Caching Documents with Active Properties

Eyal de Lara[*], Karin Petersen, Douglas B. Terry, Anthony LaMarca, Jim Thornton, Mike Salisbury, Paul Dourish, Keith Edwards, and John Lamping

Computer Science Laboratory
Xerox Palo Alto Research Center

Abstract

Caching in the Placeless Documents system poses new challenges because users can attach active properties to documents. Active properties can modify the document's content as seen by a user. Thus, the caching mechanisms must take into account that a document's content not only depends on when the document was last modified, but also on the set of personal and universal properties attached to the document and the information on which these properties depend. Interestingly, active properties can be used to help caches manage their contents by notifying them of events that affect cache consistency, by providing caches with document-specific verifiers to further check on a document's consistency, and by returning information that can aid in decisions of which documents to cache.

1. Introduction

The Placeless Documents project at Xerox has designed and implemented a document management system based on personalized document properties. In this system, document properties are statements about the context of a document or the intended behavior for the document. Sample properties are "keep at home and the office", "translate to French", or "budget related". Properties can be static labels like "budget related", or active objects that implement a desired behavior, like replicating a document between a user's home and his office. Document properties are said to be personalized because they are managed on a per-user basis. The scope of a property applies to a document within a document space that can be owned by an individual or a group of people. This scoping rule allows different users to assign different meanings or behaviors to the same document. Furthermore, properties in the Placeless Documents system can be attached to documents originating from arbitrary content sources: file systems, the World Wide Web (WWW), e-mail servers, document management systems (DMS), live video feeds, etc. Personalized properties can thus be attached to a wide range of documents and be used as a uniform mechanism to interact with all these documents.

In this paper we focus on how active properties affect caching of document content in the Placeless Documents system. Because active properties provide a mechanism to add new behavior to a document, they can change the actual content delivered to the user or an application. For example, the "translate to French" property can return an English document in French. Similarly, a "summary" property may return a condensed version of the document instead of its original in full length. Furthermore, since users can personalize their document use by attaching different active properties to a document, caching the content for these users may mean that different versions of the content need to be cached. For example, one user may attach the French translation property to a document. Other users may retrieve it in the original language or execute a different transformation, like summarization. Active properties therefore affect how much sharing of cached content can occur. Active properties also affect cache consistency. The cached content of a Placeless document depends not only on its original content, but also on the transformations applied by active properties. Thus changes to the type, number or order of the properties attached to a document, changes in the information these properties depend on, as well as changes to the original content of the document can render the cached content for the document out-of-date.

The rest of the paper describes the issues and opportunities of active properties for caching in the Placeless Documents system. In particular, we show how active properties themselves can be used to implement custom per-document caching policies. To ground the discussion we first describe the Placeless Documents design in more detail. To close we report preliminary results obtained from a prototype implementation.

2. Placeless Documents Design

The Placeless Documents system is designed to provide an uniform, individualized, property-based interface to arbitrary sets of documents. The design is based on the philosophy that documents of interest come from many different sources and often have many more consumers than authors. The system therefore needs to

[*] Department of Electrical and Computer Engineering, Rice University

accommodate the individual needs of multiple users. To this end, the system supports two types of document objects as illustrated in Figure 1: *base documents* and *document references.*

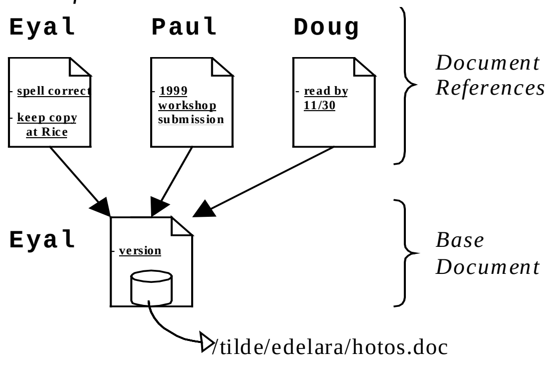

Figure 1. Properties are attached to base documents and document references

A base document is the link to the actual content of the document and is generally owned by either the author of the content or the person or group that imported the document into the local environment. A document reference points to the base document. Each user of the document owns a separate document reference. Both base documents and document references can contain properties, and we deem these properties to be "attached" to documents. Properties attached to the base document are called universal because they are seen by all users with a reference to it. Properties attached to a reference are called personal properties, and are seen only by the individual owner of the document reference.

