Flexible Open Caching for the Web

S. J. Caughey, D. B. Ingham and M. C. Little
Department of Computing Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, United Kingdom
{s.j.caughey, dave.ingham, m.c.little}@ncl.ac.uk

Abstract

Caching plays a vital role in the performance of any large-scale distributed system and, as the variety and number of Web applications grows, is becoming an increasingly important research topic within the Web community. Existing caching mechanisms are largely transparent to their users and cater for resources which are primarily read-only, offering little support for customisable or complex caching strategies. In this paper we examine the deficiencies in these mechanisms with regard to applications with requirements for shared access to data where clients may require a variety of consistency guarantees. We present 'open' caching within an object-oriented framework, an approach to solving these problems which, instead of offering caching transparency makes the caching mechanism highly visible allowing great flexibility in caching choices. Our implementation is built upon the W3Objects infrastructure and allows clients to make caching decisions for individual resources with minimal impact upon other resources which do not support our mechanisms.
Keywords: web; caching; object-oriented; weak consistency

1. The Changing Web

Since its inception the Web has been primarily concerned with the delivery of information to users. That is, the majority of users access resources published on the Web in a simple request/response, read-only mode. Servers (resource managers) play a passive role in this interaction and are frequently stateless. Updates to resources are relatively rare (with respect to reads) and generally carried out through the use of tools which are independent of the Web's protocols, e.g., a document editor. Web-based caching systems such as the Netscape proxy server [1], Harvest [2], Squid [3], the CERN httpd [4] have played a valuable role in reducing access latency, decreasing bandwidth requirements and distributing load in this environment.

The introduction of effective Web-based payment protocols and systems [5] enables the deployment of applications with more complex modes of operation. Because they have the opportunity to recoup their costs, service providers are increasingly willing to carry out complex processing and maintain long-term interactions (sessions) with clients. For example instead of simply serving data when requested, service providers can sell subscriptions and deliver specific up-to-date information to the client on a regular basis, or as the information changes. Applications which allow clients to have write access to data blur the distinction between information providers and consumers, enabling cooperating groups of users to work collectively on shared information. These more complex applications require servers to maintain client and session state and to initiate communication with clients to inform them of changes in service conditions, e.g., updates to shared information.

Experience in multiprocessors [6], distributed file systems [7, 8, 9] and distributed object systems [10] has shown that caching offers significant benefits for applications involved in shared read/write access to data and this seems likely to be true for the Web also. Existing caching mechanisms will undoubtedly continue to remain useful for access to traditional resources but, as we shall show, they will prove inadequate for some of the class of applications described above.

In this document we shall describe an approach to caching which we call *open caching*. This approach allows application programmers to define their own caching protocols and for clients of an application to drive that protocol at run-time to optimise performance. Our design is independent of the existing Web infrastructure offering additional caching choices which can be utilised alongside traditional browser and

proxy-server caching. The implementation we propose uses standard browsers and standard (modifiable) servers and has a minimal effect on access to resources which do not require our mechanisms.
We shall begin with a description of a general caching model, used to explain our ideas, and a discussion of existing caching mechanisms, outlining their deficiencies. This is followed by brief presentation of the W3Objects project [11] and a description of how flexible caching can be introduced using W3Objects. We give a number of examples of how caching may be used to provide significant performance advantages for specific applications.

