The Roma Personal Metadata Service

Edward Swierk, Emre Kıcıman, Vince Laviano and Mary Baker
Stanford University
{eswierk, emrek, vince, mgbaker}@cs.stanford.edu

Technical Report: CS-TR-00-1633
July 2000

Computer Science Department
Stanford University
Stanford, California 94305 USA

Abstract

People now have available to them a diversity of digital storage devices for their personal files. These devices include palmtops, cell phone address books, laptops, desktop computers and web-based services. Unfortunately, as the number of personal data repositories increases, so does the management problem of ensuring that the most up-to-date version of any document is available to the user on the storage device he is currently using. We introduce the Roma personal metadata service to make it easier to locate current file versions and ensure their availability across different repositories. Roma does this through the use of a centralized, available and usually portable metadata store used by mobility-aware clients. Separating out the metadata store from the respositories eases deployment of the system, since it allows us to use existing repositories without change. In this paper we describe the design requirements, architecture and current prototype implementation of Roma.

1 Introduction

As people come to rely more heavily on digital devices to work and communicate, they keep more of their personal files—including email messages, notes, presentations, address lists, financial records, news clippings, music and photographs—in a variety of data repositories. Since people are free to switch among multiple heterogeneous devices, they can squirrel away information on any device they happen to be using at the moment as well as on an ever-broadening array of web-based storage services. For example, a student wishing to record his professor's phone number could store it on his cell phone, scribble it into his palmtop computer, type it into his desktop address book, or record it in various web-based address book services.

One might expect this plethora of storage options to be a catalyst for personal mobility[8], enabling people to access and use their personal files wherever and whenever they want, while using whatever device is most convenient to them. Instead, it has made it harder for mobile people to ensure that up-to-date versions of files they need are available on the current storage option of choice. This is because contemporary file management tools are poor at handling multiple data repositories in the face of intermittent connectivity. There is no easy way for a user to determine whether a file on the device he is currently using will be accessible later on another device, or whether the various copies of that file across all devices are up-to-date. As a result, the user may end up with many out-of-date or differently-updated copies of the same file scattered on different devices.

Previous work has attempted to handle multiple data repositories at the application level and at the file system level. At the application level, some efforts have focused on using only existing system services. Unfortunately, file synchronization tools that use generic metadata provided by the system[14], such as the filename or date of last modification, are unreliable; they can only infer relationships between file copies from information not intended for such use. For example, if the user changes the filename of one copy, its relationship to other copies may be broken. Other file synchronization tools[13] that employ application-specific metadata to synchronize files are useful only for the set of applications they explicitly support.

Distributed file systems such as Coda[7] provide access to multiple data repositories by emulating existing file system semantics, redirecting local file system calls to a remote repository or local cache. Because these systems require installation of software on both the user's device and the data repository, they exclude web-based storage services and other data stores that are unlikely to support a new file system without the demand of a critical mass of users.

Our approach is to provide an application-level service that mobility-aware applications can use to give the user ubiquitous access to his or her personal files, along with additional services to integrate legacy applications. No changes need to be made to data repositories. Thus the system is easy to deploy, and a user can benefit from the system without the cooperation of anyone else.

Our system, Roma, provides an available, centralized repository of metadata, or information *about* a single user's files. The metadata format includes sufficient information to support tracking files across multiple file stores. A user's metadata repository resides on a stationary server or on a device that the user carries along with him (metadata records are typically compact enough that they can be stored on a highly portable device), thus ensuring that metadata is available even when network connectivity is intermittent. To maintain compatibility with existing applications, synchronization agents periodically scan data stores for changes made by legacy applications and propagate them to the metadata repository.

Related to the problem of managing versions of files across data repositories is the problem of locating files across different repositories. Most file management tools offer hierarchical naming as the only facility for organizing large collections of files. Users must invent unique, memorable names for their files, so that they can find them in the future; and must arrange those files into hierarchies, so that related files are grouped together. Having to come up with a descriptive name on the spot is an onerous task, given that the name is often the only means by which the file can later be found[10]. Arranging files into hierarchical folders is cumbersome enough that many users do not even bother, and instead end up with a single "Documents" folder listing hundreds of cryptically named, uncategorized files. This problem is compounded when files need to be organized across multiple repositories.

