those who provide some more subtle and non-computable service?

One reason is that some of the services provided today are simply not good enough. A popular service (at least among parents) is filtering or blocking content that is "harmful to minors". Such censoring filters have blocked access to web-sites such as middlesex.gov (due to the domain name), The Privacy Forum (due to a discussion about cryptography that was rated as criminal skills), and other subversive material such as the United States Constitution, the Bible and the plays of William Shakespeare [Neumann and Weinstein 99]. Furthermore, a search engine advertised to be "family-friendly" filtered away about 90% of the relevant hits when queried for material about the American Red Cross, San Diego Zoo and Christianity.

These results certainly indicate that there is a need for a better solution, a solution more adapted to the needs and wishes of the individual user and not coloured by the biased opinions of the middleman. As one among several alternative solutions, client-side proxies could provide a framework for making content retrieval and adaptation more flexible and better adjusted to individual needs. As there probably are situations when this is indeed a better solution than what is available today, and since the merits of this architecture are relatively unexplored, it is an approach worthy of attention.

1.3 Contributions from this thesis

The evaluation of the client-side proxy approach to information retrieval and adaptation is in itself a contribution, since this is a relatively unexplored field. The results of the evaluation should be able to provide useful guidelines for those interested in using this approach.

Blueberry, the prototype implementation that is part of this thesis, could also be viewed as a contribution to the field. As an example application, the goal is that it will provide ideas as to how a proxy can be designed to take advantage of the potential strengths of the approach. As noted, there are already a number of existing client-side proxy applications available, but this one will highlight some issues that has not been fully utilised in the available systems.

1.4 Thesis outline

Section 2 provides the background for this work, giving a brief description of other material about the use of client-side proxies for content processing and identifying some key issues regarding use of such proxies. The methods used in the evaluative sections of this thesis are outlined in section 3. Results of the evaluation are described in sections 4 and 5. The former focuses on the evaluation of client-side proxies compared to other types of applications performing similar tasks. In the latter, the architectural differences between existing client-side proxies are examined. The proxy module implementation of this thesis, Blueberry, is described in detail in section 6. Concluding the thesis, a discussion of the results and ideas for further research can be found in section 7.

2 Background

The most obvious starting point for a survey of related works is to look at other works with a similar comparative approach to the client-side proxy architecture for content processing. However, there does not seem to be any, so instead this background will survey documentation about using client-side proxy servers as a fundamental part of application architecture. What will primarily be examined is for what tasks the proxy is used, notable details of the proxy architecture, deployment experiences and, if this is discussed, the reasons why the proxy approach was preferred.

2.1 The original proxy

One original function of proxy servers is to intercept communication between client applications and remote servers in order to improve network efficiency through caching. Since network congestion is not a diminishing problem, this continues to be an important function of proxy servers [Thaler and Ravishankar 98]. As all requests go through the proxy, documents that are requested frequently can be stored locally for later use, decreasing both the response time experienced by users and the overall network load of subsequent requests.

Providing caching through a proxy is a natural choice. The proxy provides a service that is transparent to the user as well as to client and server applications. Transparency can be beneficial since users probably are more interested in the service provided than in the particulars of its functionality. Transparency also allows users to share a single proxy easily, for example on a local area network. It is in this situation that the biggest gains of a caching proxy are realised.

There will be no in-depth discussion of traditional caching functionality, since the focus of this work is on proxies working locally as single-user applications. At the same time, caching functionality in a client-side proxy could prove beneficial to the individual user, for example by increased browser independence. Through this a user gets more control of what is stored locally, the ability to switch client and still have access to the same cached documents and a consistent way to view documents off-line, regardless of client support. Another reason to include caching functionality in client-side proxies with other tasks is that these tasks themselves might result in increased response time. When caching is discussed, it will be in this context, as a way to improve the efficiency of client-side proxies.

2.2 A more versatile approach

Moving away from the traditional view of proxy servers towards the kind of proxies examined in this thesis, [Brooks et al 96] "generalise the notion of proxy servers to construct application-specific proxies that act as transducers on the HTTP stream". Normally, clients and servers expect that requested documents remain unchanged during transport from server to client, even if they are cached copies. The motivation for this transgression is that substantial value can be added by working directly on HTTP streams to view and alter the contents. The stream transducers, called OreOs, can have practically any functionality, implemented examples include URL validation, measuring network performance, creating group histories, supporting group annotation of documents and creating full-text indexes of accessed documents.

Every OreO is a specialised stream processor, with the freedom to use information from any obtainable source and to produce arbitrary output. The architecture is modular, aimed at facilitating sophisticated behaviour by aggregating highly specialised modules. This is supported by the ability to place OreOs in a chain, so that the output from one is the input for another. This kind of system can be configured with high granularity and set up to support the specific needs of different classes of users, from individuals through groups to enterprises and the public.

Introducing processing modules in the content stream affects the performance of network transactions, especially if many modules operate on the same stream. However, during tests the delay caused by introducing OreOs in the stream was mostly so small that users hardly perceived them, as they were accustomed to variations in network performance. The delay naturally depends on the efficiency and complexity of the different OreOs, but if the delays are kept small it does not have to be a big problem, especially if the added value is substantial enough.

A proposed architectural improvement is to encapsulate the content stream using a higher level of abstraction than the current low-level byte stream. This would probably help third-party developers increase their productivity, and this is indeed a notion supported by several client-side proxies today. Other issues of interest are how to achieve the benefits of a modular approach and ways to minimise the impact on performance.

2.3 Some example proxies

One system using the notion of proxy servers described above is Crowds [Reiter and Rubin 99]. It enables users to retrieve Web content anonymously, using a client-side proxy server as the backbone of functionality. The idea is to create crowds of users and relay requests through a chain of proxies in the crowd. Neither the addressed server nor the proxies along the relay path can be sure who originally sent the message. Why the proxy solution was chosen is not explicitly stated. A reasonable assumption is that it was because the task at hand is to intercept communication between the client (browser, ftp client, etc) and the server transparently.

Experiences from deployment of the Crowds system have shown that there are some potential drawbacks to the proxy approach. As already mentioned, any intermediary might slow down the retrieval of content and/or result in increased network traffic. If the proxy is aimed at improving network efficiency this is not an issue, but that is not the goal of the Crowds system, and so there will be some performance degradation.

There could also be problems when trying to use client-side proxies behind firewalls or other security constructs. In the Crowds system, the proxies communicate through non-standard network ports, which might be disallowed. A related problem is that system administrators often want to monitor user communication. However, monitoring users in a crowd is not easy, which could inspire administrators to forbid the use of such systems. This is not a problem directly related to the use of proxies, but since several existing proxies are used for enhancing the privacy of its users, it is an interesting question. These and related legal, moral and ethical questions will be discussed further in later sections.

Pavilion is a framework for developing collaborative web-based applications [McKinley et al 99]. An important part of the framework is a client-side proxy server, with both traditional proxy functionality like caching frequently requested pages and tunnelling content through firewalls, and functionality that is more versatile. The default behaviour of the Pavilion proxy is to provide a group with a common view, for example, allowing several users to automatically view the same document as the group's leader. This is achieved by multicasting information from the leader proxy to the other proxies in the group. Apart from this, the Pavilion framework uses the notion of extensible proxies, meaning that external modules can be attached to the proxy as plug-ins to facilitate type-specific processing of requests and resources before their delivery to the client application. This architectural detail is interesting since it facilitates processing of the actual content flowing through the proxy, as opposed to proxies that simply relay requests and replies, ignoring the content. Through this, Pavilion realises the notion of a content-altering proxy.

Apart from the proxy server, Pavilion also offers interfaces to popular web browsers and protocols for floor control and multicast delivery of content, both aimed at facilitating distributed collaboration. Browser integration is achieved with operating system-specific inter-process communication mechanisms. This is an approach with possible negative effects on the platform and browser independence of a system using the Pavilion framework.

In the context of this work, the Pavilion framework raises two issues to be examined further. First, the merits of extensible proxy servers will be discussed in more detail in subsequent sections. Second, the question of whether browser integration is desirable, and if so, how it should be done, will also receive attention.

Browser integration is also an issue in WebMate, a system for helping users browse and search the web more effectively [Chen and Sycara 98]. WebMate uses a local stand-alone proxy server to monitor and learn from the browsing and searching behaviour of the user. This system provides a relatively close integration with the client's browser environment, not by using browser or platform dependent methods but by inserting the user interface directly into the requested document. The user can interact with the system through a controller applet at the bottom of each document, supplying interests, providing relevant information for processing and receiving help. Whether or not this is a better solution to browser integration than the one Pavilion provides will be examined later.

The WebMate proxy is used for more demanding tasks than in previously described systems. Intercepting communication between server and client is one of the functions, but the content of this communication is not altered in any significant way. Communication patterns and user feedback is processed with machine-learning algorithms to build and refine a model of user interests based on keywords describing relevant documents. Through this model, WebMate can automatically provide documents of interest to the user. Another task is to increase the quality and relevance of search results through criteria refinement and keyword expansion. Both these tasks require advanced functionality and algorithms; functionality implemented directly in the WebMate proxy rather than provided as plug-in functionality to a modular proxy server.

2.4 Proxies in mobile environments

To revisit the more traditional proxies, one common use is to provide a bridge between different transfer protocols. For example, a web browser lacking knowledge of the gopher protocol can access gopher-based material through a proxy. The proxy acts as a translator, speaking HTTP to the browser and gopher to the server. Taking this a step further, a proxy can act as a connection between fundamentally different environments such as stationary and mobile environments. One current example of this approach is the Wireless Application Protocol [WAP 00] that utilises proxy servers to adapt standard Web content to mobile devices through negotiation and translation. Most traditional techniques assume that the location of clients and the client-server connection remains unchanged during communication sessions, which is obviously not the case in mobile environments [Jing et al 99]. The mobility of clients, differences in display technology and the relatively low bandwidth of wireless links are some of the factors that must be taken into account when adapting content from stationary networks to the needs of mobile users.

Adaptation of communication and content can be made mobile-aware using different techniques, of which transparent proxy-based adaptation is of most interest here. The proxies are rarely pure client-side proxies, since stable wireless communication often requires processing on both the mobile client and in the stationary network. Client-side proxies running on mobile devices still play an important role, providing an interface to regular servers and attempting to shield the negative effects of the mobile environment from applications and users. Transparent caching, prefetching of requested documents and support for disconnected operations are among the tasks performed by mobile client-side proxies.

Transparent adaptation to mobile environments might be detrimental to overall functionality and performance, since it is very difficult to meet the diverse needs of different applications not themselves mobile-aware. Allowing the affected applications to control parts of the adaptation process might prove useful. This issue is not specific to mobile environments, and the benefits and drawbacks of transparency will be discussed further.

2.5 A great diversity

The overall impression of the background survey is that there is a great diversity of choices made in the design of systems using client-side proxies, regarding both functionality and fundamental architecture. Despite this, one possible conclusion is that client-side proxies are most useful when the task at hand involves monitoring or altering the communication between clients and servers. This will serve as a starting-point for resolving when a proxy approach is appropriate and when it is not. Supposing such an approach is preferable, other important issues can be identified and must be evaluated.

One issue is whether the proxy architecture should be monolithic or modular. This touches on the subject of creating sophisticated behaviour by aggregation and if this should be supported by chaining, extensibility or not at all. If a proxy is extensible, how to present the content to developers of additional functionality is a relevant issue. Should a developer have access to the content as a low-level byte stream, or should the proxy parse the stream to provide a higher level of abstraction, such as wrapper objects for individual HTML elements? How to avoid performance degradation and the level of transparency are other issues related to application architecture. Also of interest is if a proxy should be integrated with or independent of browsers and operating systems and, as a related issue, how to support user interaction. Privacy concerns and legal, moral and ethical considerations are also questions that will be examined in the remainder of this thesis.

3 Method

The approach of this thesis is qualitative, a methodical stance focusing on the more intangible qualities of the research topic. The alternative would be a method focusing on quantification of research results by using for example extensive empirical studies and statistical methods. There are also other differences between qualitative and quantitative methods [Starrin 94]. In this context, two main issues regarding the method demand attention: precision of measurement and objectivity of the results.

3.1 Precision of measurement

Using a qualitative approach mostly means that results are not easily measurable. This is also true for this thesis, since the aim is to find the more or less abstract qualities of client-side proxies under different conditions.

It might be possible to measure some of the results with acceptable precision, for example by providing statistics regarding impact on network efficiency when using proxies, accuracy of filtering proxies, the number of users of different applications, etc. If it were the main goal to answer these or other quantifiable questions, a quantitative approach would be preferable.

However, the goal is to answer questions that are more general, such as when the examined approach is preferred over other solutions. Because of this, a quantitative approach would not suffice. Inevitably, using a qualitative approach means that the results will not be thoroughly validated or invalidated with empirical or statistical methods, but this is not uncommon for this kind of research.

3.2 Objectivity

Since the results are not easily measured, there must also be doubts regarding their objectivity. It is true that answers to questions about the relative qualities of a specific technique are rarely objective. The whole field of software design mostly depends on the notion of good practices, rather than on fixed truths and objective evidence. Only in low-level areas of research, such as determining the efficiency of specific algorithms for particular tasks, is it possible to obtain truly objective results.

Obviously, this work does not deal with this kind of low-level research, so there can be no claim that the presented results will be truly objective. Where some kind of conformity with the current "truths" is desirable, the evaluation will be coloured by the generally accepted good practices of software design. However, a large part of the work will be dependent on rather subjective interpretations of the design and performance of the examined systems.

3.3 Method in action

The main part of the work is an open-minded evaluation of existing client-side proxies aimed at finding the qualities of the technique. As mentioned above, this evaluation will be based partly on what is accepted as good software design but mainly on more subjective perceptions of existing solutions and hypotheses regarding the potential qualities of client-side proxy applications.

Through a comparison of proxy and non-proxy solutions, the goal of the first phase is to identify the circumstances when a proxy solution is better and which factors speak in favour of using them. Based on the issues raised in the background survey and the results of the first evaluation round, the second part of the evaluation will focus exclusively on existing client-side proxies. The aim is to investigate to what degree they realise the potential of the approach and to find ways to improve them.

Together, these two phases will give some possible answers to the introductory questions, some of which will be visualised in the Blueberry module. In total, this will provide results that admittedly are not final, but should be a useful starting-point for further evaluation and serve as a set of guidelines for those interested in the approach.

4 Task-oriented evaluation

One of the main objectives of this thesis is to examine under which circumstances and for whom proxy-based applications could provide better results than non-proxy solutions, and when these other approaches are preferable. The focus of this section is to gain a better understanding of this issue, through a comparison of applications that work as client-side proxies and applications that do not. This is a task-oriented evaluation in the sense that the systems evaluated in each subsection perform similar tasks using different approaches. It is not an exhaustive examination of existing systems, but the covered areas and applications provide insight into diverse task domains and implementation techniques. Discovering the characteristics common to all areas and those particular to some will help to resolve the issue at hand.

4.1 Protecting privacy

Applications providing privacy for Internet users comes in many flavours, secure transactions, encrypted e-mail, masquerading, etc, helping Internet users hide personal information from accessed servers. Without protection, there are many ways for a keen administrator to monitor individual users, through environment information from clients and servers, placing cookies and other techniques. So, what use is the client-side proxy approach to a user trying to protect this information from prying eyes? This section examines two approaches to anonymity, the proxy-based Freedom system [Freedom 00] and the Web service Anonymizer.com [Anonymizer 00].

