What is a File?

Richard Harper, Eno Thereska, Siân Lindley
and Richard Banks, Phil Gosset, William Odom[+], Gavin Smyth, Eryn Whitworth[*]

Microsoft Research Ltd. Technical Report MSR-TR-2011-109

ABSTRACT
For over 40 years the notion of the *file*, as devised by pioneers in the field of computing, has proved robust and has remained unchallenged. Yet this concept is not a given, but serves as a boundary object between users and engineers. In the current landscape, this boundary is showing signs of slippage, and we propose the boundary object be reconstituted. New abstractions of file are needed, which reflect what users seek to do with their digital data, and which allow engineers to solve the networking, storage and data management problems that ensue when files move from the PC on to the networked world of today. We suggest that one aspect of this adaptation is to encompass metadata within a file abstraction; another has to do what such a shift would mean for enduring user actions such as 'copy' and 'delete' applicable to the deriving file types. We finish by arguing that there is an especial need to support the notion of 'ownership' that adequately serves both users and engineers as they engage with the world of networked sociality.

Author Keywords
File, file systems, databases, cloud computing, grammar of action, metadata, generic object, ownership, possession, command, social networking, consumer device.

INTRODUCTION
It would seem perfectly reasonable for the man or woman on the street to assume that the term file, when used in connection to a PC, reflects quite closely what they mean when they use the same term to describe the things they have in their non-digital world. One might be digital and the other real, but they are the same sort of things: *file*.

Unfortunately the relationship is not as simple as this – and indeed to think it might be or that there is even any straightforward analogue between the use of this word in everyday life and with regard to computing is quite egregious. It turns out the word covers many things; a number of things in ordinary life and a rather different set of things when used in reference to computers. All told so many things, in fact, that the way that it gets used to cross refer between these domains (and indeed sometimes within these domains) often leads to solecisms – to absurd meanings and mistaken understandings.

Our purpose in this paper is not to list all such misunderstandings, nor to offer correctives to each, even if that were possible. Our goal is to suggest that something more profound is at issue, and that once that is understood a way forward is possible. It seems to us that part of the problem here has to do with the fact that both sides in this equation, the everyday ordinary user and the computer scientist, misunderstand each other by dint of sharing this much used word, file. And, perhaps more surprisingly, the word opens up further misunderstandings for systems designers, those who are engineering systems to which this concept is central. The consequences of the misunderstandings that derive are, we think, greater today than they have ever been. New forms of computer devices and networked systems are emerging and these are offering increasingly more options for 'where' and 'how to' file; 'what' is being filed and is being labelled as something 'to file' is altering too (with documents, music, images and sometimes even postings being treated as file types, for example). User practices are also altering. Key to this has been a shift from a single user negotiating with a file abstraction representative of data stored on a hard disk, to a situation where users are negotiating with other people over a file abstraction where the data itself may be stored in many places: on the local disk, on a cloud server, or on a social network site. What is of interest to the user in the latter case may not so much be the file itself, but the social life the file lets them, the user, have.

This is to put it simply. But if this holds true, then a new set of definitions for the term file is required; ones that will allow computer scientists to engineer what users require and provide a meaningful base for those users to act upon. This definition will consist of an abstraction that mediates between these two communities, the user and the engineer, allowing the user to treat content in a particular way, as a file, and for the engineer to structure the underlying protocols and data management issues as efficiently as possible. This new understanding will, it hardly needs saying, go hand in hand with developments and refinements

Main contacts: {r.harper, etheres, sianl}@microsoft.com

[+] *Carnegie Mellon University* [*] *University of Texas at Austin*
Work done while interning with Microsoft Research Ltd.

in related terms applicable to systems, though our concern in this paper will be with the file, first and foremost.

More particularly, our goal will be to propose how a refined definition of the term can be part of a larger effort to define a *grammar of action* for an HCI of the 21st century, where the practices of users are profoundly more social in nature than they were when the term 'file' was first coined. This seems quite a bold statement, especially for a single paper. Given this, we will structure the paper as follows.

First, we will presage the bulk of our argument with some careful illustration of the issues we have in mind. The reader will have to bear with us during this section, since the explanation will be somewhat ahistorical and, even though it is quite detailed, it is still just a gloss on the issues. Nevertheless we hope that this illustration will be sufficient to set the groundwork for the subsequent discussions. Here more historical characterisation as well as more technical detail about file systems and user behaviour will be provided.

The first of these will look at *file abstractions*, as a boundary object mediating between the user and the engineer. An instantiation of these abstractions in operating systems led to *file systems*, which emphasised primarily individual units of data, or a *file*. Then we will look at early attempts to move away from file systems to database systems. The latter can be seen as emphasising relationships among files, rather than the file itself, in that the unit of most interest was the type of relationship itself.

We shall then turn to the current world: the world of Cloud storage, social networks and applications like Facebook and Flickr, as well as to a world in which users have multiple devices and access points to a file. We shall remark on some of the practices that are emerging in this diverse context, with users undertaking activities that look like filing and other activities where the nomenclature ought to be different and indeed in some instances already is. We shall illustrate some of the difficulties the old file abstraction has created. These difficulties attest, we argue, not to an incorrect model of a file, so much as to the stresses that emerge when engineers and users try to orientate to the same model, one that ought to allow the seamless integration across and between applications and devices – but when the model turns out to have different assumptions for each party. We shall propose that the problem of copying a file between say, a PC and the Cloud, is harking back to a conception of a file that was applicable for a world where that and that alone made sense. Now that interaction is only one part of a much more complex and diverse set of practices and technological possibilities. The role of a file as a boundary object is thus losing its efficacy, we suggest.

