1. It's All the Same to Me: Data Unification in Personal Information Management

DAVID R. KARGER[1]

1.1. Introduction and Description of Topic*

Information fragmentation is a pervasive problem in personal information management. Even a seemingly simple decision, such as whether to say "yes" to a dinner invitation, often depends upon information from several sources---a calendar, a paper flyer, web sites, a previous email conversation, etc. This information is fragmented by the very tools that have been designed to help us manage it. Applications often store their data in their own particular locations and representations, inaccessible to other applications. Consider the information Alex maintains about Brooke. He must keep Brooke's address in his address book, his picture in a photo album, his home page in his web bookmarks, a birthday invitation he is editing with her in his file system, and an appointment with her in his calendar.

This fragmentation causes numerous problems. There is no one "directory" Alex can use to find all the information about Brooke; nor any way to "link" pieces of information about Brooke to each other. Instead, Alex must launch multiple applications and perform numerous repetitive searches for relevant information, to say nothing of deciding which applications to look in. He may change data in one place (a new married name in the address book) and fail to change it elsewhere, leading to inconsistency that makes it even harder to find information (which name does Alex use to search the photo album?).

While the computer has fragmented information, it can also be used to put the pieces together again. This chapter surveys some of the ways in which our personal information might be better unified.

1.1.1. Motivation

Data unification offers many benefits to end users. The general motivation is that users often need to work simultaneously with several information objects in order to complete a given task. Here, we explore this motivation in greater detail.

One motivation for unification is that a user may need to observe several distinct information objects in order to draw conclusions about them. Looking at them one at a time can be slow and

[1] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02138. karger@mit.edu. http://theory.csail.mit.edu/~karger.

difficult, particularly if we must return to each several times---for example, whenwe must compare two information objects element by element. Even if direct comparison is unnecessary, shifting from one object to another to access different attributes can cause a user to "lose their place" and waste time finding it again when they return to an object. Given the value of seeing all the relevant information at the same time, it is unsurprising that almost all users have adopted window managers that let them display several application windows simultaneously.

A second motivation for unification is the desire to shift data easily from one application to another. At some point you have likely found yourseld reading information out of one application and manually entering it into another, when the two applications could not agree on a common representation for that information that would allow it to be transfered by a simple cut and paste operation. As a fortunate exception, most applications nowadays offer uniform cut, copy, and paste interaction with whatever *text* information they hold.

Copying data between applications is a sign that (possibly different aspects of) the same information is being managed by multiple applications. For example, a person might appear in a user's address book, and also as the creator of a photograph in the user's album. This creates several problems that can be fixed by (and therefore motivate) better unification. One arises when the user records the information in multiple places, and gets caught by inconsistent versioning when changes to one version of the data (a new married name, for example) do not propagate to other versions of the data.

Even when users record information in only one place, they may not remember which one. This can make it hard to find. For example, a user might record a person's birthday in their calendar, but then try and fail to look it up in his address book entry about that person. Conversely, if they first discovered that birthday in their calendar, and wanted to send a congratulatory email, they would have to look up the person's email in their address book. Arguably, this is a wasted search: if we are already looking at the person's name on the screen (in the calendar), why can't the application tell us their email address without further searching? We can understand this as a failure in two ways: first, there is no easy way to "link" from the information about the person in their address book to the information in the calendar (doing so would also help fight inconsistency). Alternatively, we can argue that the functionality of sending email should not be locked up in the address book, but should instead apply to any person we encounter in any application---calendar, photo album, and so on.

In a study of users' desktop environments (also discussed in Chapter 3 of this text), Ravasio et al. (2004) observed that users are themselves aware of this value of unification. They state that "the systematic separation of files, emails and bookmarks-was determined by three users to be inconvenient for their work. From their points of view, all their pieces of data formed one single body of information and the existing separation only complicated procedures like data backup, switching from old to new computers, and even searching for a specific piece of information. They also noted that this separation led to unwanted redundancy in the various storage locations."

Another motivation for unification is our need to manipulate multiple pieces of information in ways that cross application boundaries. Users like to gather information objects into groups that

will be used together, and organize or annotate those groups in ways that will make it possible to retrieve them based on their intended usage. The artificial separations imposed by separate application data models can interfere with this organizational urge.

Besides grouping information objects, users often want to record or exploit information that directly *links* multiple objects. Much of the information about a given object lies in its relationship to other objects; thus, when viewing one object, a user may well want to see those relationships, and may want to learn more about those other related objects. For example, Brooke may want to identify the creator (a relationship) of a given song, and may then want to explore that creator in order to learn about where they lived. When data is not unified, there may be no way to record the relationship between objects stored in different applications (except by redundantly duplicating the related objects in both applications, which has the drawbacks discussed above). The World Wide Web offers perhaps the most massive example of the value of linking information objects to indicate relationships between them and help users explore those relationships.

As another motivation for linking, recent work highlights users' preferences for finding information by *orienteering* (Teevan et al., 2004, and Chapter 3 of this text). Rather than jumping directly to needed information, users often try to locate it by starting with a known object and taking repeated navigation steps to related objects, aiming to home in on the desired information. We use this approach frequently when navigating the web, or when seeking files in our directory hierarchies. To support such orienteering, it is necessary to connect information objects to other related ones. When data is fragmented, it may be impossible to record some of those linkages, meaning that they will be absent when the user needs them for orienteering. Again, Ravasio et al. indicate that users are aware of their desire for linking: “most interviewees expressed the need to have their information linked together (e.g. article author and respective address book entry, or citation and cited article, etc.) and in general, to have more content-based and context-based access to their information” (Ravasio et al. 2004).

1.1.2. Approaches to Unification

The multitude of motivations for unification is matched by a multitude of different approaches to unification that have been developed in the past or are currently being investigated. We will survey and taxonomize this variety of approaches.