Freedom protects user privacy by redirecting communication through a private network before releasing it on the Internet (figure 1). Each node (including the local client) in the private network adds a layer of encryption to the proxied data packet before passing it to another node in the network or out on the Internet. This means that no single operator has comprehensive knowledge about the user. The response is then sent back along the same path, shielding the identity of the user. This is similar to the approach used by the Crowds system, described in the background section of this work. The main difference is that Freedom uses a static set of dedicated servers instead of relaying requests through other users' local proxies. The Anonymizer service has a simpler approach. To be anonymous, a user logs in at the Anonymizer web-site and enters the requested URL in a text-field. The Anonymizer retrieves and processes the document on behalf of the user (figure 1), thus hiding the user's identity from the remote server.

4.1.1 Getting started

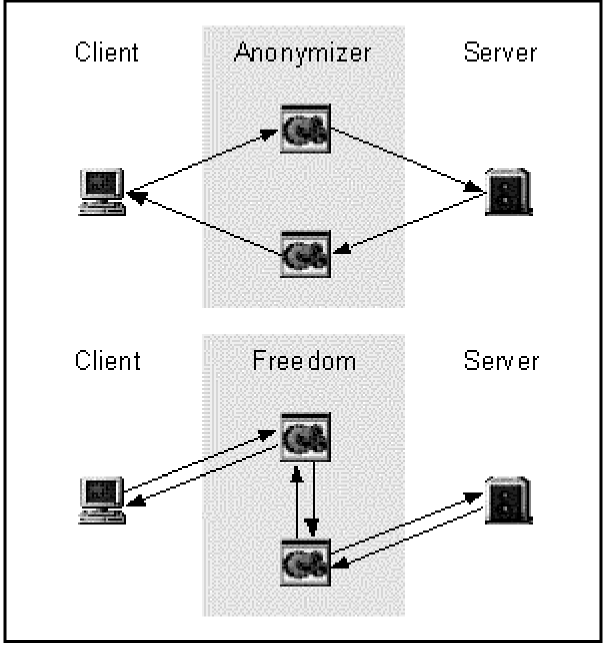
One of the strengths of web-based services, including the Anonymizer, is that there is no need for installation on the local machine, provided there is a working network connection and a compatible browser installed. Freedom requires installation, which is not trivial but reasonably easy, since the user is guided through the process. Freedom uses platform-specific network functionality, automatically enabling proxy functionality. Hence, there is no need for manual proxy configuration of client applications. Regardless of how easy Freedom is to install, it is harder than using the Anonymizer for the first time, since all that is required is to log in at the Anonymizer web-site. Clearly, all client applications must be installed on the local machine before use, and whether it is worth the trouble depends on the additional value of the client-side application.

Anonymizer, as most web-services, is relatively independent of both operating system and client applications, while Freedom integrates closely with the Windows platform. There are positive aspects of this platform integration, providing an easy installation process being one. Using platform-specific functionality to provide a truly transparent service is another. There is also the performance issue, since applications written for a specific platform mostly are faster and more efficient than platform-independent applications. An obvious drawback is that a user can not easily access the functionality from other machines than the one where the application was installed.

4.1.2 Making the user anonymous

The common function provided by both Anonymizer and Freedom is to conceal the identity (that is, the IP address) of a user retrieving Web content. In addition to this, Freedom provides anonymity for email, chatting, posting to newsgroups and telnet sessions. This is clearly a more sophisticated and exhaustive service. Indeed, it is common that client-side applications have more advanced functionality and are more configurable than web-based services. For example, Freedom lets the user decide how to treat cookies, setting preferred communication routes, control performance/privacy ratios, etc. In comparison, the Anonymizer is a blunt tool, providing no user-adaptable configuration and simply blocking cookies together with Java and JavaScript in web pages. Although these techniques are potential privacy threats, this is not a very good solution. Many web-sites depend on these techniques to function properly, and complete blocking prevents access to many sites that pose no threat to the user's privacy. Freedom does not address the issue of Java of JavaScript, leaving it up to the user to configure client applications for the preferred level of security.

Figure 1. General redirection schemes of Freedom and Anonymizer.



The Anonymizer provides an easy to use and relatively transparent service. Retrieved documents are altered so that when a user follows a link in the document the linked resource is automatically retrieved through the Anonymizer web-site. Accessing documents not linked from the retrieved page is not so transparent. As a remote service, the Anonymizer is unable to intercept document requests entered directly in the location-field of a browser. Instead, the user has to enter the request in a special text-field embedded in the processed document. This can be easy to forget, and if documents are requested directly through the client application, the user will no longer be anonymous.

The threshold for Freedom users is higher, partly because of the installation procedure, partly since it must be activated before each session. Activation takes noticeable time, but afterwards it works completely in the background unless the user wishes to change the configuration. When Freedom is up and running, the user can behave as usual since anything transmitted over the network is intercepted and anonymised by the Freedom proxy. Indeed, this is one of the main benefits of the proxy approach.

4.1.3 Increased response time

Both approaches has effects on the response time experienced by users, since they insert extra nodes on the path from client to server, nodes that might become communication bottlenecks. It seems that this is a smaller problem for Freedom users, since there are several dedicated Freedom servers distributed geographically. Because of this, it is possible to perform some optimisation of the chosen routes, initiated either by the user or by the client application. With the Anonymizer, all requests go through the same network. On the other hand, that processing performed by the Anonymizer is simpler than within the Freedom network might lessen the impact on response times.

Providing anonymity requires introducing an intermediary, resulting in a longer path between client and server. Users that want to be anonymous must pay this penalty of increased response times, regardless of the implementation of the service.

4.1.4 Security considerations

It seems clear that Freedom offers better overall protection of user privacy than the Anonymizer. One advantage is that Freedom as a client-side application has the ability to perform privacy enhancements before sending information over the network. Using the Anonymizer leaves the initial connection vulnerable and open to monitoring. It also means that the Anonymizer site has access to all the information the user wishes to hide, and the user must rely on the measures taken by the third-party site to ensure the privacy of its users. Using a client-side approach, whether it is implemented as a proxy or not, can make sure that personal information never leaves the local machine. There is still the risk of malevolent applications disclosing this information without user knowledge, but in the end, there is no such thing as total security. Apart from this, Freedom also protects communication using other protocols than HTTP, and since Freedom is more configurable, it offers levels of protection more adaptable to the needs of specific users.

4.1.5 The proxy advantage

In conclusion, both systems have their strengths. The major strengths of the Anonymizer Web service is that it is easy to use, requires no installation and is independent of both platform and client applications and thus available to all computers with Internet access. In contrast, the proxy approach of Freedom is non-portable and requires more work before use. However, for a determined user, installing Freedom is probably worth the trouble. Because it is a proxy, it intercepts and processes all communication before any information leaves the local machine, an important advantage if the task is to protect user privacy. As a client-side application, it provides more sophisticated functionality and ways to adapt the behaviour according to user preferences.

4.2 Collaborative rating

Finding relevant material on the World Wide Web can be a time-consuming task, and it can be difficult to establish the value of found documents. Collaborative rating is one way to ease the burden of individual users, providing a way to take advantage of the experiences of others. When users rate resources, they leave footprints for others to follow. To find and assess different resources, following these footprints and becoming aware of the opinions of others can be of help to the user. Two tools that facilitate collaborative rating, Alexa [Alexa 00] and SELECT [SELECT 00a], are examined in this section.

SELECT is a project funded by the European Union with the aim "to help Scientific, Technical and other professional Internet users to get and find the most reliable, valuable, important and interesting information" [SELECT 00b]. However, when speaking of SELECT in the remainder of this thesis, it refers to the client-side proxy server for collaborative rating being developed as part of the project. Alexa is a commercial navigation service, providing users with information about sites that they visit, including ratings of these sites by other users. The version examined here is implemented as a browser plug-in.

4.2.1 The price of independence

Written in Java, and therefore supposedly platform-independent, the SELECT proxy exhibits one of the possible drawbacks of platform-independent applications: difficult installation. It requires a Java installation on the client machine, manual editing of configuration files and manual proxy configuration of client applications. Some of this might be due to the prototype status of the project, but when compared to the installation process of Alexa it is a serious disadvantage. Installation of Alexa is extremely simple; the user simply follows a link on the Alexa web-site resulting in automatic download and installation. This simplicity is achieved through close integration with the latest version of the Internet Explorer browser.

Again, integration causes dependence to particular platforms and/or applications. Alexa supports different browsers with different application versions, focused on the Windows platform. There is also an older, more browser-independent version available, working more like a proxy. The version examined here is the one integrated with the latest version of Internet Explorer. Clearly, using different versions for different client applications is not an optimal solution. SELECT, as a proxy, has the potential to be browser-independent. However, this potential is not fully realised because of the use of Java applets and JavaScript, techniques supported inconsistently or not at all by different browsers.

4.2.2 The rating mechanism

The aim of Alexa is to provide useful information about accessed web-sites, user rating being only a part of the provided information. In SELECT, document rating is the central functionality. This difference in focus obviously has impact on the extent of functionality directly related to document rating. Where Alexa is limited to facilitate rating and display the average rate and number of votes, SELECT also lets users describe rated documents with keywords and provides a searchable database of these documents. In addition, the rating is more fine-grained since it applies to individual documents, while Alexa ratings apply to whole sites. A future goal of the SELECT proxy project is to provide different rating databases for different user categories. This would make the ratings even more fine-grained and information-rich.

As a browser plug-in, Alexa share the browser with the current document. Without leaving this environment, the user can rate the document by using a pull-down menu (figure 2). The average rating by other users is also in plain sight at all times. Close integration with the browser environment and a simple interface makes Alexa very easy to use.

Figure 2. Alexa user interface.

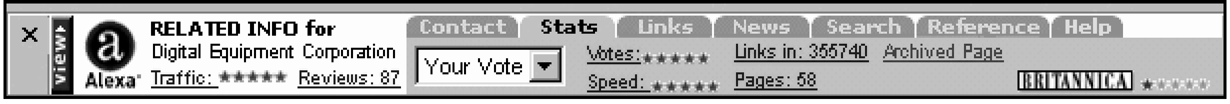
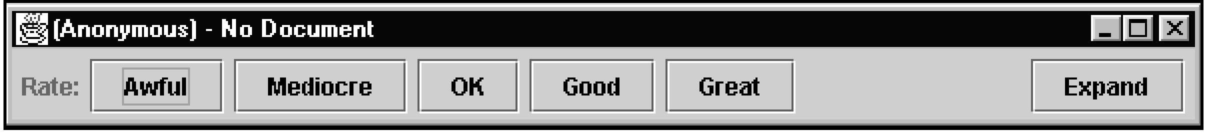


Figure 3. SELECT minimal rating interface.



The minimal rating interface of SELECT (figure 3) is also straightforward, but to log in and access additional functionality such as average rating and the searchable database, this window must be expanded. Even on fair-sized screens, the expanded window is big enough to be partially hidden behind the browser window. Since users are supposed to use SELECT and the browser in union, an independent application window is not as easy to use as a more integrated solution. This and the fact that the user interface of the SELECT prototype is both cruder and more complex speaks in favour of the plug-in approach of Alexa, at least when ease of use is an important issue.

Both Alexa and SELECT depend on the performance of remote servers, with the possible negative effects of communication bottlenecks and net congestion, but this is independent of the choice of implementation architecture.

4.2.3 The proxy disadvantage

Somewhat simplified, the minimal requirements of a system for collaborative rating is knowledge of the address of the current document and a connection with a rating-server. These requirements are fulfilled by the plug-in approach, and since it also is user-friendlier, it is preferable in this situation. A proxy approach could be more independent of platform and browser, but there is no additional functionality or greater usability to justify the additional overhead. A stand-alone proxy solution is both harder to install and operate than a more integrated solution. It is true that the SELECT proxy provides more functionality related to rating, such as database search, but this is mainly a result of the focus of the different approaches. There are simpler and user-friendlier methods to implement additional functionality. An example is provided in the SELECT project itself, letting users add a browser bookmark consisting of JavaScript code which opens a new browser window with access to a web-based rating and search interface.

To justify the use of a client-side proxy, it is crucial to provide some functionality that depends on processing the content stream. Annotation of hyperlinks depending on the rating of linked documents is one function that is discussed within the SELECT project. Other possible functions are automatic extraction of keywords describing a document and inserting the average rating into the rated document. If the SELECT proxy evolves in this direction, it might provide the functionality needed to justify the extra overhead.

4.3 Improving performance

The number of World Wide Web users has exploded since the birth of the medium and documents on the web have become more complex and graphic-intensive. Together, these factors have increased the overall load of the networks, and many users experience slow connections and disturbingly long response times. Caching is a possible method to improve the performance perceived by the user, another is to acquire faster connections. A third method is to remove unwanted material from requested pages, an approach examined here. AdWiper [WebWiper 00] and WebWasher [WebWasher 00] remove advertisements from Web pages. These ads can be quite large, especially if animated, and they are often retrieved from heavily trafficked ad-servers. Removing them accelerates page retrieval by significantly reducing the amount of data transmitted. Just as Alexa examined above, AdWiper is a plug-in for Internet Explorer, while WebWasher is a client-side proxy.

4.3.1 Installation and independence

As most contemporary platform-specific applications using installation guides, installation of both WebWasher and AdWiper is easy, done with a few button-clicks. Unlike AdWiper, but like many other proxies, WebWasher requires additional configuration. The user has to edit the proxy settings of client applications manually, but coming versions of WebWasher will automate this so that WebWasher alter browser settings at start-up and restore them at shutdown.

With the platform-independent techniques available today, providing this kind of simplicity is not easy. Both WebWasher and AdWiper are platform-dependent, available only for Microsoft Windows. AdWiper is also browser-dependent, since it works only with the Internet Explorer browser. As other proxies mentioned, WebWasher has the advantage of being browser-independent, at least if the browser supports connections through a proxy server.

4.3.2 Functionality and ease of use

The close integration of plug-ins and browser tend to make the inner workings of a plug-in transparent to the point of invisibility. It can be hard to know if the plug-in is functioning correctly, difficult or impossible to turn it on and off at user discretion and difficult to configure. This is partially true for AdWiper. To some extent, it is configurable and it is also possible to edit the rules that determine what constitutes an advertisement, but the configuration interface is an application separated from the plug-in.

WebWasher has a range of functions apart from blocking image and applet advertisements, such as filtering popup windows, stopping animated images, simple privacy enhancements and access control. WebWasher also has a user interface separated from the browser, but it is accessible via an icon in the Windows task-bar, allowing easy configuration and one-click disabling/enabling.

There is no direct relationship between implementation architecture and the usability differences of AdWiper and WebWasher. Both are easy to use, since they do not interfere with the user's real task. WebWasher has the advantage of extensive and easy configuration and more functionality, but this would also be possible to implement in a plug-in. However, making a plug-in as browser-independent as WebWasher is not feasible.

Introducing additional content processing inevitably raises the issue of performance, but in this situation the decrease in network transfers quickly compensates for the additional overhead introduced by the applications.

4.3.3 Free lunch?

Removing ads from web pages might improve performance, but it is a potentially controversial issue. In the words of Milton Friedman: "There's no such thing as a free lunch." Many sites that provide useful information and/or services depend on advertising revenues to supply a free service. If a large number of users decide to block out these commercial messages, revenues will probably fall and since somebody must finance a commercial service, users themselves might have to pay for what they want to access.

As an alternative, users could decide not to remove advertisements completely but stop only the animation of images. The overhead involved in connecting to remote ad servers still remains, but the size of the download will be smaller. WebWasher allows this, as a user can choose to break animations without completely removing advertisements. This could be a way to improve performance without jeopardising the free availability of online services.

4.3.4 The proxy advantage

Client independence is one of the strongest arguments in favour of the proxy solution. This allows WebWasher to deliver the same functionality regardless of which client application the user prefers, while AdWiper is limited to the Internet Explorer browser. That WebWasher provides functionality that is more extensive could also be construed as a benefit of the proxy approach. A client-side proxy has access to the full functionality of underlying system services, and even if some browsers give the same freedom to their plug-ins, others do not.