A conclusion will outline what we think might be a way forward given this context, a way forward that might allow designers to offer systems that reflect what users are seeking to do and which can be engineered so that the prosaic but nevertheless essential concerns about where this more social data might be stored, how it might be accessed, copied, moved, made secure and so on, might also be dealt with effectively. In short, the goal will be to ensure that user and engineering practices can be better understood and enabled through a more nuanced vocabulary. By this point in the argument it should be clear, we hope, that this can be achieved only in part by redefining what is meant by the word file and the doings associated with it; other terms will be required to reflect other aspects of the new sociality of computer use.

SETTING THE SCENE

In everyday life the term file is something of a catch all. It is used for things that are written but, whether that means written by oneself, or someone else (a sender, perhaps) is another matter. The term is also (and often) used to describe things that don't have any words in them at all – pictures, say. The owner might use the word file when referring to where things are, even to refer to 'what' they are. 'File' then, is a word for things that are documented, one might conclude, whether the documentation in question is visible or textual. It is simply the name for different sorts: words about something, written on paper, cards, envelopes, pictures and other materials that are kept [5].

Yet all things that are documents in this sense do not therefore have the same status, not for the ordinary person. Some files are kept because 'they need to be' - in case they get used in the future, for example. Copies of guarantees come to mind. Others get kept, meanwhile, for sentimental reasons: pictures come to mind for this. In the latter case, some files are kept because throwing them away would be sad, somehow; in the former for reasons of a bureaucratic nature. But one should not forget also that some things are kept, are *filed* as the saying has it, for *no reason at all*: simply because the person responsible for them couldn't be bothered to sort them out or throw them away.

As a noun, then, the term file covers a range of things. Yet the word is also a verb, referring to the status of these same things and what that implies about various doings. People do not just have a file (or a set of files), they *make* a file; they have things that have to be filed - though one might add that the actual doing of this act is as likely to be something aspired to as achieved. After all, one often hears people complain about the need to do their files when they know that there will never be enough time to do the doing.

The above suggests a certain *ad hoc* and impromptu use of the term file in everyday life, though one can be sure that people don't ordinarily have problems with this: it's a complex term, one can hear them say, albeit labelling practical things. One might propose that all these uses of the term are connected by a family relationship, as Wittgenstein might have put it in his studies of words [12]. Notwithstanding the insights of ordinary language

philosophy, one can imagine ordinary people sometimes mistaking the use of the word as a noun with the use of it as a verb: 'Oh, I thought you were just telling me about the files, not suggesting I sort them out'.

Computer scientists, if one can treat them as a single group for the moment, though doubtless familiar with this everyday practice, use the word in a different way in their own business, when engineering systems. They use it to label things that need to be handled quite carefully, and the doings that they are thinking of when they use the word are themselves quite particular and specialised [7].

For computer scientists, the stuff they deal with, data if you like in the form of bits and bytes, exist in an environment where those data (leaving aside what they consist of for the moment) can disappear. So unlike people's dealings with stuff in the 'real world', computer scientists need to cater for the practical reality that computer systems fail. They fail in part and they fail in totality; disk sectors can develop defects as a case in point (and this may mean that part of the digital store for a file is faulty) and sometimes disks can fail altogether (in which case remote back-up is required). Given this, data within this system are so arranged that if part of them is destroyed or even the whole, this event does not catastrophically affect the system's functioning, its utility. The architecture duplicates data, so that if some is lost (as it is assumed will happen), then a copy can be used as a replacement. This in turn means that there has to be a system element to manage these duplications: this keeps these versions up to date after changes are made, for example, and knows where they are stored.

Computer scientists have come up with a name for the data they are worried most about: a *file*; their name for the structure of the systems, a *file system*. A file in this sense is the name given to what one might think of as an *abstraction*; indeed this is the term computer scientist's use to label this way of thinking. In this view, a 'file' is *not* the stuff stored, *it is a way of bundling that stuff* into identifiable materials that in turn can be managed, stored, retrieved by the computer system and its component applications, the OS and so forth.

Some of these actions are mirrored in the things that users can see (and do) and some are not - though we shall come to say more about the user's point of view shortly. For example, with the abstraction in hand, a system can allow a file to be 'read' – and thus seen by the user, to put it crudely, even though the system also has the task of bringing together the thing-as-seen by the user from the various parts stored in different places. Likewise, when a user 'saves', the system 'writes' that data (i.e., the elements constituted as an abstract level as 'a file') somewhere and this means the system stores that data. Or to put this another way (though still simply), when a user 'saves', the system 'writes', when the user 'opens' a file, the system 'reads' that file, and so forth (See Fig 1.).

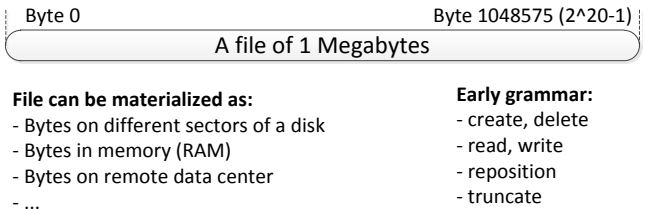

Figure 1: operating system view of a file abstraction as a consecutive, uninterrupted range of bytes ([7] pages 372-372). The abstraction relieves users from understanding how the bytes that make up a file are scattered across memory or disk, giving them a simple logical object to operate on.