2. A Caching Model

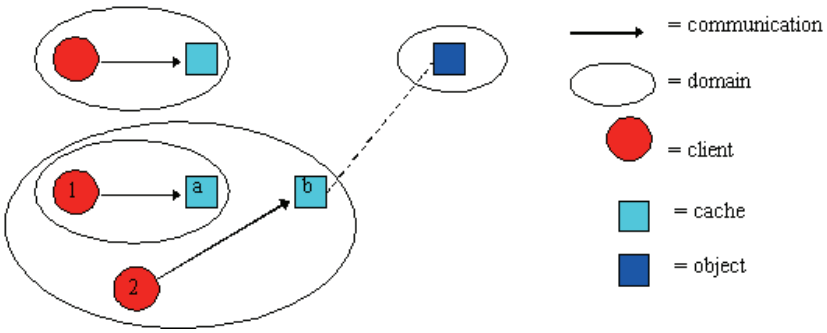

Fig. 1. The Caching Model

Our general caching model (which is not specific to W3Objects) has four components, as illustrated in figure 1. *Clients* are active entities who wish to communicate with objects. An *object* is an entity which encapsulates both state and operations (or methods) which operate upon that state. (A Web resource for example is an object encapsulating the state of the resource and the HTTP methods HEAD, GET, POST etc.). For simplicity we assume that any client may communicate with any object. *Domains* represent locality in the model and may contain clients, objects and other domains. Examples of domains include address spaces, computing nodes, LANs etc. Clients communicating with an object, i.e., invoking operations upon that object, incur communication costs whenever domain boundaries are crossed. *Caches* represent some object such that a client may invoke some subset of the object's operations upon the cache. The association, or *binding*, between cache and object is shown by the dotted lines in the diagram. Clients can decrease communication overheads by communicating with a cache whenever the cache is 'closer' than the object it represents.
Of course without some guaranteed consistency view the results returned from the cache may not be those expected. Maintaining a consistent view onto the object via the cache requires a consistency protocol involving, some or all of, the client, the cache and the real object. Consistency views range through a spectrum from strictly consistent, in which all clients see all reads and updates on the object in precisely the same order, to weakly consistent, in which the cache may be stale up to some acceptable time or consistent only after specific synchronisation actions. Each protocol can be represented within the model by the definition of a set of messages and rules defining the exchange of those messages between client, cache and object. For example one (simplified) protocol which can offer the opportunity for clients to have a strict consistent view onto read-only caches, states that i) a newly created cache sends a message to its object indicating its creation, ii) an object responds to such a message with a message containing the object's state, iii) after every update operation the object sends a message containing the object's state to every cache and iv) on deletion a cache sends a message indicating that it has been deleted.
Within the model clients can dynamically create and destroy both objects and caches, and, as caches are themselves objects, clients can create caches of caches. This is illustrated above where cache *a* is associated with cache *b*, which is associated with the real object. The model allows clients to create (and remove) arbitrary tree structures in which caches are the nodes and the real object is the root. As clients may communicate with any object, and as caches are objects, clients may share access to caches, as shown above where clients *1* and *2* share access to cache *b*. Sharing caches can be effective if clients share some working set of objects.

We shall use this model to highlight some of the deficiencies in the current Web caching mechanisms with respect to complex interactions. Although each deficiency could be catered for by extensions to HTTP or through the use of existing CGI technology, we shall show that the use of object-oriented technology, and our specific W3Object implementation of caching mechanisms, offers a means by which an application programmer can create protocols designed to provide optimal caching performance without impacting on the existing Web infrastructure.

3. Caching in the Current Web

Caching is achieved in the current Web through browser or proxy-server caching [2, 3, 4]. (In our discussion we shall ignore virtual memory caching, in which browsers or servers cache resource in main memory rather than on secondary storage). Browser caching involves configuring the browser with a cache store. On processing an HTTP GET or HEAD request the browser checks if the resource, identified by its URL, is already within the store and if so accesses the cached copy (validating it if necessary). If the resource is not cached the GET or HEAD is performed as normal and, as a result, a cached copy of the resource is placed in the cache store.

Proxy-server caching is achieved by configuring the browser to indirect all HTTP requests via a proxy-server. The proxy-server maintains its own cache store and handles requests as would a browser in the previous example. However, proxy-servers may be configured with knowledge of other parent, or peer, proxy-servers and, whenever they experience a cache 'miss', i.e., the requested resource is not cached, can interrogate those servers for the resource. Browsers and proxy-servers typically use a time-to-live (TTL) value to ascertain the validity of each cache.