Fortunately, several projects have explored the use of attribute-based naming to locate files in either single or multiple repositories[2, 4]. While Roma metadata includes fully-extensible attributes that can be used as a platform for supporting these methods of organizing and locating files, our current prototype does not yet take advantage of this data.

The rest of this paper describes Roma in detail. We begin by outlining the requirements motivating our design; in subsequent sections we detail the design and current prototype implementation of Roma, as well as some key issues that became apparent while designing the system; these are followed by a survey of related work and a discussion of some possible future directions for this work.

2 Motivation and Design Requirements

To motivate this work, consider the problems faced by Jane Mobile, techno-savvy manager at ABC Widget Company who uses several computing devices on a regular basis. She uses a PC at work and another at home for editing documents and managing her finances, a palmtop organizer for storing her calendar, a laptop for working on the road, and a cell phone for keeping in touch. In addition, she keeps a copy of her calendar on the Yahoo! web site so it is always available both to her and her coworkers, and she frequently downloads the latest stock prices into her personal finance software.

Before dashing out the door for a business trip to New York, Jane wants to make sure she has everything she will need to be productive on the road. Odds are she will forget something, because there is a lot to remember:

- *I promised my client I'd bring along the specifications document for blue fuzzy widgets—I think it's called BFWidgetSpec.doc, or is it SpecBluFuzWid.doc?* If Jane could do a keyword search over all documents (regardless of which applications she used to create them) and over all her devices at once, she would not have to remember what the file is called, which directory contains it, or on which device it is stored.

- *I also need to bring the latest blue fuzzy widget price list, which is probably somewhere on my division's web site or on the group file server.* Even though the file server and the web site are completely outside her control, Jane would like to use the same search tools that she uses to locate documents on her own storage devices.

- *I have to make some changes to that presentation I was working on yesterday. Did I leave the latest copy on my PC at work or on the one at home?* If Jane copies an outdated version to her laptop, she may cause a write conflict that will be difficult to resolve when she gets back. She just wants to grab the presentation without having to check both PCs to figure out which version is the more recent one.

- *I want to work on my taxes on the plane, so I'll need to bring along my financial files and tax-related documents.* Like most people, Jane does not have the time or patience to arrange all her documents into neatly labeled directories, so it's hard for her to find groups of related files when she really needs them. More likely, she has to pore over a directory containing dozens or hundreds of files, and guess which ones might have something to do with her taxes.

To summarize, the issues illustrated by this example are the dependence on filenames for locating files, the lack of integration between search tools for web documents and search tools on local devices, the lack of support for managing multiple copies of a file across different devices, and the dependence on directories for grouping files together.

These issues lead us to a set of architectural requirements for Roma. Our solution should be able to

1. *Make information about the user's personal files always available to applications and to the user.*

2. *Associate with each file (or file copy) a set of standard attributes*, including version numbers or timestamps to help synchronize file replicas and prevent write conflicts.

3. *Allow the attribute set to be extended by applications and users*, to include such attributes as keywords to enable searching, categories to allow browsing related files, digests or thumbnails to enable previewing file content, and parent directories to support traditional hierarchical naming (where desired). This information can be used to develop more intuitive methods for organizing and locating files.

4. *Track files stored on data repositories outside the user's control.* A user may consider a certain file as part of his personal file space even if he did not directly create or maintain the data. For example, even though the user's bank account balances are available on a web site controlled and maintained by the bank, he should be able to organize, search and track changes to this data just like any other file in his personal space.

5. *Track files stored on disconnected repositories and offline storage media.* Metadata can be valuable even if the data it describes is unavailable. For example, the user may be working on a disconnected laptop on which resides a copy of the document that he wants to edit. Version information lets him figure out whether this copy is the latest, and if not, where to find the most recent copy upon reconnection. Alternatively, if the laptop is connected on a slow network, he can use metadata (which is often smaller than its associated file) to find which large piece of data needs to be pulled over the network.