It is important to bear in mind that all of this work, all these doings are not necessarily seen or understood by users. Indeed, questions of where bytes are stored are systematically hidden from view. But the term abstraction labels how computer scientists view the issues at hand, not how the users see it. One might say that users have a different abstraction, one that reflects what they think of as the properties of a thing that is a file as constituted in everyday life.

One property of that might be, for example, that the thing is unitary, a single entity, not a composite. Of course, in some instances, a file might be made up of bits brought together from other files – though again, those other files might be singular. What is more important is to note that, at the point of interaction, at the interface between the user and the system and thus, to put it crudely, at the interface as understood by HCI, what the user sees and what the computer scientist sees is not the same thing: they are looking from different points of view and constituting what they see differently.

To give a second example, for computer scientists, all user data is considered identically precious. All things are equal in the world of digital files, whatever that data might be. In contrast and as we have noted, the user has a range of objects in mind when they think of a file or set of files, some precious, some less so; some important to keep and others not. This is true of the way that they orient to not only a file in the material world, but also those in the digital environment. They mix file types one might say, they 'do' heterogeneity.

The genius of the designers of early systems is reflected in the fact that despite these differences in orientation, the interaction that is undertaken nevertheless succeeds: users by and large get what they want, a file that can be handled as they understand it and a system that can function efficiently and effectively.

File as a Boundary Object

This description is obviously a simplification of file systems design just as it is of the orientations of everyday life. But it is sufficient we hope to convey the claim that a 'file', as a boundary object [5], stands in the middle ground between the user and the computer scientist. It links and

binds computer scientists (and their abstraction) and the user's point of view (and all that implies). The user can orient to digital content as if that content had properties similar to a corporeal file, being an object that can persist, that can be changed, kept and destroyed; and the computer scientist can architect systems that allow them to store the data constitutive of the file-as-seen-by-the-user wherever they wish, and to ensure that system errors do not destroy or corrupt data irretrievably.

These differences might of course be of little importance; beyond these semantics what does it matter that the two views of what a file is are dissimilar? As long as the user can use a file sensibly and the engineer design systems that are effective what is the issue?

Boundary objects succeed when they allow both sides to get on with their concerns without interfering with the other. They start to fail when the clarity of this distinction blurs. Unfortunately, such blurring occurs in the case of a file. Indeed as we shall see, issues, though subtle, surrounding what a *file* means in regard to computers, lead to solecisms more problematic than the distinction between what a file 'is' from the user's point of view and what the system 'writes as a set of data when abstracted as a file'. They lead to nuances across what it means to save one type of file as opposed to another, and even highlight sensitivities regarding whether certain types of object, that we might consider file-like, can be 'saved' at all.

It turns out that when efforts are made to alter the boundary object, problems begin to surface. In the following, we describe previous attempts to alter these two ways of thinking about a file, before demonstrating how an open set of problems has to do with the abstraction that computer scientists orient to, and another has to do with the new ways that users are treating what they think of as a file when they enter the networked, cloud-supported, social networking world of today.

Previous Attempts to Reconstitute Files

We have suggested that in being an abstraction, a boundary object, there is a profound difference between the everyday use of the term 'file' and the use of the same word with regard to systems. Over the years many attempts have been made to bring these different views of a file, the user's and the computer scientist's, closer together, or at least to alter this connection so that it is better able to allow users to do what they want and engineers to support those doings more effectively. In this regard one might say that there have been attempts to reconstitute the boundary object.

More particularly, and until recently, there have been two major and related themes of inquiry on this topic: the first concerning how to make documents on PCs seem more like documents from the everyday human point of view; the second focusing on how to rethink the storage of a file in a way that gets away from treating a file as an entity in a hierarchical order, and moves towards database systems where a file can be entity in more than one place, and with more than one relationship to other files (superordinate or otherwise). The latter can be interpreted as a way of realising the former, and so oftentimes the arguments seem to blur into one another.

Beginning with the first theme: documents. There have been numerous attempts to redesign computer file systems to reflect the ways that people use and understand documents. Xerox PARC's *placeless documents* research project, at the end of the last century, was perhaps the most ambitious and well thought out of these efforts, reflected in subsequent years by other similar efforts. In Dourish *et al.'s* view [1], the placeless documents paradigm can be contrasted with a hierarchical file system. This latter approach typically puts a file in only one place in a hierarchy, irrespective of the fact that users might come to a document from different points of view and concerns (there are exceptions to this with 'multiple directory entry' points being allowed on some systems, for example, and with 'shortcuts' in Windows, as another exception).

For example, a document concerning travel plans might be relevant for budgeting and scheduling. These are distinct concerns. But the nuance of these distinctions cannot be represented if the file is only in one place and where the relationship is always superordinate – where a file must be in either one or other structural order but not both; either a budget representation or a scheduling one. Hence, a hierarchical system is too constrained, the Xerox researchers argue; a more human point of view more subtle. So management of documents becomes conflated in hierarchical systems: a file can be retrieved according to one criterion, as against the multitude of criteria users might apply; the hierarchical location of a file is used to determine back up, not its salience to a user. One could go on.