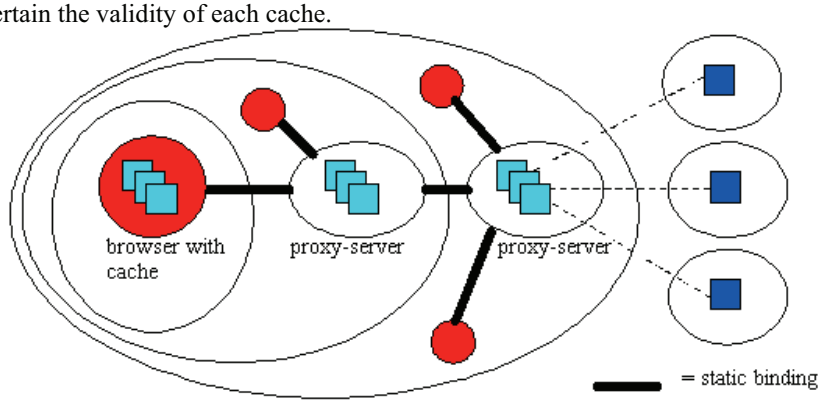

Fig. 2. Existing mechanisms

Browser and proxy-server caches are illustrated in figure 2 using our caching model. In this example all HTTP GET or HEAD requests originating from the client (browser) on the left are indirected via the browser's cache store and then two levels of proxy-server. Other clients who do not have browser caching configured but who share the proxy-servers are shown. The bindings between browsers and proxy-servers are static in that changes require reconfiguration by a responsible party. These caching mechanisms have proved effective in improving performance in the largely static, read-only environment.

The cache mechanisms in the current Web are largely invisible to clients with the advantage that clients are protected from complexity and from potential abuse or misuse of the system. However this approach does tend to offer "vanilla" caching for all clients and their resources, i.e., browsers use the same cache path (the path from client to object) for all resources and the same consistency protocol is provided for every resource. (Clients do get limited control in that they can specify, within each HTTP request, whether caching should be used and what is an acceptable age for a cache).

This use of caching assumes that the optimal cache path and cache consistency requirements are essentially identical for all resources. As an example of how cache path requirements might differ in the Web, imagine a broker who works in a department which runs a proxy-server providing a high cache hit-rate for resources encapsulating commodity prices (the business of the department). Unfortunately the proxy-server provides a low hit rate for resources encapsulating electronics industry news (the broker's specialty). However the broker does know of another (nearby) proxy-server which does offer a high hit-rate in this area and persuades the departmental administration to reconfigure their proxy-server with this

other proxy-server as a parent. The broker now finds significant improvements in the cache performance for resources involving electronics news but, unfortunately, for both himself and all of his departmental colleagues, every cache miss for commodity price resources now involves overhead (and possible delay) due to communication with the newly configured parent. Ideally the broker would like to define his own cache paths on the basis of individual resources. Additionally, although the broker is satisfied with weak consistency for some resources he requires strict consistency for others. Therefore, as this example shows, for certain applications allowing clients to specify the cache path and consistency requirements on an individual resource is necessary.

As well as this general criticism, the current cache consistency protocols in the Web rely heavily upon HTTP and are thereby restricted in two important aspects. Firstly the HTTP definition offers no support for writeable caches, yet the use of such caches can provide significant benefits for certain applications. For example, within a Web-based co-authoring service, which allows a single author to update some section of a document whilst others continue to read remaining sections. The update process in this application may consist of a large number of update operations and, without caching, every operation occurs upon the real document. This may impose a repeated communication overhead if the document is remote from the author, despite the fact that as long as the author is working these changes are not available to other readers. With support for writeable caches (and the appropriate concurrency-control) the author could work on a local cache, performing multiple updates over a period of time, and perform a single operation to writeback the cache to the real document when he has completed his task. Application which offer shared data access and which may therefore benefit from writeable caches are expected to grow in number.

Secondly, HTTP only supports polling-based validation and does not allow resource-driven invalidation of caches using callback mechanisms. That is, to ensure that a cache is consistent the cache holder must poll the resource server. This is especially inefficient in the case where consistency is important, and updates are infrequent but cannot be predicted. For example, consider a service offering up-to-date, but infrequently changing, commodity prices which clients require to be correct. Caching may be beneficial to avoid communication overheads when frequent reads are occurring. However, this requires either frequent validation or on demand validation - both can generate considerable, often unnecessary, network traffic and the latter reduces much of the latency gains offered by caching. The viable alternative in such circumstances is resource-driven invalidation where the server invokes a callback on the cache to inform it whenever an update has occurred. Although this solution involves the server maintaining knowledge of its caches there will be applications which are willing to accept these memory costs in preference to the communication costs of polling-based invalidation.