4.4 Filtering news

There are tens of thousands of Usenet newsgroups, containing staggering amounts of posted messages. Sifting through this to find what is relevant and interesting is a gargantuan task. Applications that filter groups and messages to remove spam and extensive cross-postings and highlight messages that might be of interest could be of great value to the user.

This section explores the Agent news and mail reader [Agent 00] and the filtering proxy NewsProxy [NewsProxy 99]. Agent is a full-featured news and mail application with extensive filtering functionality, while NewsProxy is a client-side proxy server, working as a supplemental filtering program to existing newsreaders. The focus here is on the filtering functionality of the respective systems.

4.4.1 Potential platform independence

Both Agent and the NewsProxy binary release are specific to the Windows platform, providing the simple installation procedure exhibited by most platform-specific systems examined so far. NewsProxy requires some additional configuration, but nothing more advanced than configuring a stand-alone newsreader. NewsProxy also comes in a source-code release, allowing the inclined user to compile the application on other platforms than Windows. However, this kind of porting is not a trivial undertaking, and most users are restricted to the binary releases available. On the bright side, open source projects tend to attract third-party developers, thus increasing the possibility that the application will become available on different platforms. As all true proxy servers, NewsProxy looks like a remote server to the client application, relying only on standardised network protocols. Ergo, any client application talking the same language as the proxy can benefit from its functionality. The use of standardised protocols could also mean that these parts of the application are easier to port to other platforms, thus providing at least some platform-independence.

4.4.2 The complexity of filter creation

Because of what it is, Agent provides much more functionality than NewsProxy. Focusing only on filtering functionality, the two are more similar. Before displaying the message headers, both apply filters that can delete or watch and mark these messages. Filters can be simple text matching of various header fields, or built with more powerful (and less intuitive) regular expressions. These are originally Unix-based expressions for parsing and matching strings of text. However, how the filters are constructed differ between the two approaches. As most integrated applications, Agent is dialog-driven (figure 4), providing users with a well-known configuration method, with the additional possibility of using message-specific information as template for new filters.

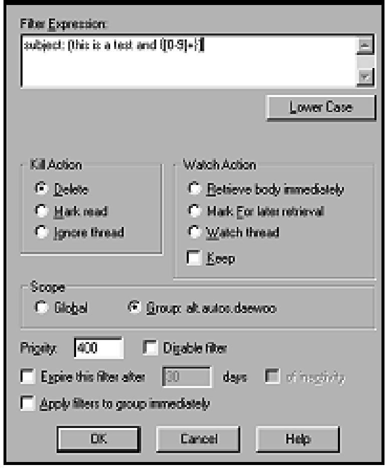
In NewsProxy, the user has to edit the configuration file manually to create and maintain filters (figure 5). While this is harder for the novice user, it does give a better understanding and overview of the filtering language and enables simple cut-and-paste changes to the filter rules and the order in which they are applied. The human readable rules are also easy to import and export, just copy a rule from a newsgroup message, an email or a Web page and insert it into the configuration file.

Figure 5. Excerpt from NewsProxy configuration file.

```
rec.*              drop        from:*i_am_a_flamer@domain*
alt.autos.daewoo   drop        subject:*this is a test*  subject:[0-9]* |
```

Manual creation of expressive filters is never trivial, and user interfaces for this kind of task can easily become complex and hard to use. This is especially true if there also is functionality unrelated to the task at hand. In Agent, the full spectrum of news and mail functionality clutters the user interface, and a user interested in the filter functionality must navigate through many menus filled with "irrelevant" options. In contrast, the only task of NewsProxy is filtering news, and the user interface focuses exclusively on this functionality. This results in cleaner and more navigable menus. The simplicity of the NewsProxy interface makes it easier to access the filtering functions than it is in Agent, a clear benefit of strongly focused applications.

Figure 4. Filter configuration dialog in Agent.



Filter Expression: subject: {this is a test and {[0-9]+}}

Kill Action: (•) Delete ( ) Mark read ( ) Ignore thread

Watch Action: ( ) Retrieve body immediately ( ) Mark For later retrieval ( ) Watch thread ☐ Keep

Scope: ( ) Global (•) Group: alt.autos.daewoo

Priority: 400 ☐ Disable filter

☐ Expire this filter after 30 days ☐ of inactivity

☐ Apply filters to group immediately

Agent obviously has the advantage of an integrated environment for all functionality, while an evaluation of the overall usability of NewsProxy must take into account both NewsProxy itself and the newsreader used. One benefit of an integrated environment is that it is sensitive to the context in which the user is working. As mentioned, Agent allows a user to create filters based on specific messages. In this way, Agent is more flexible and adaptable to the individual user, but the proxy solution also has its flexibility gains. One is that the functionality is portable between different client applications, meaning that a user can apply the same filter configuration without reinventing the filters if he decides to use another newsreader.

4.4.3 Disarm security threats by filtering?

Although there are security threats related to reading news, mainly the risk of catching viruses through message attachments, these are not as widely discussed as similar threats concerning email and malevolent Web pages. Even if this is not the aim of the examined applications, filtering could be one way to minimise risks. Spam messages are often distribution channels for viruses, links to malevolent sites and/or applications and other potential security threats, and removing these could improve the overall security of the user.

4.4.4 The proxy (dis)advantage

Both Agent and NewsProxy filter messages by analysing incoming message headers. The main difference is that Agent integrates filtering with other news-related functionality, while NewsProxy access and alter the incoming content stream before it reaches the client application. It seems clear that more sophisticated functionality is not an automatic advantage of the proxy approach, since the filtering capabilities of Agent and NewsProxy are roughly equal. The primary advantage of using a proxy is that it can enhance the functionality of client applications that lack the extensive filtering capabilities exhibited by Agent.

An integrated environment such as Agent might be beneficial, for example by using contextual information to simplify the user's task. Building filters based on specific messages is one operation, studying the behaviour of a user to automatically create filter rules could be another. It is difficult to accomplish this using the proxy approach, since proxies have less detailed knowledge about the interaction. On the other hand, if functionality is split into several layers, each layer becomes more focused and perhaps more easily understood. It also promotes easier update of different layers of functionality and more freedom for users to choose their client applications. Like many other proxies, NewsProxy visualises the benefits of a layered solution.

4.5 Blocking content

Content blocking has many similarities with the filtering task described in the previous section, especially with "kill filters" that remove a resource before the user sees it. The main distinction is that filters remove unwanted material, while blocking applications prevent access to wanted material. That is, someone with authority decides what is appropriate and not, denying other users access to inappropriate material. This could be a parental authority, keeping children away from pornographic or other unsuitable material, or it could be a corporate administrator making sure employees only access work-related resources.

Two client-side proxies for content blocking are examined further, SurfWatch [SurfWatch 00] and PureSight [PureSight 00]. NetNanny [NetNanny 00] represents an alternative approach, a standalone application monitoring the client applications themselves rather than the content stream.

4.5.1 Setting up

As the majority of the client-side applications examined so far, NetNanny, SurfWatch and PureSight are available only for Microsoft Windows, making extensive use of platform-specific functionality. Although SurfWatch and PureSight work as proxies, they do this by low-level integration with the operating system. The good side is that manual proxy configuration is unnecessary. The bad side is that platform-specific functionality makes the applications even more platform-dependent and lessens the possibility of availability on other platforms. NetNanny is equally non-portable, using Windows-specific mechanisms for inter-process communication. In addition, the administrator of NetNanny must manually decide which applications to monitor for attempts to access unauthorised material. PureSight is ready for use immediately after installation, while the other requires download of additional resources, adding a considerable amount of time to the installation process.

4.5.2 Running the applications

A common function of all three applications is blocking of alleged pornographic content. SurfWatch and NetNanny also has the ability to block other questionable material, such as gambling, racist web-sites, bad language, etc. They depend on extensive lists of sites containing this kind of material, lists of keywords and phrases to block and lists of unauthorised Usenet newsgroups. PureSight only depends on the requested material, blocking resources based on content analysis.

Due to the technologies used in PureSight, this application is more prone to make errors in judgement, blocking sites that does not actually contain pornographic material. NetNanny and SurfWatch also use technology to find explicit material, but human reviewers verify the results before updating site and keyword lists. This should mean that these lists are more accurate, although there is always the risk of human error and misjudgement. On the downside, manual updating inevitably means that new or unknown sites slip through, even if blocking would be justified. The client-side administrator must also update blocking lists frequently for the applications to function properly. In SurfWatch, down-loading and installing new lists is easy and mostly automatic, taking place entirely within the application environment. However, it can be a time-consuming task. NetNanny requires the administrator to manually download updates using a browser, remove existing lists, import new lists and then manually configure each of the imported lists. To say the least, this is unnecessarily difficult. In contrast, it is pure joy to use PureSight, since it does not require any such updates.

All three let the administrator specify additional sites/addresses that users are allowed or disallowed to access. SurfWatch and NetNanny also allow editing of keywords and phrases that are accepted or unaccepted in requested material. While PureSight focuses primarily on pornography, the others are more flexible in that they can block arbitrary categories of unwanted material, depending on the wishes of the administrator. SurfWatch also has the option to block all content except what is explicitly allowed.

Although configuration and maintenance can be a hassle, this is supposedly the responsibility of an administrator. In the eyes of the end-user, the applications run in the background without need of user involvement, and business can go on as usual without the user worrying about the workings of the blocking applications.

4.5.3 Performance and security

Both NetNanny and SurfWatch work with site lists as a basis for blocking, so there is a similar impact on performance. Matching is quite simple, not causing any noticeable communication delay. PureSight analyses each accessed page to determine the type of content, resulting in a delay, more or less noticeable, before displaying or blocking the page. Introducing demanding processing in the content stream usually has this effect, an issue that must be dealt with when client-side proxies are involved.

There could be indirect security gains by using blocking applications, as was the case with the news filtering applications. Blocking access to and downloads from distrusted sources should prevent hostile attacks, at least from these specific sources. In addition, NetNanny can monitor and protect personal information such as addresses, credit card numbers, social insurance numbers, etc. Since NetNanny is not a proxy, this clearly is not an automatic proxy advantage. However, a proxy can also be useful in these matters, as we have seen.

4.5.4 The proxy advantage

Like Freedom, the anonymising proxy described earlier, one of the major advantages of the proxy approach in this context is that it is not necessarily HTTP-centric. Although the Web is the most accessible part of the Internet, material that someone wants to block might as well reside on for example ftp servers or in Usenet messages. A proxy can provide transparent blocking of these as well as web-based material, and both PureSight and SurfWatch do this. It is true that NetNanny also can monitor different kinds of communication, but it requires configuration for each application it should monitor. This is a difficulty not directly related to the issue at hand, but the proxy approach provides smoother coverage of different applications and protocols.

In its simplest form, content blocking is matching of the address of a requested page with entries in a database of questionable sites. If this was all, a proxy approach might not be justifiable. However, all applications examined here also make a closer examination of the requested material in search of trigger keywords or other indications of unauthorised content. Not relying on site lists, PureSight works closest to the content stream, examining the content of requested pages to determine whether they contain unaccepted material or not. In contrast, the approach of NetNanny is quite peculiar, since it requires the user to decide which applications to monitor. This is a roundabout way to achieve the task; a task closely tied to the actual content stream.

This concludes the task-oriented evaluation. What has been found regarding the merits of the client-side proxy approach and when it should and should not be used will be examined from other angles in the following sections.

5 Existing client-side proxies

While the focus of the last section was when and why someone should use client-side proxies, this section focuses on how the approach is actually used. Specifically, the questions examined here concerns the implementation of available proxies and, if they do not fully realise the potential of the approach, how they might have been implemented. As a rough classification, these questions deal with the external and the internal aspects of client-side proxies. External is what is visible to the end-user, mainly the user interface. The specifics of the internal aspects, including questions about application architecture and performance, are mostly of interest to advanced users, administrators, developers, etc, but also of some interest to the end-user, since they affect the perception of usability and efficiency.

In addition to those examined in the previous section, six new proxies enter the field for closer examination. A4Proxy is a Windows-specific anonymising proxy, functionally similar to the Crowds proxy described in section 2 [A4Proxy 00]. Another acquaintance from section 2 is the WebMate proxy providing browse and search assistance [WebMate 99], again subject to scrutiny. ByProxy [ByProxy 98] and Muffin [Muffin 00] are extensible client-side proxies. Although they provide predefined modules for different processing tasks, the main feature is that they allow third-party developers to extend the functionality of the proxies by implementing modules of their own. Muffin is limited to processing HTTP streams, while ByProxy also has support for the mail (SMTP) and news (NNTP) protocols. WebMate, ByProxy and Muffin are all written in Java and supposedly platform-independent. While not entirely platform-independent, the privacy-enhancing Junkbuster proxy at least has open source-code [Junkbuster 99]. It blocks unwanted URLs, deletes unauthorised cookies and removes HTTP headers that might identify the user. Finally, Proxomitron is a client-side filtering proxy targeted at HTML text and HTTP headers, with both pre-configured filters and support for creating additional filters [Proxomitron 00]. Like A4Proxy, it is only available for the Windows platform.

This examination will not consider every aspect of every application. Of main interest are the examples that stand out, for good or bad, and these will be emphasised in the following subsections.

5.1 User interaction

For many years, the desktop metaphor has been predominant in computer-user interaction. With the advent of the Internet, and especially the World Wide Web, user interaction has partially changed shape. Today, hypertext documents viewed with Web browsers is a familiar and well understood way of user interaction and many applications and online services embrace this method to provide advanced functionality. While interaction is dependent on the relatively limited expressiveness of the hypertext mark-up language, it can provide a simple and consistent interface to different types of services. In client-side proxies it is possible to facilitate user interaction in several ways, since the proxy is neither a pure stand-alone application nor a online service, but rather a hybrid of the two. This section explores different models of interaction, their impact on platform/client independence and the overall quality of user interfaces.

5.1.1 Interaction models

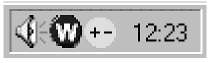
Apart from giving access to functionality and configuration, one of the responsibilities of the user interface is to communicate the state of the application to the user. In general, this means that the user interface (or parts thereof) should be visible near the client application and the processed content. Despite this, we have seen that most of the client-side proxies examined rely on application windows separate from the client applications for user interaction. In a way, this is natural, since it visualises the separation of proxy and client functionality. From another viewpoint, it is not so natural. Although there is a clear technological boundary between proxy and client, this boundary might not seem so obvious to the end-user. Rather, there is often a close semantic relationship between the processing of the content stream performed by the proxy and presentation in the client application. Possible ways to visualise this relationship is to integrate the user interface with the client application or embed it in the requested document. One obvious requirement is that the content protocol or client application supports this.

There are exceptions to the principle of visibility, for example concerning content blocking (Pure-

Sight, SurfWatch) and other prohibiting or monitoring applications. In applications like these, designed for almost complete transparency, the end-users have no need (or business) to access the inner chambers of the application. Rather, the users should be more or less unaware of the fact that the applications are performing their duties, only revealing themselves when the user tries to do something unauthorised.

On the next step on the visibility ladder, we find applications like Freedom and WebWasher. They work actively with the content stream but without requiring incessant monitoring and without producing any additional information apart from the processed content. It might be enough for these kinds of applications to indicate that they are functioning properly. The Windows-specific applications show this via icons in the Windows system tray (figure 6). Granted, this is a highly platform-specific feature, but a user-friendly one since a simple mouse click can give access to the full user interface.

Figure 6. WebWasher in the Windows system tray.