The placeless documents system, in contrast, made paramount the user orientations to documents. So, for example, since users tend to associate all sorts of apparently ad hoc properties to their documents, the system would similarly allow any property to be associated in the 'file system' (leaving aside what 'a file' is for the moment) when representing a document. Hence, any categorisation the users prefer would be acceptable to the system; any way of collating and bundling would be acceptable too. Furthermore, these properties might be related to a specific user, and so the system ought to allow this too. This is important because the person who uses a document may not be who first created it – and this may be as important as what constitutes the use itself. Metadata that allow and govern access and interaction should reflect this.

All of these arguments can be seen to turn around the idea of getting away from things in fixed places to relationships that link things and doings with those things, from a system that stores a file to databases that create links between entities, some of which may be file-like. Such a view permits the heterogeneity highlighted above, and supports

the position that not all files are equal while moving away from the notion of a hierarchy. In this view, documents are the objects in the PARC system, and metadata the material that allows for the searching, collating and use of those documents.

This leads us to the second theme: a more general interest in the relationship between hierarchical systems (documents being one instance of things in a hierarchy) and databases, where the latter was to replace the former. This interest was common in computer science research at the time of Dourish *et al*.'s efforts. It still is. Some of the titles of well-regarded papers on the topic, such as Seltzer *et al.'s* 'Hierarchical File Systems are Dead' [6], say it all. Much of the research turned around the idea of treating the things represented by a file as an object that exists within a database. Thus configured, the argument went, such an object can be associated and bundled and accessed in all sorts of more complex ways.

This was the view that drove the WinFS effort in Microsoft, as an example, undertaken at the start of the last decade [11]. In the case of WinFS, treating a file in this fashion would allow users to access a file according to a multitude of criteria, to associate that file with other files in equally diverse ways, and to render a file and its relations, graphically, in manners that reflected this diversity. In other words, WinFS would move the Microsoft OS from being file-based towards a database model.

WinFS did not end up being delivered in products, in the next intended release of the Windows OS, however. The release that did occur was of Vista, and this did not radically shift the file system design from what had been previously embedded in Windows NT. Subsequent releases have not changed the design either. One reason for this was that legacy systems would not have much metadata, so although a user might be willing to invest in creating the metadata for a new type of file, they would nevertheless be confronted with another set of historical material that would have no metadata on it. They would have, in effect, databases and files and this would all have to be rendered by the OS as somehow similar if not the same. Besides, many of the main types of file to which people would attach metadata, namely photos and music, would be adequately handled via very specialized applications outside of the Microsoft OS ecology of applications. Getting a common UI to handle the Microsoft experience and that offered by these other applications satisfactorily was thought difficult, if not impossible.

There were, of course, doubtless other reasons but when phrased this way, one can perhaps understand why WinFS never saw the light of day in Microsoft products: designing around the notion of relationships was not sufficient in this case. Nevertheless, in the intervening years between then and now the concept of a file, a file as some kind of an abstraction, has remained visible and further, has begun to shift. It is to that period we now turn.

A File as a Leaky Abstraction

In recent years, there has been an increasing proliferation of devices, and computers have gradually become ever more heterogeneous. It is certainly true to say that when Xerox were looking at their systems, most people (even in PARC) only had access to one device, the machine that sat in front of them, linked via a local network to other identical machines. But over the past ten years or so, this singularity has been replaced by a plurality of devices. People have become used to having many more devices than ever before, and technologies with distinctly new properties. These changes have had consequences for the abstraction that is a *file* in the digital world, which have further ramifications for users and computer scientists alike.

The important point that derives from this change has to do with the fact that these devices are not copies of one another; people have different devices doing different things. They will have a laptop for example, supporting many of the office and work-related tasks that the original Xerox machines were designed to support all those years ago: the creation of texts, reports, memoranda, spreadsheets. They may also have music players, cameras, Flash drives (or memory sticks in common parlance), and so on. Bound up with this change are two further shifts. Firstly, people are dealing with an increasing range of file types, encompassing music (e.g. MP3s), image (e.g. JPEGs), and movie (e.g. SWFs), to name but a few. Secondly, people are encountering their data in a number of contexts, some of which render them in ways that are less 'file-like' than hierarchical systems typically did. The implicit linking of file with application in operating systems such as Apple's iOS, where the underlying file structure is hidden from the user, is a case in point[3].

It is also in this context that the user is increasingly encountering a new term, one describing a raft of services and technologies all of which were meant to let them 'back up their files' – the Cloud. This technology, if single technology it might be, was introduced to the user as offering them new ways of storing their digital stuff and of accessing programs to create that stuff. The Cloud was also claimed to offer new ways of connecting those same people to those with whom they might want to share their stuff with. In much of this discussion the term file was used. And here lies the rub: was this new arrangement of limitless back up and continuous connectivity to be achieved around that same concept of a file as had been critiqued, say, by the Xerox researchers? Or, in the same way that WinFS prioritised relationships, was the Cloud being designed to support complex connections across data, now distributed across different people and different places? If we consider WinFS as one way of underpinning a collection of files that are hybrid and diverse, albeit a way that is ill-equipped for now, we might hope that the bundle of services and applications constitutive of both the Cloud and the things the Cloud will link to, through their common use of the

concept of a file, could offer a way of cohering this diversity.