The deficiencies described above show that the current caching mechanisms will not be sufficient for all applications. Instead application-specific consistency protocols are required. Clients of applications require flexibility in how they interact with an underlying caching system in order that they can optimise performance with regard to individual resources. Support for a wider variety of protocols could be provided by extending HTTP horizontally, e.g., by adding resource-driven invalidation of caches to the protocol but, as we have argued previously [11], improved functionality can better be introduced though the use of object-oriented technology. In the next three sections we shall describe first an overview of the W3Object technology and then our implementation of open caching within the W3Object project.

4. W3Objects overview

Many researchers have investigated the application of the object-oriented paradigm, e.g., [12, 13], as the means of introducing new functionality to the Web. It is hoped that by working within this paradigm new services can be developed quickly, be capable of modification, and be structured so as to utilise reusable components (including existing browsers and servers). We have applied our experience in both large-scale distributed systems and object-oriented technology [14, 15] within our work on W3Objects [11, 16]. Our approach is evolutionary rather than revolutionary with the provision of a parallel system supporting distributed shared objects which is largely orthogonal to the existing Web infrastructure. By this means existing browsers, servers and resources may continue to operate as before, whilst allowing the gradual introduction of new objects and object wrappers for existing resources. We begin with a brief description of the basic architecture before describing how our system is integrated into the Web.

Each W3Object (henceforth also just 'object') is an instance of some programmer-defined class. Objects encapsulate data and each supports a number of operations, defined within their class, through which the data may be accessed. Every object exists within some W3OServer (henceforth also just 'server'), with a name which is unique within that server. Each server offers a remote procedure call (RPC) endpoint, with a unique network address, via which its objects may be accessed. Each object can therefore be uniquely identified using the network address of their server and their name within that server. Every object offers management operations, e.g., objects may be migrated between servers. Client programs (and objects) may initiate servers, create objects, and invoke operations on objects using RPC. By this means a network of communicating W3OServers, each holding W3Objects, may be instantiated and managed. The run-time system supports very lightweight mechanisms for naming and locating objects and offers referential integrity guarantees, i.e., objects continue to exist whilst referenced, but are garbage collected when no longer referenced.
How then is a W3Object network integrated into the Web ? Objects may be named with standard URLs of the form :

- http://<server name>/W3Object/<object name><operation details>

We use extensible Web servers to catch requests upon objects and deliver them to the relevant W3OServer. (In our current implementation this is achieved by adding a W3Object module to an Apache http daemon [17]). Every such http daemon has an associated W3OServer and, on detecting a W3Object request, a message requesting the relevant operation (as defined by the operation details within the URL) is delivered to the named object within that W3OServer. All W3Objects provide a HTTP interface via which they may serve HTTP operations. The operations may be specified on a per object class basis so that, for example, a GET on an object can dynamically generate appropriate HTML for the response. Typically, a client performing an HTTP GET from a browser upon an object may receive in response a form offering access to other operations supported by the object. By this means any operation supported by an object may be presented to the Web client.

