Published in ACM Transactions on Computer-Human Interaction, 6(2), 133-161, 1999.

Presto: An Experimental Architecture for Fluid Interactive Document Spaces

Paul Dourish, W. Keith Edwards, Anthony LaMarca and Michael Salisbury

Computer Science Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto
CA 94304 USA

{dourish, kedwards, lamarca, salisbury}@parc.xerox.com

Abstract. Traditional document systems use hierarchical filing structures as the basis for organising, storing and retrieving documents. However, this structure is very limited in comparison with the rich and varied forms of document interaction and category management in everyday document use. Presto is a prototype document management system providing rich interaction with documents through meaningful, user-level document attributes, such as "Word file", "published paper", "shared with Jim", "about Presto" or "currently in progress". Document attributes capture the multiple different roles that a single document might play, and allow users to rapidly reorganise their document space for the task at hand. They provide a basis for novel document systems design and new approaches to document management and interaction.

In this article, we outline the motivations behind this approach, describe the principal components of our implementation, discuss architectural consequences, and show how these support new forms of interaction with large personal document spaces.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; H.3.2 [**Information Storage and Retrieval**]: Information Storage—*file organization*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrival; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*interaction styles*.

Additional key words and phrases**:** Document management, attribute/value systems, information retrieval, direct manipulation.

1 INTRODUCTION

Much of what we do when sitting in front of computers is document space management. We store, file, organise and retrieve information from a variety of electronic document collections. The emergence of the World Wide Web and the spread of the Internet mean that more and more documents flow into our document space every day, increasing our need for better tools to visualise and interact with it.

The tools that most of us use every day rely on one fundamental mechanism for organising document spaces—the hierarchy. Hierarchies occur in all sorts of document management tools, including mail systems, contact managers and web browsers, but perhaps the most pervasive is the filesystem. Documents are stored in files that live in directories or folders, which are themselves contained in other directories, creating a hierarchy that provides the structure for document space interactions. In this model, the difference between files and documents all but disappears; to all intents and purposes, the file space *is* the document space.

By implication, the structure of the document space is also mapped onto the structure of the file space. Much of our navigation through the filesystem, in browsers and dialog boxes, can be seen, in fact, as navigation through *semantic* structures that have been mapped onto a file hierarchy. When someone begins a search in the directory titled "Frame documents", and then progressively move through sub-directories called "Publications", "Conference papers", "UIST 98" and "Presto paper drafts", this movement can be seen not only as a movement from one folder to another, but also as a search procedure that exploits features of the documents to progressively focus on a smaller and smaller set of potential documents. The structure of the search has been mapped onto the hierarchy provided by the filesystem because that hierarchy is essentially the only mechanism provided to organise files.

Documents, though, are not files. Documents are organisational entities with social meaning and practical consequences, while computer files are computational artifacts whose behaviours are defined technically. Similarly, our requirements for document space organisation differ from those for managing files. We would like to provide a basis for organising documents that reflects the use of documents in working settings, and supports richer document practices for organisation, management and retrieval.

Our current research in the "Placeless Documents" project concerns precisely this question of document space organisation, and how we can support more natural and fluid forms of interaction with a document space. Examples of the sorts of problems we want to address include:

1. A filesystem provides only a "single-inheritance" structure. Files can only be in one place at a time, and so can occupy only one spot in the semantic structure. The use of links and aliases is testament to this problem, but at the same time is sufficiently complicated that it often extends no further than desktop shortcuts.

2. My conception of the structure by which files should be organised changes over time, but the hierarchy is fixed and rigid. Moving individual files around is easy enough, but reorganising large sets of files is a chore.

3. Not only does the category structure for files change over time, but it is dependent on the task. At one moment, searching for a particular document, I might want to organise my document space in terms of projects; at the next, according to time; and then, according to document content. Effectively, document retrieval depends on being able to manage multiple different views of my document space fluidly.

4. Since these documents are being stored on a computer, I would like to be able to harness computational power to help me organise them. If the computer can figure out that something is a Web document that references three other specific pages, then why should I have to do it?

We have developed a system, called *Presto,* that provides an infrastructure for experimentation with some of the issues that have arisen in the Placeless Documents project. Placeless is concerned with a broad range of concerns, and the development of a novel approach to document systems infrastructure; it is described in more detail elsewhere [Dourish et al., 1999a]. Presto is concerned with a subset of the Placeless design issues. In particular, Presto has been designed as a platform to explore new forms of fluid interaction with documents spaces, addressing the sorts of user problems discussed above.

In what follows, we first outline some motivations for our approach, and describe the basic model that Presto presents. The main body of the paper will introduce the Presto architecture, and discuss how it supports attribute-based document interaction, in both legacy and novel applications. Finally, we will discuss what we have learned from our first set of applications, and discuss some possibilities for the future.

2 MOTIVATION

The work described in this paper is inspired by the rigid, inflexible means provided by filesystem-based document repositories for managing working documents. The traditional model of hierarchical directories and (hopefully) mnemonic filenames not only imposes considerable burdens for filing, searching and retrieval, but also fails to take account of the ways in which everyday document work takes place.

Studies of filing and retrieval practices, such as those documented by Barreau and Nardi [1995], highlight the ways in which traditional mechanisms fail. Barreau and Nardi draw together findings from a number of studies of how people organise files on PCs, and how they use the desktop and directory structures. In their investigations, most users do not employ elaborate or deep filing systems, but rather show a preference for simple structures and "location-based search", exploiting groupings of files (either in folders, or on the computer desktop) to express patterns of relationships between documents and to aid in retrieval. In their response to Barreau and Nardi's article, Fertig et al. [1996] defend deep structure and search queries, observing that location-based finding is "nothing more than a user-controlled logical search". There is, however, one clear feature of location-based search which adds to a simple logical search—in a location-based system, the documents have been subject to some sort of pre-categorisation. Additional structure has been introduced into the space; this structure is exploited in search and retrieval.

Another interesting feature of this location-based structure is that it is exploited *perceptually*, rather than *cognitively.* Moving the burden of retrieval effort from the cognitive to the perceptual system has long been the direction taken by information visualisation research [Robertson et al., 1993]. However, although this approach can be highly effective, the information that these systems rely on is content-based, and extracting this information to find the structure can be computationally expensive.

In Presto, we steer a middle course, between structure-based retrieval with few structure cues (traditional hierarchies) and content-based retrieval with extensive feature extraction (as in some information visualisation approaches). Instead, we tag or annotate documents in our system with *attributes* that express attributes of the documents. Our attributes express attributes meaningful to the user in their use of the document. They can be added directly by users (like the pre-categorisation of location-based search) or extracted by software services (like content-based approaches).

A critical feature of this attribute-based approach is that it captures the *multivalent* nature of documents.[1] In everyday working, documents can represent different things to different people, or be used in different ways at different times, and it can be hard to fit these different uses into a single structure (even with the help of shortcuts, aliases or symbolic links). Sometimes I might want to handle a document as an email message, and sometimes I want to handle it is a Presto design document; and at the same time, my administrative assistant might want to organise it according to the budget center to which equipment purchases should be charged. Capturing the multiple features of documents that may be relevant for their organisation is a critical feature of our approach. The corollary is that we need to be able to fluidly switch from one view to another, reorganising the document workspace quickly to reflect the terms and structure relevant to the task in hand.

A broader goal is to understand the relationship between computational and naturalistic approaches to classification, identity, grouping and membership. For instance, investigations such as those of Sacks [1992] emphasise membership and categorisation as practical accomplishments and social actions. That is, the identification of an entity as being such-and-such-a-thing is achieved *by some particular individual* and *for some particular purposes*; for other purposes, such an identification might be quite incorrect. Inspired by such analytic accounts of the practice of category management, we are interested in how computational systems can allow for similarly fluid categorisation. This is part of a wider program of research [Dourish and Button, 1999] and will not directly concern us in this paper.