However, if it was once the case that computer scientists were of a mind that file architectures were a way of dealing with hardware fallibility and that there was a time, somewhat later, when they began to think that database systems provided a way forward, now the world that was being engineered could not be so easily described, however much the word file was used. If one might have said that users are rather capricious in their use of the term file, then any examination of systems encompassed by the desktop PC and the Cloud would say that computer scientists were coming to be lax in a different way. When they used the term, they were evoking not just an abstraction but several, each slightly different; they were designing for a range of devices too, each of which affected what that abstraction stood for. Following Spolsky [8], we might suggest that the abstraction is 'leaky'. The concept of file is an abstraction of certain details that are central to systems design. Mismatch in how those details are abstracted introduce problems. The problems that began to arise in this period (when the Cloud first started being mentioned) arose not because the abstraction was altered; it was rather that it became muddled up and mixed: what a file came to stand for was the problem.

The Elusive "Smallest Allotment"

"...From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file."[7] (p. 372)

While it may be unsurprising to a CHI audience that what users think of as a file could differ from what engineers devise their file abstractions to allow, it is perhaps more unexpected that engineers can misunderstand what the term 'file' means for other computer scientists. The term "users" in the above statement probably refers to fellow engineers. The original file abstraction places much emphasis on a file describing a single unit of data. But, users and engineers operate on different 'units'.

Let us take examples from a productivity suite such as Microsoft's office to examine what a smallest unit could be. Microsoft's Office product offers users a number of different applications. *Word* is perhaps the most well-known. Word uses the file system to store a file. But what a file is within this system, how a .DOC file can be treated and managed as a result, and, relatedly, how that links to or is understood by users of a Word file, is quite different from how, for example, OneNote, another Microsoft product, is designed or how it is understood by the user.

For example, a Word file as conceived of in the application and the OS would seem to be quite close in its abstraction to the way the user understands it. When a user creates a Word file, the application creates a single file too; when a user saves a file, after making some changes, the application saves pretty much the same thing as the user understands. The addressable object, to put it in more technical terms, is close if not identical for both the application and the user. But things start to show some delicate differences when crashes occur. In actuality, Word periodically writes to a hidden temporary file as the user makes edits; when the user saves the document, Word "commits" the changes by manipulating the original and temporary file to make it seem like the original file has been updated. If the system crashes before the user selects save, however, then any changes that a user has made since the last save are not flushed to the original file. This can muddle the user: sometimes the crash loses changes; sometimes it does not.

Now contrast this with OneNote. To the user, a OneNote *Notebook* looks very much like a Word *document*, albeit that it allows the user to add things more easily – images and pictures, cut and pastes from the Web, even their own scribbles which the system can convert through OCR to typescript. But how the application treats the thing that a user thus creates is quite different from Word. For a Notebook is not a single entity, it is not a single file. It is a collection of files. Sections are single files, but the user interacts with Pages, which are components of a section. So, when a user drags an image to a page and the system saves, the Notebook as a whole is not saved, but only the section containing the page currently being worked on. The addressable object, in other words, is not the same as with Word, but it looks like that through the file system. The file system presents the OneNote document as a single file to the user. However, the OneNote document is not a single file as far as the file system is concerned; it is made up of a collection of files.

Be that as it may, one can imagine that a typical user might find any of these differences, the smallest allotment being a Word file or a OneNote page, curious if they found out about them, but not necessarily consequential. The only time they might find these differences consequential is when things do not go as expected during an operation like copy or move. The user's intentions are to operate on the unit as a whole (atomically, as computer scientists say), but the file system operates on a different, lower unit. It is then likely that during a crash, or when things go wrong, the user might puzzle on how the unit ends up being divided, with some parts of it updated, but some not.

So with this understood, when a user sends a Word file to the Cloud, to SkyDrive say, they are likely to understand that the thing that is sent is the thing that they saved and the thing they saved most recently: not the thing that they might have just made changes to before they selected save. In contrast, if they make changes to a OneNote Notebook while they are posting it to SkyDrive, the transfer might well fail. The system will find that it has two versions and this cannot be allowed in its file model. The versions it will have are the one that it has copied and is being transferred

to the Cloud, and the one that is being worked on by the user. The system as a whole loses confidence in the status of its files: which is correct, which is consistent, which does it turn to? The result: a failure in the file directory system.

Now of course, the user may come to understand this. Besides, with OneNote, a file is only a page so it is only a page that is muddled. Thus the user might presume that the rest of the Notebook will be sound. Unfortunately not so. SkyDrive imposes a model of a file on Notebooks that is different from OneNote itself, Instead of a page being a file, a section consisting of several pages is. Thus much more is corrupted when the error of this sort occurs: the failure is greater one might say, though it is never graded in quite this way: a file is either 'good' or not 'good'.

Implications for a Grammar of Action

What this example shows is that the particular concept of a file and the associated assumptions that go with either Word or OneNote are not identical. To coin a phrase from the philosopher Oswald Hanfling, the 'grammar of action' associated with the use of the concept is different in each case [2]. The differences in grammar turn around what is understood by the term file, and relatedly, what action is meant when the term 'save' is used.