By necessity, this level of visibility must also suffice for another class of proxies, for example ByProxy, A4Proxy and NewsProxy. ByProxy and A4Proxy work with multiple types of content and protocols, a diversity that makes it nearly impossible to use any other model of interaction than separate application windows or client dependant integration. The same applies to the single protocol proxy NewsProxy, since the news protocol do not support any reasonable way to incorporate the user interface in the content stream.

For other proxies, the user interface should be close to the workspace of the user, providing immediate feedback and configuration. However, since most examined applications use their own windows for interaction, this is not the case. SELECT, Proxomitron and Muffin are examples that use application windows although the content they work with makes it possible to use and adapt the actual content stream for user interaction purposes. The content involved is mainly HTML documents, where the protocol and the content language allow applications to interact with the user through the content stream. Focusing even more on hypertext documents, which predominantly means Web pages, using the content stream for user interaction provides three major alternatives to do this with the aid of Web browsers. Integration might be supported by presenting the user interface in a separate browser window, a separate frame or directly in the requested document. Using a separate browser window has drawbacks similar to those of separate application windows - they might not catch the attention of the user or some other application might cover them. A covered window is mainly a problem for novice users, since they are not always aware that the windows represent a three-dimensional space. As opposed to integrated interfaces, with separate proxy windows it is possible for a user to arrange the desktop freely. Dedicated windows also ensure that requested documents are unaltered, as long as the functionality of the proxy does not involve content adaptation. However, separate windows do not facilitate a close union of user interface and content. Of the possible ways to integration, the only one utilised by any of the proxies in this examination is embedding the interface in the requested document. WebMate inserts a controller applet (figure 7) at the bottom of each requested page, allowing the user to access the user interface in a separate applet window (figure 8). There is also a stand-alone application window for the administration of basic proxy functionality, but all functionality directed to the user is accessible via the controller applet.

Figure 7. WebMate controller applet.

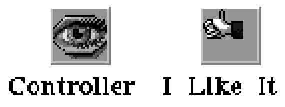
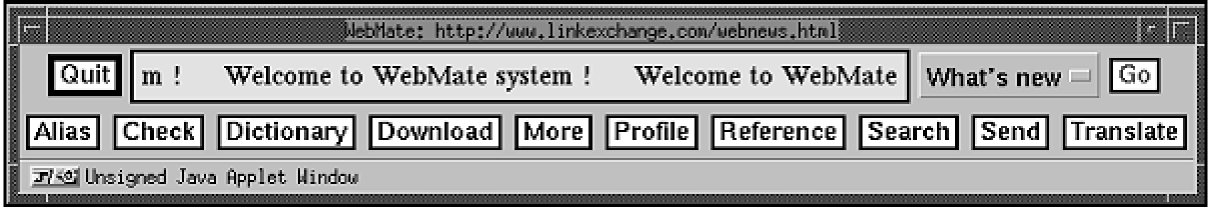


Figure 8. WebMate main interface.


WebMate: http://www.linkexchange.com/webnews.html
Quit
m ! Welcome to WebMate system ! Welcome to WebMate
What's new
Go
Alias
Check
Dictionary
Download
More
Profile
Reference
Search
Send
Translate
Unsigned Java Applet Window

It could be possible to embed the interface in other types of content. A user might for example interact with an underlying proxy through e-mail messages. In situations where it is important that the user is not interrupted and where interaction can be asynchronous, this could make sense. However, direct interaction is preferred in most cases, and "normal" user interfaces provide a way of interaction that is undoubtedly more intuitive.

Muffin and ByProxy also use their own windows but since they are extensible, it is possible for third-party developers to provide closer integration of user interface and processed content, at least for some modules. This is especially true for Muffin, being an HTTP-only proxy. ByProxy works as a proxy for multiple protocols, making it harder to provide interfaces integrated through the content stream unless the extension module focuses solely on HTTP processing.

5.1.2 Integration, separation and independence

The choice of interaction model also has implications on the client-independence of the application. Proxies that integrate their user interface with the client application are mostly more dependent on the capabilities of specific clients than those that use their own application windows. WebMate and SELECT are two examples of this, since they use Java applets and/or JavaScript embedded in the requested Web document as part of their user interface. Although the most popular browsers available today support these technologies, other browsers do not. A high degree of both integration and client independence requires the user interface to be described in the basic language supported by the client application. Fulfilling this demand is most feasible in the context of the Web and Web browsers, but inconsistent support for various HTML features can still make the user interface unsuitable for some clients. To achieve complete client independence, a clear separation of the proxy user interface and the client application is necessary.

As we have seen, separation is the approach used by the majority of existing client-side proxies examined in this work, while none uses pure HTML interfaces. Junkbuster is a proxy whose only attempt at a user interface is pure HTML, but this is merely a summary of version information and some variables that has been set during initialisation. To configure Junkbuster, the user must edit the configuration files manually.

The choice of interaction model also has implications on the dependence or independence of specific operating system platforms. An interface using native graphical functionality and components is not platform-independent. An application limiting itself to interface components readily available in the graphical toolkits of diverse platforms might be more portable and less platform-dependent. For example, applications using the Windows-specific system tray are probably more platform-dependant than those that do not. Interfaces implemented in languages like Java or HTML is certainly more independent, but they are still limited to platforms that support the chosen implementation technique. In reality, all major platforms have the ability to display HTML and, perhaps to a lesser extent, Java interfaces. However, a decision not to use platform-specific components and functionality can have other effects on the overall quality and usability of the interface.

5.1.3 User interface quality

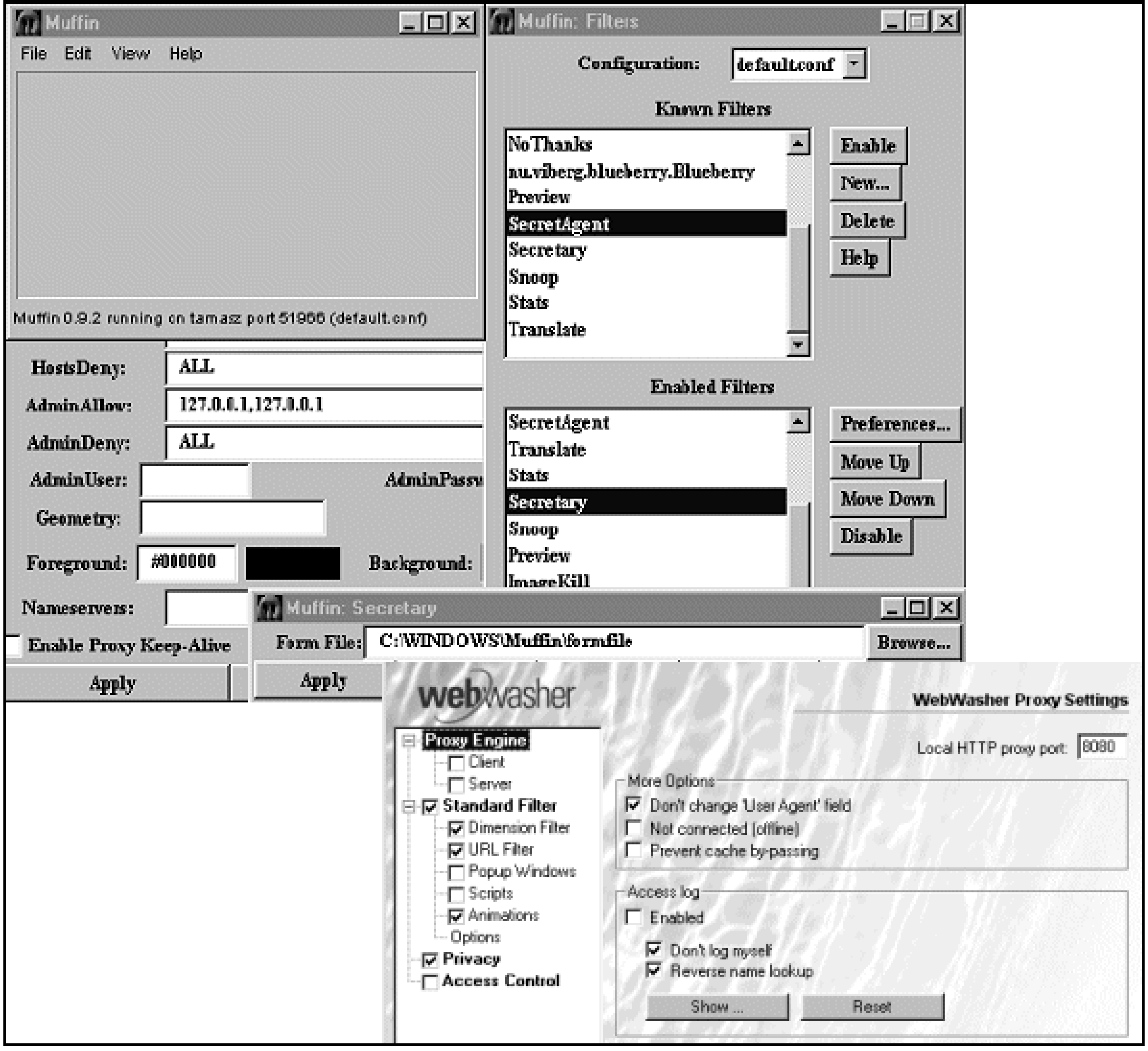
The quality of the interface has obvious impact on the usability and perceived complexity of user interaction. So what is a good interface? One determinant of a good interface is to what extent it fulfils the expectations of the user. If an interface complies with the look-and-feel of the underlying operating system, most users will consider it good enough since they are accustomed to similar environments. The standard interface components provided by operating systems more or less force platform-dependent interfaces into the appearance mainstream, thus introducing an element of standardisation. While this does not guarantee a high quality interface, at least it guarantees that users will not be completely confused. Let us consider some aspects of this, borrowed from Microsoft's user interface guidelines for Windows applications [Microsoft 00].

The first assumption is that the best interface is no interface. Instead of relying on interaction, the application just works, which in the end is what the user wants. A good example is the pornography blocker PureSight. Relying on computation rather than interaction, there is generally no need for the user to interact with the application. There is also the no-interface paradigm used by many UNIX-style applications, basing interaction on command-line arguments passed to the program at start-up, and by manual editing of configuration files. The Junkbuster proxy illustrates the approach. For a user that is familiar with this milieu, it can be a usable interface, perhaps a parallel to keyboard short-cuts in graphical environments. However, for users accustomed to the graphical interfaces, these console applications can be very frustrating to use. They give virtually no visual aid regarding the functionality of the application.

If there has to be a user interface, strongly focused applications are normally easier to manage and configure, as has already been stated in the previous examination of the news filtering proxy NewsProxy. This is also true for other proxies with tasks limited to a specific area, such as WebWasher and SurfWatch. A strong focus is imperative to create a simple interface, concentrating on essential functionality and promoting fast initial learning. To different degrees, Proxomitron, WebWasher, SurfWatch and PureSight all live up to this, having focused tasks and providing familiar environments with default configurations that allow a user to start use the application quickly and worry about the details later. Like WebWasher, A4Proxy provides access to all functionality in a single window, with a tabbed dialog to navigate through different configuration windows. However, the contents and presentation of the different windows are diverse, lending a certain degree of complexity to the interface.

The extensible proxies, Muffin and ByProxy, are generally harder to configure. The main reason for this is that, apart from configuration of the base application itself, it also requires installation and configuration of different third-party extension modules. Relating to many, possibly very

Figure 9. Interface samples from Muffin and WebWasher.



Muffin 0.9.2 running on tamasz port 51966 (default.conf)

HostsDeny: ALL
AdminAllow: 127.0.0.1,127.0.0.1
AdminDeny: ALL
AdminUser:
AdminPassw
Geometry:
Foreground: #000000
Background:
Nameservers:
Enable Proxy Keep-Alive

Configuration: default.conf

Known Filters: NoThanks, nu.viberg.blueberry.Blueberry, Preview, SecretAgent, Secretary, Snoop, Stats, Translate

Enabled Filters: SecretAgent, Translate, Stats, Secretary, Snoop, Preview, ImageKill

Form File: C:\WINDOWS\Muffin\formfile

- Proxy Engine
  - ☐ Client
  - ☐ Server
- ☑ Standard Filter
  - ☑ Dimension Filter
  - ☑ URL Filter
  - ☐ Popup Windows
  - ☐ Scripts
  - ☑ Animations
  - Options
- ☑ Privacy
- ☐ Access Control

Local HTTP proxy port: 8080

More Options
- ☑ Don't change 'User Agent' field
- ☐ Not connected (offline)
- ☐ Prevent cache by-passing

Access log
- ☐ Enabled
  - ☑ Don't log myself
  - ☑ Reverse name lookup

different tasks, it is difficult to maintain a consistent configuration view, thus increasing the complexity of the process. For example, the Muffin interface is extensive enough, but not as consistent and easily understood as the WebWasher interface (figure 9). The total amount of configuration needed might not differ that much between focused and extensible proxies, at least not if the user wants to create aggregate behaviour with multiple single-task proxies. In such situations, the overhead of configuring several different applications adds to the complexity.

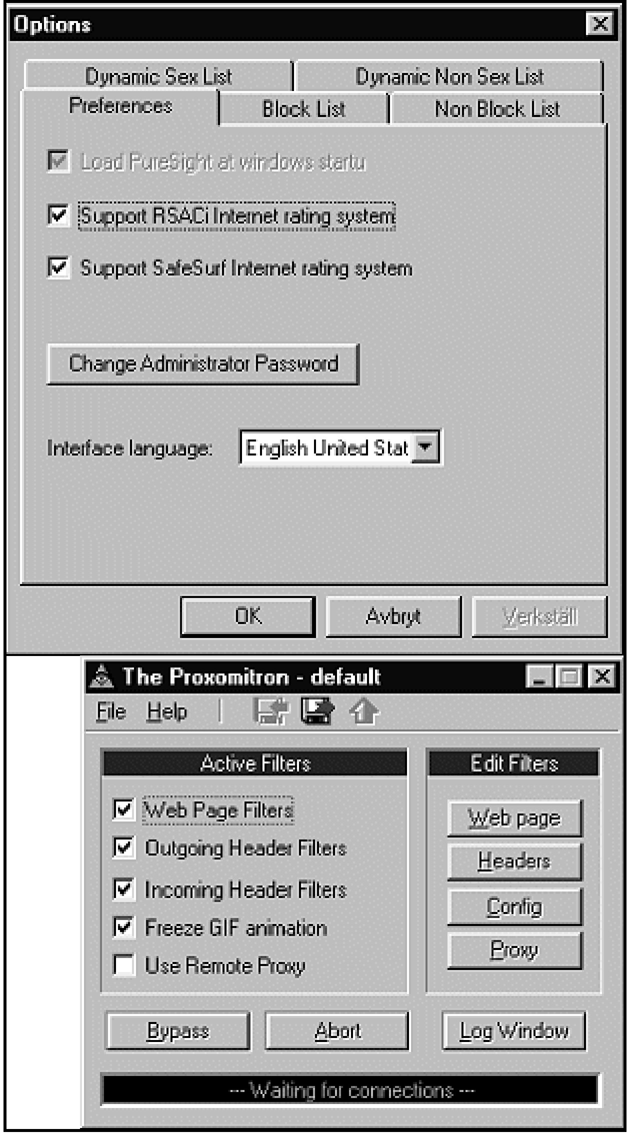
No matter the task, it is important that the user is in control of what happens. A good example is the Proxomitron proxy, where the user has easy access to functionality and can control what the proxy does to the content stream. Filter rules are accessible in table-like lists, with checkboxes to enable or disable them. Clicking on a rule brings up a dialog where the user can edit the behaviour of the specified filter. On the other side of the control spectrum is Junkbuster, allowing no run-time interaction whatsoever. Command-line arguments, raw text configuration and restarting to apply changes do not give most users a sense of control.