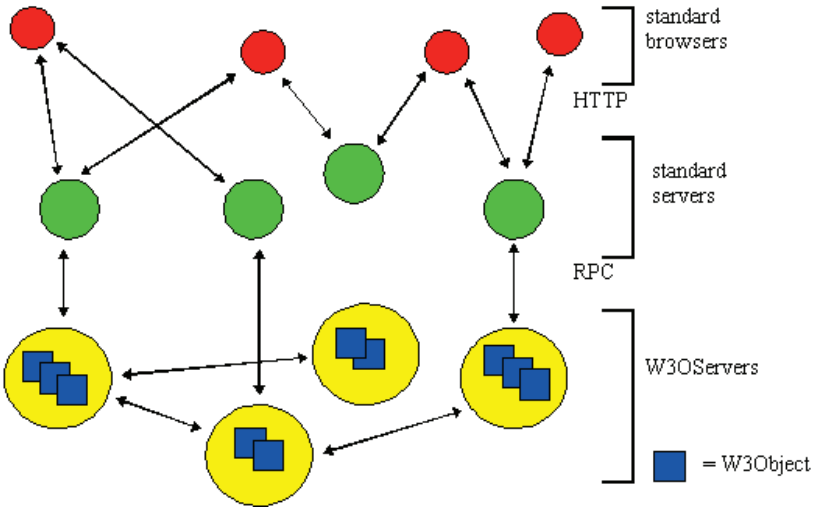

Fig. 3. W3Objects Architecture

Our overall architecture is illustrated in figure 3. Note that the Web is unaffected by the use of W3Objects in its normal non-W3Object operation. Communication between browsers and servers continues to use standard HTTP, whilst communication between servers and W3Objects (and between W3Objects) uses RPC.
The data encapsulated within an object may be in the form of a standard Web resource, e.g., an HTML file, gif etc. so, as all objects support the HTTP interface, objects may act as object wrappers around such resources. The object wrapper may however enhance standard resources with a richer interface offering improved management of the resource or additional operations upon the resource state. Alternatively an object might encapsulate some existing, or entirely new service.

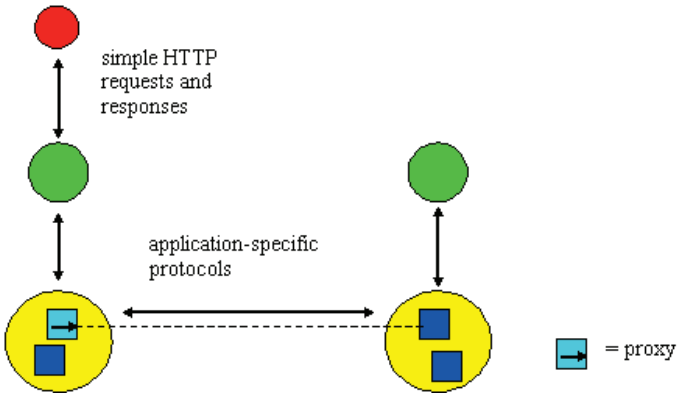

Fig. 4. Application-specific Protocols

W3OServers are themselves presented to clients as W3Objects enabling clients to perform management operations and to browse the server, i.e., ascertain and communicate with the objects it contains. Clients may also invoke operations upon servers in order to explicitly create (and delete) *proxies* bound to some remote object. The proxy represents the object and, in its normal mode, operations performed upon the proxy are redirected to the remote object. As illustrated in figure 4, proxies can engage in protocols with the remote object, with both proxy and object capable of holding state regarding the status of a session. For example all proxies exchange messages with the real object in order to ensure the referential integrity guarantees mentioned previously. By developing suitable objects and corresponding proxies, clients may communicate with proxies (via HTTP) in order to drive whatever protocols are required.

The W3Objects class library provides a set of off-the-shelf classes which support single-writer/multiple-reader concurrency-control, state-based checkpointing, crash-recovery etc. all of which utilise the ability of proxy and object to maintain state and to communicate together to implement their functionality. Programmers can obtain this pre-defined functionality for their object classes through the use of inheritance. Programmers can also define their own classes which are then also available for reuse.

5. Caching using W3Objects

The need for application-specific caching, as described in the section 'Caching in the Current Web', can be satisfied through the provision of *open caching*, i.e., highly visible caching mechanisms. A client may access these mechanisms from a standard browser at run-time to explicitly configure arbitrarily complex caching hierarchies and to drive a dynamic caching protocol best suited to their changing requirements of *individual resources*. The mechanisms operate independently of existing browser and proxy-server caching. Therefore clients may choose to use our explicit mechanisms for a subset of resources with particular caching requirements whilst continuing to use traditional Web caching for other resources. Although it is possible for clients to be in complete control of caching (if such a high degree of flexibility is required) W3Object programmers can choose to hide caching details and offer an appropriately simple interface thereby protecting clients from the dangers of protocol misuse.

As part of the W3Objects library, we have developed a *Cacheable class*. The class offers a small set of primitive caching operations which may be performed upon a *Cacheable object*, i.e., any object derived from the Cacheable class. These operations, in conjunction with appropriate concurrency-control, constitute the basic building blocks used by a client to explicitly specify whichever caching protocol is required. Invoking the basic operations in different sequences provides different protocols. One of these basic operations enables the proxies of Cacheable objects to obtain the state of their remote object. Client operations performed upon the proxy may then occur locally, i.e., the proxy can act as a cache. Another operation enables the proxy to write back the state to the object.