2.1 Attribute-Based Document Interaction

In Placeless Documents, and hence in Presto, users conceive of and interact with documents in terms of document attributes. Attributes are user-level properties of documents, expressed as key/value pairs. For instance, attributes of a document might include that it is in Framemaker format, that it is a journal paper, that it is in preparation, that it is intended for TOCHI, and that it is about Presto. Another document may contain attributes stating that it is an email message from Doug about meeting times, and that it was read last Tuesday. In other words, document attributes are intended to capture information that is meaningful to document users and creators. This information is captured and recorded as a set of key/value pairs, such as "MIME Type=application/msword" or "Sender=terry@parc.xerox.com".

These document attributes form the basis of interaction with the document space in Presto. Documents can be retrieved, indexed and organised according to their attributes. Attributes can be added by users or by Presto services, and can store arbitrary information. A document can contain any number of attributes. Document attributes can be used as the basis of queries, and queries can be used to specify "live" document collections with dynamic membership.

In this way, the structure of the document space, and hence the structure of *interaction* with the document space, is based on meaningful properties of documents, rather than the structure in which they were filed. Using document attributes in this way means that interaction is more strongly connected to the users' immediate concerns and the task in hand rather than an extrinsic structure. In addition, the structure of the document space reflects changes in the state of documents, rather than

1. By "multivalent", we mean that documents have multiple values and play multiple roles concurrently. This is not quite the same as the "multivalent documents" of Phelps and Wilensky [1996], which are documents whose content is the composition of multiple components.

simply their state when they were filed. However, collections still appear inside collections, and standard filing information—such as document ownership, modification dates, file types, etc.—are still preserved by our system, appearing as document attributes maintained by the infrastructure. In this way, we can recapture more traditional forms of structured interaction with document spaces. New forms of interaction emerge as a logical extension of traditional practices.

2.2 Design Goals

Presto is our first prototype exploring the use of attributes as a uniform mechanism for document interaction. As such, it had two primary goals: to demonstrate the validity of the approach, and to provide us with some early design experiences to inform future explorations. In addition, we set a number of approximate criteria for success. Our goal was to develop a system efficient enough to hold document sets numbering in the thousands; robust enough for everyday use, at least by ourselves and a small set of colleagues; and practical and effective enough to be used as a primary interface to our own document spaces, with the documents that we use every day. This "practicality" criterion implied that our system needed to interoperate with the applications that we use on our regular desktop platforms (Solaris and Windows NT). Expressed simply, our position was that a document system that can't launch Word and fetch Web documents is no document system at all.

At the same time, we decided that a number of issues were explicitly *not* our concern in Presto, including document replication and coordination, migration, and security. Our focus for the moment is on interaction rather than the systems issues.

3 THE PRESTO ARCHITECTURE

Before we discuss the details of our design, let's consider Presto's basic architectural model, and how it provides support for novel forms of document interaction. Many of Presto's architectural elements are shared with the Placeless Documents system, as outlined earlier; we discuss them here purely in the context of Presto, and will have more to say about Placeless Documents later.

3.1 The Presto Document Model

Presto provides a uniform document model for documents of all sorts. A document is an entity to which attributes can be attached. Attributes are arbitrary name/value pairs. They are arbitrary in a number of ways. First, there can be arbitrarily many attributes attached to a document; indeed, since attributes are the primary means of organising and retrieving documents, our principle is "the more, the merrier". Second, attributes can have arbitrary names; users can always add new sorts of attributes even if the system has never heard of them before, and the system will impose no restrictions (other than access control) on who can add attributes, when and how. Third, attributes can have arbitrary values. Although we primarily use simple text string values in our examples here, Presto can actually record any structured data item, including runnable code, in attribute values.

Note that the fact that a document may contain content is, essentially, incidental. Of course, most documents do, and Presto provides convenient access to whatever that content might be; and most applications are naturally focussed around content. However, content is not an absolute requirement, and the Presto implementation itself makes use of attributes on "content-less" documents as an associative object storage facility.

Users can add their own personal attributes to the documents of other people. If I like a paper that you wrote, I might mark it as being "interesting", for my own purposes, without interfering with

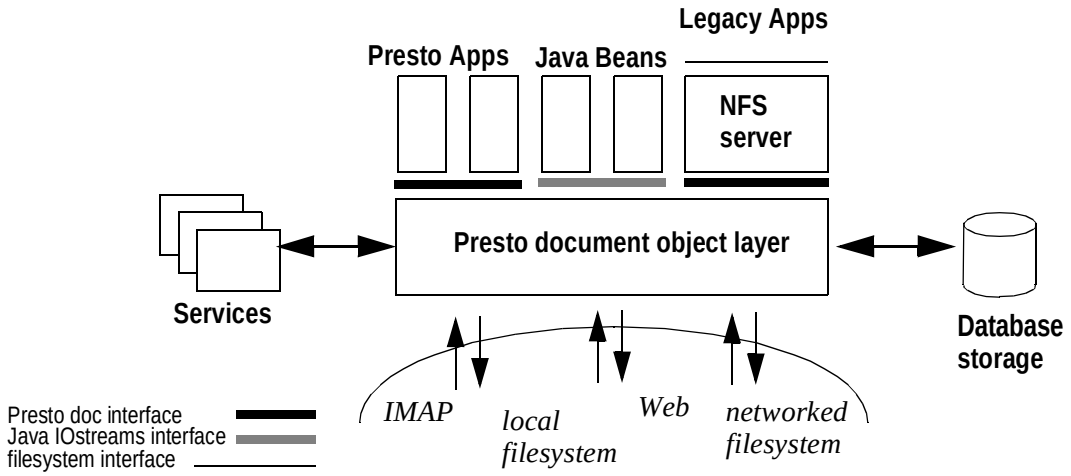

FIGURE 1: The Presto kernel architecture provides a single point of interaction supporting multiple application interfaces and multiple repository interfaces.

your interactions, and without everyone else having to see that attribute. (Just because I think it's interesting doesn't mean that everyone else does too.) This is achieved through document "references", which are personal reference objects to other documents. References appear as documents themselves, and can have attributes.

Documents can be grouped together into collections; a single document can appear in multiple collections, or in none at all. Document collections are, themselves, documents; the same document features apply to them. Document collection membership can be defined both statically and dynamically. Collections will be explored in more detail below.

3.2 Core Architecture

The core architecture of the Presto system is shown in figure 1.The centre of the system is the *Presto Document Object Layer*. This implements Presto's document model, providing documents with document attributes. Collections, which may include dynamic queries, are one type of document, and so are implemented by this layer. The Document Layer exports two forms of interface. The first is the "*Presto Document*" interface, a Java class API for applications that are fully Presto-aware. Presto-specific applications are built to this interface. The second is a standard Java *IO-stream* interface, for integratable Java applications that do not understand Presto. We use this interface to integrate Java Beans to provide viewing and editing of particular document formats, for instance.

For legacy applications that are completely Presto-naïve and may not be written in Java, Presto implements a Network File System (NFS) server [Sun Microsystems, 1989] so that the Presto database can also be accessed as a regular filesystem. Applications simply read from and write to the filesystem as they would normally; the NFS interface serves Presto documents behind the scenes so that those filesystem operations actually apply to Presto documents. Not only does this allow legacy applications to use Presto seamlessly, but it also allows Presto to maintain relevant document attributes (such as modification dates) for activities that happen in these external applications.

Background applications are integrated into Presto through the *Service* component. Presto services are applications that introduce information into the system, often processing structured files in order to turn content into attributes. For example, a mail service operates on electronic mail files and pro-

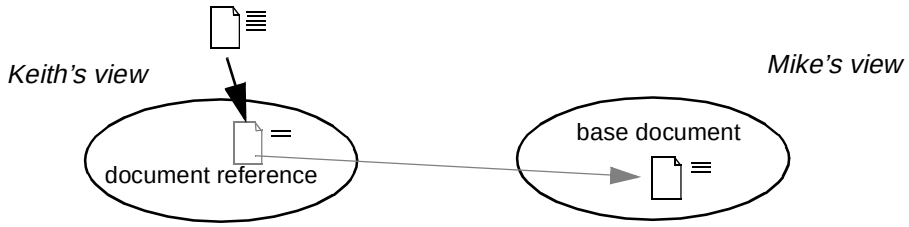

FIGURE 2: When a user interacts with a document reference, they see a combination of their local attributes and the base document's attributes. Keith sees both Mike's attributes and his own..

cesses them so that the Presto documents are annotated with details from the email headers as document attributes. Services can be scheduled to operate periodically (e.g. late at night, or every five minutes) or to act on specific events (e.g. whenever a document's contents are changed).