3 Architecture

At the core of the Roma architecture (illustrated in Figure 1) is the *metadata server*, a centralized, potentially portable service that stores information about a user's personal files. The files themselves are stored on autonomous data repositories, such as traditional file systems, web servers and any other device with storage capability. Roma-aware applications query the metadata server for file information, and send updates to the server when the information changes. Applications obtain file data directly from data repositories. Agents monitor data stores for changes made by Roma-unaware applications, and update file information in the metadata server when appropriate.

Roma supports a decentralized replication model where all repositories store "first-class" file replicas—that is, all copies of a file can be manipulated by the user and by applications. To increase availability and performance, a user can copy a file to local storage from another device (or an application can do so on the user's behalf). Roma helps applications maintain the connection between these logically related copies, or *instances*, of the file by assigning a unique *file identifier* that is common to all of its instances. The file identifier can be read and modified by applications but is not normally exposed to the user.

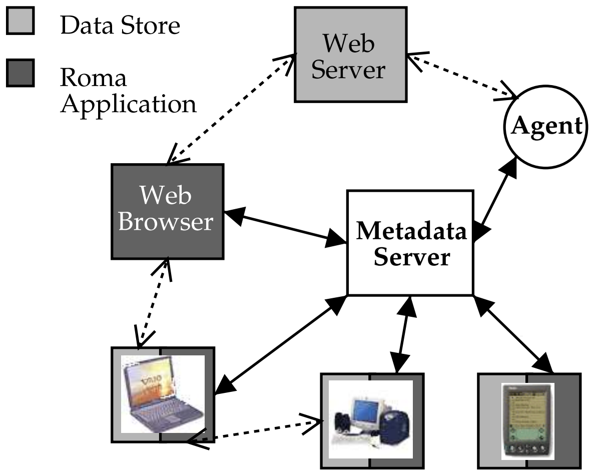

Figure 1: The Roma Architecture. Applications are connected to the metadata server, and possibly connected to a number of data stores. Agents track changes to third-party data stores, such as the web server in this diagram, and make appropriate updates to the metadata server.

Once the file is copied, the contents and attributes of each instance can diverge. Thus Roma keeps one *metadata record* for each file instance. A metadata record is a tuple composed of the file identifier, one or more data addresses, a version number and optional, domain-specific attributes.

The *data address* specifies the location of a file instance as a Universal Resource Identifier (URI). Files residing on the most common types of data repositories can be identified using existing URI schemes, such as `http:` and `ftp:` for network-accessible servers and `file:` for local file systems. When naming removable storage media, such as a CD-ROM or a Zip disk, it is important to present a human-understandable name to the user (possibly separate from the media's native unique identifier, such as a floppy serial number).

The *version number* is a simple counter. Whenever a change is made to a file instance, its version number is set to be one greater than the previous greatest version number of all the file's instances. We are investigating the use of version vectors to store more complete and flexible versioning information[9].

Roma-aware applications can supplement metadata records with a set of *optional attributes*, including generic attributes such as the size of a file or its type, and domain-specific attributes like keywords, categories, thumbnails, outlines or song titles.

These optional attributes enable application user interfaces to support new modes of interaction with the user's file space, such as query-based interfaces and browsers. Autonomous agents can automatically scan files in the user's space and add attributes to the metadata server based on the files' contents. Section 6 briefly describes Presto, a system developed by the Placeless Documents group at Xerox PARC that allows users to organize their documents in terms of user-defined attributes. The user interaction mechanisms developed for Presto would mesh well with the centralized, personal metadata repository provided by Roma.

3.1 Metadata server

The metadata server is a logically centralized entity that keeps metadata information about all copies of a user's data. Keeping this metadata information centralized and separate from the data stores has many advantages:

- Centralization helps avoid write conflicts, since a single entity has knowledge of all versions of the data in existence. Potential conflicts can be prevented before they happen (before the user starts editing an out-of date instance of a file) rather than being caught later, when the files themselves are being synchronized.