At its root, unification aims to bring information together for some useful purpose. To achieve this goal, any unification approach *must* choose some “least common denominator” into which it can shoehorn *all* the information it aims to encompass. Although an application need not store its information in any particular standard form, it must offer hooks through which its data can be treated as instances of the common information representation. For if information is to be unified, there must be some implicit or explicit contract about how that information is modelled and manipulated. There are many possible common denominators, each imposing different constraints and offering different benefits. Modern window managers treat all applications as rectangular regions of pixels, so they can be placed next to each other and manipulated on the screen. File systems treat each data object as a collection of bytes, so it can be read and written

(though perhaps not understood) by any application, or attached to an email without constraint on its type. The World Wide Web assumes that every object has a URL, which can therefore be easily referred to on any web page.

Using such a common denominator is in tension with applications' needs for rich, specialized representations of their content. Rich representations let applications offer powerful domain-specific operations. The tradeoff is that a simplified shared representation lets applications interact in a unified fashion with data from many applications and domains without needing to understand a multitude of rich representations.

In taxonomizing different approaches to unification, we ask the following questions about each unification paradigm:

- **What is its API?** What is the commonality that it assumes about all the data that is to be unified? Into what common model/representation does all the data fit? What primitive operations does that common model support? What does an application have to offer if its data is to be part of the unification, and what can it assume about data offered by other applications?
- **What can you do with it?** What information that user cares about can fit in the common model? What information management activities can be performed via invocation of the API primitives?
- **Why is it useful?** How do the supported information and activities enhance the users' ability to work with information?
- **What are its limitations?** Where does the unification fail to provide what the user needs? What information will still need to be stored in an application-specific fashion because it does not fit in the model?

Below, we will explore three major categories of unification, outlined in Table 1:

- **Visual Unification** aims to place multiple data objects in view side by side, and is one benefit offered by modern window managers that support multiple applications. Visual unification lets users see and relate the multiple objects that are relevant to a given task, but offers no mechanism for directly indicating, or letting users manipulate, data reltionships between objects that are managed by different tools.
- **Standard Data Types** such as text and files have relatively undemanding semantics (sequence of characters or bytes) and are pervasive, so they tend to be supported by almost all applications. Thus, one application can move another application's file, or paste text copied from another application. The undemanding semantics is the strength of this approach (because every application is willing to support these types) and its weakness (because much of information's rich meaning is lost when it is squeezed into these semantics-poor representations).
- **Metadata** treats information objects as opaque, but offers a standard model for talking *about* those objects in assorted ways. Examples include grouping (as in file directories), annotating (as in ID3 tags for media and del.icio.us tags for Web pages) and linking (as in

the World Wide Web). As we will argue below, much information management relies only on object metadata, so can be supported over data objects of all types. While current metadata standards (such as ID3) are limited to specific application data-types, new frameworks such as XML and RDF offer the possibility of unifying substantial portions of people's information spaces using only metadata.

Table 1: Different unification techniques are offered by different software systems. Each unification supports different types of operations and enables different data management activities by the end user.

technique		offered by	operations	enable
visual unification		window manager, wincuts, embedding	layout, tile, show, hide	simultaneous view of information
standard datatypes		text, files	cut/copy/paste, read/write	unified searching, data transfer
metadata			refer, describe	list below
	grouping	directories, taskmaster, Presto	add/remove items from group	organize, browse, simultaneous view
	cross-reference	OLE, COM WWW	embed, traverse	simultaneous view, orienteer
	attribute/value	XML, Presto	annotate, query	annotate, search, organize, browse
	relations	RDF, Databases, Haystack		record relationships, unified search

1.1.3. Overview

Having motivated and taxonomized unification, we will now explore in detail a variety of unification approaches that have been developed, and discuss the tools that make use of them. Then we will turn to considering a number of research efforts that are exploring new ways to unify information. Among others we discuss Wincuts, an effort to improve visual unification of information; the World Wide Web, the greatest example of the power of linking; the Presto system for organizing files using arbitrary metadata annotations; the Taskmaster system, which tries to turn email into an environment for unified information management; and the Haystack system, which emphasizes the power of recording, displaying, and navigating arbitrary relationships between arbitrary information objects.

One type of unification that we raise just to set aside is *physical* unification of information. The information a user works on is often stored in several physically distinct locations. It is a significant burden for a user to move physically from location to location to get the information they need. Thus, in the absence of appropriate communication mechanisms, it may be effectively impossible for a user to access and unify some remote piece of data. For cases where

communication *is* possible, protocols such as FTP (for data transfer) and X windows (for display transfer) have long existed to bring data from where it is to where a user needs it. More recently, tools such as network file systems (NFS) and the World Wide Web have tried to free the user from even having to think about the physical location of the desired data. A user can visit to a directory on a networked file system in a way that is indistinguishable from a directory on the local machine, and a user can click on a web link and navigate to the target without considering the machine that is delivering it.

While recognizing that it is a prerequisite for much of the unification we discuss, we will generally ignore the issue of physically remote information in order to avoid drifting into a consideration of networking and operating system protocols.

1.2. Visual Unification

The first strategy we consider, unification at the display, relies on the fact that most of today's applications run in desktop environments where the final step in the presentation of information to a user is rendering it into pixels on a graphical display. We can therefore choose, as the common denominator for all these applications, rectangular groups of pixels, and ask how such groups of pixels can be unified. The window manager is the current solution. Applications run inside one or several windows, and delegate to the window manager the control over those windows being stacked, tiled, moved, resized, opened, and closed. Thanks to the window manager, a user can simultaneously view and manipulate all the information objects they care about, in multiple application windows laid out side by side on the display.

On the downside, it often seems that each application wants the entire display to itself. To get at the data managed by an application, users typically need to launch the entire application, which lives in a large window with its attendant menus, toolbars, jumping-off points, and default presentations. Because the window manager treats the application opaquely as a rectangle full of pixels, it cannot select out the one piece of the display that the user actually cares about. A common consequence is *window clutter*---a desktop filled with tens of windows, all obscuring each other, each bearing a small fragment of information that a user cares about. To get at it, users must continuously locate and rearrange windows to find the fragments they need. One often sees significant effort invested in laying out windows just right, so that the desired information in one window obscures only unimportant information in another window. But this requires a significant investment of effort, is unlikely to remain the "right" layout as the user continues to manipulate the information, will be lost when the applications are exited, and is often simply impossible due to the topological constraints imposed by the location of the key information in each window. One coping mechanism, virtual desktops (Henderson and Card, 1986), allows a user to maintain a distinct desktop for each "context" or task that they are currently involved with, holding only windows relevant to that context. Often, however, even the windows of a single context are too much for a single desktop. And frameworks for actually managing the contexts remain primitive.