In important feature of the Presto design is that document content does not, itself, life in Presto. Presto stores only the attribute metadata; what Presto calls "documents" are, in fact, internal objects that stand for content stored elsewhere. We use the term "repository" to describe a store for document content. The normal filesystem, for example, is a repository; it stores content that can be associated directly with Presto documents. When an application (or a user) reads a document, Presto fetches the content from the external repository, so the separation between metadata and content is invisible to Presto users.

The primary benefit of this design is that Presto can integrate content from a variety of external repositories. Presto applications can take advantage of documents stored on the Web, since the Web is one of our supported repositories. What is more, these documents can be processed alongside file-system documents; since Presto insulates users from the mechanisms used to access content, documents at the Presto level are handled seamlessly.

The core Presto architecture has two run-time configurations. First, it can be run with the Presto document object implementation and one or more applications in a single address space. In this configuration, Presto is essentially a run-time document management library dynamically linked in with an application. Second, it can be run with the Presto document object implementation as a server, and Presto applications as clients. This second mode allows Presto applications to run in Java applets or in network computing environments with a "thin client" architecture. Applications can be run seamlessly in either configuration; the Presto layer is encapsulated within a Presto database object which may be local or may, in fact, be a proxy for a remote database kernel and speak an RPC-based protocol to a remote Presto server.

3.3 Network Architecture

The ability to decompose the Presto run-time configuration into client and server components is also the basis for a more explicit remote network separation. Recall that our document model allows users not only to refer to other users' documents, but also to record attributes on those documents. This creates the notion of a document reference, comprising a set of attributes and a pointer to a remote document. As shown in figure 2, a document reference is manipulated as if it were a base document, and it appears to have all the base document attributes as well as the reference attributes

that a specific user has added. Using this mechanism, users can add attributes to documents to create their own structures, express their own interests and manage them according to their own criteria. Most documents have more users than they do authors, and so our system provides support for each document user to have their own view.

Document references involve remote connections between document spaces managed by individual Presto kernels. We also provide a means for queries to cross spaces. In our current approach, we have extended the query language to allow queries to be expressed in terms of *another user's* view of the document, with their local attributes. Thus, in addition to asking "what documents are interesting?" (which will return documents that have been tagged as interesting either by me or by their author), I can also ask "what documents does Anthony think are interesting" and perform a search which includes local reference attributes in Anthony's document space. This can be used to provide a basic form of collaborative filtering [Goldberg et al., 1992; Resnick and Varian, 1997].

3.4 Implementation

At the time of writing, our prototype Presto system comprises approximately 45000 lines of Java 1.1 code. This includes the core and network components, the NFS server, services (including email, HTML and Java), two browsers and a Lisp interpreter for scripting. The implementation uses JDBC, the Java Database Connection, to talk to any SQL database back-end (for attribute metadata), and the user interfaces are implemented using Swing, JavaSoft's implementation of the cross-platform Java Foundation Classes. The cross-platform nature of Java means that we can run the same Presto code files on PCs running Windows 95 or NT with a Microsoft Access back-end, and on Sun workstations running Solaris using either the public-domain MySQL or a commercial database (Oracle) as our back-end.

The overview presented in this section is intended to set context for the discussions in the rest of the paper. The following sections will describe the Presto system in more detail, focussing in particular on the relationship between architecture and interaction. We will separate the components of the system into two sorts—those aimed primarily at supporting our model of use and *interaction*, and those aimed primarily at supporting the *integration* of the Presto system into conventional everyday environments. This division, while not absolutely clean, helps to separate the intent behind elements of the design. Both aspects, however, are critical to supporting the style of interaction we want from the system.

4 SUPPORT FOR INTERACTION

The basic motivation behind our explorations in Presto is the desire to support a new form of interaction with large document spaces. This new approach is based on high level document attributes that are meaningful to users and are the primary resource for document typing, organisation, searching and retrieval. In this model, document attributes provide a uniform means to express document categories, groupings and desired behaviour, as well as to organise and search document repositories.

This style of interaction places a number of criteria on our design. The first is *performance*; since all document activity is managed through attributes, then attribute management must be fast enough to support interaction by direct manipulation. The second is *coherence*; our focus should not simply be on individual documents, but across sets of documents. A third criterion is perceptual *stability*;

although attributes, and hence document collections, are subject to continual change, no-one can use a system that is constantly changing under their feet.

These criteria are reflected in the design of those elements of our architecture that deal with document attributes.

4.1 The Document Layer

The Presto document layer provides a uniform model of documents and their metadata attributes. Essentially, the document layer has three functions:

1. It unifies access to these various back-end repositories with a single document model. Documents can be managed uniformly, wherever they are stored;
2. It introduces the document attribute mechanism and provides a means to attach, remove and search arbitrary document attributes via a single coherent mechanism;
3. It adds a unified document collection service, itself based on document attributes. Since collections are managed in terms of Presto documents, a single collection can group documents stored in heterogeneous repositories.

The document layer uses a back-end database service for persistent document metadata storage. In our implementation, we talk to this database service via JDBC so that our code is independent of the particular database product being used. While many attribute values are simple strings or string lists, attribute values are stored as serialised Java objects, so that arbitrarily complicated data structures can be recorded as document attributes. This is particularly important when document attributes are used to record application-specific information.

4.2 Document Collections

A document system organised entirely around individual documents would be, at best, tedious to use. Most interactions in Presto are with documents as elements of document *collections*. Along with filesystem documents and Web documents, document collections are implemented as a document type, and so they are subject to all the same operations that can be applied to documents (including having associated attributes, search and retrieval, and themselves being members of collections).

Presto was designed to act as a bridge between different document worlds — local storage, remote document services like the Web, application-specific document management like email folders, etc. As a result, our notion of collection needs to be able to capture the different grouping modalities that these document worlds offer, from simple directories to search engines.

In our implementation, document collections comprise three elements (each of which can be null). The first is a *query term*. Query terms are specified in terms of document attributes. Queries can test for the presence or absence of particular attributes on a document, can test the specific value of an attribute, or can perform type-specific value comparisons (for instance, a wide range of date specifications can be provided, such as “changed within 2 hours” and “modified last week”). Query terms in document collections are “live”—the collection contains the matching documents at any moment, so that documents may appear or disappear depending on their immediate state. Since these queries are embodied in collections, they are persistent.[2]

In addition to the query term, the document collection stores two lists of documents, called the *inclusion* and *exclusion* lists. Documents in the inclusion list are returned as members of the collection whether or not they match the query. Documents in the exclusion list are not returned as members of the collection even if they do match the query. When the query is null, the inclusion list effectively determines the collection contents.

So, the contents of the collection at any moment are the documents in the inclusion list, plus those matching the query, minus those in the exclusion list. We call these three-part structures "fluid collections". Our goal with this implementation of document collections was to support a natural style of document organisation and retrieval. A query can be used to create an initial collection, or to specify the default membership. However, membership can be refined without having to reformulate the query, but by direct manipulation of the document collection contents; dragging items into and out of the collection causes the system to add them to the inclusion and exclusion lists, so items can be added and removed to override the results of the query, and these changes will be persistent. Our browser, which will be described in more detail later, also supports the direct manipulation of query terms, so that reformulating the query is a fairly straight-forward operation.