5.1 Single-level Caching

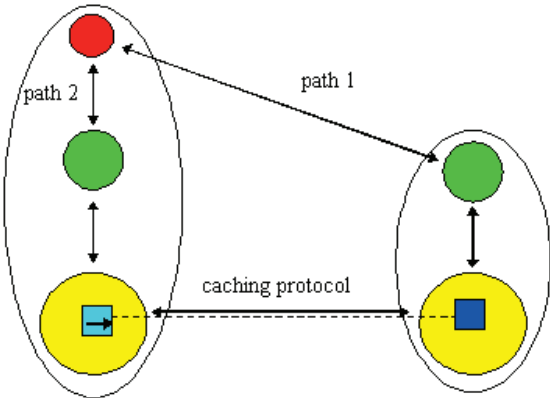

Fig. 5. Single-level Caching

We shall now illustrate two, of very many possible, caching protocol implementations which involve single-level caching (the next section covers multi-level caching). The first of these provides writeable caches for a the simple co-authoring application presented earlier. The class utilises the single-writer/multiple-reader concurrency class provided by the W3Objects class library. The application programmer implements a class which can support writeable caches by inheriting from this class and the Cacheable class. Figure 5 shows an object (an instance of the co-authoring class representing a document) which resides in the W3OServer on the right. The client may perform operations on the object, identified by its URL, by delivering messages, via HTTP, to the server and hence to the object (path 1). However, when the object lives in a different domain from the client, the application allows a client who anticipates a significant number of sequential write operations, to create a local writeable cache. To do so the client requests that a local W3OServer create a proxy bound to the remote object. The proxy has its own URL which the client may now use to invoke operations directly upon the proxy. Our client requests that the proxy obtain a write lock from the remote object and, if the lock is granted, a cache of the object's state. Subsequent operations invoked upon the proxy are performed locally, i.e., upon the cache (path 2). Whenever the client has finished his updates he requests that the proxy writes back the cache and releases the write lock. Finally, the client requests that the server discard the proxy.

Our second example illustrates a protocol utilising callback-based resource-invalidation which uses two of our other Cacheable operations - one which allows a client to register a callback for the purposes of resource-invalidation and another by which clients can explicitly increment a version number held by an object. An extension to the application described above allows clients to continue to read stale copies of the document whilst it is in the process of being updated, but refreshes those caches after the updates are complete. To create a read-only cache which is kept consistent in this manner the client requests, as before, that a local server create a proxy bound to the remote object. The client then requests that the proxy register a callback with the object and then obtain a copy of the object's state (correct as of the time of the last synchronisation event i.e. write lock set or release). Our client may then invoke read-only operations upon the proxy, all of which occur locally. Whenever some other client updates the object (which automatically increments the version number) the object invokes a callback to invalidate each of its caches. The next operation invoked upon the proxy refreshes the now invalid cache by automatically obtaining the new state of the object. Whenever the client has finished with his cache he requests that the proxy deregister the callback. Finally, the client requests that the local server discard the proxy.

In the examples above the implementation requires the clients to explicitly drive every step of the protocol from the browser. However, as we said earlier the protocol details can be hidden from the client by the application programmer. For example, the first access to the application could return a Java applet responsible for automatically managing caches. Alternatively, the object could offer simpler caching operations than the ones described here which hide complexity, e.g., in the first example the two operations *create_write_cache* and *remove_cache* might be sufficient.

As well as allowing clients to drive cache refreshes by explicitly requesting the proxy obtain a new copy of the state, another Cacheable operation allows clients to specify an acceptable age for the cache. When an

invocation is handled by a cache a refresh of the cache occurs automatically if the cache is 'older' than the acceptable age. (This is similar to the use of the *max_age* field in HTTP).