As a further drawback, even if users can get the window layout exactly right, this only gives them a convenient *view* of the information. Since only the display, and not the underlying data model, is unified, each piece of information must still be managed by the application responsible for it. Since each window is managed by a different application, there is still no guarantee that the user will be able to effectively manipulate information across the application boundaries. A window manager does not offer expanded opportunities to pass data from one application to another, or to create machine-usable linkages between data from multiple applications. The fact that Alex can display Connie's address in an address book and, at the same time, display a photograph of Connie in a photo album offers no guarantee that he will be able to use either application to associate Connie with her photo.

1.2.1. WinCuts

One significant limitation of the window-manager approach is its "all or nothing" management of application displays. When an application window is open, one sees the whole window, even if only a small portion of it is relevant to the given task. Tan et al. (2004) have addressed this problem by developing *wincuts.* The idea of wincuts is that a user can select a (rectangular) sub-region of an application's window, "cut" it out of the application, and "paste" it into an entirely different region of the display. The user can then close the original application window while leaving the wincut open, now displaying only the portion of the application that is of interest at the present time. The wincut is alive: application updates of the display are propagated to the wincut, and user-interface actions within the bounds of the wincut region are mapped back (indistinguishably to the application) as if they were applied within the application window. Reducing the screen space occupied by one application means that more applications can be visually unified.

Perhaps the greatest appeal of the wincuts approach to unification is its universality: essentially *all* current desktop applications work through the window manager, and are thus amenable to with wincuts approach, without any application modification whatsoever. Conversely, wincuts is very much aimed at graphical displays: it is not clear what analogous approach might work with speech interfaces, for example.

Wincuts suffer from some of the same limitations as traditional window managers. The information layouts persist only as long as their application is running. Thus it is difficult for a user to permanently modify the way information is presented to them; instead, they must re-customize it each time they tackle a given task. Also as in window managers, the views, not the data, are unified, so cross-application manipulation, annotation, or linking of the data is not possible.

1.2.2. Embedding

Besides the window manager, we find that applications nowadays will often manage subwindows of their own, *embedding* another application in some subregion of their display to support visualization of data managed by that application. Historically, Microsoft has offered this capability in their Office products under the name *OLE---Object Linking and Embedding.*

Nowadays, many plugins are available for Web browsers as tools to handle, inline, a variety of datatypes that are encountered on the Web, including various video datatypes and flash. As with window managers, the containing application generally cedes all control of the embedded region to a different application; this means that the data can be displayed and manipulated but not linked to the data of the containing application. And such embedding is often not under the user's control: the content embedded in a web page is determined by the creator of that web page, not by the reader.

A particular example of an embedding-oriented application is the Web Montage system (Anderson and Horvitz, 2002). Web Montage lets users assemble a single page containing embedded versions of numerous other web pages, so that the user can see all of them simultaneously---it is effectively embedding html documents in html documents. Because many pages are squeezed into one, each is cropped so that all can fit; the details of crop-sizing can be controlled by the end user. Web Montage offers a capability that is already present in the desktop window manager---in theory, Web Montages could be created simply by launching, sizing, and tiling multiple web-browser windows and navigating to each page of the montage in a separate window. Unlike the window manager, however, Web Montage is able to persist both the content and the layout of its aggregation after the application exits. This is a significant benefit for content that a user expects to interact with repeatedly. Such persistent aggregation of views is also offered by various "portal" web sites such as Yahoo, which allow users to build complex pages by choosing from a menu of possible inclusions.

Web Montage and other web portals achieve view persistence by limiting their content to web pages, unlike window managers which unify the display of anything. In Web Montage, the data to be embedded can be named by a URL. Because there is no standard model by which applications expose "what they are currently looking at," it is not feasible for a window manager to offer Web Montage's persistent visualization over an arbitrary class of applications and data.

1.3. Unification by Standard Data Types

We have discussed the benefits of a unified display, but observed that they are only "skin deep"---that the simultaneous visualization of different information fragments need not offer any way to unify those fragments as data. To achieve this deeper level of unification, we need a *data* common denominator. Here we identify standard data representations that are so pervasive that any application developer feels compelled to support them in any application. Historically, several such standard data types including text and files have played an important role in data unification. Looking ahead, it appears that XML and possibly RDF will offer richer standard data types that offer better unification of a variety of information.

1.3.1. Text

A significant portion of the data managed by many applications is text. It may come formatted in many different ways: within HTML documents on the web, in bulleted lists in Microsoft Powerpoint, formatted in a word processing document, or typeset in Adobe PDF. But strip away the formatting, and one is left with a sequence of characters.

When a data type is shared in this way by all applications, it becomes possible to easily transfer information from one application to another, achieving one of the unification goals stated above. Most applications are able to offer text to and accept text from other applications---typically with the assistance of a clipboard application with which they all communicate. The quotes offered earlier in this chapter could therefore be "cut" from Adobe Reader and "pasted" into an Emacs word processing buffer. The typical linear flow of text means that even interfaces for those operations are somewhat standardized---mouse drags from the beginning to the end select the text to be cut, and clicks within the text indicate the "insertion point" of text to be pasted.

Of course, this textual "lowest common denominator" is necessarily low. One generally loses finer aspects of the information when it is represented this way---the specific nesting of a Powerpoint slide, or the serifs and precise inter-character spacing in a PDF document. But this is likely the reason for the success of the standard: any more complex representation would have been too demanding for many applications to support.

The common model of text also means that most applications can contribute some (stripped down) representation of their data to a *text search engine*. Such an approach gives the end user a well-understood framework for searching by content over all of their data, independent of which application owns any given piece. Significant enthusiasm has developed around the recent set of "desktop search engines" such as Google desktop and MSN desktop that offer this capability. These systems are discussed in detail in the following chapter.