In Figure 1, Eyal owns the base document since he created the draft of the HotOS paper. A special active property on the base document, called the bit-provider (shown as a disk in the figure), is responsible for retrieving the actual content from its repository. Eyal also attached an universal property to the base that saves an old version of the paper each time someone opens it for writing. Eyal, Paul and Doug personalize their interactions with the paper through personal properties attached in their references. One of Eyal's personal properties maintains a copy of the content both at PARC and at Rice, whereas one of Doug's properties indicates "Read by November 30th". Because Eyal is not a native English speaker, he also attaches a personal property that corrects the paper's spelling. Paul is less involved in writing the paper and only marks it as a "1999 workshop submission". All three users see the versioning information resulting from the universal property on the base document, but nobody else needs to know that Eyal keeps two copies of the document, nor do Eyal and Paul know that Doug plans to review the paper by November 30th.

As demonstrated by the example, properties can be either active (replication between PARC and Rice) or static ("1999 workshop submission"). Static properties tend to be statements about the context of a document, while active properties serve to provide extended functionality for the document. Active properties are event driven. The properties register for events that can occur on a document, such as getInputStream, getOutputStream[1], modify property, set property, timer, etc. When an event occurs, all registered properties on that document are invoked. In the example, both the spelling correction and the versioning properties are dispatched when getOutputStream operations are invoked, whereas the spelling corrector is also invoked on getInputStream. The replication property is invoked only as a result of timer events, assuming that Eyal's replication between PARC and Rice occurs only once at the end of the day.

Caching of document content is mostly affected by active properties that transform the input or output streams. Thus, to further understand how document content flows between applications and the storage systems through the Placeless Documents middleware, consider the previous example which has been expanded in Figure 2. Imagine that Eyal is editing the paper-draft from MS-Word. When Word issues the save/write request, it results[2] in a getOutputStream call on Eyal's reference to the paper draft. This call is forwarded from the reference to the base document, which in turn invokes the call on the bit-provider[3]. The bit-provider, in this case an NFS client, opens the corresponding file for writing and returns the handle to the base document. At the base document all attached active properties interested in the getOutputStream operation get dispatched for execution. There is only one such property, which creates a new version of the content by generating a copy of the existing document and adding a new static property to the base with a link to that copy. The base document then returns the output-stream handle to the document reference. The reference dispatches all its active properties interested in the getOutputStream operation, which in this case means that it invokes the spelling corrector. Finally the reference returns an output-stream back to the application.

For an active property like the spelling corrector to intercept and actually modify the content of the write operations, the property creates a new custom output-stream as it processes the event triggered by the getOutputStream operation. This custom stream implements the transformations required by the property,

[1] The document content I/O model in Placeless is based on Java Input and Output streams.

[2] Read and write operations from off-the-shelf applications are translated into Placeless I/O operations by a NFS server layer. Newly developed applications invoke the Placeless API directly.

[3] The API actually does not contain calls directly on document references or base documents, but instead on *document spaces*, which are the system components that manage base documents and document references on a per-user basis. For simplicity of the description, however, we gloss over these details in the text.

for example, it corrects the spelling of the document being written. When a write operation is invoked by the application the active property can operate on the content by virtue of the custom output-stream. In fact, the active property hands the custom stream to the next property in the calling chain for the getOutputStream event, or if it is the last to the application. Basically, active properties that modify the document content create a chain of custom output-streams that will each operate subsequently on the content that is being written. Similarly, active properties that modify the content on read operations create chains of custom input-streams.

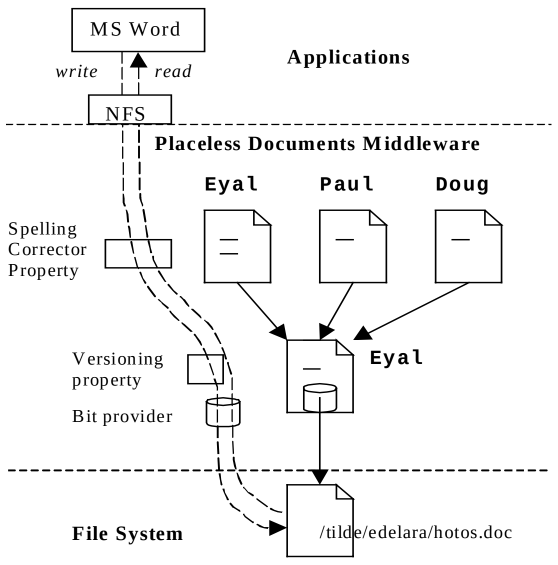

Figure 2. Read and write path through the active property mechanism.