Why introduce this three-part mechanism? Why not simply have manipulations of collection membership result in transformations of the query? There is no technical reason for our approach; it is a design decision based on a number of factors. We will give three here. First, we believe that "specifying exceptions" and "transforming the query" are, conceptually, different operations; our approach provides both rather than a one-size-fits-all approach. Second, we believe that indirect query transformation breaks the direct manipulation model around which our interfaces are built. Third, we believe that the indirect query transformations are very hard to explain to users; it is a considerable challenge to present a coherent account of the behaviour of such an interface without requiring a detailed understanding of query syntax, precedence rules and so forth. However, the design of refined query and collection mechanisms to suit the needs of specific applications is one area of ongoing research.

Collections, queries and attributes are the basis of all interactions with the Presto document space, and so the performance of the attribute engine was a key concern in the development of the system. Our database engine provides sufficiently crisp performance to support our requirements for interactive response. On a small test database (342 documents, 4911 attributes), evaluating the query "Mail.From=dourish" took 30ms to return 8 documents, while the query "MIME Type=text/html or MIME Type=text/java and read within 1 month" took 140ms to return 32 documents. The same queries on a large database (2558 documents and 27921 attributes) took 90ms (8 documents) and 620 ms (300 documents) respectively[3]. These are fast enough to avoid the "submit the query, await the results" model of interaction; instead, the interface can be updated rapidly enough to give the user a sense of directly exploring the data store. We will see later how this is incorporated into an interface.

4.3 Services

A document repository organised in terms of document attributes is only of use if the documents actually have attributes. There are two sources of attributes on documents.

2. Of course, simple one-time searches are also possible.

3. All times are the mean of 15 runs, and were measured on a 200MHz Pentium Pro workstation.

Firstly, attributes can come from the users. We allow users to attach arbitrary attributes to documents so that they can create their own structure. Indeed, the basic goal of the system is to allow users to create their own structures by creating sets of attributes relevant for their tasks and then using them to organise and retrieve documents.

The value of interaction through Presto comes when there are many attributes for each document. However, since it would be tiresome to have to tag all documents by hand. So as a second mechanism, we also provide a means for documents to be tagged with attributes automatically. Some document attributes are generic, such as their type, their length, their creator, the date they were created, and so forth, and these are obvious ones for us to maintain directly. Other relevant attributes might be content-specific. For instance, an email message can be tagged with information about its header contents; or an HTML file can be tagged with information from its header, or the other document links that it contains.

Presto provides a mechanism for processing files and deriving this sort of information. A *service* is a piece of code that can be used to analyse files in this way. We provide services for common structured file types such as email messages and HTML documents, as discussed above. We can also provide more specialised or complicated services; one example is a Java service which parses Java source files and can encode information about packages, imports and method definitions in attributes on the document. We also provide service wrappers for other pieces of software that exist outside the system. For instance, we have taken the document summarisation tool from a Xerox product and incorporated it into Presto through the service model, so that the document contents will be summarised, with key words and sentences made available as document attributes.

The service mechanism is also a route to building application-specific Presto spaces. One example is a processor that understands the format of the internal database of summer intern applicants to our research center; each application record document can be tagged with the applicant's skills and interests, information on their school, degree, topics of interest and so forth. The Presto system can then be used to analyse and organise the applicants. In this case, all the application-specific processing is localised in the service; there is no need for the Presto engine itself to be aware of any application details.

Services like these enhance interaction with the document space by increasing the number of available attributes for any given document. Using services to extract attributes from documents allows us to extract content information and encode it in the attribute-centric document structure, bridging from content-based to structural approaches.

Because we rely on services to provide this link, it is important that they be responsive to changes in document content. Services can be scheduled to run at various points in order to keep information up to date. Services might run at a particular time of day (e.g. doing major processing at 4am), at particular intervals (e.g. every ten minutes), or on particular events (e.g. when an attribute is added to the document, or when the document is written).

5 SUPPORT FOR INTEGRATION

A practical aspect of the everyday world is that document systems simply have to be integrated and extensible. Our system is, after all, intended to provide support for organising and searching exist-

ing document spaces, and existing document spaces employ a wide variety of formats, structures and applications.

The model of interaction that was our original motivation leads to new forms of document interaction, which will clearly be embodied by new applications, which can take advantage of the sorts of features Presto has to offer. At the same time, however, the need to support existing applications was a strong requirement for our system.

In order to accommodate existing data and applications as well as providing for the development of new ones, Presto offers three application interfaces (API, IOStreams and NFS), and provides a pluggable mechanism for integration with content repositories.

5.1 Support for Native Applications: API and IOStreams

The *Presto document interface* provides access to documents as Java objects. Applications can make use of this interface by importing the relevant package in their Java code, and coding to the API we provide for accessing documents, collections and attributes. This is the standard means to build new Presto-aware applications and to experiment with new interaction models. The supplied browsers (see below) can be regarded as Presto applications and are built at this level. The Presto document interface provides Document and Attribute classes, with specialised subclasses supporting all the functionality described here (such as collections, access to WWW documents, etc.) Applications can provide a direct view of Presto documents, perhaps with a content-specific visualisation, or can provide a wholly different interface, using Presto as a attribute-based document service back-end. (We will discuss applications experience in more detail later.)

Secondly, we provide access to Presto documents through the *Java IOStream interface*. Presto IOStreams subclass the standard Java streams model, and so make Presto functionality available to any standard Java application. In our implementation, we have made use of this model to incorporate Java Beans, such as for images and HTML files, that can provide access to document content without the overhead of starting a new application.

5.2 Support for Legacy Applications: NFS

Our third level of access is through a server implementing the NFS protocol. This is a native NFS server implementation in pure Java (and which runs on top of a pure Java RPC layer). The Presto NFS server provides access to the Presto document space to any NFS client; we use the server to allow legacy applications such as Microsoft Word to make use of Presto documents; on our PC's, Presto simply looks like another disk to these applications, while on our UNIX machines, Presto looks like part of the standard network filesystem.

Critically, though, what we achieve through this mechanism is that Presto is directly in the read/write path for legacy applications. The alternative approach would be to attempt to post-process files written to a traditional filesystem by applications, such as Word, that could not be changed to accommodate Presto. By instead providing a filesystem interface directly to these applications, we can ensure that relevant attributes (such as ones which record when the document was last used or modified) are kept up-to-date. Even though the application is written to use a filesystem, the Presto database remains up to date, because Presto *is* the filesystem.

As part of its interface to the Presto database layer, NFS provides access to the query mechanism. Appropriately formatted directory names are interpreted as queries, which appear to "contain" the

documents returned by the query, similarly to the "Semantic File System" approach of Gifford et al. [1991]. However, this is implemented largely for completeness; `cd` is not intended as a Presto interface for everyday use.

Although Presto provides this NFS service, Presto is not a storage layer. Documents actually live in other repositories. However, using the NFS layer provides uniform access to a variety of other repositories (so that documents available over the Web appear in the same space as documents in a networked file system). The combination of this uniformity along with the ability to update document attributes by being in the read and write path makes the NFS service a key component for our desired level of integration with familiar applications; without this Presto would be a demo tool rather than a working part of our environment.

5.2.1 Mapping The Document System to the File System

Although mapping the Presto document space into a virtual file space is a natural mechanism for integrating with legacy applications, it also introduces a number of technical problems. These stem from the fact that Presto's document semantics are richer than those of traditional filesystems, and in particular, that a document is not uniquely identified by a name and location in Presto, as it is in a filesystem. In Presto, a document may have multiple names, or no name at all, or may appear in many places, or in none. Applications that use filesystems as their storage layer expect to be able to exploit a set of constraints that do not hold in Presto. As a result, our NFS implementation has to go to some lengths to interpret the actions of filesystem applications so as to understand their intended consequences. We will consider two examples here.

Most document systems adopt a two-stage process to save files, so that the file will not be lost in the case of a write failure (e.g. a disk filling up). So, in order to save file FOO.TXT, an application might actually save the contents to a file called FOO.TMP. Once that file has been written successfully, it may delete the old file named FOO.TXT, and rename FOO.TMP to FOO.TXT. This way, should the write fail, then the old file contents have not been lost. This procedure relies on the assumption that a file's name in a given location is the entire source of the document's identity; a new file with the same name will take the place of the old file.