This support is not universal---had the PDF source document instead been postscript, which does not provide a text model, it would have been necessary to manually retype the quotes instead of cutting and pasting them. Another unfortunate omission is error messages: it is often impossible, when a dialog box pops up with an error message, to cut and paste its text into an email to technical support.

1.3.2. The File System

The other obvious common data format is that of the file. All applications can make use of the notion of a sequence of bits that can be read from, copied, or written to. An email application can "attach" a file for delivery to another computer without any understanding of which application created the file or what its contents mean.

The file system also offers a unified interface for the end user. Using the now-standard model of hierarchical directories or folders, a user can organize files according to whatever principle they find useful. For example, a user may gather into a single directory all the files necessary for accomplishing a particular task, regardless of which applications manage those files. Working

inside that directory gives the user immediate access to all of those files.[2] The file system lets users name individual files and list the names of files in a directory, an important aid to organizing and searching. Most applications let users specify files to open or save.

On the negative side, the semantics of files are so weak that, unlike text, they offer relatively little opportunity for data sharing. Except on a planned case by case basis, one application is generally unable to construct meaning from the bits it reads out of a file written by another application. Significant effort goes into writing *converters* that translate a file written by one application into one that can be understood by another. The lack of standard file semantics also means that any significant manipulation of any file requires launching an appropriate application. This will take a user away from the directory view of all files relevant to a given task, and back to an application view that shows only some of the information they want to work with.

Additionally, files are typically large. An address book, for example, will usually be stored as a single file, rather than as one file per entry. This precludes using the file system for fine-grained organization. A user will not be able to put, into a directory aimed at writing a particular paper, the address-book entries of all his co-authors. He can certainly place a link to the address book, but from within that directory he would need to launch the address book and then go through an address-book specific process to find the co-authors.

1.3.3. XML

Recently, enthusiasm has grown for XML as another potential standard data type for unification and sharing of data. XML offers the possibility of standardizing a richer structure than files' bit sequences. XML offers a standard syntax and model for representing arbitrary hierarchically structured information. For example, XML can use its standard syntax and model to represent that a playlist is made of a collection of songs, that each song consists of a media file, a title, a genre, and a creator, and that a creator has of a name, an age, and a birthplace. This richer representation holds out the possibility of transferring data from one application to another and having the second application take advantage of some piece of the hierarchical structure that makes sense to it. For example, given a media file, an address book may be able to work with some address-oriented parts of the "author" object, even if it cannot work with the media file at all. XML's standard for representing attributes of information objects also offers useful benefits for unified organization and searching, as we will discuss below.

XML does not solve the entire unification problem, because any given XML fragment obeys some *schema* that characterizes its different parts, and any tool aiming to work with that fragment needs to understand the schema. Thus, a media application looking for a "creator" fragment in an XML document could be stymied if some other application instead recorded an "author" fragment---these two descriptive terms may mean the same thing, but there is no way to tell just by looking at the data. Someone must still take responsibility for unifying different schemas that talk about the same information. However, the fact that different pieces of the

[2] While directory organization may seem inextricably entwined with the file system, we will see below that directories are more accurately characterized as a form of *metadata-based* unification.

structure are clearly indicated means that an application only needs translation for the specific fragments it aims to use, so the translation problem becomes smaller and local. Furthermore, XML may lead to the emergence of standard schemata---for example, a consensus that one should always use the term "author" to talk about the creator of a document---that become dominant schemata the same way that text and files have become dominant data formats. Agreeing on names for particular fields seems less demanding than agreeing byte-for-byte on file formats for all applications' data. In particular, someone writing an application can use the standard schemas where they are relevant (e.g., for an author), and simultaneously use idiosyncratic schemas for representing the nonstandard parts of the data they work with (e.g., color of the ink). The standard XML syntax means that tools that understand the standard schemas will be able to work with the standard elements, while ignoring the idiosyncratic ones.

1.3.4. SQL Databases

In the corporate world, relational databases with SQL interfaces are the dominant paradigm for handling the massive amounts of data companies need to manage. As corporations merge or begin to cooperate, they invariably need to merge the incompatible databases they have maintained. The field of information integration is concerned with developing tools and procedures for making such mergers easy, and making queries to the merged data efficient. While somewhat connected to the data unification themes we are discussing here, information integration is focused more on dealing with the process of forcing together heterogeneous repositories, as opposed to developing frameworks within which an individual user's data can be kept from becoming heterogeneous in the first place.

The database community has argued for decades that we would all be better off storing all our personal information in (personal) databases. This clearly has not happened, most likely due to the apparent complexity of interacting with a database. No one has yet come forward with applications that hide the complexity of installing and maintaining a database, designing the schemas for the data to be stored, and creating the queries that will return the desired information. And people seem generally allergic to having all their information presented to them as lists of tuples. It is noteworthy, however, that both XML (discussed above) and RDF (discussed below) have representational power equal to that of databases; thus, the work on PIM tools that make use of these formats may indicate that we are creeping up on the database community's perspective from a different direction.

1.4. Unification by Metadata

Perhaps the simplest general unification strategy is to ignore the complex structure of objects themselves and instead to record *metadata* that talks about the objects from the outside. In this section, we will discuss three powerful uses of metadata: *grouping* related information objects together (as in file directories), *annotating* objects with interesting attributes and values (*e.g.*, recording the title and composer of a particular song in ID3 tags), and *linking* complex objects to each other (as we do with pages on the World Wide Web).

1.4.1. What's in a name?

While a metadata scheme can ignore the complex formats of the objects it is talking about, there must still be a common API for talking about those objects. In particular, any metadata scheme needs some standard way to indicate *which* object is being talked about. This may be a standard scheme for *naming* objects (e.g. file names or URLs) so that their names can be used to talk about them, or it might be accomplished by *embedding* the metadata with the object being annotated (as with ID3 tags in media files).