Properties are on the read path of a document when they express interest in the getInputStream operation and interpose their own custom input-stream to intercept all read operations. Similarly, properties are on the write path if they interpose a custom output-stream to intercept all write operations for a document. The execution of custom input stream functionality on the read path occurs first at the base document and then at the document reference. Inversely, custom output-streams on the write path are first executed at the document reference and then at the base document. Remember that the read and write paths for different users have different document reference components, but share the base document. With this background on the Placeless system architecture, we now discuss how active properties affect document content caching.

3. Document Content Caching

Caching document content in the Placeless Documents system is important for several reasons. Document access latencies are affected by the interposition of active property execution. Document accesses also require content to be sent from the storage repository to at least one, possibly two, Placeless servers – one for the base and possibly another one for the reference, which increases network traffic and execution time at each of the servers. Finally, properties may be used to state Quality-of-Service (QOS) requirements such as “access time < .25 seconds”, which in turn can benefit from caching.

The novel features of Placeless Documents that affect the design of its caching architecture are:

1. Custom active properties can modify the content of a document on a per-user basis, which may require multiple versions of the same document to be cached.
2. The properties attached to a document and the order in which they execute can change the resulting document content. Cache consistency hence depends not only on update operations on the content, but also on transformations by properties attached to the document.
3. Documents originate from any number of repositories, many of which provide different mechanisms to handle cache consistency.

The next subsections describe each of these issues in more detail. First we focus on the issues related to cache consistency. Second we address the issues of how to determine which document content can be cached, how to manage the cached content, and how to handle cache replacement.

Cache Consistency

The content of a Placeless document seen by an application depends on the original content stored at a repository (WWW, file system, e-mail server, video camera, etc.) and the transformations applied by the active properties on the read path for a particular user. Content cached after the transformations applied by active properties can therefore become invalid in four ways:

1. The original source is modified. Source modifications occur either through the Placeless system or by changes that are not within Placeless control. When an application, like MS-Word, updates content through the Placeless system, as in the example above, the system can snoop on all update operations. On the other hand, source modifications at the original site, like updates to pages at a web-site or applications interacting with files directly through a file system, do not allow the Placeless system to track these changes. This dual update model can already be found in the WWW, where an HTTP PUT operation can modify a page, but pages can also be updated without involvement of the HTTP server that services them to clients. Because web-servers so far manage consistency only based on a time-to-live (TTL) invalidation scheme, they do no need to handle these two cases differently. The

Placeless system, however, would like to support a cache management mechanism that accommodates both update models. Managing consistency with respect to original sources is further complicated in the Placeless system, because the consistency mechanisms used by the original repositories can vary dramatically.

2. **Active properties are added, deleted or modified**. For example, when a language translation property is added to a document, the cached content in a different language is no longer valid.

3. **The order of the active properties changes**. For example, the result of applying a spell checking property to a document varies whether it is applied before or after a language translation property.

4. **Information used by active properties changes**. Active properties may rely on information that is completely external to the Placeless system, for example current time, data stored in databases and other on-line sources, or internal to the system, such as values of other document properties like "preferredLanguage=Spanish". Again, some of these issues appear in existing systems. On-line, web-based, active information services, like financial portfolio tracking and travel status, update their web-pages when the underlying sources for the information change: stock market, SABRE database, etc. In the WWW, the most common solution to this problem is to make these pages uncacheable. In Placeless Documents, all documents can be customized through active properties and hence the issue exists at a much larger scale and needs a better solution.

In summary, the validity of cached content depends on operations under the Placeless system control and operations outside its control. Under Placeless control are content updates through the Placeless system, property modifications, and changes to the ordering of properties. Outside of Placeless control are content modifications at the source repository of the document and changes to external information that the active properties depend on.

Notifiers and Verifiers

To handle these different causes of cached content invalidation, we have experimented with two mechanisms: *notifiers* and *verifiers*. Notifiers are active properties themselves that are used to invalidate cache entries resulting from changes through the Placeless system. Notifiers send a notification to each of the affected caches to invalidate the corresponding entries. Notifiers are similar in nature to file-system update callbacks [4], although, as described next, they can be extended by active properties to provide property-specific notifications. Notifiers are more closely related to semantic validators and callbacks [5], in that semantic callbacks are triggered only if some predicate is satisfied. Notifiers, in fact, integrate the notion of semantic validators and callbacks into one mechanism. On the other hand, verifiers are pieces of code returned to the cache along with the document's content. They are executed each time an entry is retrieved from the cache and can determine whether the entry is still valid at that time. In particular, verifiers can check for conditions that may change outside of Placeless control. Verifiers are returned by active properties. For instance the bit-provider will most likely return a verifier for the original source of the document. Verifiers are similar to the idea of cache-applets proposed in the active caching architecture of [2].