Consider these complexities in the light of earlier attempts to design actions around a file, undertaken when file systems were less complex and less diverse. The development of Xerox Star, for example, was predicated on the notion of *generic objects*. These objects could be treated the same way throughout the OS, being manipulated through a set of *generic commands* (move, copy, delete, etc.) that were designed into the system, each performing *"the same way regardless of the type of object selected"*. Smith *et al*. [8] continue, *"They strip away extraneous application specific semantics to get at the underlying principles, and embody fundamental computer science concepts and are consequently widely applicable. This simplicity is desirable in itself..."* (p. 523)

The point that Smith et al. make highlights the importance of the slippage we have described. These differences are significant to engineers who try to link applications like Word to Cloud services, and for those who are trying to design Cloud services for this and other types of user orientated applications: the grammar that each is relying on is different, this simplicity is compromised, and this is because the generic nature of the objects, some of which are files, that are central to systems can no longer be taken for granted. The function of the concept 'file' as a boundary object is failing; it is no longer an intermediary that binds alternative views, but one that muddles them by dint of only seeming to bind.

Let us return to examples relating to the Xerox Star system. Smith *et al*. note an additional *"subtle advantage"* of the simplicity of utilising generic objects mentioned above: *"it makes it easy for users to form a model of the system"* (p. 523). Implicit here is the assumption that users understand the nature of systems and of the doings that these enable: a consistent grammar aids understanding of what actions are available. These actions pertain to the things the system provides, such as 'a file'. The set of generic actions that can be imposed on a file reinforce the perception of that thing, that file as a generic object, as an instance of 'a file'. When the doings that are bound up with files lose this consistency, confusion is likely to ensue: users lose confidence that what they have at hand is 'a file'; engineers too begin to wonder what abstraction they are working to.

A WAY FORWARD

To this point we have put forward the argument that the term file is fundamental to user experience as well as serving as a central concept for computer scientists. It has acted as a boundary object. But as increasingly diverse applications and networked services have emerged, so the reliance on the term file has begun to break down. This stretching shows itself in the grammar of actions associated with various digital objects, some of which are file-like and some not. For example, it is no longer always clear what it means 'to save' – the term can mean something different across applications and even within the same application; it can mean different things with different types of data entities; and all of these and other distinctions are compounded by questions to do with when 'saving' is done to different locations: one's own computer or the Cloud, for example. Does one save to the Cloud, or does one save first to one's PC, for example. If one is a back-up of the other, does synching solve the riddle of what version was saved most recently?

At first glance this might simply suggest a need for increased consistency, in service of the engineering community as well as to support users in understanding these systems. But though laudable, this would be to ignore the fact that users are likely to want to have files and *other* digital types, things which are not file-like. Indeed, looking at the way that new social networking services have been adopted in particular demonstrates that there are now a range of data types that people produce, share and engage with, and these things go alongside what may be thought of as file-like. Given this, reworking the abstraction of a file is only one part of what might be developed, but nevertheless, changes even as regards this apparently partial component of the current world opens up an opportunity for something much bolder. A reconstitution of what a file is could be a *necessary part* of a new grammar of actions. In allowing file-like behaviours, other behaviours become possible through being distinct. It will allow users to navigate and appropriate as they see fit and in ways that suit the current landscape. It will allow users to separate what are postings, say, from what are action records (such as likings and playlists), and those digital phenomena that they have an especial relationship with, those objects that are file-like,

but somehow present and shared in the networked, multi-device, collaborative tagging world of today.

Towards an Abstraction that Encompasses Metadata

"...A file has certain other attributes, which vary from one operating system to another, but typically consist of...[a] name, ...[a] type, ...[a] size, ...access-control information, ...[a] time, data and user identification..."[7] (p. 372)

The first suggestion for a way forward is perhaps the most obvious: it entails rethinking the role of metadata. This is becoming central not only to applications such as OneNote, but also to current technological ecosystems, including recent offerings by Apple, Google and Microsoft, where the application is represented as being bound up with the file. But metadata is also now becoming central to what users understand as a file, though they might not always think of tags, comments, playlist information and so forth as metadata. For what a file is *is* now often bound up with the things added to it, not only by the originating user but by others too.

Consider for example, behaviours reported by [5]. In their study of teenagers and their virtual possessions, participants reported that part of the value of photos posted on Facebook was the metadata associated with them: comments and 'likes' were so pertinent that they were sometimes printed out alongside photos and pasted onto bedroom walls as a collection. This materialisation of the digital is indicative of a difficulty associated with the current technological landscape.

It is not clear how one would digitally export a Facebook photo in order to view it in this way with another computer program or application, and this remains so despite recent innovations in the Facebook service. Yet it is not surprising that users should want to treat these entities in the way they treat a file. If they can upload their photos to Facebook, and given that they do so the photos are file-like objects, why can they not download them again, while retaining the value they have accrued, but still with the benefits of file-like properties? Although it is now easier for users to export their data from Facebook, these exports, once represented simply as 'a file' on a hard disk, lose their potency. They are disconnected from the social life they were bound up with; they are the bare bones of the thing that the original file became when it was posted on Facebook.

An analogy might be helpful. If in the past a file was a single entity to the user, but the system broke the file up in to bits (and bytes) when it came to storage, the value of social networking is to make what starts as a single entity become a 'network of stuff', a composite of the file and the metadata accreted through use on the social network. Hence, when a creator of the original file wants to download the thing that it has become on Facebook, they want to download not the single thing that was the file, but the various objects constitutive of the stuff (entities) that have derived from the social discourse around it. They send up a single entity, and want the system to send back a bundle of bits, whether these be pointers to data of various types, stored in various places, or a large entity, originally called the file but now lined with tags of various kinds. This bundle, this new 'file' type, is not merely a complex data type; the important thing from the users' point of view is that it is a mirror of the social life that the file enables.