While embedded metadata is common, it tends to belong to a single application or data type: Microsoft office manages embedded author and title information for its documents; MP3 software manages ID3 tags; email programs understand Sender and Date headers. In a sense, these embedded metadata become part of the complex data object, and are understood by the programs that understand those objects. Name-based metadata seems more often to span multiple applications: for example, many applications including web browsers, word processors, email clients, and music players offer special handling of URLs. This is likely because naming requires only understanding of the names, while embedding requires understanding the file format of the object that contains the embedded metadata. Also, if embedded metadata talks about multiple objects, it may need to be embedded in multiple locations, and can therefore become inconsistent as we discussed in the Introduction.

Technically, unification by naming objects is already often available, in that users can use various text fields in their applications to refer to other objects by names that make sense to them. For example, a user might set the filename of a photograph to include the name of the individual in the photograph, or the location where that photograph was taken. Or, they might use the "memo" field in an accounting application to name the individual who received a certain payment.

But this form of unification burdens the user with all the work of interpreting the name and then remembering exactly how to retrieve the named information. The user will have much less work to do if names are designed to be machine readable, and applications provide built in support for working with named objects. For example, many applications today will make URLs or email addresses "live"---clicking on them will launch a web browser or email client to view the given URL or compose a message to the given address.

Although we have mainly set the issue aside, it is also worth noting that a name-based scheme lets us talk about and organize information that is not actually accessible: for example, a web page can link to and discuss another web site that is temporarily down.

Besides referring to the objects, there needs to be some standard way to describe the metadata itself. We will discuss several standards below, including XML and RDF.

1.4.2. Grouping

Grouping is a powerful organizational technique that requires only naming the entities that make up the group. This technique is pervasive. For decades, the file system has exploited the idea of named opaque objects. Since every file has a unique file identifier, tools such as the directory

manager (and graphical interfaces for it), with no need to interpret any file's contents, can let users manipulate their file organizations. The file system places no restriction on the types of files that can be in its directories. The same folder interface typically used to manage these directories, with its well understood click-to-open and drag-to-move semantics, is also used by many applications to manages the groups that arise in those applications: mail messages by a mail program, bookmarks by a web browser, songs in a media application, people in a contact manager, and so on. In each case, the applications manage only "their" data. But the typical organization steps---inserting, deleting, and moving an object from one collection to another---require only a reference to the object, not any understanding of its internals. If all these objects were subject to a common naming scheme, it would be possible to group files and all these other information types without any change to the user interface. This would allow the data currently separated by application boundaries to be jointly organized. A user could create a "directory" containing some files, some email messages, a few relevant address book entries, some payment records from an accounting program, and so on.

Recently, another interesting use of grouping has appeared in the "social tagging" web services. Tools like Delicious (for web pages) and flickr (for photographs) let users *tag* objects with text terms germane to those objects. Users can then navigate to the group of objects that have received a particular tag, or perform queries to locate objects with all of a set of tags (thus intersecting groups). Tagging has begun to appear as an option on mainstream web sites such as Amazon.

Although tagging offers a unified mechanism for grouping arbitrary web pages, it also highlights a potential unification failure in grouping: failure to agree on the name of a group. A single tag might seem plausible for two distinct groups (e.g. "apple" for computers or fruit) or a single group might plausibly be named by several tags (e.g. "hci" or "chi" for human computer interaction). When there is more than one possibility, a user may run into trouble if they make inconsistent choices during organization and retrieval. Delicious explicitly recommends tagging a page with all plausible tags, so that any one of them will locate the page later. This failure of unification in tagging has been noted (Guy and Tonkin 2006), and debate continues about whether it is a bug (because it impairs retrieval) or a feature (because it allows for serendipitous discovery of new information).

1.4.3. Annotation

Annotation is another organizational technique that, in principle, depends only on the ability to refer to the subject of the annotation. Everywhere we manage information, we find tools that let us associate certain *attributes* and certain *values* of those attributes with our information objects. Files have creators, creation times, modification times, and types. Email messages have senders and subjects. Music files have genres, composers, artists, and bit-rates. Photographs have subjects, resolutions, and shooting conditions.

Annotations are extremely useful for organization and search. Users will often identify an object by some of its metadata ("the email that Brooke sent me last week") so will be able to find it easily when search over that metadata is supported. Metadata also provides an opportunity to

make information coherent for browsing---media management tools will offer music organized by artist, and subdivided by album. As was the case with grouping, many applications offer almost identical frameworks for using their metadata: tabular lists with one column per data attribute, sorting on a given attribute by clicking on the attribute's column header, and search dialogue boxes that let users specify and filter by certain metadata attributes. All of these steps require access *only* to the metadata and not to the annotated object.

Ironically, given that metadata is generally extrinsic---it talks "about" the object without being part of the object---many metadata reprentations are embedded, in domain specific ways, in the objects they manage. For example, ID3 tags are stored in a specialized format as part of the music file they are annotating.[3] A similar situation holds for images.

The emerging XML standard[4] may help to resolve this problem. XML offers a natural syntax for recording metadata about objects. To the extent that metadata representation becomes standardized, it may become possible to offer unified organizational tools across data types---such as tabular metadata presentations that explore music and photographs at the same time.

While there appears to be only one notion of "group," the set of possible attribures is potentially unlimited. As we discussed in Section 1.3.3, divergent choices of annotations can lead to a new kind of fragmentation---even if the representation of the annotations can be understood, the annotations themselves may not be. However, as we discussed there, the simple standards such as XML for representing annotations mean that an application can seek out and manipulate the annotations it understands even when they are mixed with other unknown annotations.

1.4.4. Linking

Often, discussions of metadata emphasize the attachment of some "literal" information---a date, a name, or a price---to some complex object. But there is also great value in linking pairs of complex objects. Links serve as a richer version of metadata---allowing one to connect a mail message not just to the email address of the sender (a string) but to an entire rich address book entry. Beside simply providing the information about the connection, links provide a natural step for users to follow from one information object to another. This in turn supports the popular *orienteering* strategy we discussed in the introduction: navigating from object to related object in order to home in on the desired information (Teevan et al., 2004, and Chapter 3 of this text).

Ravasio et al. (2004) indicate that users explicitly desire linking: "most interviewees expressed the need to have their information linked together, e.g. article author and respective address book entry, or citation and cited article, etc."