However, in Presto, document identity is more complicated than that; each newly created document has a separate identity. What is more, we have probably attached attributes to the document, and we would like them to be associated with the new document. We want to make it look as though the document content has been changed, but the document has remained the same. The Presto NFS server achieves this by essentially detecting the sequence of behaviour that the application is engaging in, and working behind the scenes to restore the filesystem invariants that would otherwise no longer apply. When the Presto NFS server receives a request to delete a document, it does not delete it, but simply hides it. If another file is written shortly afterwards with the same name, or another document is given the same name as the document that was recently deleted, the NFS server actually swaps the old document back in, updating its content from the new one. The effect is that, within Presto, the "right thing" seems to have happened; the application has updated the content of the original document, even though the application knows nothing about the Presto document model.

A second example concerns ancillary files. Some applications may store ancillary files along with an original; for instance, a word processor might want to maintain backup files, spelling dictionaries and linked image files along with the original document, and so it will store them in files in the same

folder. However, in Presto, “storing in the same folder” is not a straight-forward proposition. The original document might not appear in a folder at all, for instance; or it might appear in many folders, either statically (by inclusion) or dynamically (by query). Clearly, we do not want the ancillary files to appear in every collection that might dynamically hold the original document, and yet the application will fail if it does not find those files associated with the original document next time it is run.

This is another case where the Presto NFS server can interpret the application’s behaviour and work to restore the filesystem invariant that the application expects. In this case, the solution relies on the fact that users invoke operations on Presto documents, not on filenames, and so Presto is responsible for determining the filename that is actually given to the application. When a user runs an application such as a word processor on a Presto document, Presto creates a unique pathname for that document within the Presto NFS filespace. The filename looks like a normal folder, but in fact the folder name is tagged with an identifier for the document. This means that all attempts to create a new ancillary file will result in a “new file” request that contains, in the name of the file, the unique tag that identifies the base document. For example, if the user invokes an operation on document #456, then the filename that the application sees might be something like “/presto/#456/document”, and so the application’s ancillary file requests will be for files such as “/presto/#456/backup” or “/presto/#456/dictionary”. Since Presto has included the unique tag “#456” in the path name, then the NFS server knows that a new file called “/presto/#456/backup” *must* be associated with a document #456, since only an application processing that document could use that path name. Once this association has been made, the rest is straightforward. The Presto server records the association between the files as attributes of the ancillary documents, and makes them appear in the same virtual folder. This means that, in future, when an application is run on a document, the virtual folder will be recreated and all the associated ancillary files will appear there, satisfying the application requirement even though they do not actually appear in the Presto collections that hold the document.

5.3 Support for External Repositories

Integration with applications is all very well, but we also need to be able to work with existing data. In fact, Presto always works with existing data. Presto does not, itself, store content. Instead, our document objects record an association with content stored in an external repository.

Internally, Presto maintains a number of different document types that can interact with different repositories, such a filesystem documents and web documents. These distinctions are, however, largely invisible to Presto users. The user is not presented with a distinction between one document type another other; they see a seamless document model in which all documents support the same operations: reading and writing, adding them to collections, setting properties on them, searching for them and so forth. When a Presto user reads or writes the document’s content, the system automatically fetches content from whatever underlying repository stores it, and delivers it directly to the application.

The approach allows users to work with their own documents from within Presto easily. Another advantage is that, since the access protocols for different repositories are managed below the level of the document objects, documents can be mixed fluidly throughout the system. Documents can appear in the same collection despite the fact that they reside in completely different repositories,

in different servers, or are retrieved via different protocols. Similarly, all documents can be managed using the same uniform attribute operations.

Note that it is *documents* that are imported into Presto, not repositories. So, Presto is not intended to operate as a "browser" for the filesystem, or for the World Wide Web. Client programs that we call *importers* can be used to walk over a document tree and incorporate its contents as documents in Presto, but a Web document, for example, is not implicitly incorporated into the Presto space simply because we speak HTTP. So, document searches will not scour the web for new documents that have not already been incorporated into the Presto document space. The goal of our design is to be able to exploit existing content, rather than to provide a new view of an existing infrastructure. Mechanisms to incorporate external content more generally is one of the Placeless Documents project's current areas of investigation.

6 INTERACTING WITH PRESTO

The previous sections have introduced the major elements of the Presto architecture and shown how they support our design goals. However, Presto was designed as an infrastructure and framework for experimenting with the interaction aspects of attribute-based document management. Our first Presto-based applications, then, are ones which focus on how users can interact with documents in these sorts of spaces. The interface is where the features come together.

Although Presto includes many features that are centred around particular forms of user interaction (such as the collection design and the query mechanism), it is not designed as an integrated interactive system. Rather, it is an infrastructure over which a variety of interfaces can operate. Clearly, each application introduces its own particular interactive demands. However, the basic design of Presto results in a number of common design issues in any user interface. As has been mentioned in the discussion of legacy application support, for example, a number of traditional constraints and invariants, familiar from filesystem interfaces, no longer hold in a system like Presto. These features make themselves felt at the interface as well. In this section, we will discuss a number of issues that have arisen in the design of interfaces to the Presto document space, and some solutions that have been developed.

We are experimenting with a number of different interfaces to Presto. Some of these are application interfaces, designed for specific applications that make use of Presto as a document storage layer, but which are not, themselves, directly interfaces to Presto. Others, however, are relatively unconstrained browsers for the Presto document space. Here, we will consider one of these in particular, a browser called Vista.

6.1 Vista, a Presto Browser

Vista is a generic browser for Presto document spaces. It is not optimised for any specific application activity; instead, like a traditional PC desktop, it gives a generalised view of the document space, supporting organisational tasks and launching other applications.

The goal of Vista was to explore the issues arising in direct manipulation of the Presto document space, and in particular, how users could be given a sense of direct interaction with attributes and dynamic collections. Vista uses a combination of dynamic collections and multiple workspaces [Henderson and Card, 1986] to allow users to switch easily between different workspaces config-

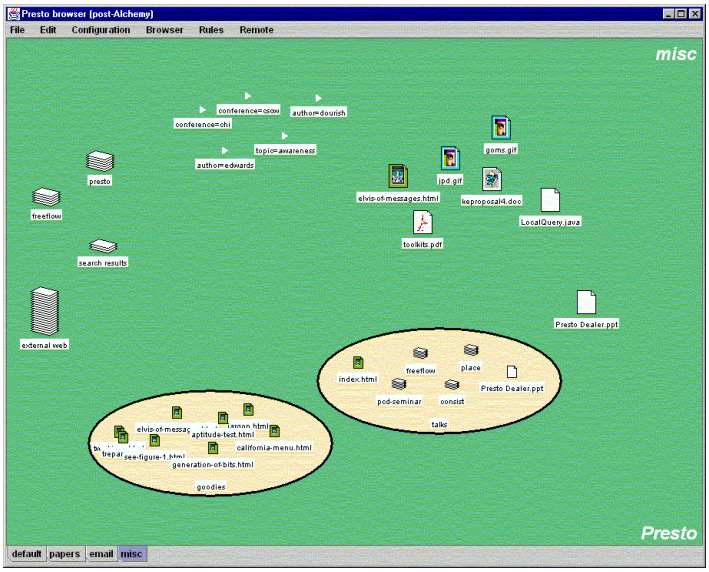

FIGURE 3: Vista, a prototype Presto browser. In addition to a number of documents, this particular desktop also contains four closed collections (shown as piles on the left), two opened collections (ovals in the lower portion of the screen), and a set of predefined attributes (triangles in upper middle).

ured for different tasks, with different representational structures for the same underlying document corpus.

Just as Presto in general uses attributes as a uniform mechanism for document operations, so Vista is organised primarily around the use of attributes. Attributes are used to control documents, to cluster and group them, and to search for them.

6.1.1 Elements of the Vista interface

Figure 3 shows Vista in use. The body of the interface is the current workspace; the tabs at the bottom of the frame move the user to other workspaces. In the workspace area, there are four basic entities being displayed: documents, attributes, collections and piles.