- Centralization allows easier searching over all of a user's metadata because clients only have to search at a single entity. The completeness of a search is not dependent on the reachability of the data stores. In contrast, if metadata were distributed across many data stores, a search would have to performed at each data store. While this is acceptable for highly available data repositories connected via high-bandwidth network, it is cumbersome for data stores on devices that need to be powered on, plugged in, or dug out of a shoebox to be made available.

- Separation from the data store allows easier integration of autonomous data stores, including legacy and third-party data stores over which the user has limited control. Storing metadata on a server under the user's control, rather than on the data stores with the data, eliminates the need for data stores to be "Roma-compliant." This greatly eases the deployability of Roma.

- Separation also provides the ability to impose a personalized namespace over third-party or shared data. A user can organize his or her data independent of the organization of the data on the third-party data store.

- Separation enables clients to have some knowledge about data they cannot access, either because the data store is off-line, or because it speaks a foreign protocol. In essence, clients can now "know what they don't know."

The main challenge in designing a centralized metadata server is ensuring that it is always available despite intermittent network connectivity. Section 5.2 describes one solution to this problem, which is to host the metadata server on a portable device. Since metadata tends to be significantly smaller than the data it describes, it is feasible for users to take their metadata server along with them when they disconnect from the network.

3.2 Data stores

A data store is any information repository whose contents can somehow be identified and retrieved by an application. Roma-compatible data stores include not only traditional file and web servers, but also laptops, personal digital assistants (PDAs), cell phones, and wristwatches—devices that have storage but cannot be left running and network-accessible at all times due to power constraints, network costs, and security concerns—as well as "offline" storage media like compact discs and magnetic tapes. Information in a data store can be dynamically generated (e.g., current weather conditions or the time of day). Our architecture supports

- data stores that are not under the user's control.
- heterogeneous protocols (local file systems, HTTP, FTP, etc.). There are no *a priori* restrictions on the protocols supported by a data store.
- data stores with naming and hierarchy schemes independent of both the user's personal namespace and other data stores.

In keeping with our goal to support legacy and third-party data stores, data stores do not have to be Roma-aware. There is no need for direct communication between data stores and the metadata server. This feature is key to ensuring the easy deployability of Roma.

3.3 Applications

In Roma, clients are applications used by people to view, search and modify their personal data. These include traditional progams, such as text editors, as well as handheld-based personal information managers (PIMs) and special-purpose Internet appliances. Clients can be co-located with data sources; for example, applications running on a desktop computer are co-located with the computer's local file system.

Clients have two primary responsibilities in our system. The first is to take advantage of metadata information. This includes an appropriate presentation of useful metadata to the user, but also includes the client's own use of metadata information. For example, a client can automatically choose to access the "nearest" or latest copy of a file.

The clients' second responsibility is to inform the metadata server when changes made to the data affect the metadata. Mundanely, this means informing the metadata server when a change has been made (for versioning purposes), but can also include updating of domain-specific metadata. We are investigating how often updates need to be sent to the metadata server to balance correctness and performance concerns.

It is assumed that while clients are in use, they are connected to the metadata server. However, clients are not necessarily well-connected to all data stores. They may be connected weakly or not at all. For example, a client might not speak the protocol of a data store, and thus might be effectively disconnected from the data store. Also, a data store itself may be disconnected from the network.

3.4 Synchronization agents

Roma synchronization agents are software programs that run on behalf of the user, without requiring the user's attention. Agents can do many tasks, including:

- providing background synchronization on behalf of user
- hoarding of files on various devices in preparation for disconnected operation
- making timely backups of information across data stores
- tracking third-party updates (on autonomous data stores, or data shared between users)

Agents can be run anywhere on a user's personal computers or on cooperating infrastructure. The only limitation on agent's execution location is that the agent must be able to access relevant data stores and the metadata server. Note that the use of a portable metadata server precludes agents from running while the metadata server is disconnected from the rest of the network.

3.5 Examples

To illustrate how Roma supports a user working with files replicated across several storage devices, let us revisit Jane Mobile, and consider what a Roma-aware application does in response to Jane's actions.