Probably the most recent big success of this linking approach was the World Wide Web. One of the most important contributions of the WWW was to define a single, shared URL namespace

[3] This can cause some interesting problems in practice. The act of playing a music file, which seems like a read-only operation, can actually cause that file to change (as the ``last played'' metadata is modified). This in turn can cause backup or synchronization systems to make a new copy of the entire file, based on those few-byte changes.
[4] http://www.w3.org/XML

that lets users craft names for arbitrary web objects. By placing references to those objects in other web pages, authors give users the ability to navigate smoothly from object to related object.

As with annotation, many tools support linking of complex objects within their own domain. A mail program may let you navigate to the address book entry describing a sender by clicking on the mail header, or navigate to a web page by clicking on a URL in an email message. But the absence of a single standard hampers cross-application linking. It is not currently possible to link from the composer of a music file to their address book entry, or from the location of a meeting in one's calendar to a map of that location, unless a specific application has decided to offer that specific functionality.

1.4.5. RDF

RDF, a relatively new model being propounded by the World Wide Web Consortium[5], may help us take better global advantage of the linking paradigm. Central to RDF is the perspective that *anything*, not just web pages, can receive a *URN* (essentially a URL, but not necessarily "fetchable" on the web) so that it can be referred to elsewhere. Like XML, RDF can be used to record the values of arbitrary attributes of a given object. But while XML typically records attributes that are literal values such as strings or numbers, RDF can record other URNs as values. While it would be typical, using XML, to record that the person in a given photo was named "Connie", RDF might instead be used to record that the person in the photo is the one identified by the URN "urn:a74fj83jfn". The URN offers two key advantages. First, it removes ambiguity: to seek an address associated with the name "Connie", Alex might need to do a search in his address book, look over the multiple individuals named "Connie", and decide which one he intended at this time. But the URN identifies a single individual: with an address book that supports URNs, Alex could get the right address with a single click and no cognitive effort. The second benefit is to separate the (machine-usable) URN identifier from a human being's preferred descriptive name. In the introduction, we discussed how the change to a new married name could create inconsistency that makes it harder to find information; using a URN to identify the person ensures that a (human) name change does not break the association between photo and individual.

1.5. Systems that Unify

Many researchers have realized the opportunities data unification can offer. Below we discuss a number of systems that explore this concept. We have already discussed wincuts and Web Montage in the course of introducing various unification approaches; here we discuss several other systems and the way they apply the unification techniques we have discussed.

[5] http://www.w3.org/TR/rdf-primer

1.5.1. The World Wide Web

Much of the impact of the World Wide Web can be attributed to its success in unifying a tremendous range of information. The Web rides on the concept of the universal (we might say unified) resource locator, or URL: every web page has one, and any web page can use one to link to any other web page. As the web grew, many different entities---institutions, people, books, recipes, and so on---became represented as HTML documents on the web, linked to each other with no restrictions as to type. The standard representation as HTML meant that a single tool, the web browser, could present all the different information types. Also, the fact that much of the content was text meant that search engines could be built to search the entire web, years before anything similar was attempted for the end-user desktop.

The names linked to in web pages can refer to objects that a web browser could not interpret; this failure becomes apparent only if the user chooses to actually navigate to the named object. This works quite well in an environment where some users have and other have not installed plug-ins and extensions to handle a variety of new object types. Users can see references to and discussions of the named objects even without the extensions, and can then make individual choices about whether it is worth the work of extending their own environment to handle the new object type.

With this appealing unification tool already present, we can ask why it has not been adopted as the primary environment for personal information management. Hypothetically, a user could create a separate web page for each email message, each directory, each music file, each calendar appointment, each individual in their address book, and so on. Editing these pages, the user could indicate arbitrary relationships between their information objects. Feeding these web pages to a tool like Google would give users powerful search capabilities; combining them with the orienteering opportunities offered by the user-created links would surely enhance users' ability to locate information.

In response, we observe that HTML is quite an impoverished data representation, offering little more than a rich formatting of text and images. Users' rich, structured, personal information spaces can certainly be “boiled down” to HTML for presentation, just as they can be boiled down to pixels, but such a representation loses much of the semantics of the underlying data. While an application could hypothetically store its richly structured information in HTML, it would have to select canonical rules for representing it, and would not be able to cope with a user or another application making modifications to other valid HTML that does not obey the canonical rules. At the interface level, while the web browser is certainly a good tool for *browsing* data, it is often too limited a tool for *manipulating* data. Web forms are useful only for relatively primitive data entry, not the sophisticated manipulations offered by applications' carefully specialized interfaces and operations. This demonstrates that while the web may seem unified at first glance, there are fundamental ways in which the web’s data remains highly fragmented---there are few tools that let us bring together the pieces of it that we want.

1.5.2. Presto

Presto *(Dourish et al., 2000)* is one system that tries to give users a powerful unified data organization environment. Dourish et al. realized that, as discussed above, many of the operations that users want to apply when organizing information, such as grouping, annotating, and linking, are independent of specific document types, and can be offered by a system that treats the documents themselves as opaque objects. They also observed that the file system's typical model of a hierarchical directory structure, with each file occupying a single position in the directory structure, fails to let users organize documents in natural ways.

Presto operates over (references to) arbitrary files, and lets end users define arbitrary *attributes*, represented as attribute/value pairs such as "author/David R. Karger" that can annotate documents and that can then be used to group or search for them. Such attributes can be extracted by type-specific automated tools. Alternatively, a simple drag and drop interface lets users associate new attributes with any document. Presto operates in the same way over locally and remotely stored documents, since when organizing based on extrinsic attribute/value pairs, the question of where a document is located is as unimportant as the precise details of the document's internal representation. As Dourish et al. state: "The user is not presented with a distinction between one document type and another; 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.... 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."

Presto gives substantial attention to the management of collections. Users can create collections by defining queries over certain attributes, and can also explicitly place documents in collections. These collections can be shown open (displaying the documents in them) or closed, where they are represented as "piles" whose size indicates the size of the collection. Because collections are often defined dynamically, documents can easily appear in multiple collections. Indeed, since all collections are considered somewhat fluid, there is no real "main" collection containing a document, and no distinction between the "true containment" (hard links) and aliases (soft links) that are distinguished in traditional file systems.