PureSight, Proxomitron, WebWasher and the other operating system-specific proxies use platform-native interfaces, which generally are faster and more responsive than platform-independent solutions. Java interfaces suffer from the overhead of the virtual machine, and the connection between proxy and client needed for HTML interfaces introduce communication overhead. The Java-based proxies show this clearly, since the interfaces of SELECT, WebMate, Muffin and ByProxy are all slower and less responsive than their platform-native counterparts.

Another thing that determines the perceived quality of an interface is the way it handles different modes. An application is in a special mode when it displays for example a dialog window that demands user attention before it is possible to continue normal operation. The ideal solution is a modeless interface that never interrupts the user. WebWasher exhibits such an interface, while the common case is that interfaces occasionally force a switch of mode. If this is necessary, the mode should at least be obvious and visible, such as file dialogs. A bad example of modal behaviour is the Muffin proxy. During configuration, numerous different windows might be opened, causing confusion as to what mode the application is currently in and what the results of an action will be.

For a user to feel in control of application behaviour, the interface must also provide directness. A user should be able to manipulate information directly within the application, and the interface should give access to all of the application's functionality and configuration options. This is normal behaviour for user interfaces, since their purpose is to be the link between user and application. Accordingly, a majority of the examined proxies provide access to the full spectrum of relevant information directly through their interface. However, some proxies store important information in configuration files separate

Figure 10. Consistent within the operating environment - PureSight and Proxomitron.



- [✓] Load PureSight at windows startu
- [✓] Support RSACi Internet rating system
- [✓] Support SafeSurf Internet rating system

Interface language: English United Stat

- [✓] Web Page Filters
- [✓] Outgoing Header Filters
- [✓] Incoming Header Filters
- [✓] Freeze GIF animation
- [ ] Use Remote Proxy

from the application and the only way to access the information is through manual editing. Most notably, this applies to Junkbuster, having no interface, and NewsProxy, where the interface does not facilitate filter configuration. Available and visible information and presentation of possible choices reduce the reliance on a user's ability to recall the right actions. It is easier to recognise the appropriate actions, and directness in an interface thus alleviates the mental burden of the user.

That it is easier to recognise something than to recall it from memory leads to the next ingredient of a good user interface, consistency. There are two levels of this, consistency within the application and consistency within the operating environment. If an application is consistent with the general look-and-feel of the surrounding operating system, users already accustomed to this environment can transfer their existing knowledge to new software. A familiar and predictable interface facilitates quicker learning, which enables the user to focus more on the task at hand. Platform-specific proxies generally look and behave like other applications on the same platform (figure 10).

Freedom, SurfWatch, A4Proxy, Proxomitron, PureSight and other applications that are consistent within the operating environment have a lower learning threshold than for example Java-based applications. Since the Web also has become a familiar environment for many users, hypertext interfaces have a similar advantage. Although the interface does not look like the surrounding operating system, it looks like other Web documents. Users that understand the design of the Web will consider the interface consistent within its environment. In contrast, consistency is not a common characteristic of platform-independent Java applications. The applet interface of WebMate and the stand-alone Java interfaces of SELECT, Muffin and ByProxy lack the common design style that is one of the strengths of platform-specific applications. Although standardisation is not the only path to usable interfaces, it is de facto very important.

Following general design guidelines, environment-consistent applications are in general also consistent within themselves. This level of consistency requires that command names, presentation style, behaviour of operations, placement of elements, etc remain the same throughout the interface. An example of inconsistent behaviour is the rating buttons of the SELECT proxy. In the minimal rating interface, the buttons are located at the top of the window, which is inevitable since the window contains only these buttons and a button to expand the interface. When a user expands the interface, the rating buttons suddenly are close to the bottom of the window, creating an unnecessary inconsistency in the interface.

Users also expect some kind of response on the actions they perform, and application developers should make the effort to provide noticeable feedback on user actions. Again, the normal behaviour of the examined proxies is to provide feedback, communicating application status through messages, animations, etc. As usual, there are also exceptions. Editing filter rules in NewsProxy does not result in immediate response, since editing is separate from the application. To detect syntactical errors in the edited rules, the user has to restart the application. Another "feature" of systems with lacking feedback is frozen screens. The SELECT proxy demonstrates this. Whenever network communication takes place, the interface dies and is not resurrected until (and if) the communication is finished. Another annoying detail is that when a user switches between the minimal and the complete interface, the interface completely disappears for quite a while before it appears again.

The major determinant of a good interface is simplicity, providing smooth access to the complete functionality of an application. Extensive functionality might work against simplicity, and interfaces that maintains a strong focus and reduce the available information to the base requirements are generally simpler and more usable then more complex interfaces. For a proxy, the base requirements might be no interface at all, since proxies mostly run in the background. Depending on the task and the additional information produced, the interface design is more or less important to the proxy user. Nevertheless, even proxies providing completely transparent run-time services require installation and some configuration. A well-designed interface is always better than a poorly designed, even if it is seldom used.

5.2 Application architecture

Just like with people, interior qualities are harder to evaluate than exterior. It requires an in-depth examination of what happens inside to get a thorough understanding. Gaining such an understanding of computer software internals requires study of application source-code or detailed system documentation. This poses a problem, since sources or documentation might not be readily available, especially for commercial systems. The extent and complexity of source-code also makes the task time-consuming beyond the limits set by the scope of this work. With this method disqualified, a black-box approach must suffice, looking at the outer signs to draw conclusions about the architectural issues regarding client-side proxies.

5.2.1 Monolithic or modular

One architectural issue is whether the application is modular or monolithic. Somewhat simplified, a monolithic application consists of one large application file, while a modular application is split into different modules with specialised functionality. Modular applications create links to external modules dynamically, while running. Applications built with statically linked modules at compilation time are not modular. In the context of this section, a modular application is one that uses dynamic linking. Among other things, the choice between static and dynamic linking has impact on application efficiency and ease of updating.

Of the examined client-side proxies, Junkbuster seems to be the only monolithic application, although built from modular source-code. Other applications might seem monolithic at a glance, but they probably use dynamic linking of platform-specific libraries, for example to gain access to graphics and network functionality. It is difficult to be certain of this using a black-box approach, but it is standard behaviour for modern platform-dependent applications. What is certain is that the Java applications SELECT, WebMate, Muffin and ByProxy are modular. All linking is dynamic in Java.

A modular approach could facilitate run-time loading and unloading of functional modules. By loading only basic functionality at start-up, application initialisation might be considerably faster. Loading additional functionality only when demanded could lessen the application's overall use of memory and processing power. However, the overhead introduced by dynamic loading might have negative effects on performance, and monolithic applications are generally faster. For Java applications, with both completely dynamic linking and dependence on the virtual machine, performance is often a problem. On the good side, a modular, dynamically linked application might be easier to update, since it does not need complete reconstruction after every update. This is particularly apparent in Java environments. Simply replace a class file containing a certain module with an updated version, restart the application, and the changes take effect. Easy updates of individual modules can improve the overall stability of an application. Of course, an updated module can also introduce new problems resulting in serious errors in dynamically linked environments, while the compiler might have discovered the problem at compile time in a monolithic application.

5.2.2 Transparency

Designed as invisible middlemen working to improve the perceived performance of network communication, a major feature of the original proxy servers is transparency. Transparent service is also a trademark of the more versatile client-side proxies examined in this thesis. To behave as was originally intended, a proxy should perform its duties without drawing attention to itself. Monitoring and adaptation of the content stream should be invisible or appear as part of the functionality of the client application or operating environment. Ideally, the user should forget about the proxy once it is started.

Freedom, PureSight and SurfWatch provide the most transparent service, working with the low-level network functionality of Microsoft Windows. They access the content stream directly through the operating system, offering easy installation and complete transparency. There is no need to configure specific client applications since the operating system automatically monitors all communication on behalf of the registered proxies. The obvious gain is that no communication can bypass the proxy, but the downside is that a user can not decide to exclude some particular client application from proxy interference. Due to the smooth low-level integration with the operating system and the fact that these applications do not produce additional information separate from the content stream, a user can normally ignore their existence. After installation and configuration, the single-task filtering proxies WebWasher, Proxomitron, A4Proxy, Junkbuster and NewsProxy are equally unobtrusive. However, they rely on intra-machine communication for their functionality, which normally requires manual configuration of different client applications to make them send their requests through the proxy. While making installation slightly more complex, it lets the user decide which clients to subject to proxy processing.

In the end, the task performed by the proxy determines whether true transparency is possible. The basic architecture of a proxy server provides transparency, but if a developer builds functionality that requires user interaction on top of the proxy, there is no guarantee for transparency. The extensible proxies Muffin and ByProxy exemplify this. The basic proxy functionality is running in the background, invisible to the user. An extension module has the option to be as transparent as the surrounding application, but it can also supply functionality that demands the user's attention. For example, the SELECT proxy for collaborative rating is built on top of Muffin. However, since this task clearly demands user interaction, the SELECT proxy is less transparent than Muffin and the average proxy. Although WebMate also requires interaction, it is more transparent. By providing interaction through the client application, it might seem to the user that the client environment and not a separate application provide the functionality. However, confusion might arise if the user moves to a machine where the proxy is not installed. Contrary to expectations, the client application does not provide the anticipated functionality. This is a problem common to all transparent services and applications, whether they are proxies or not.

5.2.3 Sophistication through aggregation

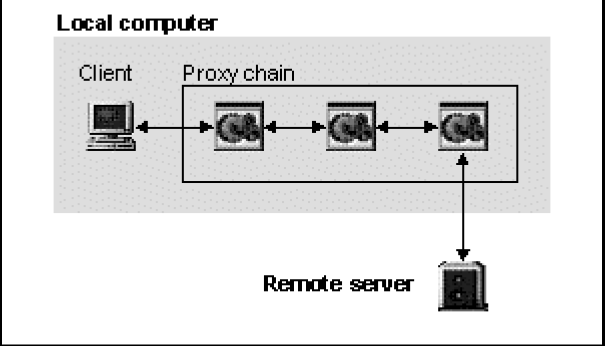
Most proxies focus on a specific task, such as breaking animation, removing personal information from requests, filtering, etc. A user might want to submit the content stream to many types of processing before it reaches the client application and for this, the proxies must have some support for aggregation.

Chaining is one way to support this, meaning that the output from one proxy is the input for another. Communication passes through multiple proxies on the way between client and server (figure 11), making the aggregate functionality of all proxies available to the user. This is the most common way to support aggregation, probably because the basic requirement is simply to change the network port (and optionally, the host) through which communication flows. All examined proxies except ByProxy and NewsProxy support chaining. SELECT does not seem to support chaining either, although it is based on Muffin which has chaining support. A4Proxy is not so straightforward regarding chaining, since its task involves relaying requests through external, privatising proxies. With the option to set a default relay proxy, chaining of local proxies is possible, but not a wise choice. To ensure privacy, the A4Proxy must be last in the chain, applied to communication just before it leaves the local machine. In this way, the proxy can relay communication through any remote, anonymising proxy it wishes.

Order could be important in proxy chains. A privacy-enhancing proxy should be the last stop between the client and the remote network. It also makes sense that a content blocking proxy performs its task before the document is processed by other proxies. Normally, users can control the chaining order by configuring the individual proxies. Although Freedom, PureSight and SurfWatch support chaining, close platform integration hides this aspect of configuration from the user. It is not possible for a user to decide in which order to apply these proxies to the content stream.

Chaining of proxies is a simple and well-supported way of aggregating behaviour. It does require configuration of multiple applications and it could be bad for performance, as will be discussed later. An alternative is to support aggregation through extensibility, an approach we recognise from the Pavilion framework in section

Figure 11. Proxy chain between client and server.



2. An extensible proxy allows developers to implement plug-in modules to extend proxy functionality. This supports aggregate behaviour without configuration of multiple applications and without the overhead of communication between chained proxies. It is also possible to apply extensions in a user-defined order, and since configuration is limited to one application, changing the order is probably simpler in extensible environments. Apart from some client-side proxies, many different applications use this approach to enable third-party developers to extend the basic functionality of the application.

What distinguishes an extensible application is that it allows dynamic loading of extension modules, modules possibly developed long after the first release of the application. A developer only needs to know about the application's programming interface and nothing about implementation particulars. With this knowledge, the developer can develop functionality extensions using the full expressiveness of supported programming languages. ByProxy and Muffin are the only extensible proxies in this examination, and their support for third-party extensions is the topic of the next subsection.

5.2.4 Development of third-party extensions

The extensible proxies Muffin and ByProxy are both implemented in Java, which is probably no coincidence. A basic requirement for extensibility is dynamic loading of extension modules, and Java has built-in support for run-time loading of classes. In addition, Java interfaces make it easy to enforce that an object provides the methods required of an extension module, regardless of module internals.

In Muffin, a developer must provide a FilterFactory that among other things maintain the state of the application between sessions, with the help of configuration functionality supplied by Muffin. As the name implies, the factory also supplies Muffin with Filter instances that receive and process content. What aspects of the content a filter can access depends on what interface(s) it implements. A ContentFilter can process requested documents directly through the stream flowing between client and server. A HttpFilter can intercept requests and send anything back to the client and a RedirectFilter intercepts a request and redirects the client to another resource. A ReplyFilter filter replies from remote servers, and finally, a RequestFilter does the same with client requests. Muffin pre-parses the content stream to give developers easy access to the information of the stream, creating Reply and Request objects that encapsulate header information from client requests and server replies. The content stream is transformed from the original byte-stream format to a stream of specialised objects providing high-level access to the HTML content, such as tags, tag attributes, character data, etc.

Instead of using internal streams, ByProxy reads the stream into byte-buffer objects. For reading and writing header information, ByProxy provides high-level reply and request objects, named BrowserRequestHeader and ServerDocumentHeader. In addition, ByProxy provides IncomingEmail and OutgoingEmail objects, encapsulating mail-specific information. Through these objects, an email filter can easily access message headers, content body, server information, etc. There are no news-specific objects. Instead of using predefined interfaces, a ByProxy extension specifies the types of objects it is interested in processing. For example, a filter can specify that it wants access only to IncomingEmail objects, and when the proxy receives an email, it calls the sniff method of extensions with registered interest in the object. The sniff method should be available in an object called a sniffer. The sniffer is responsible for acting on or manipulating the data it receives from a so-called proxy agent. The agent handles communication monitoring, and notifies the sniffer when it encounters something of interest. There is no interface to enforce the existence of the sniff method, but it must be available for ByProxy to function properly.

One of the major arguments for extensible solutions is to increase the productivity of third-party developers. Since the basic proxy functionality is available through the base application, a developer does not have to worry about the miscellany of the underlying technology. Indeed, this is a trademark of all layered solutions, such as operating systems, network protocols, etc. It also means that the overall application can evolve and become more attractive without constant involvement of the original developers. To increase the productivity of third-party developers, there must be a stable and understandable framework in which to develop extensions. Muffin's consistent use of interfaces ensure some degree of stability, while ByProxy's lax approach could result in serious run-time errors. Well-documented interfaces also visualise what is required of an extension module and because of this, it is probably quicker and easier for a third-party developer to produce extensions for Muffin than for ByProxy. The strength of ByProxy is that it is multilingual, allowing developers to process several types of content within the same application environment.

Neither Muffin nor ByProxy provide a user interface specialised for presentation of processing results. They do provide a graphical interface for configuration, but the extension module itself must supply any other interface. Although a module that process HTML content easily can display what it wants in the processed documents, the lack of interface support has potential negative effects. Several filters adding their information will clutter the requested document, developers creating their own interface will experience productivity loss, and modules without interface could be less user-friendly.

5.2.5 Platform independence

The client-side proxy architecture is not inherently platform-independent. A proxy relies on the same more or less platform-dependent programming languages and operating environment functionality as other solutions. Nonetheless, there are four discernible levels of platform-independence exhibited by the proxies in this examination.