To illustrate the use of notifiers and verifiers consider the HotOS paper draft example we used before, but assuming a cache that is interposed between the application (MS-Word) and the Placeless system. When Eyal first opens the paper from MS-Word, a notifier property is attached to the base document to invalidate the cache if the file is opened for writing by another user. Another notifier at the base tracks any additions or deletions of active properties that could modify the content. At Eyal's document reference, a third notifier is attached to watch for active property additions, deletions and for changes in the "spelling corrector" property. If Eyal were to upgrade his spelling corrector to a new release, this would trigger an invalidation of the cached content. Similarly, if Doug were to update the document, one of the notifiers at the base document would invalidate Eyal's cached version. The bit-provider for the file corresponding to the paper draft returns a verifier that polls the last-modification time of the file. The verifier can thus detect that the file has changed in the file system and invalidate its cached entry.

The power of notifiers and verifiers is that they can be specific to both document types and active properties applied to a document. For example, if the cached document were a WWW document, the verifier could implement the TTL timeout as specified in the HTTP response. Verifiers can also serve documents that are composed of multiple sources, like news summaries constructed from several web sites; in that case, verifiers can check the consistency of each of the sources. For a document with heavy customization, like a financial portfolio page, the verifier may invalidate the cached entry only if there has been significant change in the stock quotes or even modify these values as needed.

Similarly, more sophisticated notifiers can be constructed as needed. Furthermore, invalidation policies could either be placed in a notifier or a verifier. For example, tracking external information that an active property depends on could be handled by a notifier installed by that property or a verifier returned by the property to the cache. In general, verifier execution trades-off cache consistency with cache access time latencies, while notifier execution adds load to the Placeless system. The evaluation of these tradeoffs is future work.

Cache Management

In addition to determining how to handle cache consistency, active properties can influence whether documents are cached at all, how cached content should be managed, and how to handle cache replacement.

The type of functionality an active property implements may determine how a document's content should be cached. For example, properties that change the content of the document or the bit provider may deem a document uncacheable if the retrieved content changes each time it is accessed, e.g., its source is live video. Other active properties may need to intercept operations only to invoke a service but will do nothing with the content itself. As an example, an active property that creates a read-audit-trail for a document only needs to know when read operations occur, but does not need to receive the actual content being read. Placeless lets active properties inform the cache of whether and how content should be cached through a *cacheability indicator*.

Currently, we provide three cacheability options: uncacheable, cacheable but operation events need to be triggered, and unrestricted caching. The three cacheability options are set by all active properties on the read-path (caused by a cache miss and load), and these choices aggregate to the most restrictive value. Thus, when all properties on the read path have executed, in addition to the document's content and the consistency verifiers, the cache receives the cacheability indicator that specifies how the content should be handled. When a property enables caching but requests the cache to trigger operation events, the cache will forward the operation, but the Placeless system will not execute them fully, instead just use them to trigger active properties that have registered for these events. Assuming a write-through cache, it is sufficient for just the properties on the read-path to set the cacheability indicator. With a write-back cache, active properties on the write-path may need to register their cacheability requirements as well. Although for most properties it is likely to be sufficient if they execute on the write-back operation and hence do not need write operations to be forwarded at all times, some may want to know exactly when each write-operation occurs. In that case these properties should set the cacheability indicator so that getOutputStream operations get forwarded to the Placeless system.

The issue of caches hiding operations that need to be tracked has been discussed extensively in the context of the WWW [2], however there the solution generally is to make those pages for which operations are tracked uncacheable. For Placeless that seemed an unreasonable restriction. In particular, because the number of caches storing any particular document for a user is likely to be small, it is reasonable to assume that the caches can collaborate with the Placeless system.

The next issue to be addressed is how entries are identified in the cache. Since the content returned by users' references to the same document can vary, the cache manager must be able to distinguish among these entries in the cache. Our current implementation tags content with both a document identifier and the user to whom the version of the document belongs. Using both the document identifier and user enables cache implementations to be shared between users, yet distinguishes between different versions of the same document. However, this approach enables no sharing of cached entries even when the cached content for different users actually is the same, such as when no active properties transform the content or when all the transformations requested by the users are the same. When a document is first accessed by a given user, the cache manager can not easily determine whether it can return the cached content that is maintained for some other user. However, for subsequent accesses, content entries could be shared if the cache maps a pair of document and user identifiers to a content signature (e.g., MD5 hash) and in turn these signatures map to the actual content. On a cache miss for an already cached version of the same content, only the document and user identifier mapping to the content signature needs to be established.