Presto also offers a task focused "workspace" model similar to that in the Xerox Rooms system (Henderson and Card, 1986) to help reduce clutter. Individuals can set up a different workspace for each task, and populate the workspace with those documents (and attributes) that are relevant to the given task.

On the downside, Presto continues to focus on the document as the entity that should be annotated; it does not appear to consider working with finer-grained information objects such as individual address book entries or appointments. Presto also does not support the linking of complex object by relations: the value of an attributes is always a primitive value such as a number or string.

1.5.3. Taskmaster

Bellotti et al. (2003) performed field studies that demonstrated how end users have shoehorned a substantial portion of their data inside their email clients---a kind of ad hoc unification. Bellotti et al. then developed a new email client, Taskmaster, with this usage in mind, offering a more effective unified environment centered on email. Taskmaster is discussed in greater depth in Chapter 11.

As the Presto group did in defining task-specific workspaces, Taskmaster takes a task-centric view of the world. Taskmaster lets end users define and manage "thrasks" (task threads) containing all the information items relevant to a given task: the email messages, their attachments, and any other files that become relevant at some later time. Taskmaster thus breaks through two levels of opacity that typically interfere with the organization of email: that the email clients offer their own data representation (so that the desktop environment cannot manage individual email messages) and that attachments stay locked up inside individual email messages (so that even if each email message were stored as a file, the desktop environment still could not manage the attachments as individual files to be organized). In Taskmaster, a user tackling a given task has direct access to any individual relevant item (through a folder holding all the thrask items). A user can move a particular attachment to a different thrask than its containing message, and have the attachment persist even after the containing message is deleted, and can add tast-relevant items that did not originate as attachments.

In addition to grouping, Taskmaster allows the user to attach metadata such as "deadline" or "action" to each information object, regardless of type, and to use these attributes to organize and retrieve information in the system.

Thrasks could contain not only files (email and attachments) but also links to objects on the web. Taskmaster was therefore able to blend data that was typically managed separately by the file system and the bookmark manager---an explicit wish of some of the users in the study by Ravasio et al. (2004).

Taskmaster offers a "preview pane" that presents the currently-selected information object (as displayed by its application) without launching a new window. As discussed above this kind of "embedding" lets a user see information about neighboring objects without leaving the context that they are currently occupying.

1.5.4. Haystack

Haystack (Huynh et al. 2002, Sinha and Karger 2004, Bakshi and Karger 2005) takes the notion of "opaque information objects with attributes an links" to a much finer grain than individual files or email messages. We observe that even inside individual applications, much of the information management reflects the creation and usage of (binary) relationships connecting information objects. For example, people are a data type appearing in various applications that manage their relationships to email messages (as senders in an email application) to music (as composers in a jukebox program) and to appointments (as people to meet in a calendar program). For some users (in the entertainment industry) those sets of people might overlap. It is therefore worth exposing the individual information objects, and the relationships between them, in a

unified manner, and allowing those exposed objects to be annotated, organized, and linked independent of their "home" application.

As was discussed above, the only real prerequisites for such an approach are to give each information object a unique identifying name and to choose a representation for the metadata. Haystack uses RDF[6], an emerging World Wide Web standard model for naming information objects and for recording relationships about those objects. In RDF, any object can be given a Unique Resource Name (with a syntax similar to URLs), and any two information objects can be linked by connecting their URIs with a *statement* naming (in a machine readable fashion) the relationship between them. In the Haystack data model, a typical file will be shredded into many individual information objects of various types connected by application-specific relationships.

Haystack's user interface is responsible for taking these many small objects and assembling them into traditional-looking information displays. But since each information object visible in, say, an email management interface is itself a distinct entity in the data model, a Haystack user can operate directly on any object in view. Haystack gives the user a web-like navigation paradigm: by clicking on (say) the author of the message, the user can navigate to a view of that author (which is constructed by looking up important objects related to that person, and laying them out in the style of an address book). Users can benefit from being able to orienteer from an email message to its authors, to a photograph of that person, to a representation of the location where that photograph was taken, to a map of that location, and so on. Similarly, traditional drag-and-drop operations can be applied by users to create collections of related objects, or to create annotations linking information objects together.

Haystack also offers Presto-like specification of collections of arbitrary objects (not just files) using queries over the metadata about those objects; Alex could assemble a "Birthday folder" containing all the people coming to Edna's party, all the bills not yet paid to the caterer, and the predicated weather, and have that folder stay up to date as RSVPs come in, payments go out, and the weathermen change their minds. This is a capability that Haystack shares with Semex, discussed in Chapter 8. However, we believe that basic linking, viewing, and orienteering are likely to make up a more substantial part of most users' interactions than complex queries, so Haystack's user interface focuses on enabling those more elementary behaviors---Haystack aims to feel less like a database, and more like an application.

Tools like Presto and Taskmaster continue to work with a fixed set of information types (files in Presto, email, projects and task in taskmaster). Haystack focuses almost entirely on the metadata surrounding objects, and thus does not need to understand anything about the objects themselves. Thus, Haystack is able to smoothly support the inclusion of arbitrary new types of information by an end user. Anyone interested in knitting can easily introduce objects representing different types of yarn, different weaves, and different needles, and craft presentations of those objects that integrate smoothly with the rest of their information space. Thus, unification is not limited to preexisting data types, but can stretch to incorporate whatever a given user considers important.

[6] http://www.w3.org/TR/rdf-primer

1.6. Conclusion

We have described some of the benefits of offering users a unified data model, one in which information is not firmly partitioned into the domains of distinct applications but can instead be viewed, grouped, annotated and linked flexibly according to the end-user's needs. We have argued that the key step for such unification is the choice of an appropriate common API---a representation of the data and interfaces to it that can be accessed by all applications. Such an API must be simple enough not to discourage application builders from supporting it. We have touched on a few such APIs: pixels, text, files, groups, annotations, and links. We have discussed the represntations thos APIs offer and the actions they enable.