On the first and most independent level, we find the Java applications. SELECT, WebMate, Muffin and ByProxy can run virtually unchanged on any machine with a proper virtual machine installed. In theory, they are platform-independent, but in reality, they are dependent on platforms with Java support. Despite this, they are more independent than any application targeted at a specific platform, since the virtual machine shields them from the particulars of the underlying operating system. On the next level, Junkbuster and NewsProxy are more platform-dependent, but since their source-code is available, they are at least portable to different platforms. As already discussed, porting is not a trivial undertaking and most users are limited to the pre-ported versions. However, as the Linux operating system and the GNU software has shown, open source projects tend to attract third-party developers whose effort results in availability for more platforms than the commercial alternatives.

The third level houses Proxomitron, WebWasher and A4Proxy. Although tied to the Windows platform, they behave as standard proxies and communicate through local network ports. This kind of network functionality is common across different platforms, and these applications should be portable without extensive structural changes. This is probably not the case with Freedom, SurfWatch and PureSight, all depending on platform-specific network functionality provided by the Windows operating system. They access the content stream directly through the operating system, a possibility that is not as common, or at least not as consistent, as the socket communication used by other proxies. Freedom, SurfWatch and PureSight constitute the fourth level, being thoroughly platform-dependent.

Regardless of the platform-independence of a specific proxy application, proxies are mobile. They can be located on the client machine, on a local network or anywhere on the Internet, and still be accessible to the user. Therefore, moving a proxy to a computer on the network where it is executable makes a platform-dependent proxy independent, at least in the eyes of the user. An obvious requirement is that the proxy has no user interface or the ability to display the interface through the content stream, such as Junkbuster or WebMate. However, moving the proxy to the network has negative side effects. Some of the benefits of a local proxy are lost, such as the ability to enhance user privacy before communication leaves the client machine, and the possibility to utilise local processing power for demanding tasks. In addition, network-based proxies will probably be multi-user systems, adding the complexity of multi-user environments to development and administration.

5.2.6 Performance impact

The introduction of proxies between server and client will have impact on performance, primarily through increased response times. Several factors influence the degree of performance degradation. If the goal of the proxy is to improve performance, the gains of processing should obviously compensate for the cost. The only proxy for performance enhancement in this examination is WebWasher. By removing advertisements from requested Web pages, WebWasher clearly improves the overall performance. The proxy eliminates requests for ads from busy servers, resulting in faster retrieval of Web documents.

Another factor is the simplicity of the task. Simple processing has less impact on performance. One example is the relatively straightforward text matching used by Proxomitron, SurfWatch, Junkbuster and NewsProxy. While simplicity is a way to minimise performance loss, it generally leads to less sophisticated behaviour. In cases where the processing is more demanding, asynchronous processing might be a way to alleviate the performance impact. This is the approach used by SELECT and WebMate, since they only need a quick glance at document-specific information. After extracting this information, the proxy releases the content stream to the client application and continues its processing. Obviously, there is a period of waiting before the processing results are available, but it allows the user to view the document while waiting. Although the overall loss in performance might be considerable, it is not as noticeable as when all processing must be finished before the document can be displayed.

Where asynchronous processing is not possible, performance could certainly be a problem. Examples are Freedom and A4Proxy, since they encrypt communication and/or introduce privacy-enhancing detours from the optimal path between client and server. The porn blocker PureSight can not use asynchronous processing either, since the content analysis must be done before deciding whether to show or to block the requested document.

Chaining multiple proxies for aggregate behaviour might have considerable impact on performance, since chaining requires socket communication between different proxies and all content processing is lost at each movement along the chain. For example, a proxy could adapt the content to simplify processing. Before sending it to the next proxy in the chain, the application must restore the content to its original state, and every proxy in the chain might repeat this procedure of parsing and restoring. From the performance viewpoint, the extensible approach of Muffin and ByProxy could be preferred, since it only performs pre- and post-processing of the content once. However, most client-side proxies do not support extensibility, and even those that do might maintain the view of the content as a datastream. Such a proxy uses internal streams to give extension modules access to the content. This means that a stream is sent to a module, which parses it and writes the result to another stream that is passed to the next module, and so on, until all modules has had access to the content. This is clearly inefficient compared to building a higher-level data structure from the stream and passing pointers to the structure to the interested modules.

Apart from simplifying the task and use extensibility rather than chaining, there are other ways to minimise the performance impact. Caching of documents comes to mind, since it is a function many ordinary proxies provide, but none of the examined proxies use internal caches of any sophistication. Moving ahead of the user to fetch documents that has not yet been requested is another way to improve at least the perceived performance. Pre-fetching increases the overall network traffic, but performance will probably improve for users following links in Web documents. WebMate provides pre-fetching of documents.

While caching, pre-fetching and other performance-enhancing methods could be valuable in a single-proxy environment, they might cause problems in multi-proxy chains. If several proxies attempt to cache or pre-fetch documents the results are likely to be confusing and inconsistent. From this viewpoint, it is understandable that the performance-enhancing functionality provided by some of the examined proxies is limited to maximise the performance of the individual proxy. For example, the Freedom proxy allows the user to set the length of the privacy-enhancing detour in favour of either performance or security, and PureSight has the ability to remember previous processing results so that the same page does not have to be processed every time it is accessed.

Up to this point, we have mapped out the territory of client-side proxies. Now it is time to leave already trodden paths, and step into hitherto unknown domains. The next section introduces Blueberry, a prototype proxy extension. Although deeply rooted in the proxy environment, it stretches the boundaries set by other client-side proxies appearing in this work.

6 Blueberry

Developed as a part of this thesis, Blueberry is a framework for processing the content of Web documents. Building on the proxy functionality of the extensible Muffin proxy, Blueberry provides an environment for swift and simple development of extension modules. This section also introduces BackLink, an example extension module. Blueberry is provided to visualise ideas about the implementation of client-side proxies, and not as an exercise in imaginative algorithms or a showcase for pretty programming. Hence, this section merely gives an overview of components and functionality. Readers interested in the details are invited to review the application, source-code and package documentation, available online [Blueberry 00].

6.1 Goals and design choices

As noted in the previous section, the extensible proxies Muffin and ByProxy do not provide an interface close to the content. Since an extensible proxy can contain modules with diverse functionality, a consistent and intuitive user interface is important. The first goal of Blueberry is to provide such an interface, a decision that rests on the assumption that content processing requires user interaction. A common look-and-feel for the proxy environment both helps and forces developers to provide user interaction that is consistent within the Blueberry environment. Consistent interaction helps users manage the configuration of multiple extension modules. Another assumption is that content processing produces additional information of interest to the user, and hence requires an interface that can display the information.

The second goal is to provide a solution that is both integrated with the client application and effectively client-independent. The choice of integration rather than separation follows from the decision to provide an interface. Since there is an interface, this should be close to the workspace of the user, visualising the relationship between processing and presentation. In the context of Web documents, the workspace is the browser. The common denominator of all browsers is HTML and Blueberry will provide a pure hypertext interface. The next choice is whether to display the interface in a separate browser window, a separate frame or embedded in the document. Separate browser windows have similar drawbacks as stand-alone application windows, and probably require Java or JavaScript to function properly. Inserting the interface in the original document is the easiest way to integration, but it destroys the intended layout of the document. What remains is to present the interface in a separate browser frame. This minimises the impact on the original document, makes it easy to distinguish the requested document from the interface, and it is still close to the user's working environment.

The third goal is to increase the productivity of third-party developers. Providing a ready-to-use interface is one way to do this, high-level access to the content is another. Muffin works with streams of high-level objects, and ByProxy works with byte-buffers. Both these approaches require extension modules to perform additional parsing to access the required content elements. The approach of Blueberry is to build a high-level data structure from the content stream, maintaining the internal hierarchy exhibited by HTML documents. Extension modules access the structure through object references, references that point directly to the type of content the modules are interested in. There is no need for additional parsing, and it is easy to navigate the nested hierarchy of each structure element. This should also prove beneficial to the overall performance of the application, but to some extent, the more demanding parse algorithm and the complex data structure lessen the gains.

As a side effect of these design choices, Blueberry is practically platform-independent, since it relies only on Java and HTML.

6.2 Limitations

An obvious limitation is that Blueberry only supports processing of Web content. Request and reply headers, request redirection, and other details of HTTP communication are not accessible through Blueberry. However, the underlying Muffin proxy supplies this functionality. A Blueberry extension could choose to also implement the interfaces required by Muffin and register itself as a Muffin filter, thereby gaining access to these parts of the communication. Neither Blueberry nor Muffin supports non-HTTP communication.

The most notable deficiency is that Blueberry does not handle framesets or internal frames well. In the context of content processing, the content of framed documents is more interesting than the enclosing frameset document. At this point, there is no solution to the problem of treating framed documents as a single entity. In a best-case scenario, frame documents display correctly but will not be subject to processing. Following links in framed documents will probably cause problems, and nested framesets are never displayed correctly. Until this is resolved, behaviour regarding frames is unspecified and unstable.

Since Blueberry is a prototype implementation and not a production-quality release, there are inevitably other limitations. The functionality is not thoroughly tested, and there might be bugs and inconsistencies in the basic application and the programming interface for third-party developers. The code is not optimised for performance, although it should run well on most contemporary machines.

6.3 Blueberry architecture

The Blueberry framework uses the extensible proxy Muffin to provide its own extensible environment. The major architectural components, depicted in figure 12, are Blueberry itself, an SGML parser and the programming interface for extension modules.

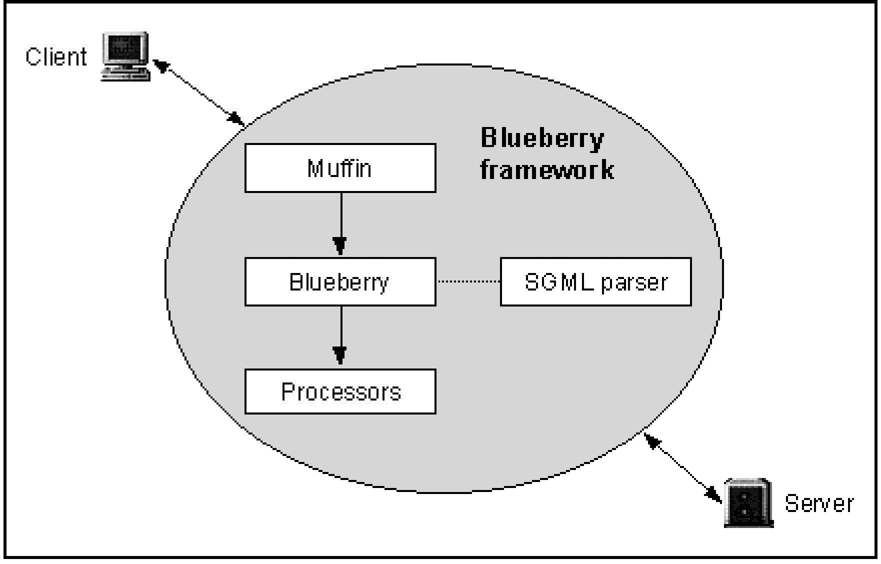
6.3.1 Blueberry, a Muffin filter

Blueberry is an extension to the Muffin proxy. The Blueberry class, implementing Muffin's FilterFactory interface, the BlueberryFilter class that implements the HttpFilter and ReplyFilter interfaces, and various helper classes constitute an environment for content processing and user interaction. The basic tasks are extension handling, content parsing and user interface creation.

At initialisation, Blueberry loads all registered extension modules into memory. As a module is instantiated, it is queried for the element types it is interested in processing. This decides what the modules will get access to during the processing phase. Through the ReplyFilter interface, Blueberry intercepts replies from remote servers. Reply objects provided by Muffin give access to the raw content stream, which is processed by the SGML parser described below. The next step is to traverse the hierarchical tree structure created by the parser. For each HTML element in the structure, extensions that have registered interest in the element type are called upon to perform processing before the tree traversal continues.

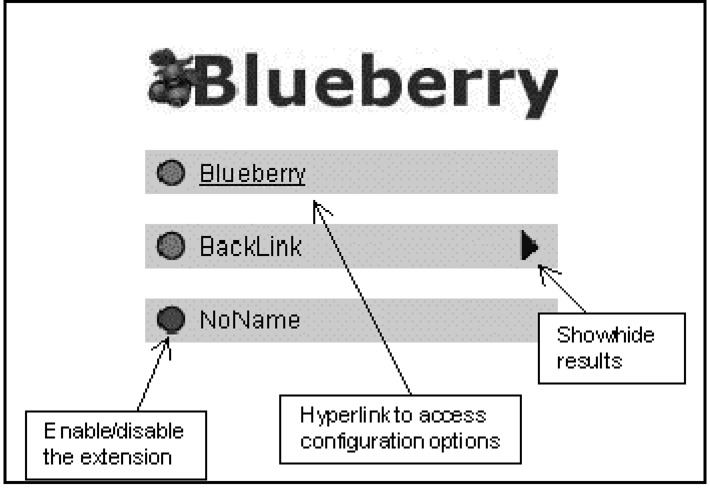
When the requested document is processed, Blueberry transforms it to a frameset document; the left frame contains the user interface and the right frame the original document. The interface

Figure 12. Blueberry architecture.



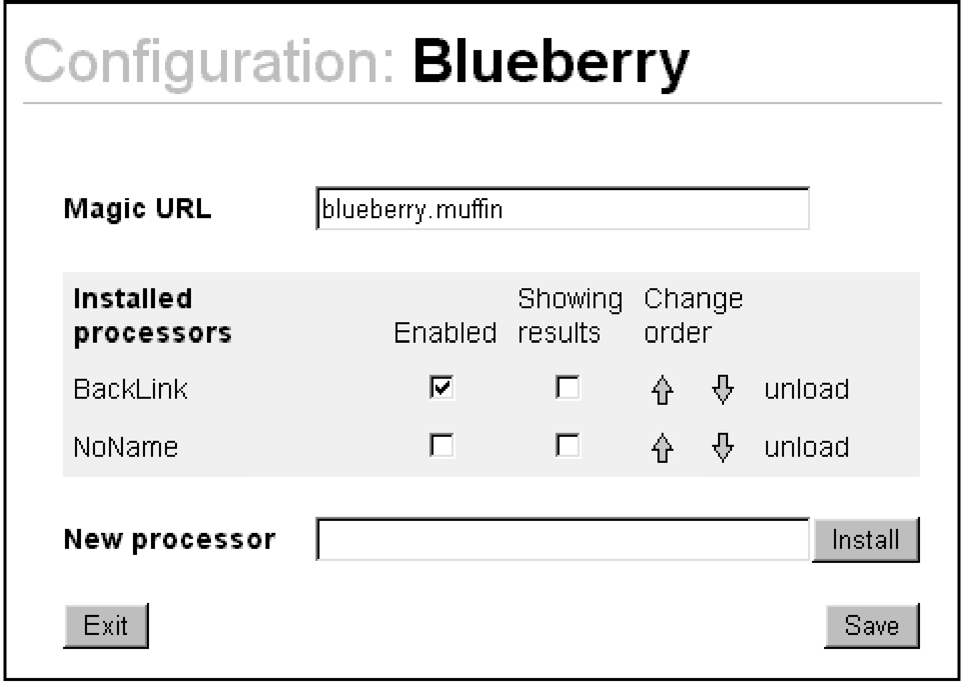
gives the user control over the available functionality. Individual modules can be enabled, disabled and configured (figure 13). Naturally, the interface is re-created for each requested document, and Blueberry collects the processing results of all enabled modules and presents them to the user. General configuration of Blueberry is also accessible from the interface frame; most important is the extension administration. Existing modules can be re-ordered, enabled, disabled or completely shut down, and new modules can be loaded and configured (figure 14). It is also possible to edit configuration files manually, but all functionality is accessible from within the client environment.