The action of copying a file actually has two different results, depending on her intent; the application should provide a way for her to distinguish between the two:

- *She makes a file instance available on a different repository* (in preparation for disconnected operation, for example). The application contacts the metadata server, creates a new metadata record with the same file identifier, copies all attributes, and sets the data address to point to the new copy of the file.
- *She copies a file to create a new, logically distinct file based on the original.* The application contacts the metadata server, creates a new metadata record with a new file identifier, copies all attributes, and sets the data address to point to the new copy of the file.

Other actions Jane may take:

- *The user opens a file for updating.* The application contacts the metadata server, and checks the version number of this instance. If another instance has a higher version number, the application warns the user that he is about to modify an old version, and asks user if he wants to access latest version or synchronize the old one (if possible).
- *The user saves the modified file.* The application contacts the server, sets version number of this instance to one plus the current highest, and updates any attributes, such as the file's size.
- *The user brings a file instance up to date by synchronizing it with the newest instance.* The application contacts the server, finds the metadata record with the highest version number for this file, and copies all attributes (except the data address) to the current instance.

4 Implementation

In this section we describe the current status of our prototype Roma implementation. The prototype is still in its early stages and does not yet support synchronization agents.

4.1 Metadata server

We have implemented a prototype metadata server that supports updates and simple queries, including queries on optional attributes. It is written in Java as a service running on Ninja[5], a toolkit for developing highly available network services. Metadata is stored in an XML format, and we use XSet, a high performance, lightweight XML database, for query processing and persistence[16].

We have also implemented a proof-of-concept portable metadata server. Though the metadata server itself requires a full Java environment to operate, we have implemented a simple mechanism to migrate a metadata repository between otherwise disconnected computers using a PDA as a transfer medium. As a user finishes working on one computer, the metadata repository is transferred onto his PDA. The next time he begins using a computer, the metadata repository is retrieved from the PDA. In this way, though the metadata server itself is not traveling, the user's metadata is always accessible, regardless of the connectivity between the user's computer and the rest of the world.

4.2 Data stores

Currently, the data stores we support are limited to those addressable through URIs. Our clients can currently access data stores using HTTP and FTP, as well as files accessible via a standard file system interface such as local file systems, NFS and AFS.

4.3 Clients

We have implemented three Roma-aware clients. These clients allow users to view and manipulate their metadata and data from a variety of devices.

Our first client is a web-based metadata browser. It provides hierarchical browsing of one's personal data. The browser displays the names of data files, their version information, and the deduced MIME type of the file. In addition, if the file is accessible, the browser will present a link to the file itself. We have also written a proxy to enable "web clipping" of arbitrary web content

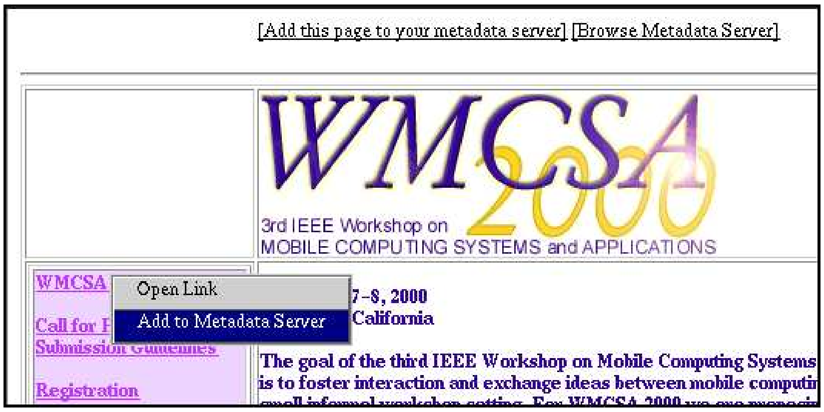

Figure 2: A screenshot of the web-clipper proxy. As the user browses the web, the proxy adds links on the fly, allowing the user to browse the metadata server and to add pages to his personal file space.

into the user's personal file space, as displayed in Figure 2.