We have discussed a number of unification-oriented research projects. One commonality we observe is a move towards *finer granularity*: grabbing small fragments of windows in WinCuts, breaking up objects into fine-grained hierarchical representations in XML, and referring to arbitrarily small objects in RDF and Haystack. Such approaches appear to recognize that it is not possible to predict and design a single monolithic data model or presentation that will be appropriate in all circumstances, so users should be able to work with the small pieces they actually need at a given time.

1.6.1. Looking Forward

Given that users would like to cross domains in their organizations, and that the interfaces are already often nearly identical in each domain, one can ask why we do not already see unified organizational tools. As one possible answer, note that to expose their data to be organized by such unified tools, applications need some standard naming scheme by which they can refer to their data items. One such scheme would be file identifiers. But this would require that each data item be stored as a file, which could be highly inefficient if the items are numerous, small, and frequently changing.

The two new related data representations, XML and RDF, hold out a possible solution to this problem. XML offers a standard syntax and model for defining a hierarchical nesting of information objects, their attributes, and values, while RDF offers a model for representing arbitrary information objects connected by arbitrary relationships. Either of these models can represent heterogeneous grouping, and both implicitly assume that the individual information objects may be small and numerous, requiring support from tools more like databases than file systems. Most importantly, in each of these models, the individual information objects can be referred to individually by machine-readable names. Using XML or RDF, an email program can expose its individual emails, a calendar program its individual appointments and todo items, an address book its individual addresses, and a document its individual authors and sentences, to be organized or annotated into heterogeneous task-specific collections.

Some, when faced with the idea of such a unified organizational framework, have objected that it could have a significant downside, as all information is suddenly squashed into a “one size fits all” (and therefore ill-fitting) organizational structure. It is important to distinguish, however, between a *unified* organizational system and a *single* organizational system. Even if it is possible

to create heterogeneous collections, there is still likely to be value in seeing a homogeneous collection of, say, all of a user's email. Since references to an individual items can appear in multiple places, we can simultaneously maintain today's application-driven data partitions (to the extent that they are useful) and also offer task-specific, cross-application collections of the same information.

It is important recognize that unification is not a complete solution, but an *enabler* of solutions. A unified data model does no good if it sits intert on disk; interfaces must be designed to exploit the unified model. Much research is ongoing to find the right metaphor or paradigm for letting people interact with their unified information. Lifestreams () offers a purely chronological metaphor, giving users access to all the information they used at a given past time. Presto sticks to our current files and folders metaphor of organization. ContactMap () centers information organization around the people with whom one is interacting to accomplish tasks. Taskmaster, discussed above and in Chapter 11, aims to frame all its information management as a generalization of email interaction. The Universal Labeller (Jones et al., 2005) uses "projects" as the framing device, without wrapping them in email like Taskmaster---the authors argue that planning a project, and breaking it down into subprojects with annotations about planned actions, provides an effective information organization. The Umea system (Kaptelinin, 2003) instead tries to identify projects implicitly, by observing the user, and gathering together all the information they seem to be using consistently at one time. Haystack adopts the web metaphor of individual information elements linked for orienteering.

All these metaphors are worth an entire additional chapter of discussion, and the question of which metaphor "works," and whether more than one is needed, is central to PIM research going forward. Common to all of them, however, is the recognition that the partition of information induced by diverse data formats and solitary applications is *not* the organizational metaphor consistent with people's uses of their information, and that a unified data model is a critical contribution towards organizing and presenting information to people the way they want it.

1.7. References*

Anderson, C. R. and Horvitz, E. (2002). Web montage: a dynamic personalized start page. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 704--712, New York, NY, USA. ACM Press.

Bakshi, K. and Karger, D. (2005). End-user application development for the semantic web. In *SemanticDesktop2005 Workshop, International Semantic Web Conference (ISWC).*

Bellotti, V., Ducheneaut, N., Howard, M., and Smith, I. (2003). Taking email to task: The design and evaluation of a task management centered email tool. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, volume 5.

Dourish, P., Edwards, W. K., LaMarca, A., Lamping, J., Petersen, K., Salisbury, M., Terry, D. B., and Thornton, J. (2000). Extending document management systems with user-specific active properties. *ACM Transactions on Information Systems*, 18(2):140--170.

Guy, M. and Tonkin, E. (2006). Folksonomies: Tidying up tags? *Dlib*, 12(1). Digital publication, available at http://www.dlib.org/dlib/january06/guy/01guy.html.

Henderson, Jr., D. A. and Card, S. (1986). Rooms: the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface. *ACM Trans. Graph.*, 5(3):211--243.

Huynh, D., Karger, D., and Quan, D. (2002). Haystack: A platform for creating, organizing, and visualizing information using {RDF}. In *Semantic Web Workshop at WWW2002*, Hawaii.

Jones, W., Munat, C., & Bruce, H. (2005). The Universal Labeler: Plan the project and let your information follow. In A. Grove (Ed.), *68th Annual Meeting of the American Society for Information Science and Technology (ASIST 2005).* Charlotte, NC: American Society for Information Science & Technology.

Kaptelinin, V. (2003, April). Integrating tools and tasks: UMEA: translating interaction histories into project contexts. *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2003),* Ft. Lauderdale, FL.

Ravasio, P., Schär, S. G., and Krueger, H. (2004). In pursuit of desktop evolution: User problems and practices with modern desktop systems. *ACM Trans. Comput.-Hum. Interact.*, 11(2):156--180.

Sinha, V. and Karger, D. R. (2004). Magnet: Supporting navigation in semistructured data environments. In *Conference on Innovative Database Research (CIDR), 2004.*

Tan, D. S., Meyers, B., and Czerwinski, M. (2004). {WinCuts}: Manipulating arbitrary window regions for more effective use of screen space. In *Extended Abstracts of Proceedings of ACM Human Factors in Computing Systems CHI 2004*, pages 1525--1528.

Teevan, J., Alvarado, C., Ackerman, M. S., and Karger, D. R. (2004). The perfect search engine is not enough: a study of orienteering behavior in directed search. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 415--422. ACM Press.