Finally, cache replacement policies are also affected by active properties. Among other things, the latency of reading a document's content can vary drastically depending on the number and execution times of the active properties attached to a document. A cache may wish to tailor its replacement policy to favor documents with numerous or complicated active properties to increase the benefit that caching provides.

The mechanism we are exploring to handle cache replacement policies with input from the properties is as follows: as document content is returned through the read path to the cache, properties define the *replacement cost* for this document. This value is initialized with the cost determined by the bit-provider to retrieve the original content from the storage repository and with that covers the varying access latencies of different document repositories. Then as properties execute along the read path, they add their costs to this initial value. Currently, the cost values used in the implementation are the execution times of each of the active properties.

4. Current Implementation

We have implemented a prototype of the mechanisms for cache consistency and cache management described above in the Placeless system. Cache consistency was implemented using per document notifiers (at the base document and/or the reference) and verifiers that execute at the cache to verify validity on cache hits. Properties also define whether documents can

	Original Source (size)		
	parcweb (1915 bytes)	*www.rice.edu* (10,883 bytes)	*www.xerox.co.uk* (1104 bytes)
no cache	822	1462	2284
cache miss	861	1582	2303
cache hit	10	10	10

Table 1. Document content access times in milliseconds for an application-level cache.

be cached and for what operations they can be cached. The replacement policy used in the implementation is a version of the Greedy-Dual-Size algorithm [1], based on the replacement cost supplied by the properties and bit-provider, as well as on the size of the document and the access frequency of the document at that cache. We also experimented with caches co-located with the Placeless server and on the machine where applications are run.

Very preliminary results show that caching can effectively hide the latency of a property-based system like Placeless. Table 1 shows the type of document access times that the system can achieve when hitting in an application-level cache (running on the same machine as the application). It also shows the raw overhead of filling the cache on a miss. No active properties were associated with the documents at either the base or the reference in this experiment. Thus, the results show that the overhead to create a minimum set of notifiers (to track additions and deletions of active properties) and the returning of one TTL-based verifier is small when servicing a cache miss.

5. Future Work

Understanding the tradeoffs between notifier and verifier usage for various types of documents and document repositories, and how best to allow active properties to influence cache replacement policies are areas for future work. For example, Quality of Service (QoS) properties, like “always available” or “access time < .25 seconds”, may need to specify caching requirements to tailor cache replacement policies. One possibility for QoS properties to influence cache replacement is to inflate replacement costs. However, we have yet to evaluate how programmers can best set the cost values for QoS requirements, and it may be necessary to add a more flexible mechanism for these types of properties. Similarly, mechanisms that tailor caching for related documents (e.g., contained in a collection) have not been investigated.

6. Conclusions

Document customization has been an emerging trend in the WWW. However, in the WWW most of the customization occurs at the original servers, like my.yahoo.com, or by designated servers for network or client adaptation [3]. The Placeless Documents system allows individual users to customize their documents through active properties. This generalization introduces interesting new issues to the problem of document caching: (1) per-user versions of the same document need to be cached, (2) cache consistency depends both on content operations and on operations that manipulate properties, and (3) the system needs to support the diversity of cache consistency mechanisms supported by the documents’ source repositories.

The Placeless Documents system allows properties to implement custom, per-document caching policies. Cache consistency is supported through notifier and verifier mechanisms. In particular, notifiers and verifiers can be used to interact with the cache consistency mechanisms of the original document sources. Properties can also expose information used to tune replacement policies by setting replacement costs and decide how the cache should manage cached content by supplying cacheability information. Evaluation of the tradeoffs in these mechanisms is future work. However, the implementation shows potential to achieve good performance without sacrificing functionality or flexibility.

7. References

[1] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pp. 193-206, Dec. 1997, Monterrey, California.

[2] Pei Cao, Jin Zhang and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. Proceedings *of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pp. 373-388, September 1998, Lake District, England.

[3] A. Fox, S. D. Gribble, E. Brewer, and E. Amir. Adapting to Network and Client Variation via On-Demand Dynamic Distillation. In *Proceedings of the ACM 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, Oct. 1996

[4] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51-81, Feb. 1988.

[5] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proceedings of the 15th Symposium on Principles of Distributed Computing*, pp. 1-7, Philadelphia, PA, 1996.