Figure 13. Blueberry user interface.


Blueberry
Blueberry
BackLink
NoName
Show/hide results
Enable/disable the extension
Hyperlink to access configuration options

The interface is quite large, as shown in figures 13 and 16. This could be a problem, especially with small screens. The assumption is that the information provided is valuable enough to justify this, but it might be necessary to reconsider this choice or at least make it possible to minimise the interface. In addition, the vertical frame might force users to scroll horizontally to view the main document. This is clearly an unwanted situation, and a future enhancement could be to let the user choose if the interface frame should be horizontal or vertical. Finally, Blueberry is not a transparent solution, at least where transparency is equal to invisibility. However, it is transparent in the sense that it integrates all its functionality within the browser environment, making it appear as part of the enclosing application.

Figure 14. Blueberry configuration interface.



Configuration: Blueberry

Magic URL: blueberry.muffin

Installed processors	Enabled	Showing results	Change order		
BackLink	✓	☐	⇧	⇩	unload
NoName	☐	☐	⇧	⇩	unload

New processor: [ ] Install

Exit Save

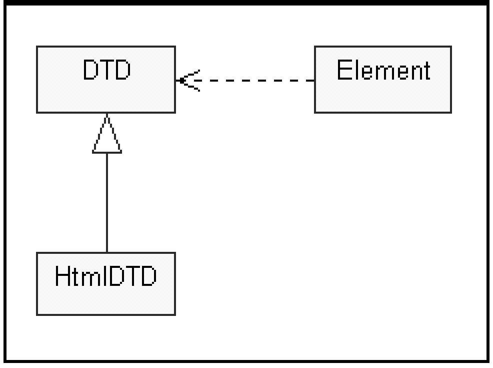
Blueberry uses a simple protocol to support user interaction through hyperlinks, HTML forms, etc. All requests to a "magic URL" are intercepted through the HttpFilter interface of Muffin. The magic URL is http://blueberry.muffin/ by default, but it is user-definable. To decide what should happen, additional information is appended to the URL. This information has syntax similar to the queries created by the GET method of HTML forms. Blueberry parses the information and performs the desired action, either directly or by delegating it to the extension that initiated the interaction. This enables specific modules to provide interaction of their own, and it is also the method used to communicate directly with the Blueberry framework.

That Blueberry provides an environment for both processing and presentation can give third-party developers a sense of freedom, since they can focus entirely on the specific processing task performed by the extension. Other developers might feel that the framework is too prohibitive, since it forces extensions to behave in a certain way, especially regarding presentation of processing results. Indeed, it is limiting to demand that modules present their results as part of the enclosing Blueberry interface, but this is a conscious choice. It is necessary to circumscribe the freedom of individual developers to maintain a consistent interface.

6.3.2 SGML parser

The main vehicle for providing high-level abstraction and access to the content stream is a SGML parser (figure 15), responsible for transforming the content from a low-level byte-stream to a high-level hierarchical data structure.

Figure 15. Overview class diagram of the SGML parser.



The basic building block of the structure is an Element, encapsulating content elements and their associated attributes. An element can encapsulate standard mark-up elements, comments, character data, whitespace, and other types of content that appears in an SGML document. Since the structure is hierarchical, an element can also contain any number of other elements nested within its structure. The Element class provides methods for navigating the nested elements, finding specific elements, displaying elements, etc. It is also possible to create Element objects manually, for example by passing a string to the constructor or by using the element and attribute access methods.

While Element objects represent the actual content, a DTD object represents the data type definition, i.e. the grammar, applying to a certain document. The DTD enforces these rules by splitting the content into the components prescribed by the grammar, and by making sure nesting of elements is done according to the rules.

The abstract DTD class supplies all functionality for parsing and rule enforcement, making it easy to tailor the parser for other languages derived from SGML. A subclass must define nesting rules and characteristics of tags, comments and attributes in the specific mark-up language. The HtmlDTD class extends the DTD class to provide support for parsing HTML documents. At this point, there is no strict enforcement of the HTML data type definition, but rather a liberal parsing. The goal is to preserve the look of the original document, not to force it into syntactic correctness.

Although the structure created by the parser gives efficient access to individual elements, it makes progressive processing impossible. In a stream-based solution, already processed parts of the content can be progressively delivered to another proxy or to the user's client application before the processing is complete. In the high-level tree structure used here, the top-level elements are the last to be completed. This means that Blueberry must process the content completely before it can be restored to its original shape and released, which could have impact on the performance of proxy chains.

6.3.3 Additional processors

A module wishing to process content within the Blueberry framework must implement the BlueberryProcessor interface. This interface defines the methods that a module must provide, of which the most important are described here.

The handleElements method returns an array of strings containing the types of HTML elements the module wants to process. If a module registers interest in the anchor tag (A), the process method of the module is called every time an anchor appears in the content stream, with an Element instance and the address of the processed page as arguments.

When a document is completely processed, the hasDisplay method is called on all modules that are enabled and showing, to see if they have anything to display. If they have, Blueberry gathers the resulting Element objects by calling the display method of the modules, and displays the Elements as part of the user interface.

The methods for module configuration have a similar structure. If a module indicates that it is configurable, through the hasOptions method, Blueberry will display the name of the module as a hyperlink in the user interface. Clicking on the link will result in a call to the options method of the module, returning an Element object that Blueberry displays. Finally, the message method of the BlueberryProcessor interface is the medium for direct interaction between user and extension module. For example, a developer can use HTML forms to handle module configuration. When the user submits the form data, the module receives it through the message method. The BlueberryLink class encapsulates the specific format of these messages.

6.4 BackLink

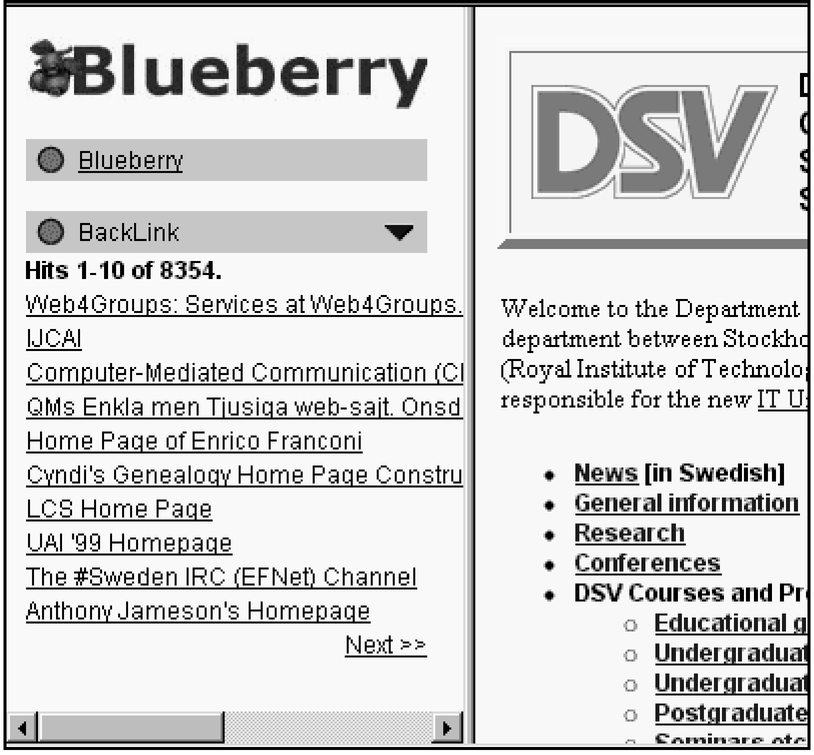
BackLink is an example Blueberry extension. For each visited page, it displays the "back-links" of that page, i.e. links to other Web pages that contain hyperlinks to the current document (figure 16). In its own right, BackLink would hardly qualify as a client-side proxy candidate. The only information it needs is the URL of the current document, and it could as easily be implemented as a browser plug-in. However, it takes advantage of the functionality of the Blueberry framework to gain access to content and to display results, visualising how easy it is to extend functionality without losing the consistent look-and-feel of the extensible framework.

BackLink consists of three classes. The BackLink class implements the BlueberryProcessor interface, acting as the link between the Blueberry framework and the BackLink functionality. The BackLinkDocument class is the abstract base class for queries to different search engines. It extends the Element class, inheriting the capability to build high-level data structures from the content. It provides BackLink with results to display, and it supports navigation of queries resulting in multiple-page replies. The Evreka class extends BackLinkDocument to provide specialised querying functionality. It handles queries to the online search engine Evreka (www.evreka.com), and parsing of query results. These classes can query remote search engines, parse replies and interact with the user, with less than 200 lines of (spacious) code.

If many people should use BackLink, it would probably have to use more of the content processing functionality provided by Blueberry. In the current version, it queries online search engines, parses the reply and displays the result. On a small scale, this is acceptable, but on a larger scale, there should probably be a dedicated BackLink server handling these queries. One way to maintain the server's database could be to let individual BackLink processors extract link information from visited pages and report the results to the server. In its simplest form and by using the processing functionality of Blueberry, implementing this function should not require more than a few lines of code. In this scenario, the proxy extension approach is better and more scaleable than browser plug-ins.

The Blueberry framework has visualised an approach not used by any of the other proxies examined in this work. The major difference is the close integration of user interface and client application. Now, all that remains is to examine the results of this and earlier sections, discuss them from a more general viewpoint and draw conclusions regarding the good and bad aspects of client-side proxies for content processing.

Figure 16. BackLink in action.



Blueberry

Blueberry

BackLink

Hits 1-10 of 8354.

Web4Groups: Services at Web4Groups.
IJCAI
Computer-Mediated Communication (Cl
QMs Enkla men Tjusiga web-sajt. Onsd
Home Page of Enrico Franconi
Cyndi's Genealogy Home Page Constru
LCS Home Page
UAI '99 Homepage
The #Sweden IRC (EFNet) Channel
Anthony Jameson's Homepage
Next >>

Welcome to the Department
department between Stockho
(Royal Institute of Technolo
responsible for the new IT U

- **News [in Swedish]**
- **General information**
- **Research**
- **Conferences**
- **DSV Courses and Pr**
  - **Educational g**
  - **Undergraduat**
  - **Undergraduat**
  - **Postgraduate**

7 Conclusions

Before drawing any conclusions from the previous sections, let us recapitulate the aim and purpose of this thesis. The overall context is the Internet and its abundance of resources. The purpose here is to investigate the merits of the client-side proxy approach as a way to help users find interesting information through content processing, adaptation and information retrieval. From a general viewpoint, this section focuses on the questions posed in the introduction: When are client-side proxies better and when could other approaches be preferable? Do existing client-side proxies realise the potential benefits of the approach? Are there ways to improve them? Table 1 provides parts of the answers, summarising the characteristics of different approaches for content processing.

Table 1. Characteristics of different solutions.

	Client-side proxy	Client plug-in	Integrated client[1]	Web service
Usability[2]	Medium to low	High	High to medium	High
Performance impact	Medium to high	Low	Low	Server-dependent
Access to user's machine	Yes	Client-dependent	Yes	No
Sophistication[3]	Arbitrary	Client-dependent	Arbitrary	Simple
Supports aggregation	Yes	Possibly	Possibly	No
Platform independence	Potential	Low	Low	High
Transparency[4]	High	High	Low	Low
Client independence	High	Low	Low	High
Interface integration	Medium	Medium	High	High
Access to content	Direct	Through client	Direct	Indirect
Exhaustiveness[5]	High	Low	Low	Low
Privacy[6]	High	Low	High	Low

1. A platform-specific application, such as stand-alone Web browsers, newsreaders, etc.
2. The level of usability includes installation, configuration and overall ease of use.
3. The sum of implementation language expressiveness, available processing power, access to operating system functionality, etc.
4. A transparent solution runs in the background or as an integrated part of the client environment.
5. An exhaustive solution can handle different types of communication and intercept it before it leaves the local machine.
6. The ability to perform privacy-enhancing processing.

7.1 Use client-side proxies or not?

The first thing to consider is whether to use client-side proxies at all. Compared to other solutions, is there anything that gives a proxy the upper hand?

7.1.1 Exhaustive access to content

One of the trademarks of proxies is that they have direct access to the content stream. Sitting in the middle of communication, they can easily intercept everything of interest. This is clearly an advantage compared to client plug-ins and remote Web services. A plug-in is subject to the good will of its parent environment. It might get complete access to the content through the client, but inherits the limitations of an integrated application. Web services have only indirect access to communication between client and server. An example is the Anonymizer, described in section 4.1. A user must manually request documents on the Anonymizer site or through a special text-field in an anonymised document. If the user does not deliver this information, the Anonymizer has no access to the content. Was it a proxy, it would automatically intercept all requests without placing cognitive demands on the user. Hence, the proxy approach is more exhaustive than Web services. It is also more exhaustive in the sense that it can handle virtually any kind of communication - Web documents, email, ftp, news, telnet, etc. Integrated clients also have this ability and direct access to the content, but they are less exhaustive. An integrated application can not process content accessed through other client applications. Since plug-ins rely on their parent applications for content access, the same limitations apply to them.

Because the proxy in theory can intercept anything from anywhere, it has big potential to perform privacy-enhancing processing, such as encryption and anonymisation. That it can apply this processing before the content leaves the local machine is a strength it has in common with stand-alone clients. Again, these applications can only handle their own communication, while the proxy can process all communication before releasing it to the network. When Web services are involved, the initial communication is always unprotected.

In general, developers should consider the client-side proxy approach when the task at hand demands direct access to the content stream. If the task also involves privacy protection and exhaustive interception of different kinds of communication from different client applications, the proxy solution is clearly the best choice. The Freedom proxy of section 4.1 is a good example.

7.1.2 Processing power and sophistication

Since a client-side proxy is located on the end-user's local machine, it has access to the full functionality and processing power of the local operating environment. Like integrated clients and plug-ins, but unlike Web services, it can perform demanding tasks close to the destination of the content, where it is most efficient. An illustrative example is PureSight, described in section 4.5, that uses demanding artificial intelligence algorithms for content analysis. Even if Web services run on powerful servers, a large number of users share the processing power. It is easier to provide a fast and reliable service if the functionality is reasonably simple.

Although it is better to perform more advanced and power-demanding processing locally, dedicated servers are better at relatively simple but large-scale operations. Web page indexing performed by search engines and online directories is one example where the local environment is simply too small to store the information. An interesting solution would be to use the processing power of client-side proxies in conjunction with the power of large-scale central servers to facilitate collaborative processing. A real-world example of this is the SETI@home project, part of the Search for Extraterrestrial Intelligence program at UC Berkeley [SETI 99]. Interested Internet users download a small part of the massive amounts of data collected through the SETI programs, and when their local computer has processed the data, the results are returned over the Internet. The project does not use client-side proxies, but it shows the strength of collaborative processing. A client-side proxy could download or gather data, process it and deliver the results to a central server, utilising the local processing power.

The processing power available to an application has obvious impact on what it can do and what levels of sophistication it can reach. Somewhat simplified, a local approach has potential to be more sophisticated than a remote Web service. If the task involves applying demanding algorithms to relatively small amounts of data, such as single Web documents, the local approach is preferable. It does not matter whether it is a proxy, an integrated client or a plug-in, as long as they have access to the local machine. In this sense, the local approach can be more sophisticated. On the other hand, if the involved algorithms are simpler, but the processed data more extensive, a high-end Web services could be better. The massive storage capacity required is better utilised if many users share the resource.