Documents are displayed as individual entities, and can be moved, deleted and launched. Launching a document, by double-clicking on it, invokes the relevant application for editing its content. There is no distinction in the interface between documents whose content resides in different repositories; they can be handled seamlessly. Double-clicking on a Word document loads its content into word whether the document is stored on my local disk, a file server, or the Web.

Document *collections* appear in two forms, opened and closed. Opened, they are displayed as ovals, showing the documents they contain. Closed, they appear as small icons depicting a “pile” of documents. Displaying closed collections as piles provides a natural means to give cues as to their size, which is particularly useful since fluid collections, backed by dynamic queries, can grow and shrink independent of user activity. Our use of piles is inspired by the prototype work done at Apple [Mander et al., 1992]; the support for “casual organisation” that drove their design is also a goal of ours; however, we have not yet implemented some of the richer features of the Apple piles prototypes, including “messy” and “neat” piles, as well as rippling through piles to search for documents.

Finally, individual *attributes* can also be stored as browser objects, and appear on the desktop (attributes are presented as triangles labelled with names and values). Although users can always use generic interfaces to add new attributes, Vista allows them to create sets of attributes that live

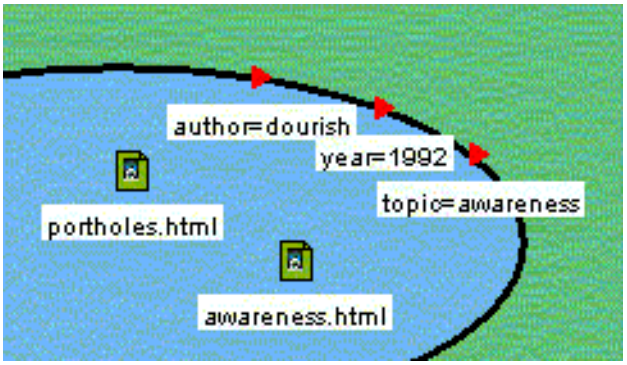

FIGURE 4: Attributes can be used to control the terms of a dynamic query.

on the desktop. Different attributes can be present in different workspaces, so that the workspaces can be tuned to different tasks; in one workspace, for example, I might have a set of attributes for organising my publications, while in another, I keep attributes that are relevant for programming tasks. Attributes have two roles in the browser interface. The first is that they can be dropped onto documents to add that particular attribute to the document, as might be expected. The second is that individual attributes can be used as query terms. Using attributes as query terms provides direct manipulation of queries, which we will now explore in more detail.

6.1.2 Directly Manipulating Fluid Collections

Recall that in the Presto document model, collections can not only contain specific other documents (including sub-collections), just like folders and directories in traditional filesystems, but that they also contain a query component, which specifies dynamic content. So, for any collection, we can specify a set of query terms which make up the dynamic portion of the collection's definition. Documents in the system that match the query will be included in the collection (unless they have been specifically excluded).

Query terms can specified by using a traditional dialog box interface, but also by direct manipulation, through attribute objects. Dragging an attribute onto a query collection adds it to the list of query terms for that collection. So, if the attribute is "project = presto", then that attribute can not only be added to documents, but can also be dropped onto a collection so that "project = presto" is added to the current set of query terms for that collection. Any number of attributes can be added to a query, providing simple conjunction; dragging them off again removes them from the query set.

Attribute icons representing the current set of query terms appear around the circumference of the collection object (as shown in figure 4). As these iconic representations of query terms are dragged on and off the query object, the query is updated in a separate thread. The result is that queries dynamically respond to the manipulation of query terms in real time, giving a very direct sense of the query as a configurable filter on the document space. It was an important element of the design that this direct manipulation of queries provide the sense of directly exploring a set of possible values, rather than the more traditional "submit the query, await the results" model; attribute searches are sufficiently fast that response rarely takes as much as a second. The effect of such direct manipulations is similar to that of UMD's dynamic queries [Ahlberg et al., 1992], although in our model we currently only provide for the addition and deletion of query terms, not the modification of query term parameters.

However, it is important that collections with a query component should still feel to the user like collections, not queries. The interactive style that this browser tries to capture is grounded in the manipulation of a document space, rather than the creation and execution of queries. Although the dynamic response of collections to the addition and deletion of query terms helps to give the sense of manipulation, we can also use the other components of collections, inclusion and exclusion lists, to help support the experience of manipulation.

Inclusion and exclusion lists in fluid collections lend them a feeling of stability that is critical to the interactive feel we are trying to support. So, in addition to the query component that dynamically maintains the collection contents, direct manipulation controls the use of the inclusion and exclusion lists to modify the results. Dragging a document out of a query collection causes it to be added to the exclusion list for the collection. This means that it won't reappear in the collection the next time the query is run (which happens regularly in the background). Similarly, dragging a document into a collection means that it should be added to the inclusion list, since it would otherwise not be included as an element of the collection. Using this mechanism, users can drop attributes onto a query window to create the query that expresses their basic set of interests, and then refine the result by adding or removing specific items. The resulting collections feel more like "real" entities than dynamically executed queries, but new documents of interest still turn up when they're added to the system.

One of the most radical changes that our attribute-based model introduces to the interactive experience and to the design of interfaces is that documents can appear in multiple places at the same time. In the model, documents can be members of multiple collections at the same time, and so two collections can display the same document concurrently. Performing a search that returns a document that is already displayed on the screen will result in it appearing twice, etc.

While care can be taken in the design of the interface to ensure that this does not become a problem (e.g. when adding a document to a collection in which it already appears), the notion that a document can be in multiple places at once tends to upset our intuitions about what facilities can be added to the interface. For instance, it might be nice to be able to draw lines between icons to illustrate relationships between linked documents; but when any document can appear on the screen multiple times, there is no stable and unique notion of "the icon for this document", and so the display of relationships quickly turns into a dense and tangled web.

Since it is part of the inherent structure of the domain, Vista allows documents to appear in multiple places in a workspace. However, it attempts not to allow a single document to appear twice in the same context (that is, on the same background pane, or in the same collection).

6.1.3 An Activity-Based Interface

Vista incorporates a workspace approach similar to Rooms; users can switch back and forth between different workspaces, each of which is a persistent view of the document store. Each workspace is a different desktop, but provides an interface to the same underlying database; a search, for instance, will return the same set of documents no matter which desktop launches it. So, the workspace model is not designed to manage different document corpora, but rather different activities that users might perform with those documents.

Workspaces can be customised to different activities by setting up dynamic collections and attributes that reflect the way documents should be organised to handle the tasks of that activity. For

example, in a “Papers” workspace, a user might have arranged a set of collections that dynamically select documents according to attributes such as as conference, year, venue and topic; perhaps the workspace has a couple of collections predefined for common queries, with attributes available so that new queries can be constructed quickly using the drag and drop approach. Meantime, another workspace might be organised around projects, and set up to allow quick and easy access to documents according to project needs (such as the projects they are associated with, or their current state). Of course, the same documents are available in each workspace; they reflect the different ways in which a user might want to approach documents based on the different activities they need to perform.

In each workspace, users can create new collections, move documents around, assign new attributes or perform queries, using traditional direct manipulation techniques. By providing very lightweight facilities to move from one workspace to another, this provides another way manage overlapping organisations for document collections.

6.1.4 Other Approaches

Again, we are exploring a variety of styles of interaction with documents in Presto; Vista is simply one example. Other interfaces have provided different facilities, or have emphasised one approach or another. One of our design goals was to enable exploration of different interactive design approaches. In addition to generic “browser” interfaces, we have also developed some applications that incorporate interactive features as part of their exploration of new ways to use attribute-based document management.

7 EXPERIENCES WITH APPLICATIONS

We firmly believe that the best way to learn from an experimental artifact like Presto is to put it into use, so we have been working with other projects at PARC to explore how they could make use of our infrastructure. In another paper, we discuss two specific projects that we have been working with [Dourish et al., 1999b]. Here, we will focus particularly on lessons learned.