Our second client is a set of command-line tools. We have written Roma-aware `ls` and `locate` commands to query a metadata server, a `get` command to retrieve the latest version of a file from remote data stores, and `import`, a utility to create metadata entries for files on a local data store.

We have also implemented a proof-of-concept PDA client. Built using a Waba VM and RMILite[15, 1], our PDA client can query and view the contents of a metadata server. Currently, the PDA client does not access the actual contents of any file.

Our clients have added a metadata attribute to describe the data format of files. If available, our command-line tools use the Unix `magic` command to determine the data format. Our web-clipper determines the data format based on the mime-type of the file.

5 Design issues and future work

In this section we describe some of the issues and design decisions encountered so far in our work with Roma, along with some of the work that remains for us to do.

5.1 Why "personal"?

One important design issue in Roma is the scope of the types of data it supports. There are several reasons behind the choice to support only personal files, rather than to tackle collaboration among different users as well, or to attempt to simplify system administration by handling distribution of application binaries and packages.

First, restricting ourselves to personal files gives us the option of migrating the metadata server to a personal, portable device that the user carries everywhere, to increase its availability. This option is described in more detail in the next section.

Second, it avoids one potential source of write conflicts—those due to concurrent modifications by different users on instances of the same file. Such conflicts are often difficult to resolve without discussion between the two users to agree on the modified version. Conflicts can still occur from modifications by third parties working on behalf of the user, such as an email transfer agent appending a new message to the user's inbox while the user deletes an old one. However, these conflicts can often be resolved automatically using knowledge about the application, such as the fact that an email file consists of a sequence of independent messages.

Third, it lets us exploit the fact that users are much better at predicting their future needs for their personal files than for other kinds of files[3].

Fourth, it lets us support categories, annotations and other metadata that are most meaningful to a single person rather than a group.

Finally, we believe there is a trend toward specialized applications tailored for managing other types of files:

- Tools like the RedHat Package Manager (RPM) and Windows Update are well-suited for distributing *system-oriented data* such as application packages, operating system components, and code libraries. These tools simplify system administration by grouping related files into packages, enforcing dependencies, and automatically notifying the user of bug fixes and new versions of software.

- Groupware systems like the Concurrent Versioning System (CVS), ClearCase, Lotus Notes and Microsoft Outlook impose necessary structure and order on access to *shared data with multiple writers.* Email is often sufficient for informal collaboration within a small group.

- The web has become the best choice for distributing *shared data with many readers.*

Since these applications handle system data, collaborative projects and shared read-mostly data, we believe that the remaining important category of data is personal data. We thus focus on handling this category of data in Roma.

5.2 Ensuring availability of metadata

Since our overarching goal is to ensure that information about the user's files is always available to the user, we need to make the system robust in the face of intermittent or weak network connectivity—the very situations that underscore the need for a metadata repository in the first place.

Our approach is to allow the user to keep the metadata server in close physical proximity, preferably on a highly portable device that he can always carry like a keychain or necklace. Wireless network technologies like Bluetooth will soon make "personal-area networks" a reality. It is not hard to imagine a server embedded in a cell phone or a PDA, with higher availability and better performance than a remote server in many situations.

One difficulty with a portable server is the issue of third-party agents acting on behalf of the user and modifying data in the user's personal file space. If the network is partitioned and the only copy of the metadata is with the user, how does such an agent read or modify the metadata? In other words, we need to ensure availability to third parties as well.

Our solution is to cache metadata in multiple locations. If the main copy currently resides on the user's PDA, another copy on a stationary, network-connected server can provide access to third parties. This naturally raises the issues of synchronizing the copies and handling update conflicts between the metadata replicas.

However, our hypothesis is that updates made to the metadata by third parties rarely conflicts with user updates. For example, a bank's web server updates a file containing the user's account balances, but the user himself rarely updates this file.

Testing this hypothesis is part of our future work in evaluating Roma.

5.3 Supporting legacy clients

Though we explicitly support legacy, Roma-unaware data stores, our architecture does not explicitly support legacy applications. We believe that for the user to take advantage of metadata, application user interfaces need to be designed to present a view of the metadata. However, if we are willing to accept restricting a user's direct access to metadata, we do have options for supporting legacy clients in Roma.