Since the Web came into being, the major usability focus has been on ease of learning for novice users. Simplicity has been the obvious gain, but at the cost of sophistication, especially for experienced users. As an example, the Anonymizer Web service is simple and straightforward, but decidedly not as sophisticated as the client-side proxy approach of Freedom. Although Web services often are simpler than local applications, the functionality does not have to be trivial [Nielsen 00]. As users become more loyal to specific web-sites and as they come to depend on web-based functionality in their daily work, these services must adapt to the needs of experienced users. Improving navigation by providing something similar to keyboard shortcuts in local applications is one example. As Web services evolve towards more sophisticated functionality, they will challenge the client-side applications. A sign of the times is the propaganda for thin clients and application service providers.

Sophistication can also be realised by aggregating the efforts of several actors. Most proxies, both client-side and others, support aggregation of behaviour under user control. Non-proxy plug-ins and applications might also support some notion of aggregation, but not in the simple and standardised way of the proxy. Proxies commonly support aggregation through chaining, or more rarely, through extensibility. Both these approaches are described in section 5.2.3. Through its support for aggregation, the proxy approach is more flexible and allows individual users to extend the functionality by simply adding another proxy or proxy extension.

7.1.3 Independence or integration

All the examined proxies have shown, to different degrees, that one of the major benefits of the proxy approach is client independence. By placing the processing functionality in a layer independent of client brands and versions, the proxy can deliver the same functionality regardless of user preferences. The proxy has this property in common with Web services.

On the other hand, an integrated application has a closer relationship with the user that it can exploit for detailed monitoring of user behaviour, down to single mouse movements and keyboard actions. While it is true that a client-side proxy can analyse the outer aspects of user behaviour, such as what resources are requested and time between requests, a more integrated solution can create more fine-grained and advanced user profiles.

Connected to integration is the question of transparency. An approach is fully transparent if it works completely in the background or appears as an integrated part of the functionality of another application. A stand-alone client is obviously not transparent, and neither are Web services. A plug-in is transparent, since it is an extension to the functionality of the parent application and appears to be a part of this application. A client-side proxy can also be fully transparent. It can run completely in the background, as the content blocking applications of section 4.5, or it can integrate both interaction and presentation with the client application like the WebMate proxy or the Blueberry framework (sections 5.1.2 and 6, respectively). These approaches integrate closely with the content, appearing to be part of the requested documents or the overall functionality of the client application.

Although some of the proxies examined in this work are independent of platform as well as client, it is not an inherent quality. Proxies rely on the same technologies available to stand-alone applications and plug-ins. If platform independence is crucial, a web-based service is the natural choice as the only one providing true independence. Otherwise, platform-specific applications have many benefits, whether they are proxies or not. Integration with a familiar environment can make the application more visually attractive and easy to use, as opposed to the sluggish interfaces exhibited by the independent proxies examined in this work. Another important benefit is that direct use of platform-specific functionality can improve the overall performance of the application.

Platform independence could be important for software developers. That proxies use standardised network protocols and network functionality common to many operating systems indicates that this approach could be more platform-independent than applications with closer platform integration. When an application is developed for multiple platforms, using independent solutions can reduce the time, cost and complexity of the development process. An application developed as a platform-independent proxy is widely available for testing, with the option to tailor subsequent production releases to the most important target platforms to provide the benefits of platform-specific applications.

For the end-user, platform independence is probably not an important issue. Client-side proxies are supposed to run on single machines, and single machines usually provide a single operating platform. Of course, some users work on multiple platforms, and if they want to use the proxy functionality on every machine, a platform-independent solution is preferred. In general, users are presumably more interested in the gains of platform-dependence - easy installation, a familiar user interface, better performance, etc - than in the vision of platform independence.

7.1.4 Performance impact and usability

Even if client-side proxies have several virtues, as we turn to overall performance and usability we must acknowledge that other solutions generally are better. Although content processing always has negative impact on performance, unless the processing is explicitly aimed at enhancing performance, proxies can be even worse than the other approaches. A major reason is the local socket communication required for most proxies, while integrated clients and plug-ins work in the client application environment. Close platform integration, as exhibited by for example the Freedom and PureSight proxies, could be a way to alleviate the performance impact, since low-level communication methods are more efficient.

Close integration could also improve usability, which is a weakness with many proxies. In general, they are more difficult to install and configure since they require configuration of both the proxy itself and the client applications whose content they want to process. Platform-specific approaches could alleviate this burden. If ease of use is crucial, a web-based approach should also be considered. As already discussed, a major feature of these services is high usability, and they require no installation at all.

The ease with which an application can be uninstalled is also a usability factor, but proxies are generally as easy or difficult to uninstall as other solutions. Most proxies use the same uninstallation procedures available to all kinds of applications. However, it is an issue when proxy chains are involved. If a proxy that is part of a chain is uninstalled, the chain is broken and the user must reconfigure the neighbour proxy. When this happens, uninstalling a client-side proxy is more complicated than uninstalling an ordinary application.

It might seem discrediting to platform-independent techniques that the independent proxies of this examination exhibits slower, clumsier and visually unattractive interfaces compared to the platform-specific systems. The lack of common design guidelines for platform-independent applications is partly to blame, but such guidelines will probably evolve as the approach matures.

7.1.5 Legal and ethical considerations

Apart from purely technical considerations, there might be situations where it is not possible or desirable to use client-side proxies, due to legal or ethical considerations. The problem with ad removers such as WebWasher has already been mentioned in section 4.3. If many users decide to remove advertisements, it could become harder for providers to supply free services. Anonymisers such as Crowds or Freedom might also be viewed unfavourably. Companies could forbid their employees to use them, since they make it difficult for administrators to monitor users' online behaviour. Shopping sites could refuse to accept orders from anonymised connections, since anonymisation makes it harder to trace fraudulent users. Furthermore, it could be irritating for a information provider if the information is processed and changed on the way to the user. All factors combined, there is a risk that online actors will take steps to inhibit the use of such applications, unless they handle these issues in a manner that is acceptable to all parties. Of course, these concerns apply to any content processing application, but many of the existing client-side proxies focus on this kind of tasks.

7.2 Today and tomorrow

What is most distinguishing of the client-side proxy approach is the natural and direct access to the content stream. Several potential benefits that can make the client-side proxy better at content processing originate in this closeness. Let us take a closer look at how the proxies of today utilise the potential of the approach, and how to make them better in the future.

The first potential benefit is, obviously, to easily access, analyse and adapt the content. In the context of this work, this is what client-side proxies are all about. The close tie to the network could also make retrieval of additional information a natural part of proxy functionality. Apart from BackLink in section 6.4 and the SELECT proxy for collaborative rating (section 4.2) that provides information about other users' ratings, this approach is not so common. Through analysis of the communication flow between client and servers, a client-side proxy could also build models of the user, that could be used to refine the behaviour of the proxy to provide support optimised for individual users. WebMate is the only example of this approach. In general, functionality for building user models and for retrieving additional information is scarce among the proxies examined in this work. This is an area with great potential, and it might be good to take further advantage of it.

Close to the content but separated from the content presentation, the proxy approach is basically client-independent. Although some proxies, for example WebMate and SELECT, use techniques that slightly circumscribe this independence, it is still a strong point in favour of the proxy approach. A user can take advantage of the functionality of a client-independent proxy regardless of the application used for presentation, and this is definitely something that existing and future proxies should uphold.

Related to independence is transparency. With functionality placed in a layer separate from presentation, client-side proxies can do their work in the background in the same manner as operating systems services. If we leave out installation and configuration, most proxies perform in the background. However, the focus on content processing poses a problem. It is common that processing generates information that should be visible to the user. The standard solution is that the proxy provides an application environment of its own, making it less transparent. An alternative is to provide this information as part of the content, when the content protocol allows this. In reality, this kind of integration is feasible only with HTML content. Blueberry carries this notion to the extreme, incorporating the complete user interface in the processed content. WebMate is more cautious - the interface is accessible through the content but displayed in its own windows. That incorporation of content and interface is possible is also an effect of the direct access to the content. It is assumed here that processing creates interesting results, and that these results should be displayed as close to the working environment as possible. However, this is an opportunity that should be used with caution, since it imposes great structural changes on the requested document and occupies a largish part of the client application's workspace.

The potential for sophistication, either by utilising the local machine for demanding processing or by aggregation of functionality, is partly fulfilled by the client-side proxies of today. Most processing is relatively simple text matching and filtering, but there are also attempts at more powerful processing, most notably in WebMate and PureSight. Compared to the simpler approaches, the content analysis performed by PureSight minimises the need for user interaction and manual updates, resulting in a more usable application. This is an example from which others could learn. Although most proxies perform relatively simple processing, they generally support sophistication through aggregation. The common way to combine the functionality of several proxies is chaining. This is straightforward and rather flexible, but a well executed extensible approach might be better for performance and usability, conducting all processing and user interaction within a single application. The extensible proxies Muffin and ByProxy (section 5.2.4) partly live up to this notion, but the higher-level content abstraction and integrated interface of Blueberry shows a possible way to utilise the potential even more.

7.3 Who will use a client-side proxy?

Compared to other approaches, client-side proxies have architectural strength. The combination of direct access to the content, client independence, access to the local operating environment and the inherent support for aggregate behaviour is a compelling argument to use proxies for sophisticated content processing. The problem is that even if the architecture has merits, other solutions generally exhibit better usability and performance. Most of the examined proxies will probably be considered only by advanced users, while the other will prefer the simpler and more familiar alternatives - integrated clients, plug-ins and Web services. There are exceptions, such as WebWasher and PureSight, that combine the strengths of the proxy with the usability of integrated applications, but this is not the usual case.

There is also the risk that client-side proxies will not be used, simply because they are not discovered by potential users. Most of the proxies examined here have a low profile, at least compared to heavily advertised integrated clients and Web services. It is more rule than exception that marketing strength is more important than technical merits in deciding which solution will be commonly accepted and used.

On the good side, there is a close and natural tie between proxy, content and network. As use of the Internet increases and client-side machines and applications become more integrated with networks and remotely hosted services, applications with network capabilities have the competitive edge. This is clearly an advantage for the client-side proxy, since networking is the foundation of its existence. If this advantage is utilised together with a stronger focus on usability issues, the future of the client-side proxy might not be so bleak.

7.4 Further research

The primary focus of this thesis has been the potential merits of client-side proxies for content processing, but several related areas should also receive attention. The legal aspects of content processing and adaptation are interesting.

Changing content provided by others might be a copyright violation, and displaying retrieved Web pages within a framework such as Blueberry might be viewed unfavourable by the information providers. A survey of the opinions of these providers regarding client-side proxies and content processing could be of value. If integration of user interface in processed documents is better than clean separation, and how it should be done to be accepted by users also deserves a more in-depth answer. As wireless communication becomes more important, proxies could be a bridge between earthbound and ethereal resources. Whether client-side proxies have anything to contribute to these mobile environments could be investigated. Finally, the specific processing tasks involved need to be continually examined and improved. There has been much research regarding these topics, such as methods to retrieve information fulfilling the needs of individual users, building sophisticated user profiles, making navigation easier, etc. However, due to the fast pace of technology-changes and growth of available information, this area requires constant attention.

8 References

[Agent 00]

Agent News and Mail Reader, 2000. http://www.forteinc.com/agent/index.htm

[Alexa 00]

Alexa Internet, 2000. http://www.alexa.com

[A4Proxy 00]

Anonymous Internet Surfing: Software: Anonymity 4 Proxy, 2000. http://www.inetprivacy.com/a4proxy/

[Anonymizer 00]

Anonymizer, 2000. http://www.anonymizer.com

[Blueberry 00]

blueberry : tamasz towers, 2000. http://www.dsv.su.se/~tomas-vi/stuff/java/blueberry/

[Brooks et al 96]

Brooks, C., Mazer, M., Meeks, S., and Miller, J., 1996. Application-specific proxy servers as HTTP stream transducers. In *Proceedings of the 4th International World Wide Web Conference.*

[ByProxy 98]

ByProxy -- Take Control of the Internet, 1998. http://www.besiex.org/ByProxy/index.html

[Chen and Sycara 98]

Chen, L., Sycara, K., 1998. WebMate: a personal agent for browsing and searching. In *Proceedings of the second international conference on Autonomous agents, 1998, pages 132-139.*

[Freedom 00]

Freedom, 2000. http://www.freedom.net

[Ganesan 99]

Ganesan, R., 1999. The Messyware Advantage. In *Communications of the ACM, Vol. 42, No. 11, November 1999, pages 68-73.*

[Jing et al 99]

Jing, J., Helal, A. S., and Elmagarmid, A., 1999. Client-Server Computing in Mobile Environments. In *ACM Computing Surveys, Vol. 31, No. 2, June 1999, pages 117-157.*

[Junkbuster 99]

Internet Junkbuster Headlines, 1999. http://www.junkbusters.com/ht/en/ijb.html

[McKinley et al 99]

McKinley, P. K., Malenfant, A. M., Arango J. M., 1999. Pavilion: A Middleware Framework for Collaborative Web-Based Applications. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work, 1999, pages 179-188.*

[Microsoft 00]

Official Guidelines for User Interface Developers and Designers, MSDN Online Library, 2000. http://msdn.microsoft.com/isapi/msdnlib.idc?theURL=/library/books/winguide/welcome.htm

[Muffin 00]

MUFFIN.DOIT.ORG, 2000. http://muffin.doit.org

[NetNanny 00]
Net Nanny filtering software for your PC, 2000. http://www.netnanny.com/netnanny/netnanny.htm

[Neumann and Weinstein 99]
Neumann, P. G., Weinstein, L., 1999. Inside Risks: Risks of Content Filtering. In *Communications of the ACM, Vol. 42, No. 11, November 1999, page 152*.

[NewsProxy 99]
nFilter Home Page, 1999. http://www.nfilter.org

[Nielsen 00]
Nielsen, J., 2000. Novice vs. Expert Users (Alertbox Feb. 2000). http://www.useit.com/alertbox/20000206.html

[Proxomitron 00]
The Proxomitron - Universal Web Filter, 2000. http://members.tripod.com/Proxomitron/

[PureSight 00]
PureSight - Homepage, 2000. http://www.puresight.com

[Reiter and Rubin 99]
Reiter, M. K., Rubin, A. D., 1999. Anonymous Web Transactions with Crowds. In *Communications of the ACM, Vol. 42, No. 2, February 1999, pages 32-48*.

[SELECT 00a]
SELECT server at SZTAKI, 2000. http://samson.aszi.sztaki.hu/SELECT/

[SELECT 00b]
SELECT Project Overview, 2000. http://cmc.dsv.su.se/select/select.html

[SETI 99]
SETI@home, 1999. http://setiathome.ssl.berkeley.edu

[Starrin 94]
Starrin, B. 1994., Om distinktionen kvalitativ - kvantitativ i social forskning. In *Kvalitativ metod och vetenskapsteori*. Studentlitteratur, Lund.

[SurfWatch 00]
SurfWatch, 2000. http://www1.surfwatch.com

[Thaler and Ravishankar 98]
Thaler, D. G., Ravishankar, C. V., 1998. Using Name-Based Mappings to Increase Hit Rates. In *IEEE/ACM Transactions on Networking, Vol. 6. No. 1, February 1998, pages 1-14*.

[WAP 00]
Wireless Application Protocol, 2000. http://www.wapforum.org

[Waxman 00]
Waxman, J., 2000. Internet Trends Report, Issue 4Q99, February 2000. Alexa Research, San Francisco.

[WebMate 99]
Agent: WebMate, 1999. http://www.cs.cmu.edu/~softagents/webmate/

[WebWasher 00]
webwasher.com, 2000. http://www.webwasher.com/index.htm

[WebWiper 00]
WebWiper, Inc., 2000. http://www.webwiper.com/frameset.htm