Rethinking the Grammar of Copy

This suggests much more than an extension of the scope of the thing 'copied' or downloaded, however. The shift that we describe above, towards an abstraction encompassing metadata, that in itself reflects the social life of the object in question, has a number of implications.

For the sake of simplicity, let us continue with examples taken from actions related to Facebook (and disregarding the variety of actions that are supported by different social networking services). Things like the 'author' and 'place' tags, as well as the 'likes' and 'comments' that can be appended to images and other posts, create rich layers of data on an originating file, which can imbue this file with greater meaning. Reflecting on this, we have suggested that it is sensible for a user to be able to interact in file-like ways with this combination in order to retain this value.

However, this immediately raises complexities. For instance, images posted to Facebook might be copied not only by the person who posted them, but also by others. In these circumstances, should these others be able to copy the metadata, the tags as well as the thing-itself? If so, what of the rights of the owner or, if you prefer, the maker of the initial file? When people copy an originating file, would they be creating a new file or would their new entity be a version of the original one? Is there an order of precedence that we are proposing and ought this to be reflected in the concept of a file that might apply?

It seems to us that there is a distinction that ought to be made between things that are put on the web, which the originator wants to have file-like properties (even as that thing develops a social life once on the web), and those things that are posted that the user does not want to have file-like properties. The properties we are thinking of have to do with questions like whether ' making a copy' means making a copy, a version of the thing itself, or having and owning (as it were) the originating thing itself and all that has ensued in that thing's social life.

The issue here is what grammars of action are implied for these related but evidently distinct objects, some file-like and some not, and how this spills out in terms of the actions possible that mediate the social relations in question. There are evidently subtleties here. We have begun to point out some; it is to others we now turn.

Rethinking the Grammar of Delete

Consider this quote, taken from interviews reported in user studies we have done on the discontinuities between

people's expectations about what they can do with their digital material and what they can in fact do when they place it on social networking sites.

"I guess I can delete them (photos on my computer)… online, well I can try to delete something but who knows? Who deletes the deleted? Where does it go when I delete it? I don't know but I don't think it disappears and that way it feels like I don't have control over it…"

What is implied here is the possibility, conceived from this user's point of view, that a digital object is something that can be done away with. At least, this is their understanding of what seems to happen when they interact with things – certainly this is their understanding of what happens when they interact with their PC (notwithstanding the subtleties of this for the moment).

This individual is making a contrast however, between interacting with their PC and when they venture elsewhere, onto Facebook (or Flickr, say, for want of another illustrative context). Though only one person's account, it seems to us that this can be taken as representative of the view held in common. After all with a computer, the abstraction representative of a file has for some years now allowed a user to treat a file in this way: as something that can be done away with. Never mind that computer systems have not been designed so that a file is truly eviscerated when the 'delete' command is selected (instead simply readdressing the digital space used by the file in question). For the purposes of the user, this is sufficient for them to get on with their doings: for their practical intentions, their delete action does way with the file.

How different the situation is as both regards this basic interaction and the essential status of the thing filed and-or deleted when the 'place' that this interaction is occurring on is remote, on some server, either the Cloud or on some social networking service. It is in this sense that the doubts that this interviewee expressed, though tentative, are accurate. They are right to ask, albeit rhetorically, 'just what does happen when delete is selected?' 'What is implied here?' They are asking, 'What is left unstated but necessarily relied upon when I press delete?' Their understanding, as represented in this single quote, is not naïve so much as too knowledgeable: just as they understand that on a computer, to delete doesn't mean to completely eviscerate or destroy a file, so now they worry that same will apply in this new landscape.

It is precisely because of issues such as these that the grammar of action that was devised for the PC cannot be the solution that is applicable for the current multiple device, cloud-linked, multiple file type, social networking world we have now. Something more is required than was true in the past if one is to copy or delete in this new context. The grammar must imply more.

What is needed is not only a file abstraction through which the user's desire to hold on to the metadata that makes their files meaningful can be encompassed, when a file gains what one might say is its 'social life'. It is that, in addition, this thing, distributed as it is, can nevertheless be done away with, removed, taken out of play, ended. A boundary object needs to be developed that can bridge the abstraction of the user and the one of the engineer, who needs to worry about where this thing that keeps growing and changing, and where the locale of storage changes too, such that when a user says 'delete', the thing whatever it is and wherever the entities constitutive of it are, are indeed, done away with.

Expanding the Grammar of Action: to Own

These examples of the thing that is a file, of the copying of that thing and, last of all, the deleting of a file, show how reconstituting a file abstraction needs to be done mindfully. Nevertheless, one might say that these are still actions that resonate with the world that existed when engineers at Xerox were developing the Star. But the world is much more different than is suggested by the continuing applicability of these terms. Numerous technological shifts are already underpinning various actions that were not possible then: 'synching' and 'streaming' are amongst this novel set of behaviours. Devising a new file abstraction also requires that some actions implied but not stated in the original concept of 'a file' now require explicit attention in ways that would have startled the Xerox engineers more than the introduction of the concept of 'synching' would.

Take the following two quotes, from two different participants in our user studies, as illustrative here:

"the more I talk about it, the more the idea of owning something online seems lost in translation."