One particular feature of Presto applications so far is that they reflect a duality in Presto’s conceptual design. The duality is that Presto is both a document system and an object system. Clearly, Presto stores documents, groups them together into collections, and allows users to read and write content using their conventional desktop applications such as Word, Powerpoint or Photoshop. The user experience is of Presto as a document system. On the other hand, it also provides a means to associate arbitrary data directly with documents through the attribute mechanism. The programmer experience is closer to that of a simple object system.

This “object system” perspective results in a new model for application development. Application data tends to be stored directly on documents, rather than in separate configuration files or application databases. Once the application data has been moved out onto the documents that the application is managing, so application functionality can also be moved out, by mapping it onto dynamic collections and queries over document attributes. This results in a model of “exploded” applications whose state and behaviour is distributed through the document space.

For example, consider an email application. In a conventional approach, if we wanted to build an email application, we would write a monolithic system capable of reading mail information from a server using a protocol like POP or IMAP, presenting the individual messages to users, managing

a set of folders, providing search facilities over the message headers, and storing private information about which messages had been read, replied to, forwarded and so on. When implemented on top of Presto, however, this takes on a very different feel. Access to an IMAP server can be provided through Presto's facility for interacting directly with different document storage repositories. This makes individual messages directly manipulable as Presto documents. Information about activities over this messages (such as how they have been filed and processed) can then be recorded as document attributes. This in turn allows dynamic collections can be defined for automatic categorisation into folders. At the same time, an application-specific service (such as the one already provided in Presto) can annotate the document with information from the headers so that email-specific searches can be performed using the generic search mechanism. In other words, our "application" in this case is actually a set of small individual components which bridge between the specifics of the application domain and the generic facilities provided by the Presto infrastructure.

8 RELATED WORK

We have discussed some related work in the course of introducing various aspects of our system; however, the relationship between our work and other research deserves further examination.

We have already alluded to work in the domain of information retrieval and visualisation investigating user-centered organisation for document spaces. For instance, BEAD [Chalmers, 1993] uses a landscape metaphor in which relative document positions are derived from content similarity metrics. BEAD does not support Presto's user-imposed structures. Most systems in this domain operate on the basis of content.

The Lifestreams system, originally developed at Yale [Freeman and Fertig, 1995], uses a timeline as the major organisational resource for managing document spaces. Like our approach, Lifestreams is inspired by the problems of a standard single-inheritance file hierarchy, and instead seeks to use contextual information to guide document retrieval. Unlike our approach, however, Lifestreams replaces one superordinate aspect of the document (its location in the hierarchy) with another (its location in the timeline).

The Semantic File System of Gifford et al. [1991] introduces the notion of "virtual directories" that are implemented as dynamic queries on databases of document characteristics. The goal of this work was to integrate an associating search/retrieval mechanism into a conventional (UNIX) filesystem. In addition, their query engine supports arbitrary "transducers" to generate data tables for different sorts of files. With the NFS server implementation, Presto essentially provides the same sort of functionality which was provided by the Semantic File System. However, Gifford and his colleagues were largely concerned with direct integration into a filesystem so that they could extend the richness of command line programming interfaces, and so they introduced no interface features at all other than the file name/query language syntax. In contrast, our concern is with how such an attribute-based system can be used in day to day work, and with how our interfaces can be revised and augmented to deal with it; the fact that Presto acts as a filesystem is simply in order to support legacy filesystem-based applications, rather than as an end in itself.

DLITE is the Stanford Digital Libraries Integrated Task Environment, a user interface for accessing digital library resources [Cousins et al., 1996]. DLITE explicitly reifies queries and search engines in order to provide users with direct access to dynamic collections. The goal of DLITE, however, is to provide a unified interface to a variety of search engines, rather than to create new models of

searching and retrieval. So although queries in DLITE are independent of particular search engines, they are not integrated with collections as a uniform organisational mechanism as they are in the Presto interfaces.

Like the developers of DLITE, we have been strongly inspired and influenced by ideas developed and expressed in various generations of the Self user interface [Chang and Ungar, 1993; Maloney and Smith, 1995]. In particular, their concern with providing a very concrete interactive experience is one we share whole-heartedly.

9 DISCUSSION AND DIRECTIONS

Presto is an early prototype, and is still in the process of development. We are currently engaged in developments in both the infrastructure and the interface.

We are investigating ways of enriching the interactive experience. Further support for working directly with the documents in the space is critical; this can be supported by incorporating thumbnail images and intermediate forms of document collections that reveal more details of the documents themselves. We are interested in adopting other aspects of the Apple piles prototype, such as using the "messiness" of the pile to convey aspects of the accumulation of documents. We are also developing mechanisms in Presto to support the creation of "toolsets" for particular tasks, allowing users to create customised sets of queries and attributes, etc. Magic lenses [Bier et al., 1993] provide one potential vehicle for managing these in an interface.

Similarly, we are trying to understand better the relationship between the Presto infrastructure and applications that might be developed and deployed on top of it. As we have discussed, Presto can serve different roles to applications, some of which challenge our intuitions about the structure of applications. A number of application development efforts are currently under way which explore this relationship in different ways.

These applications also provide us with an opportunity to understand how people might make use of an attribute-based document management system. Our design has been driven by a set of intuitions developed from the literature on the use of both paper and electronic documents, which is a currently active area of research (see, for example, Sellen and Harper, 1997; Adler et al., 1998; Bellotti et al., 1998). Part of our strategy in developing applications for deployment in our own environment is to study the role that document attributes can play in on-line document management. To this end, we are working with other groups at PARC to jointly develop task-specific applications based on the Presto infrastructure. In one project, for instance, we are especially interested in how applications can introduce structure into the attribute value space, and how they can coordinate their work through sharing and manipulation of a structure that describes the task-specific categorisations of document corpora [Dourish et al., 1999c]. This work draws upon previous investigations of the collaborative nature of customisation activity [MacLean et al., 1990; Trigg and Bodker, 1994], and current ethnographic investigations of document work practices [Trigg et al., 1999].

In general, our goal is to understand how attribute-based interaction can become a central part of the user experience in managing and organising document spaces. Presto is an infrastructure testbed directed towards this goal, and so it will continue to develop as we learn more about this new model of system interaction.

These goals are being pursued in the context of a larger project, called Placeless Documents, of which Presto is one component. Presto is essentially a prototype implementation of a subset of the functionality of Placeless Documents. The most significant features that Placeless Documents adds to Presto are, first, a more robust and scalable distributed architecture, and, second, active properties. In contrast to the static attributes provided by Presto, active properties can carry code with them which will be executed when operations are performed on the documents to which the active properties are attached. In this way, users of the Placeless Documents infrastructure can use properties not only to organise and manage files, as described here, but also to customise their behaviour and configure document services. Extending a Presto-like document management service to support active properties in addition to static attributes introduces a number of technical challenges; we describe our approach and current implementation in another paper [Dourish et al., 1999a].

10 CONCLUSIONS

Documents in the real world serve different purposes at different times for different people, but the systems we provide for people to store and organise electronic documents are typically based on rigid, uniform hierarchies. In the Placeless Documents project, we have been investigating how a document storage and management infrastructure can be based on document attributes rather than on names and locations. In our model, users interact with their document sets in terms of high-level attributes that capture aspects of the documents that are meaningful for everyday activities, and are provided and fluidly managed by the users themselves.

Presto is an experimental system that we are using to explore the issues of attribute-based document management in a system such as Placeless Documents that is organised entirely around attributes. Presto is not, itself, a document repository; instead, it provides uniform coordinated access to a variety of repositories such as networked fileservers and the World-Wide Web. Presto provides the means to organise and search document spaces in terms of document attributes. Attributes play an intermediate role between traditional structure-based retrieval schemes and more extensive content-based approaches; using Presto services, relevant aspects of document content can be handled as attributes, and used as query terms or collection criteria. Presto's architecture is designed for fast and fluid access to document spaces organised dynamically around attributes, and it was developed particularly to explore the interactional consequences of an attribute-centric approach.