Our first option is to use application-specific extension mechanisms to add Roma-awareness to legacy clients. For example, we implemented a Roma-aware proxy to integrate existing web browsers into our architecture.

Our second option is to layer Roma-aware software beneath the legacy client. Possibilities include modifying the C-library used by clients to access files, or writing a Roma-aware file system. This option does nothing to present metadata to the user, but can potentially provide some functionality enhancements such as the intelligent retrieval of new versions of data. We are basically trading support for legacy data stores for support for legacy clients.

A third option is to use agents to monitor data edited by legacy clients in the same way we monitor data repositories not under the user's control. This option neither presents metadata to the user, nor enhances the functionality of the client. It does, however, ensure that the metadata at the server is kept up-to-date with changes made by legacy clients.

5.4 Addressing personal data

Our current Roma implementation uses a URI to identify the file instance corresponding to a particular metadata record. Unfortunately this is an imperfect solution since the relationship between URIs and file instances is often not one-to-one. In fact, it is rarely so.

On many systems, a file instance can be identified by more than one URI, due to aliases and links in the underlying file system or multiple network servers providing access to the same files. For example, the file identified by `ftp://gunpowder/pub/paper.ps` can also be identified as `ftp://gunpowder/pub/./paper.ps` (because `.` is an alias for the current directory) and `http://gunpowder/pub/ftp/pub/paper.ps` (since the public FTP directory is also exported by an HTTP server).

The problem stems from the fact that URIs are defined simply as a string that refers to a resource and not as a unique resource identifier. Currently we rely on applications and agents to detect and handle cases where multiple URIs refer to the same file, but if an application fails to do this, it could cause the user to mistakenly delete the only copy of a file because he was led to believe that a backup copy still existed. In the future we would like Roma to address this problem more systematically.

5.5 Improving application support

One key feature missing from our current implementation is a set of APIs that lets client applications communicate with the Roma metadata repository, via an interface more tightly coupled to the client's operating system than the generic RMI interface we provide. Developing these APIs would offer two major benefits. First, application developers would not have to worry about supporting different transport protocols for accessing the metadata repository, whether it resides on a remote server or on a local device.

Second, legacy applications could be ported quickly using a library that conforms to the platform's file system API, while new applications could take full advantage of the metadata repository using a richer interface. This is particularly important to meet our goal of easy deployment, because we expect that existing applications will be ported and new applications written only when a critical mass of Roma users exists.

6 Related work

Helping users access data on distributed storage repositories is an active area of research. The primary characteristic distinguishing our work from distributed file systems, such as NFS[11], AFS[6], and Coda[7], is our emphasis on unifying a wide variety of existing data repositories to help users manage their personal files.

The Coda distributed file system, like Roma, seeks to allow users to remain productive during periods of weak or no network connectivity. While Roma makes metadata available during these times, Coda caches file data in a "hoard" according to user preferences in anticipation of periods of disconnection or weak connectivity. However, unlike Roma, users must store their files on centralized Coda file servers to benefit fully from Coda, which is impractical for people who prefer to maintain several active "computing bases." Even when users do not prefer to maintain more than one data repository, they may be obliged to if, for instance, their company does not permit them to mount company file systems on their home computers, or if they use several different computing devices some of which they do not synchronize to a central data store.

The Bayou system[9] supports a decentralized model where users can store and modify their files in many repositories which communicate peer-to-peer to propagate changes. However, users cannot easily integrate data from Bayou-unaware data stores like third-party web services into their personal file space.

The Presto system[2] focuses on enabling users to organize their files more effectively. They have built a solution similar to Roma that associates with each of a user's files documents a set of properties that can be used to organize, search and retrieve files. This work does not specifically address tracking and synchronizing multiple copies of documents across storage repositories, nor ensuring that properties are available even when their associated documents are inaccessible. However, the applications they have developed could be adapted to use the Roma metadata server as property storage.