"it feels like there is this illusion that they are mine, that I own them. But they could disappear at any moment."

These quotes are suggesting that the relationship a user can have to digital stuff can be one where *ownership* is applicable. At first reading one might think they are alluding to digital rights management. But further reflection brings this into doubt. They appear to be thinking of something that they had been able to take for granted hitherto, something that went hand in hand with their understanding of what a file is.

Later on in these same interviews, these participants talk about how it used to be that they knew where a file was. They stated that they used to have a desire to put a file "*on a CD"* so that "*it could be safe"*. But they go on to say that they find that this is hard to do in the context of Cloud storage. The reasons why they wanted to do this (before we remark on the difficulties) were that, for them, where a file is could act as an instrument of ownership. Being "*here",* "*on their PC",* or *"in a CD",* could make it *"theirs"*. That they can see *"it is here"* could assure them that their ownership has not been violated.

What is being pointed towards is a set of assumptions, relating to the functioning of a file on PCs that harks back to the discussion of deletion above. The thing that can be a file, and hence the thing deleted or in this case owned, is treated as if it has a physical locale, a knowable place where it lives. This somewhere used to be (of course) "*there"*, in a particular machine, on their desk at home or at work. But when it comes to the current world, where this 'there' might be is no longer clear; there is often no knowing where a file is, certainly from a user's point of view.

What follows on from this is the possibility that what once was taken for granted can no longer be. Before one's ownership of digital data was manifest in the physical presence of the devices that housed that data. Now that proxy relationship no longer applies. And users are right to wonder about what ownership means in this new context. Amazon's continuing efforts to specify how Kindle users can lend each other (whose?) books highlight the complexities in this space; it certainly doesn't offer a way forward and out of them. We think that a new concept of what a file might be does: ownership is what we are thinking of, when ownership stands as proxy for what used to be knowledge of location and responsibility for that location. What was once a relationship between a user and a physical thing now needs to stand as a relationship between a user and a digital thing. Just what this ownership might be and how it might function in terms of what is specified in this new entity we are thinking of, one that somehow has the properties we have described above and which also allows this new characteristic, we have begun to outline but a beginning is all it is. Much more needs to be done.

CONCLUSION

Whatever future work does need undertaking – and there is obviously plenty of opportunities here – these examples have been presented to assert our view that users sometimes want a particular type of digital entity. This entity needs to let them do certain things, a particular job. A new version of what users think of as a file can let them do this, we have proposed. But we are also proposing that this new entity needs to be engineerable, too. The thing that will result may well not look a file as conceived of in file architectures; nor will it have quite the same assembly of interactional features, the same grammar of action as we have put it, as a file on a PC.

We have noted that the devising of a new abstraction needs to be done in a way that is cognizant of the grammar of action that it will imply. Enduring actions, such as copy and delete, need to be re-thought, and new actions may be needed, for example to provide a sense of ownership of data. A new abstraction might allow users to *eradicate* a file that is stored in the cloud, or *withdraw* one from a social network. It might allow them to knowingly *place* a file in a particular location, one that is tied to a physical locale. It might resolve issues surrounding the *loaning* of digital media. Although we conclude with these suggestions, we make them tentatively. A new abstraction, and a new associated grammar of action, will require a good deal of thought and experimentation. In this case, diligent HCI research is warranted more than ever.

ACKNOWLEDGMENTS

We would like to thank our colleagues at Microsoft for discussing with us several of the issues raised in this report.

REFERENCES

1. 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 Trans. Inf. Syst. 18.
2. Hanfling, O. 2000. Philosophy and Ordinary Language: The Bent and Genius of Our Tongue. London: Routledge.
3. Harter, T., Dragga, C., Vaughn, M., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H., A File is Not a File: Understanding the I/O Behavior of Apple Desktop Applications. In Proceedings of Symposium on Operating System Principles (SOSP). 23-26 October 2011. Cascais, Portugal. (to appear).
4. Marshall, C. 2007. McCown, F., Nelson, M. 2007. Evaluating Personal Archiving Strategies for Internet-based Information. Proc. of IS&T Archiving, 151-156.
5. Odom, W., Zimmerman, J., Forlizzi, J. Teenagers and their virtual possessions' (2011) In CHI '11. ACM, New York, 1491-1500.
6. Seltzer, M. and Murphy, N. 2009 Hierarchical file systems are dead. In Proc. HotOS'09. USENIX Association, Berkeley, CA, USA.
7. Silberschatz, A, Galvin P.B and Gagne G. (2002) Operating system concepts, (6th Ed). Wiley, New York.
8. Smith, D. C et al, 1982.The Star Interface: An overview, in Brown,R & Morgan,H. (Eds) AFIPS'82, pp 515-528.
9. Spolsky, J. 2004. Joel on Software. Apress.
10. Star, S.L. 1989, The structure of ill-structured solutions: Boundary objects and heterogeneous problem solving, Gasser & Huhns, Eds, Distributed AI, Vol 2, pp37-54.
11. WinFS. Wikipedia, accessed in 2011. http://en.wikipedia.org/wiki/WinFS.
12. Wittgenstein, L. 1953. Philosophical Investigations. Trans A. N.Anscombe. Oxford: Blackwell.
13. Xerox Star. Wikipedia, accessed in 2011. http://en.wikipedia.org/wiki/Xerox_Star