5.2 Multi-level Caching

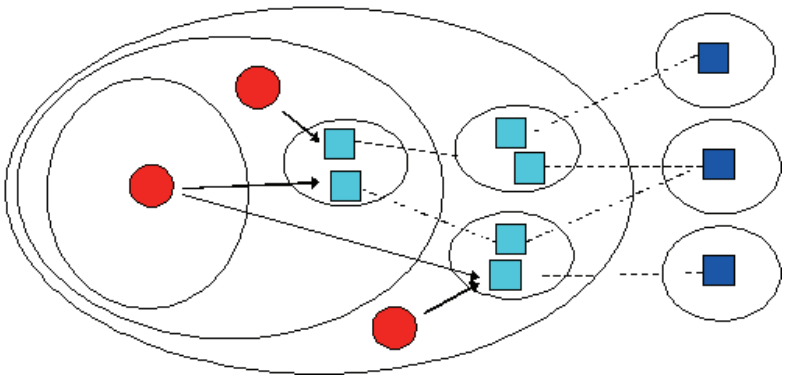

Fig. 6. Multi-level Caching

As we have already described clients may create, or access, W3OServers within the network. Clients may also create, or access, proxies within those servers which are bound to an object. Servers may be created to act as cache servers and used by cooperating clients as a shared cache store. As caches may be bound to caches, many levels of cache may be constructed into complex hierarchical trees.

An example of multi-level caching is illustrated in figure 6. Note that in this example, as compared to figure 6 (a similar diagram illustrating existing Web-based caching structures) clients do not have to be statically bound to a particular proxy server in order to obtain caching. Clients may access a number of different cache stores. Also different cache paths may be set up on an individual object basis for use by individual clients and groups of clients. Different cache protocols may be running on each path offering different consistency guarantees. We believe this flexibility will prove useful for applications where clients wish to retain control over caching of particular resources and where groups of clients share similar requirements.

For example let us return to the broker described in an earlier example. The broker is a subscriber to a real-time newspaper where different sections of the newspaper are represented by objects updated independently as the news changes. Many of these sections are of little interest to the broker and so he makes no caching decisions, being happy to communicate with the application remotely on the rare occasions when he does require access. However he does explicitly cache the main news, politics and sports pages within his personal cache server and sets the policy to ensure that each page is at most four hours old. He also caches the financial page which he accesses frequently but, in this case, he sets a policy utilising resource-invalidation so that when the financial news changes, at any time during the day, his cache is automatically updated.

Now let us examine the broker within his social context. Unsurprisingly, the broker shares similar interests to his colleagues and there are therefore benefits to be obtained from utilising a shared cache within his department. The shared cache server (which could be offered by the application) holds the main news and sports sections of the paper as these are the most popular within the department and the user can set up his individual caches for those pages to be bound to the relevant sections cached there. The fact that many of the broker's colleagues are accessing the same caches means that our broker is more likely to receive cache hits. Similarly the broker could bind other of his caches to other cache servers at disparate geographic locations.

6. Current Status and Future Work

The (non-caching) W3Objects system has been implemented using modified Apache http daemons, as described previously. Clients who act as object administrators are provided with a W3Object's interface with which they can manage their objects - migrating them as required and offering referential integrity guarantees to other clients. The W3Object system is built using the *Shadows* [15] distributed shared object system, where W3Objects are Web-flavoured Shadow's objects. The caching described in this paper has

been fully integrated into Shadows so that clients may create their own object-based multi-level caching hierarchies and define and control their own caching policies.
We are now in the process of developing W3Object applications which will utilise open caching. Our first application will be multi-level shared hotlists. The application will allow groups of Web users to create shared hotlists which can be updated with new entries and browsed to examine new entries (hopefully of common interest to the group) inserted by other clients. Individual entries in the hotlist represent caches of the resource and each cache may be bound to the remote object or to entries in other hotlist servers. Individual clients may also have their own (private) hotlist servers and may specify the consistency requirements of each entry. Holding an entry in a hotlist also has the benefit that it offers referential integrity guarantees, i.e., the object referred to will continue to exist whilst the entry exists. Hotlists will also help support disconnected operation (whether due to the use of a mobile computer or due to network partition) as placing an entry in the list guarantees the existence of a local cache.
Our work so far has concentrated on caching opacity, i.e., clients make explicit decisions to cache particular objects and access those caches using different URLs from those of the real objects. We also wish to examine transparent caching for W3Objects using our mechanisms. A modified proxy-server could catch all W3Object requests and forward them to its W3OServer. This server could check if the required object was already cached and if so deliver the request to it. If the object was not cached the server could create the cache, e.g., using the mechanism described above. The advantage of this form of caching over traditional proxy-server caching is that clients may browse their cache server and specify caching policy on a per object basis.