Both Presto and the Semantic File System[4] enable legacy applications to access attribute-based storage repositories by mapping database queries onto a hierarchical namespace. Presto achieves this using a virtual NFS server, while the Semantic File System integrates this functionality into the file system layer. Either mechanism could be used with Roma to provide access to the metadata server from Roma-unaware applications.

The Elephant file system[12] employs a sophisticated technique for tracking files across both changes in name and changes in inode number.

7 Conclusions

We have described a system that helps fulfill the promise of personal mobility, allowing people to switch among multiple heterogeneous devices and access their personal files without dealing with nitty-gritty file management details such as tracking file versions across devices. This goal is achieved through the use of a centralized metadata repository that contains information about all the user's files, whether they are stored on devices that the user himself manages, on remote servers administered by a third party, or on passive storage media like compact discs. The metadata can include version information, keywords, categories, digests and thumbnails, and is completely extensible. We have implemented a prototype metadata repository, designing it as a service that can be integrated easily with applications. The service can be run on a highly available server or migrated to a handheld device so that the user's metadata is always accessible.

8 Acknowledgements

The authors thank Doug Terry for his helpful advice during the early stages of the project, and Andy Huang, Kevin Lai, Petros Maniatis and Mema Roussopoulos for their detailed review and comments on the paper. This work has been supported by a generous gift from NTT Mobile Communications Network, Inc. (NTT DoCoMo).

References

[1] Mike Chen, Mohan Lakhamraju, Eric Brewer, and David Culler, "Jini/RMI/TSpace for Small Devices." http://post-pc.cs.berkeley.edu/rmilite/

[2] Paul Dourish, W. Keith Edwards, Anthony LaMarca and Michael Salisbury, "Uniform Document Interactions Using Document Properties." *Proc. ACM Symposium on User Interface Software and Technology (UIST '99).*

[3] Maria Ebling, "Translucent Cache Management for Mobile Computing," Thesis, School of Computer Science, Carnegie Mellon University, March 1998.

[4] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole, Jr., "Semantic File Systems." *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, October 13–16, 1991, Pacific Grove, California.

[5] Steve Gribble, Matt Welsh, Eric A. Brewer, and David Culler, "The MultiSpace: an Evolutionary Platform for Infrastructural Services." *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS '99)*, August 1999.

[6] M. L. Kazar, "Synchronization and Caching Issues in the Andrew File System." *Proceedings of the Winter 1988 USENIX Technical Conference*, February 1988.

[7] James J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System." *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, October 13–16, 1991, Pacific Grove, California. Pages 213–225.

[8] Petros Maniatis, Mema Roussopoulos, Ed Swierk, Kevin Lai, Guido Appenzeller, Xinhua Zhao, and Mary Baker, "The Mobile People Architecture." *ACM Mobile Computing and Communications Review (MC^2R)*, July 1999.

[9] Karin Petersen, Mike J. Spreitzer, Douglas B. Terry, Marvin M. Theimer and Alan J. Demers, "Flexible Update Propagation for Weakly Consistent Replication." *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, October 5–8, 1997, Saint-Malo, France. Pages 288–301.

[10] Jef Raskin, *The Humane Interface.* Addison-Wesley, 2000.

[11] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System." *Proceedings of the Summer 1985 USENIX Technical Conference*, June 1985.

[12] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton and Jacob Ofir, "Deciding When to Forget in the Elephant File System." *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, December 12–15, 1999, Charleston, South Carolina. Pages 110–123.

[13] Stu Slack, "Extending Your Desktop with Pilot." *PDA Developer Magazine*, September/October 1996. http://www.wwg.com/newsview/palmdesktop.shtml

[14] Andrew Tridgell and Paul Mackerras, "The rsync Algorithm." Technical Report TR-CS-96-05, Australian National University.

[15] Wabasoft, Inc., "Wabasoft: Product Overview." http://www.wabasoft.com/products.html

[16] Ben Y. Zhao and Anthony D. Joseph, "XSet: A Lightweight Database for Internet Applications." Submitted for publication, May 2000. http://www.cs.berkeley.edu/~ravenben/publications/saint.pdf