Presto serves two roles. Its first role is as a proof-of-concept design for attribute-based document space management. In this capacity, it has been successful. Presto clearly demonstrates that an attribute-oriented mechanism for document sets numbering in the thousands can be made responsive enough to support direct interaction; that the service mechanism integrates system-manipulated and user-manipulated attributes flexibly enough to allow both to be used as part of the document space organisation; and that integrating attribute-based queries into enhanced "fluid" collections gives a concrete but malleable basis for document space organisation and interaction. Presto is already in use in a number of projects in our laboratory, supporting not only traditional document management activities but also more fluid interactive applications. Presto's second role is as a scouting expedition for other attribute-based document systems. In this capacity, we have learned a number of valuable lessons, including the value of document/object duality, the relationship between the document system and filesystems as a mechanism to support legacy applications, and a number of interface design concerns that arise specifically from the use of attributes as the primary means for document interaction.

Our initial experiences with Presto have been extremely positive, but of course, they are simply the first step. In collaboration with colleagues, we are currently developing our research program in two directions. At the infrastructure level, we are working on the development of a second-generation system in which document attributes can be used to control and manage a wider range of document activities in a distributed setting. At the same time, we are also considering how to specialise this form of interaction to particular tasks and particular domains, and developing Presto-based tools for a range of specific tasks and activities. We hope to use these investigations to ground our investigations of this novel form of document space interaction.

ACKNOWLEDGMENTS

Many other people have contributed to the development of these ideas. We would particularly like to thank the other members of Placeless Documents group – John Lamping, Karin Petersen, Doug Terry and Jim Thornton – for their significant contributions to the work described here. We also thank our early Presto customers for their forbearance and sense of adventure.

REFERENCES

1. Annette Adler, Anuj Gujar, Beverly Harrison, Kenton O'Hara and Abigail Sellen, "A Diary Study of Work-Related Reading: Design Implications for Digital Reading Devices", *Proc. ACM Conference on Human Factors in Computing Systems CHI'98* (Los Angeles, California), May 1998.
2. Christopher Ahlberg, Christopher Williamson and Ben Schneiderman, "Dynamic Queries: An Implementation and Evaluation", *Proc. ACM Conf. Human Factors in Computing Systems CHI'92* (Monterey, CA), May 1992.
3. Deborah Barreau and Bonnie Nardi, "Finding and Reminding: File Organization from the Desktop", *SIGCHI Bulletin*, 27(3), July 1995.
4. Victoria Bellotti, Annette Adler, Sara Bly and Katie Candland, "Papering Over the Cracks: Document Practices in the Networked Professional Workplace", Xerox PARC report, 1998.
5. Eric Bier, Maureen Stone, Ken Pier, William Buxton and Tony DeRose, "Toolglass and Magic Lenses: The See-Through Interface", in *Proc. SIGGRAPH'93* (Anaheim, CA), August 1993.
6. Matthew Chalmers, "Using a Landscape Metaphor to Represent a Corpus of Documents", *Proc. European Conference on Spatial Information Theory*, Elba, September 1993.
7. Bay-Wei Chang and David Ungar, "Animation: From Cartoons to the User Interface", *Proc. ACM Symposium on User Interface Software and Technology UIST'93* (Atlanta, GA), 1993.
8. Steve Cousins, Andreas Paepcke, Terry Winograd, Eric Bier and Ken Pier, "The Digital Library Integrated Task Environment", Technical Report SIDL-WP-1996-0049, Stanford Digital Libraries Project (Palo Alto, CA), 1996.
9. Paul Dourish and Graham Button, "On Technomethodology: Drawing Foundational Relationships between Ethnomethdology and Interactive System Design", *Human-Computer Interaction*, 13(4), 395–432, 1999.
10. Paul Dourish, Keith Edwards, Anthony LaMarca, John Lamping, Karin Petersen, Michael Salisbury, Douglas Terry and James Thornton, "Extending Document Management Systems with Active Properties", Xerox PARC report (submitted to ACM Transactions on Information Systems), 1999(a).
11. Paul Dourish, Keith Edwards, Anthony LaMarca and Michael Salisbury, "Using Properties for Uniform Interaction in the Presto Document System", *Proc. ACM Symposium on User Interface Software and Technology UIST'99* (Asheville, NC), 1999(b).
12. Paul Dourish, John Lamping and Tom Rodden, "Building Bridges: Customisation and Mutual Intelligibility in Shared Category Management", Xerox PARC report, Palo Alto, CA, 1999(c).
13. Scott Fertig, Eric Freeman and David Gelernter, "Finding and Reminding Reconsidered", *SIGCHI Bulletin*, 28(1), January 1996.

14. Eric Freeman and Scott Fertig, “Lifestreams: Organizing your Electronic Life”, *AAAI Fall Symposium: AI Applications in Knowledge Navigation and Retrieval* (Cambridge, MA), November 1995.
15. David Gifford, Pierre Jouvelot, Mark Sheldon and James O’Toole, “Semantic File Systems”, in *Proc. Thirteenth ACM Symposium on Operating Systems Principles* (Pacific Grove, CA), October 1991.
16. David Goldberg, David Nichols, Brian Oki and Douglas Terry, “Using Collaborative Filtering to Weave an Information Tapestry”, *Communications of the ACM*, 35(12), December 1992.
17. Austin Henderson and Stuart Card, “Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical Interface”, *ACM Transactions on Graphics*, 5(3), 211–243, 1986.
18. David Karger and Lynn Andrea Stein, “Haystack: Per-User Information Environments”, Technical Report, MIT Artificial Intelligence Laboratory (Cambridge, MA), February 1997.
19. Allan MacLean, Kathleen Carter, Thomas Moran and Lennart Lovstrand, “User-Tailorable Systems: Pressing the Issues with Buttons”, *Proc. ACM Conf. Human Factors in Computing Systems CHI’90* (Seattle, Washington), April 1990.
20. Richard Mander, Gitta Salomon and Yin Yin Wong, “A ‘Pile’ Metaphor for Supporting Casual Organization of Information”, *Proc. ACM Conf. Human Factors in Computing Systems CHI’92* (Monterey, CA), May 1992.
21. Tom Malone, “How Do People Organize Their Desks? Implications for the design of office information systems”, *ACM Transactions on Office Information Systems*, 1(1), 99–112, January 1983.
22. John Maloney and Randy Smith, “Directness and Liveness in the Morphic User Interface Construction Environment”, *Proc. ACM Symposium on User Interface Software and Technology UIST’95* (Pittsburgh, PA), 1995.
23. Thomas Phelps and Robert Wilensky, “Towards Active, Extensible, Networked Documents: Multivalent Architecture and Applications”, *Proc. ACM Conf. Digital Libraries DL’96* (Bethesda, MD), March 1996.
24. Paul Resnick and Hal Varian (eds), Recommender Systems, special issue of Communications of the ACM, 40(3), 1997.
25. George Robertson, Stuart Card and Jock Mackinlay, “Information Visualization Using 3D Interactive Animation”, Communications of the ACM, 36(4), April 1993.
26. Harvey Sacks, *Lectures on Conversation* (Jefferson, ed.), Blackwell, Cambridge, MA, 1996.
27. Abigail Sellen and Richard Harper, “Paper as an Analytic Resource for the Design of New Technologies”, *Proc. ACM Conference on Human Factors in Computing Systems CHI’97* (Atlanta, Georgia), April 1997.
28. Randy Trigg and Susanne Bodker, “From Implementation to Design: Tailoring and the Emergence of Systematization in CSCW”, *Proc. ACM Conference on Computer-Supported Cooperative Work CSCW’94* (Chapel Hill, North Carolina), October 1994.
29. Randy Trigg, Jeanette Blomberg and Lucy Suchman, “Moving Document Collections Online: The Evolution of a Shared Repository”, *Proc. European Conference on Computer-Supported Cooperative Work ECSCW’99* (Copenhagen, Denmark), September 1999.
30. Sun Microsystems, *Network File System Protocol Specification (RFC 1049)*, DDN Network Information Center, SRI International (Menlo Park, CA), March 1989.