7. Conclusions

In this paper we have presented a means of providing open caching for Web clients. Clients may make caching decisions for individual resources and may explicitly drive the consistency protocol for their cache in order to obtain optimal performance. Cache hierarchies may be created and used by groups of clients to obtain higher performance and other benefits of sharing.
Our mechanisms offer a very high degree of flexibility over caching at the expense of additional complexity as perceived by a Web client and/or application programmer. However, users are free to make decisions about whether to accept this tradeoff on a per resource basis and choosing to do so has a minimal performance effect on access to all other resources. The implementation presented here uses standard browsers and standard modifiable servers. We believe the flexibility our solution offers coupled with its low-impact upon the existing Web-infrastructure makes our approach a promising one.

Acknowledgments

The work reported here has been partially funded by grants from the Engineering and Physical Sciences Research Council (EPSRC) (Grant Number GR/K34863), Hewlett-Packard Laboratories and GEC-Plessey Telecommunications.

References

[1] Netscape Communications Corporation "Netscape proxy server manual".
http://home.netscape.com/comprod/proxy_server.html

[2] A.Chankhunthod, et al. "A Hierarchial Internet Object Cache" University of Colorado, Boulder.
ftp://ftp.cs.colorado.edu/pub/techreports/schwartz/HarvestCache.ps.Z

[3] Squid Internet Object Cache.
http://squid.nlanr.net/Squid/

[4] A. Luotonen, et al., "CERN HTTPD public domain full-featured hypertext/proxy server with caching", 1994.

http://www.w3.org/pub/WWW/Daemon/Status.html

[5] Mastercard and Visa, "Secure Electronic Transaction (SET) Specification, Book 1: Business Specification", June 1996.
http://www.visa.com/sf/set/SETBUS.PDF

[6] J. Archibald, et al., "Cache Coherency Protocols : Evaluation using a Multiprocessor Simulation Model", ACM Trans. on Computer Systems, vol. 4, pp. 273-298, Nov. 1986.
[7] J. Howard et al. Scale and Performance in a Distributed File System. ACM TOCS, Feb. '88.
[8] R. Sandberg et. al. Design and Implementation of the Sun Network Filesystem. In Proceedings of the USENIX 1985 Summer Conference.
[9] M. Nelson et al. "Caching in the Sprite File System". ACM Transactions on Computing Systems, Feb. '88.
[10] M. Nelson, et al. "Caching in an Object-Oriented System" Proceedings of the 3rd Workshop on Object Orientation in Operating Systems, Ashville, North Carloina, December December 1993.
[11] D. B. Ingham, M. C. Little, S. J. Caughey, S. K. Shrivastava, "W3Objects: Bringing Object-Oriented Technology To The Web," The Web Journal, 1(1), pp. 89-105, Proceedings of the 4th International World Wide Web Conference, Boston, USA, December 1995.
http://www.w3.org/pub/Conferences/WWW4/Papers2/141/

[12] N. Edwards, et al., "A Web of Distributed Objects", The Web Journal, 1(1), Proceedings of the 4th International World Wide Web Conference, Boston, USA, December 1995.
http://www.w3.org/pub/Conferences/WWW4/Papers/85/

[13] P. Merle, et al., "CorbaWeb: A Generic Object Navigator", Proceedings of the 5th International World Wide Web Conference, Paris, France, May 1996.
http://www5conf.inria.fr/fich_html/papers/P33/Overview.html

[14] G. Parrington, et al., The Design and Implementation of Arjuna, USENIX Computing Systems Journal, Vol. 8, No. 3, Summer 1995, pp.253-306.
http://arjuna.ncl.ac.uk/arjuna/papers/designimplearjuna.ps

[15] S. Caughey, et al., SHADOWS: A Flexible Support System for Objects in a Distributed System, Proceedings of the 3rd IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS), Ashville, North Carolina, USA, December 1993.
http://arjuna.ncl.ac.uk/arjuna/papers/shadows-iwooos93.ps

[16] D. B. Ingham, S. J. Caughey, and M. C. Little, "Fixing the Broken-Link Problem: The W3Objects Approach," Computer Networks and ISDN Systems, 28(7-11), pp. 1255-1268, Proceedings of the 5th International World Wide Web Conference, Paris, France, May 1996.
http://www5conf.inria.fr/fich_html/papers/P32/Overview.html

[17] The Apache HTTP Server Project.
http://www.apache.org/